Fintrix: An Agentic AI Powered Customer Service Solution for the Banking Sector

SAIM CHISHTI
01-136221-045
MUHAMMAD TAHA HASNAT
01-136221-018

Supervisor: Dr. Faryal Nosheen

Bachelor of Science in Artificial Intelligence

Department of Computer Science
Bahria University, Islamabad

October 2025

Certificate

We accept the work contained in the report titled "Fintrix: An Agentic AI Powered Customer Service Solution for the Banking Sector", written by Mr. SAIM CHISHTI AND Mr. MUHAMMAD TAHA HASNAT as a confirmation to the required standard for the partial fulfillment of the degree of Bachelor of Science in Artificial Intelligence .

Approved by . . . :

Supervisor:

_____

Internal Examiner:

_____

External Examiner:

_____

Project Coordinator:

_____

Head of the Department:

_____

Abstract

The fast pace of adoption for AI is creating a new landscape in financial services where intelligent automation, improved decision-making, and powerful defenses against increasingly sophisticated digital attacks are now possible. This thesis introduces Fintrix, anAgentic AI-wielding intelligent banking assistant that consolidates various AI functionalities in a unified, self-contained entity. Fintrix is composed of three main modules: (i) a Customer Support Agent which provides efficient, contextual and personalized counseling using advanced natural language understanding; (ii) a Financial Advisory Agent, based on predictive analytics, time series modeling and behavior analysis that offerspersonalized financial advice and insights; nd (iii) a Fraud De- tection Agent which employs machine-learning techniques together with anomaly-detection methods to recognize fraudulent activities in real-time. Fintrix is grounded on modular, scalable and explainable (XAI) methods for natural language understanding with the intuition behind Agentic AI, Large Language Models (LLMs), andXAI approaches. These are just a few possible advantages of suchan inter-relation of operations, which provides the systems with self-management to perform ultra complex economic operations on their own responsibility and anyway always give content clear and understandable to the end user. Synchronous running of multiple agents "Fintrix ensures that rendezvous between the state rooms can be accomplished bad decision making and overall accounting overhead onthe bank's computer. In summary, Fintrix demonstrates the practicalitiesand relevance of a decentralised MAS approach in the modern financial sector. It also demonstrates how Agentic AI drives improved customer service, better riskmanagement and data-driven financial planning

Acknowledgments

We are extremely thankful to oursupervisor Dr. Faryal Nosheen whose tremendous support in every step and valuable comments all along the thesis was a great favor for us. We are also very thankful to Department of Computer Science, Bahria University Islamabad for creating an academic environment along with technical resources and research infrastructureneeded for the completion of this project. The challengingcurriculum and hands on learning approach of the department played a key role in shaping our skills and teaching us how Agentic AI systems are designed. Furthermore, we acknowledge the faculty members and administrative staff who contributed indirectly by maintaining a supportive and well-organized academic structure. Their efforts ensured that we had access to the tools, libraries, and facilities required for the successful execution of this research. This thesis represents the collective effort, dedication, and academic support we received throughout our degree, and we are grateful to everyone who played a role in enabling the completion of this work.

SAIM CHISHTI
Islamabad, Pakistan

MUHAMMAD TAHA HASNAT
Islamabad, Pakistan

October 2025

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

AI	Artificial Intelligence
AML	Anti-Money Laundering
ANN	Artificial Neural Network
API	Application Programming Interface
CI/CD	Continuous Integration / Continuous Deployment
CNN	Convolutional Neural Network
FAQ	Frequently Asked Questions
GDPR	General Data Protection Regulation
GNN	Graph Neural Network
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
LIME	Local Interpretable Model-agnostic Explanations
LLM	Large Language Model
LSTM	Long Short-Term Memory
MAS	Multi-Agent System
ML	Machine Learning
PCI DSS	Payment Card Industry Data Security Standard
RAG	Retrieval-Augmented Generation
RAM	Random Access Memory
ReAct	Reasoning + Acting
REST	Representational State Transfer
RNN	Recurrent Neural Network
SHAP	SHapley Additive exPlanations
SVM	Support Vector Machine
UI	User Interface
UML	Unified Modeling Language
UX	User Experience
XAI	Explainable Artificial Intelligence
XGBoost	Extreme Gradient Boosting

Chapter 1

Introduction

1.1 Background and Motivation

The financial services industry is in themidst of a rapid digital shift because customers demand better service, security rules keep getting more rigorous and banks feel pressure to do things more efficiently. Legacy banking systems are often incapable of keeping pace — theyare slow to respond, provide very broad support to customers, and do not fully rely on numbers. These limitations erode customer experience andconfidenceespecially without timely, personalized helpbeing provided to individuals [1, 2]. Meantime, financial It's becoming moresophisticated, and that users as wellas the banks are more exposed than ever. state of art fraud detection solutions are showing promising results, however the majorityof them either at or out-of-the box and deployed in most industrie s.func tiona lly only applicable to the financia l services sector. not specific enough andthey'rebeen working in isolation thus are capable to detect dodgy behaviour after the event, while still leaving banks vulnerable to new methods offraud. attack [3, 4]. The majority of the traditional static content driven advisory tool basedsystems are depended on historical data. and rules-based reasoning,making it challenging to offer individualizedfinancial advice. or product offerings, as theydo not utilize up-to-the-moment behavioral information [2]. Taken together, all these factors—lack of individual personalization, execution inefficiency and risingrisks for fraud underscore theneed for smarter systems that can adapt and be. virtually self-sufficient with potential for concurrent banking activities[5, 6].

1.2 Problem Statement

Even though artificial intelligence is being used in different parts of the financial industry, most of its applications are still very limited. Customer service bots usually answer only fixed or common questions, fraud detection tools often work as separate systems, and many advisory platforms still can't adjust to real-time user behavior [3, 2]. Because everything is

so separated, a clear research gap appears: There is still no unified, agentic AI framework that can handle customer support, financial advisory, and fraud detection together in one connected and mostly autonomous system. This gap is what motivates us to work toward a multi-agent, autonomous platform that can offer more secure, personalized, and efficient banking solutions while also reducing how much human involvement is required.

1.3 Proposed Solution: Fintrix

To address these limits, this thesis introduces Fintrix, an agentic AI–powered banking assistant that brings customer support, financial advisory, and fraud detection into one connected platform. Fintrix largely runs autonomously based on a multi-agentarchitecture inspired by ReAct (Reasoning+Acting) approach [5, 1]. The Customer Support Agent manages stuff such as card failure/dispute, balance checking and simple technical troubleshooting through natural language understanding, whereas the FinancialAdvisory Agent leverages predictive analytics and behavior modeling to provide more personalized real time suggestions that adapt to changes in users' behaviors. Fintrix, for detecting fraud, performs rapid analysis leveraging curated knowledge and historical transactions using a lightweight XGBoost modelto identify patterns of interest. XGBoost was chosen because it's fast, scalable and stillexplainable, which is beneficial for real-time banking, even though models like LSTMs or GRUs might achieve slightly higher accuracy but also add delay and extra complexity. Fintrix is in general that they work standalone but still brings the same agentic workflow ideas and ambition to put more decision powerinto something Publisher's hands.

1.4 Technical Challenges

Developing Fintrix has presented a challenging task, because it requires confluence of agentic AIwith autonomous reasoning and a rigid financial-domain logic. A key problem is multi-agent coordination, because the three core modules work independently but they depend on processing and passinq urser turn input/render results correctly and comply with its ownflow meanwhile maintain conversation consistent context during all interaction (which for that its merely hard to control) [1, 3]. Real time reasoning is yet another issue since, eachworkflow has to form its interpretation promptly about what the user intends to communicate and fetch the appropriate knowledge and choose a proper tool or model, with very low latency (for friction-free banking [5]).

We also struggled with trackingsession and state since personalization relies on maintaining fairly large memory over many turns in the conversation. As Rau put it, "baked into all that is this omnibox for integrating with knowledge base whichyou don't want to pull everything plus the kitchen sink: the laws and rules and policy but also transactions

history. . . but without slowing your system down." And, inevitably privacy and compliance makeit that much harder; when you're handling sensitive financial data you do have to ensure that you respect the likes of GDPR and PCI DSS. Securing the proper implementation of encryption, access control and audit logs insystem security is important as it can directly influence integrity [6]. Nevertheless we see in preliminary trials impressive results over indicia like fraud prediction accuracy, customer support response time or quality of financial advices which show that Fintrix performs well with management situationsover realistic banking environment.

1.5 Research Contributions

This thesis has several implications to AI-enabled technology for agentive systems (systems in which an agent and designer are clearly distinguishable) and autonomoussystems. First, it provides a convergent multi agentmodel which can consider both customer support as the single thing and financial advice, fraud detection on a related scenario as opposed to each of the tools separately. We also have a real time fraud detection pipeline with the help of human curated knowledge, transaction pattern analysis andfast xgboost which is more active anomaly warning system. And another contribution is provided in the advisory module with which we make use of predictive analytics, machine learning and time-series to provide richerand more personalized financial insights that adapt with usage. "I think a lot of this is very much tied to the customer support rep," which draws on language models so it can have a response that sounds more natural — and perhaps somehow more apropos to whatever the user is asking for —than answers that are simply regurgitated from prewritten ones. And the system is also very deliberately designed withan eye toward configurability, that new features of a banking nature can be added without too much fuss, and it's intended to accommodate future enhancements like explainability tools as those are needed.

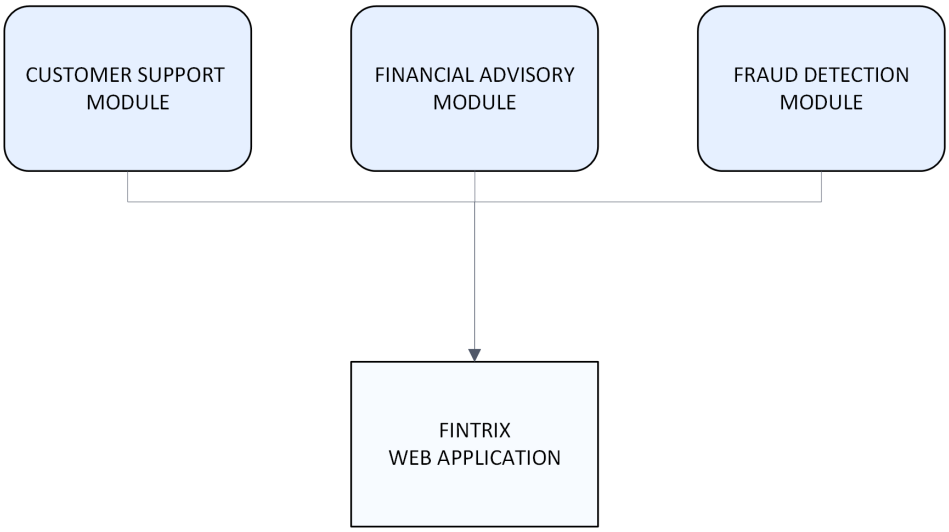


Figure 1.1: High-level overview of the three independent Fintrix modules

Chapter 2

Literature Review

2.1 Introduction to the Literature Review

One of the forces that has been remaking the financialindustry over the past several years is artificial intelligence. Banks are alreadyapplying AI to automate routine work, process big data and create new digital services. At the sametime, there are growing expectations from customers for round-the-clock support, greater personalisation of products and stronger protection against fraud and cyber attacks. These new pressures are becoming much harder for traditional rule-based and siloed old-systems to respond as quick as change [2, 7].

Re: point 1 : From a macro perspective, AI in finance is by nowa fairly mature research topic. Cao gives a thorough survey of different AI technologies for finance, including their historical ML and DL algorithms as well as the new ones insuch applications as trading strategies, risk management, credit scoring, and compliance [8]. The review also touches on perennial issues like cumbersome and volatile data, burdensome regulations and the need for models that remain robustin fast-moving markets. All of that really matters when an AI model needs towork in the wild west of banking instead of the sanitized lab.

Deeplearning has expanded that further still. Ozbayoglu et al. have observed that "deep models form the backbone of systems used for trading, portfolio optimisation and risk assessment up to fraud Socher et al (2013)." Unlike traditional methods, these deep models are more accurate but also less- interpretable economic transaction analysis when only shallow handengineered features wereconstructed." Some other researchers have further emphasized that deep models become increasingly deeper and thus expensive as we train better [9].These modelsdo capture transactions data patterns that humans tend to miss, but they also come with all sorts of luggage — things like slower reaction times, explanations that don't make a lot of sense and models behaving strangely when the data moves around just slightly. Because Fintrix attempts to blend fraud checks with advisory work and customer support under one roof, we had to somewhat juggleaccuracy, speed and how comprehendible the system remains, which is not always an easy task. A few newerstudies

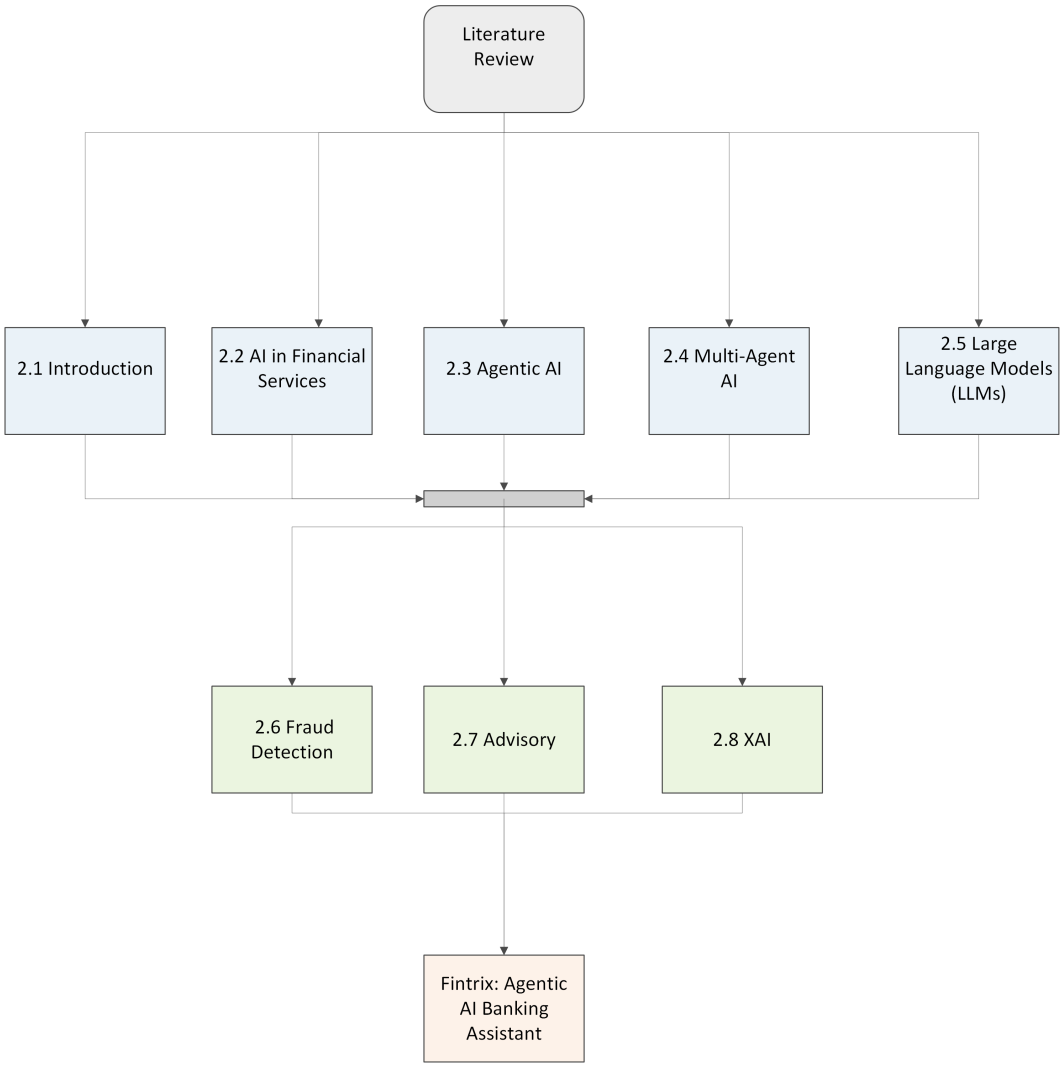


Figure 2.1: Structure of the Literature Review

are solely dedicated to AI in banking. Fares et al. addresses AI applica-tions in the banking domain and categorizes previ- ous works in three themes: strategy, process, customer [7]. Their research reveals that banks apply AI mainly inchatbots, credit scoring and back-office processes as well as risk management, however many of these tools remain narrow-focused and expert-oriented. Godolja and Domi also find that AI has been shiftingfrom a pure decision-support role to more integrated systems of support which affects most areas within the banking value chain [10]. These studies reveal that the banks know theyneed AI to compete but are held back by challenges including legacy systems, skills shortages and issues around trust. Explainability has now emerged as another hotarea. I as AI models shape high-stakes decisions —whether to block transactions, approve or deny loans— regulators and users are demanding that companies explain why a decision was made. C˘ ernevic˘iene˙ and Kabašinskas presenta review of explainable AI in the financial sectorand claimed it not only to be regulation required but a key factor for trust and adoption [11]. . Their work shows that XAI is being applied in credit risk, fraud detection, and portfolio management, but there is still a gap between technical explanations and what non-experts actually understand. This gap matters a lot for agentic systems like Fintrix, which need to explain decisions to customers as well as internal bank teams. Industry reports also show that generative AI and more autonomous "agentic" systems are starting to move into real-world deployment. NVIDIA's *State of AI in Financial Services* report notes that many financial institutions already use generative models for document processing, research, and personalised banking [12]. Similar reports from McKinsey and EY show that most large banks are running or piloting generative AI projects, and interest in agentic AI—tools that can plan, call external systems, and manage workflows—is rising quickly [13, 14]. In this context, Fintrix is positioned as an integrated agentic AI banking assistant that brings together customer support, personalised advisory, and real-time fraud detection in one coordinated system. Instead of treating these tasks separately, Fintrix uses a multi-agent design built around large language models and lightweight machine learning components so the modules can share context and automate more complex workflows. The rest of this chapter expands on this background and reviews related work in five areas: AI in banking and finance, multi-agent and agentic AI, AI-based customer service, AI-driven financial advisory systems, and fraud detection with a focus on explainability and reliability.

2.2 AI in Financial Services

Artificial intelligence has slowly shifted from being just a supporting tool to becoming a core part of how modern financial systems work. Banks, insurance companies, payment networks, and investment firms now depend on AI to make sense of huge amounts of transactional data, automate repetitive tasks, and help with bigger strategic decisions. Compared to older financial automation—which mainly used fixed rules and simple

Table 2.1: Key research streams relevant to Fintrix

Research stream	Typical focus in literature	Relevance to Fintrix
AI in finance	Surveys of AI techniques for trading, risk, credit scoring, portfolio management [15, 16]	Provides overall toolbox and limitations for financial AI models.
AI in banking	Systematic reviews of AI use cases in banks: chatbots, process automation, risk [17, 18]	Shows where banks already use AI and where gaps still exist.
Deep learning in finance	Application of CNNs, RNNs, transformers etc. to market data and transactions [16]	Informs model choices for fraud detection and behavioural analysis.
Explainable AI in finance	Methods to make black-box models more transparent and compliant [19]	Supports the need for interpretable fraud alerts and advisory explanations.
Generative & agentic AI in banking	Industry reports on genAI pilots, autonomous agents, and scaling challenges [20, 21, 22]	Motivates a multi-agent, LLM-driven architecture like Fintrix.

prediction methods—today's AI mixes machine learning, natural language processing, and real-time analytics to build solutions that are more adaptive and aware of context [23, 24]. This section looks at the key developments shaping how AI is used across financial services today, including how it's being adopted, where it's applied, and the main opportunities and challenges that come with it.

2.2.1 Evolution and Adoption Trends

AI use in finance has grown fast over the past decade. Early systems focused on credit scoring, document sorting, and basic fraud detection using traditional machine-learning methods. These worked but struggled to adapt to fast-changing financial conditions. According to Fares et al., institutions began moving toward deeper automation around 2015, adding conversational agents, behaviour analytics, and model-based risk tools [17]. More recently, deep-learning models—such as LSTMs, CNNs, and attention networks—have entered financial workflows for forecasting, fraud detection, and market prediction [16]. They offer better accuracy but also bring issues like high compute cost and weak interpretability. Industry reports from NVIDIA, McKinsey, and EY confirm that AI is now used across customer service, compliance, lending, and personalised banking, making it a core part of modern financial operations [20, 21, 22].

2.2.2 Major Application Domains

Several main AI areas consistently appear in financial services.

Customer Service. AI chatbots handle routine questions and guide users through digital banking. LLM-powered agents now offer more natural and context-aware responses [17, 18].

Credit Scoring. Machine-learning models and alternative data improve loan evaluation, though fairness and explainability remain major regulatory concerns [19].

Fraud Detection. AI models analyse large transaction streams to detect unusual patterns, with newer graph-based methods helping uncover organised fraud [24].

Trading and Investment. Deep learning and reinforcement learning support prediction, sentiment analysis, and portfolio automation, though concerns around stability remain [23].

Compliance and Back-Office. AI supports AML checks, document review, and regulatory reporting, reducing false positives and analyst workload [21].

2.2.3 Opportunities and Challenges

AI brings a lot of benefits to financial services, like better predictions, quicker decisions, and more personalised customer experiences. But the literature also points out several issues that keep coming up. Černevičienė and Kabašinskas stress that explainability is especially important in areas like lending and fraud detection, since unclear or wrong decisions can easily lead to mistrust or even regulatory trouble [19]. Banks also still struggle with things like poor data quality, models drifting over time, security risks, and the difficulty of connecting new AI systems to old legacy setups. Overall, these developments create the background for systems like Fintrix, as the industry moves away from narrow prediction tools toward more autonomous and explainable AI that can combine customer interaction, advisory tasks, and fraud monitoring in one place.

2.3 Agentic AI

Agentic AI has become a hottopic in AI research because it does more than make predictions. In contrastto older-style ML models that end at an output, they can plan tasks, take actions, call out tools or APIs, change what they're doing based on feedback [25]. "It makes so much more sense forfinance, where most of the workflows are extremely messy and decisions need to happen usually now," he said.

Most work says Agentic AI operates based on three main principles: it forms a model of the human user's desired goal and decompose that into a sequence of finer grained actions;

it relies on cognitive simulations performing inference in different tools or databases as required to make progress toward the overall goal; and it maintains an ability to revisit its plans if some external input changes [26]. This is a lot more flexible thanthe older rule-based agents.

It is LLMs that have accelerated thisspace even further. With more powerful language understanding and reasoning, agents are able to follow instructions,perform multi-hop reasoning, and pull in external info when the task demands it. Li et al. show agents can solve much more complicated taskswith LLMs in the loop [27], and Xi et al. show that modern planning + LLMs outperformearlier agent architectures in both accuracy and task success [28]. That's partly why you can find many of them working in banking orrisk management, where situations change constantly and simple rules are woefully insufficient.

Banks have already been deployingagentic interfaces for customer service, fraud verification checks, document assistance and personalized recommendations. Multi-agent applications such as regulating goods packets [29] and automated loan processing or delivery ofan e-service across a number of government departments are now being deployed. One agent, for example, might be able to look at a customer's history, run a scoringmodel on her and check rules in looking over the deal, and then put out a brief summary of all that would-be user-generated report on its own.

But there are problems too. Agentic AI is not controllable, soresearchers emphasize the requirement for guardrails — boundaries on tool use, logging and with continuous monitoring and controls to prevent hallucinations from swaying crucial decisions [30]. Explainability is also very importantsince banks need to explain why their system made a decision.

In sum, Agentic AI is a move beyond mere tools for prediction toward systems that can plan andact on their own. This is an excellent fit for Fintrix as it relies on orchestrating multiple smartmodules. As banking becomes more sophisticated, agentic systems provide a flexible platformfor future AI applications.

2.4 Multi-Agent Systems in Financial Services

Multi-agent systems (MAS) consist on multiple independent agents moving in the same area, and they come tobe more intersting when they work somehow together. Instead of using one big centralplant agent, MAS opposed the control problem in to smaller individualagents that have been independent agents and duringtesting scenario we can increase or decrease the numbers of internal knowledge bases it was said also (wooldridge2021multiagent) splitting the work across several entities.It often make system scalable as well as a bit more robust when interval processing will be malfunction [31].

MAS is pretty well-suited for banking, because financial services is already about a lot of things happening partaking at once — transactions, fraud checks, customer

queries, regulatory guff, investmentsuggestions — being managed alongside each other. Thereis rarely a single model to do it all seamlessly. With MAS, a single agent can monitorsuspicious activities, another one checks customer's data, one sums up a document, etc., then sharing the information via simple communication channel [32].

The great thing is that MAS allows you to perform tasks inparallel. Sowhen a high-risk transaction appears, one agent examines the history of such transactions, another views credit behavior, another executes an anomaly detection model and someone else checks compliance rules. Together they provide a quicker and more comprehensive answer than one model making an attemptto do everything in sequence, which is why MAS are popular for fraud-detection, compliance, credit-scoring etc [33]. And banks appreciate that they can bring on new agentswithout having to redesign the whole system.

Morerecently, MAS have been complemented with LLMs (Jeckmans et al., 2017), by bringing language to the coordination and control of things using a "language-based agent does the telling what to do" ap proach so that other agents may focus on heavy analytical tasks. Research has shownthat this makes cooperation easier and reduces conflicting or duplicated work, which could be very useful in onboarding flows, support escalation and even investment research [34].

There are downsides too. It's not always easy coordinating amess of agents, particularly when they don't have full information and may even have slightly different objectives. Communication overhead can increase rapidly,and the explanation of decisions taken in a MAS can become messy due to many small agents forming an answer. For finance—which essentially needs explainablity andaccountability—this is a genuine concern [35]. Nevertheless, MAS are a good fit for systems like Fintrix whenthey get the correct structure where customer service, advisory and fraud checks need run in parallel that still makes sense together.

In general, MAS are a flexible and quite powerful method, leading to intelligentfinancial systems. Their distributed style aligns well with how banks already operate,and the more complex AI services become, the more likely that MAS will serve as a hub.

2.5 Large Language Models (LLMs)

Models like GPT-3, PaLM, Claude, and LLaMA basically opened the door to much more advanced capabilities. They can summarise documents, extract info, answer questions, write structured content, and even solve math when paired with the right tools. Studies show that LLMs handle many tasks without needing retraining, so banks can often use one model for customer support, document review, compliance checks, and a bunch of other workflows [36]. This makes them a strong base for intelligent banking assistants.

Recent research also looks at LLMs as autonomous decision-making parts. Mialon et al. show that LLMs can plan tasks, critique what they wrote, and refine answers when

Table 2.2: Literature on Agentic AI Systems

Study	**Focus Area**	**Key Contributions**	**Limitations**
OpenAI (2024) [25]	Agent Architecture and Safety Guidelines	Defines core principles for building safe and controllable agentic systems; identifies planning, tool use, and self-correction as foundational behaviours.	High-level conceptual guidance; lacks empirical results or benchmarks.
Wang et al. (2024) [26]	LLM-Based Agent Reasoning and Tool Use	Comprehensive survey covering reasoning, planning, tool-use pipelines, and evaluation strategies for LLM-driven agents.	Still theoretical; limited real-world financial applications analysed.
Li et al. (2024) [27]	LLM-Augmented Agents	Demonstrates how LLMs improve task decomposition, multi-hop reasoning, and adaptive planning in agent workflows.	Models suffer from hallucination risks; requires guardrails for high-stakes domains.
Xi et al. (2023) [28]	Planning Algorithms with LLMs	Shows that hybrid planning approaches outperform classical agents in complex reasoning tasks; proposes new LLM planning benchmarks.	Focuses mainly on synthetic planning tasks; limited coverage of enterprise use cases.
Accenture (2024) [29]	Industry Adoption of Agentic AI in Finance	Reports real-world use cases such as autonomous customer support, regulatory workflows, document automation, and multi-agent pipelines in banks.	Industry-focused; does not provide technical evaluation or model-level performance metrics.
Weng (2023) [30]	Agent Alignment and Reliability	Explains risks of autonomous agents, discusses alignment challenges, and proposes control strategies for real-world systems.	Opinion-based review; lacks controlled experiments or quantifiable measures.

supported by memory or reasoning frameworks [37]. This shift—from just predicting text to actually "reasoning and acting"—is what enabled agentic systems, where an LLM coordinates tools and other agents. In financial settings, this means an LLM can start a fraud investigation, analyse customer behaviour, generate advisory insights, or pass a tricky case to a human.

LLMs are also becoming common in customer service. Since they understand context better, they can manage long conversations, track intent, and keep the flow consistent. Industry reports say LLM-based chat systems reduce call-centre load and improve customer satisfaction because responses feel quicker and more personal [38]. Analysts are also using LLMs for research—summarising reports, pulling data, and helping with risk assessment.

But there are challenges too. The biggest one people mention is hallucination—cases where the model sounds confident but gives wrong facts. In finance this can be risky, especially with regulatory documents or high-stakes decisions. LLMs also need strong safety checks, continuous monitoring, and domain-level tuning to stay reliable. Another issue is traceability: regulators expect clear explanations of how a decision was made, but LLMs don't always offer that level of transparency [39].

Even with these concerns, the literature shows LLMs have strong potential when used carefully. Their reasoning, context understanding, and flexibility make them key parts of agentic architectures. In Fintrix, the LLM acts as the main reasoning engine—it reads user inputs, coordinates the other modules, and explains advisory or fraud-related decisions. With proper guardrails inside a multi-agent setup, LLMs become a powerful foundation for intelligent financial automation.

2.6 Fraud Detection in Financial Services

And fraud is, if anything, more common and, I won't say less blatant, actually I will —it has become both subtler and obviously worse. Many banks instead rely on old-style rule-based systems with hard-coded limits or blacklists: easyto create rules for, but not powerful enough when fraudsters change their tricks. These systemsare flooding with false positives, annoying legitimate users and increasing the cost of investigation [8, 9]. Even as such, regulators won't settle for not just aset of handmade rules; they want fraud controls that are stronger and explainable.

Machinelearning models were also proposed for this purpose. Reviews of existing work reveal that SVMs, random forests and boosting models are heavily used in credit-card fraud detection, insurance claims analysis and other financialapplications [8, 9]. They often outperform rules of thumb-driven systems in terms of accuracy, but nonetheless face challenging sub-problems such as extreme class imbalances, concept drifts and the necessity to operate at extremely low latencies especially for real-time screening [6, 8].

Deep learning pushed things further. CNNs, LSTMs/GRUs and Transformer-style models learn time series patterns such as “a strange device sequence” or “odd sequence of merchants,” but are even more “black box”, hence it is difficult for analysts toexplain why a payment got flagged [6].

An additional rapidly spreading practiceis the representation of financial data as networks. Many fraud schemes rely on connections— common devices, mule accounts, related merchants — not a single dubious transaction. Graph-based techniques and GNNs leverage this concept, and recent studies indicate they are capable of enhancing the detection accuracy in densely connected financial networks [10]. This is particularly important for digital banks, where the same fraud ring may operate across cards, wallets and instantpayments.

At the same time,accuracy is no longer enough. Bankswant to use fraud models that are effective and also explainable. XAItoolssuch asSHAPandLIME are employed to generate

an investigator-understandable explanation at the case level [11]. Even some more recent works also include federated learningwith XAI for the banks to be able to train shared models without opening their raw data but still demonstrating interpretable local explanation [12].

“With Fintrix we borrowed from some of these newer deep learning and graph based approaches for the Fraud Detection Agent but then you still end up presenting risk scores, key features, simple narratives soour analysts don’t feel like they are working with a black box,” Demarest said. The aim is to land in the middle: more advancedthan old rule-based systems, but not so opaque as to obstruct day-to-day compliance and fraud-team work.

2.7 Financial Advisory and Personalisation

Personalised financial advice has turned into a big deal in digital banking, mainly because people now deal with banks through apps and websites instead of sitting with an advisor. Older advisory tools were really basic, mostly rule-based stuff like “age = this portfolio” or simple income brackets, and they obviously miss a lot of the real-life variation in how people actually handle money [13]. Recent studies show banks shifting toward data-driven models. Machine-learning systems look at spending habits, income changes, saving patterns, risk tolerance... all that, to try and give more personal suggestions. Things like random forests, boosting, even clustering are used to group customers and match them with products, but the downside is these models depend a lot on good-quality data and sometimes the patterns are noisy or just weird [14]. Deep-learning pushed this a bit further. LSTMs, transformers and similar models can read transaction histories over time and forecast things like spending spikes or early financial stress, and some research says they beat older models in predicting savings or risky behaviour [**?**]. They can also use unstructured text, like messages or notes, which gives extra context but also makes things more complex. Then came generative AI and LLMs, which basically made advisory more

conversational. Instead of clicking through dashboards, customers can just ask, "Why did my spending jump?" or "Is this loan actually good for me?" and the LLM summarises activity, compares options, or simulates outcomes in plain language. Early industry use says people trust these systems more because they feel clearer and more personal [**?**].

Explainability is a huge part of this. People follow advice only when they understand why, and studies show explainable systems get better reactions than black-box ones [40]. In banking, it's not just nice to have — regulators require explanations, especially for credit and investment advice.

For *Fintrix*, the Financial Advisory Agent mixes these ideas: it uses behaviour analytics to form personalised insights, and the LLM layer delivers them in a natural conversation. It also gives simple explanations (like key spending categories, risk levels, budgeting trends) so users aren't left guessing why a suggestion was made.

2.8 Explainable Artificial Intelligence (XAI)

Explainable AI (XAI) is becoming essential in finance because automated decisions need to be understandable for customers, auditors, and regulators, and modern AI models—while accurate—often act like black boxes, which is a problem in places that demand transparency [39]. The literature usually divides explanations into local ones, which tell you why a specific prediction happened, and global ones, which describe the model's general behaviour; tools like LIME and SHAP are often used because they give simple feature-based explanations analysts can review without digging too deep into the model [41]. Recent studies also point out that explanations need to be user-friendly: customers prefer short, plain-language reasons, while analysts may want more detailed feature contributions for auditing and compliance tasks [42]. In Fintrix, XAI is kept lightweight—we provide straightforward justifications, like the main factors behind a fraud alert or the behaviours influencing an advisory suggestion, so the system stays easy to use and still transparent for both customers and internal teams.

2.9 Summary of Literature Review

This chapter outlined key research questions tackledduring the development of Fintrix on how finance has evolved from rule-based to more flexible, sufficiently data-driven and indeed agentic AI methodologies. Earlier sections demonstrated how banking AI moved from simple classifiers to deep learning, LLMs and multi-agentmodels that are able to deal with longer multi-step tasks. The sections on Agentic AI (2.3) and MAS (2.4) explained that contemporary systemsrely on groups of smaller agents collaborating rather than one large model doing everything. Sections 2.5 and 2.7 on LLM and advisory have disclosed that conversational AI as well as behaviour modelling are the two key components for

consumer protection in end-user facing solutions – both of which play directly into the workings of theFintrix Advisory Agent. The (2.6) fraud part described the evolution of ML, graph-based detection and explainabilityto combat shifting fraud tactics. And section 2.8 — the XAIsection — made it clear we still must have transparency in finance, so Fintrix uses short simple explanations (not long technical ones) as you can see here in the doffn ("depth of first fit node"). broadly speaking, theliterature seems to point toward adaptive, multi-agent, explainable AI systems – and these ideas inform the design decisions in the next chapter.

Table 2.3: Key Literature on Multi-Agent Systems in Financial Services

Study	**Focus Area**	**Key Contributions**	**Limitations**
Wooldridge (2021) [31]	Foundational MAS Concepts	Provides theoretical grounding of multi-agent systems, communication models, coordination structures, and formal agent behaviour.	Primarily theoretical; lacks domain-specific applications in finance.
Yang et al. (2024) [32]	MAS for Banking Automation	Proposes MAS frameworks for fraud detection, customer operations, and compliance automation in financial institutions.	Limited empirical testing on real banking infrastructure.
Barkat et al. (2023) [33]	MAS for Fraud Detection	Shows how cooperating agents improve anomaly detection, reduce false positives, and manage high-volume transactional data.	Complex coordination schemes may increase computation and communication overhead.
Qi et al. (2024) [34]	LLM-Assisted MAS	Introduces hybrid MAS with LLM-driven coordinator agents that enhance collaboration and reduce redundancy among agents.	Early-stage research; performance depends heavily on LLM reliability.
Abraham et al. (2024) [35]	Explainability in MAS	Proposes XAI methods to make MAS decisions interpretable for credit scoring, AML, and risk auditing.	XAI techniques increase system complexity and may not scale well for very large agent networks.

Table 2.4: AI-based financial fraud detection approaches relevant to *Fintrix*

Study	**Fraud domain**	**Main methods**	**Key insights for Fintrix**
Ali et al. (2022) [8]	Multiple financial sectors	Systematic review of ML-based fraud detection using SVM, ANN, random forests and other classifiers; highlights strong class imbalance, dominance of credit card fraud datasets, and the need for careful feature engineering.	Classical ML methods help but still struggle with imbalance and evolving fraud patterns, supporting Fintrix's use of more adaptive models.
Ben Mekhlouf et al. (2025) [9]	General financial fraud	Review of machine-learning and deep-learning approaches for fraud detection; discusses scalability issues as transaction volume increases.	Supports the move from rule-based systems to data-driven AI in Fintrix.
Cheng et al. (2025) [10]	Networked financial transactions	Survey of graph neural networks (GNNs) modelling accounts, cards and merchants as nodes with transactions as edges.	GNNs help detect organised fraud rings and complex relationships, informing the Fintrix Fraud Detection Agent design.
Zhou et al. (2023) [11]	Various fraud types	User-centred XAI framework using SHAP-based explanations for ensemble fraud models.	Shows the need for explainable-risk scores for analysts, guiding Fintrix's XAI layer.
Aljunaid et al. (2025) [12]	Banking payment fraud	Federated learning-based fraud detection combined with SHAP/LIME explanations.	Suggests future deployment of Fintrix across multiple banks with privacy-preserving training.

Table 2.5: AI-based financial advisory and personalisation approaches relevant to *Fintrix*

Study	**Advisory focus**	**Main methods**	**Key insights for Fintrix**
Jain & Kumar (2023) [13]	Automated financial advisory	Review of personalised advisory systems; compares rule-based and ML-based approaches for budgeting, saving, and product recommendations.	Highlights the shift from generic advice to personalised, behaviour-driven recommendations — a core principle in Fintrix's advisory module.
Fernandez & Singh (2023) [14]	Customer profiling & recommendation	Machine learning techniques (RF, GBM, clustering) for customer segmentation and personalised financial product recommendations.	Supports the use of customer behaviour modelling in Fintrix for customised insights and adaptive recommendations.
Kim & Park (2024) [24]	Behavioural finance analytics	Deep learning models (LSTM, transformers) for spending pattern analysis, financial behaviour prediction, and early detection of financial stress.	Informs Fintrix's ability to analyse transaction histories and generate predictive personalised advice.
PwC (2024) [23]	Conversational financial guidance	Industry report demonstrating how LLMs enable natural-language advisory, scenario simulation, and personalised financial explanations.	Justifies Fintrix's LLM-based conversational layer for advisory support.
Sundar & Patel (2023) [40]	Explainable recommendation systems	Explores XAI-enhanced recommendation models that improve user trust, with narrative-focused explanations.	Reinforces the need for clear, user-friendly advisory explanations within Fintrix.

Chapter 3

Requirement Specifications

Thischapter provides an overview of how today's banking systems operate, what remains to be desired, and why it makes sense to build Fintrix in the first place. -Andthen it also walks through the primary functional and non-functional requirements of the system, finishes with some of everything the key use cases that drive how we architected to the whole thing.

3.1 Existing System

Even today, mostbanking institutions still use a mishmash of antiquated systems (legacy), human-driven workflows and a lot of older software tools that quite frankly weren't built for real-time or intelligent decision-making. And while bankshave begun employing AI in a small number of areas, the overall systems tend to be pretty fragmented and don't really scale to make tasks personalised, automated or coordinated at any kind of meaningful scale.

3.1.1 Customer Support Workflows

Customer service, meanwhile, continues to be very reliant on call centers, scripted chat bots and other manual-response systems— and that infrastructure breaks down when there are exceptionally high volumes of inbound inquiries or long lead times (and also: let's just admit it: not a whole lot of insight into what is actually going on with the customer). Considering KPMG's banking predictions up to 2025, much of the banks that have been caught in legacy support models whereAI tools still are in early pilot and not fully positioned across [43]. Which is to say that customers receive inconsistentresponses, duplicated questions and in general slower resolution of issues.

3.1.2 Advisory and Personalisation

The advisory platforms that banks "use today are stillbased on rather simplistic rule-based logic and great customer segments." This is how they give generic advice according to predefined parametersrather than actually considering someone's real financial habits. It is stressed in the 2025 BCGreport that- "Banks have invested heavily in digital channels, but truly customized ("segment-of-one") advice remains largely not yet possible at scale. So they get advice thatfeels off and isn't particularly relevant, which diminishes trust and engagement.

3.1.3 Fraud Detection and Risk Controls

In many locations, fraud detection still relies on static rules, simplethreshold-based alerts and rudimentary anomaly-detection systems. The false positive rate on these is very high, and they tend to miss neweror cross-channel fraud patterns. There are some scalable fraud detection systems in banks but many of them fail to go beyond a smallscale and they lack the capability to leverage behavioral analytic or graph-based detection which modern systems offer . That leads to analystfatigue — too many alerts, and too many blocks for real customers.

3.1.4 Limitations of Current Systems

The endgame of all this is several unresolved conflicts. All responses take long becauseeverything is being held up by manual, or semi-automatic, workflows. Personalisation is minimal because you end up being put into large buckets rather than beingtreated as an individual. The operations across support, advisory and detection of fraud remain disconnected, so nothingreally works together. False positives are still high since legacy systems can't keepup with shifting fraud tactics. And explainability ispoor, because many alerts or recommendations don't include very plain reasons that customers or analysts can actually comprehend.

Table 3.1: Overview of existing banking solutions and their limitations

Area	Current Implementation	Limitations
Customer Support	Call centres, basic chatbots, manual ticket systems	Slow response, limited context, repetitive interactions
Advisory Systems	Rule-based engines, broad customer segmentation	Generic advice, low relevance, poor engagement
Fraud Detection	Static rules, threshold filters, basic ML prototypes	High false positives, weak behavioural insight, low scalability

3.2 Proposed System

Current banking systems have inherent gaps and it proves that they just are not adaptive or personalized enough, hence we envisioned Fintrix, a modular AI assistant modelledto cater for three scrip

separate components. Each module has its own language model, its ownlittle reasoning loop and its own job. That goes well with what the writersrecommend but also makes scaling and, I don't know, updating later easier without the black truck wrecking through everything.

One of the early back-and-forths we had was basically: Should one giant AI model be doing support and advisory and fraud detection all inone? We looked atresearch on agentic or modularized AI, and for the most part it seemed that having one model with a long chain of decisions about how to act was bad for performance (and not something you want if you later want to make changes). Three distinct categories required different explanations, soit made more sense to tear them apart.

3.2.1 Customer Support Module

This moduleleverages its own LLM to process natural-language questions, pull down info from the simulated backend and handle answer routine banking stuff. It's tuned not just for smoother conversations and consistent context across turns, but also because itis autonomous: Updating that model doesn't fumble the others.

3.2.2 Financial Advisory Module

This module still has its own LLM, which examines (simulated) financial history,spending profile, level of income stability, saving behaviour and so forth. We justkept asking ourselves, what would make advice feel real, not like the old rule-based systems? Sooperation-oriented modelling was the model and then it explains in straightforward, simple terms even when the patterns are a bit dirty.

3.2.3 Fraud Detection Module

Fraud detection required more thought, because it involves a delicate balance of numbers, as well ascareful timing. We didn't want to pretend we could do this perfectly, so wedeveloped a simple hybrid solution:

- an LLM which interprets transaction descriptions, context, location/device movements or changes andtiming irregularities, and
- and a lightweight ML model which compares numerical patters to that of the customer's pastbehavior.

Combined, they flag strange spikes, weirdmerchants, odd timing. and give short explanations that tell users whysomething doesn't look right.

3.2.4 Design Rationale

The modular architecture has advantages: each module can be replaced separately, failures don'tbring down the others, each uses a LLM best for its own job and modules can be added later (without having to totally redesign everything).

Explainability is kept lightweight too. Rather than fully XAI pipelines, each module would produce shorthand reasons—such as whatspending pattern led to an advisory tip or what behavior caused a fraud alert —keeping things relatively transparent without additional overhead.

In general, Fintrix aims at being a lightweight but intelligent assistant where each module worksautonomously yet also addresses the main limitations identified in Section 3.1. The following section describes the environmental system parameters that dictatethe design.

3.3 System Requirements

This part describes the criteria thatare followed when designing Fintrix. From all what wehave argued before, current banking systems, real needs of users and the architecture proposed for it, where are three groups of some functional and non-functional requirements but also system logical constraints. These guidelines are really just designed to try and make certain that thesystem behaves in a consistent way, remains secure, and functions how you'd expect users on both ends of the financial transaction to experience it — without crashing down in real-world use.

3.3.1 Functional Requirements

Functional requirementsare what the system needs to do. As Fintrix is composed of three AI distinct modules, we parse the tasks regardingwhat each module must do, in addition to some general system tasks.

General System Requirements.

- Issue FR 01: The system should accurately identify which module—that is,Support, Advisory or Fraud Detection—would be appropriate to respond to the user query.
- FR02: The system should maintain dialog history through an interaction to avoiduser repetitions.
- FR03: The system needs to be able to retrieve the relevant customer or transactiondata from the simu- lated backend.

- FR04: The output of the system should be in plain and understandable normal naturallanguage.

Customer Support Module Requirements.

- FR05: The module shall be able to read natural-languagequeries question from the users.
- FR06: It should be able toaddress typical banking requests such as account balance, transactions or branch information.
- FR07: If the module can't resolve/answer something, it should escalate or provide different guidance rather than just leaveuser hanging.

Financial Advisory Module Requirements.

- FR08: Historical spending patterns in thesimulator customer profile have to be considered by the module.
- FR09: Itshould provide personalized recommendations and insights based on that behavior.
- FR10: The module must describe the reasons for sp ending trends in u- ence s o thatusers are not confused.

Fraud Detection Module Requirements.

- FR11-The module shall analyze new transactions based on both behavioral trends and historical behavior.
- FR12: It shall produce a fraud-risk score using thelightweight ML model.
- FR13: It is required to recognize abnormal activity—uncharacteristic spending patterns, new places,unusual transaction times.
- FR14: The system shall offer brief and concise reason why a transaction might bedistrustful.

3.3.2 Non-Functional Requirements

Non-functional requirements specify the quality attributes of the system, i.e. how well it must perform, how safe it should be, and how easy it is to use.

3.3.2.1 Performance Requirements

- **NFR01:** The system should take no more than approximately 3 seconds to respond under average load.
- **NFR02:** The system should be able to serve requests from multiple users concurrently without being perceived as slow.

3.3.2.2 Security Requirements

- **NFR03:** Secure communication (e.g. encrypted channels) shall be used for all internal and external data transfers.
- **NFR04:** Customer data access shall be constrained by a predefined set of authorisation rules.

3.3.2.3 Usability Requirements

- **NFR05:** The user interface must be simple and easy to understand.
- **NFR06:** The system's explanations should be written in clear language and be understandable even for non-technical users.

3.3.2.4 Reliability Requirements

- **NFR07:** If one module fails or returns an error, Fintrix should continue running and process the remaining modules.
- **NFR08:** The system should produce consistent, non-variable outputs for similar inputs.

3.3.2.5 Maintainability Requirements

- **NFR09:** Each module shall be upgradeable independently of the others.
- **NFR10:** The routing logic should be easy to modify when new modules or behaviours are added later.

3.3.3 System Constraints

- **C01:** The platform operates with a simulated backend instead of a real banking infrastructure.
- **C02:** The quality of suggestions and fraud alerts depends on the characteristics of the simulated dataset used for training and evaluation.

- **C03:** As each module is largely self-contained, cross-module communication is intentionally kept minimal by design.
- **C04:** Resource utilisation needs to be limited so that the system can run on common, commodity hardware.

3.4 Use Cases

This section explains the main ways people (and the system) actually interact with Fintrix. Each use case basically shows how one of the three modules behaves — Customer Support, Advisory, or Fraud Detection — and helps make it clear what each one is supposed to do.

3.4.1 Use Case 1: Customer Support Interaction

Actors: Customer, Customer Support Module

Preconditions:

- The user is logged in.
- Support module is running fine.

Postconditions:

- The customer gets the info or help they asked for.

Main Flow:

1. The customer sends a natural-language question (like "what's my account balance?" or something similar).
2. The system sends that query to the Support Module.
3. The module tries to understand the question using its LLM.
4. It grabs whatever data it needs from the simulated backend.
5. A response is generated that (hopefully) makes sense for the situation.
6. The system shows the reply to the customer.

Alternative Flows:

- If the module doesn't understand the question properly, it asks the user to clarify.
- If the info isn't available, it suggests something else the customer can do.

3.4.2 Use Case 2: Personalised Financial Advisory

Actors: Customer, Advisory Module

Preconditions:

- The customer's financial history exists in the simulated backend.
- Advisory module is active.

Postconditions:

- The user gets a personalised insight or suggestion.

Main Flow:

1. The customer asks for advice (e.g., "How do I save more money?").
2. The system routes the request to the Advisory Module.
3. The module looks at spending patterns and old financial behaviour.
4. Its LLM generates advice in plain natural language.
5. The advice is shown to the customer.

Alternative Flow:

- If there isn't enough data, the module either asks for more info or tells the user it can't give a full recommendation.

3.4.3 Use Case 3: Fraud Detection Alert

Actors: Fraud Detection Module, Customer (sometimes), Fraud Analyst (if needed)

Preconditions:

- A new transaction appears in the backend.
- Fraud module is running.

Postconditions:

- A fraud-risk score is created.
- A short explanation is generated.
- Warnings show up if the transaction looks risky.

Main Flow:

1. A new transaction enters the system.

2. The system passes it to the Fraud Module.
3. The ML model scores the transaction using patterns + past behaviour.
4. The LLM generates a short explanation about why it looks suspicious (or not).
5. The module returns the score + explanation.

Alternative Flow:

- If it matches the user's normal behaviour, it's marked low risk.
- If risk is high, it may be escalated to a human fraud analyst.

3.4.4 Use Case Diagram

A UML use case diagram representing these interactions is provided in Figure 3.1. It shows the three main actors (Customer, Fraud Module, and Analyst) and their interactions with the corresponding modules.

Table 3.2: Summary of Functional Requirements for *Fintrix*

Req. ID	**Requirement Description**
FR01–FR04	General system behaviour (routing, context, data access, response generation)
FR05–FR07	Customer Support Module requirements
FR08–FR10	Financial Advisory Module requirements
FR11–FR14	Fraud Detection Module requirements

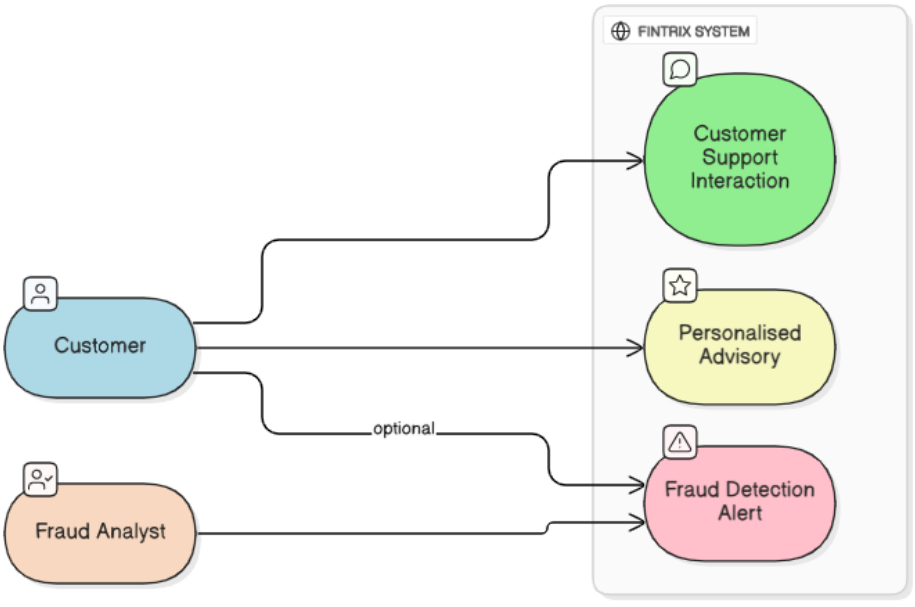


Figure 3.1: Use Case Diagram for the Fintrix System

Chapter 4

Design

This chapter gives ageneral understanding of Fintrix's structure. Where the last chapter concentrated on what the system is designed to do, your attention here shifts to how it works—the wayits various parts are structured and integration in which data transfers occur. Our intention is simply to demonstrate thegeneral architecture of the system and describe how modules—both weakly and strongly coupled—operate together. In Fintrix we have followed a minimalist modular approach where trying to support all the ways components can change/replace themselves but without making it less stable as awhole. The chapter is consequently dividedinto several key sections. We start by describingthe general structure that we initially imagined. We then presentour decision based on simple design considerations. We also discuss the global architecture we initially described, and nally, but not least, we closethis section by making a summary of how it all comes together to form a ¯nished system – at least on paper.

4.1 System Architecture

The architecture of Fintrix is already intentionallymodular and interpretative enough, such that allowing for a scalable growth without an unacceptable increase in complexity. Instead of one gigantic model that does everything, the system breaks down responsibilities into 3 different ones:Customer Support, Financial Advisory and Fraud Detection. Every partkeeps its own logic, works with a provided language model and is focused as narrowly as possible. This separation has the advantages of providing conceptual transparency, supporting independent iteration and minimizing unexpected interactions between system domains while modifications orupgrades are deployed. In development it'srunning locally on my machine, connecting out to a Supabase database in the cloud. Potentially anywhere is its database of everything from customer profiles to transaction histories, behaviour

patterns anddata on fraud. Each module just takes what it needs from there, does something with it and then sends the response back down to thebackend.

Fintrix has front-end pagesthat users have to interact with. Both the Support and Advisory modules have simple interfaces based on user requests, while the Fraud Detection module is implemented as atoy example where the user submits a transaction and receives an outcome of fraud or non-fraud. In forthcoming editions, this section will belinked to actual transactions (not an entered adjustment). Inside the backend is a little routing mechanism which determineswhich module should be responsible for the request, and passes back to the user what was requested.

The high-level context diagram is shown in Figure 4.1, it shows the main actors, the local backend, the three AI modules and the connectionto our Supabase database. The context diagram describes relations to the outside, while Figure 4.2 shows the high-level internal architecture of Fintrixbackend. Itexplains that routing logic routes requests to the appropriate module, and each module has its dedicated components. The FraudDetection module consists of a behaviour analyser, an XGBoost classifer and an LLM explainer that showcases the hybrid ML–LLM design. All of them talk to the Supabase backendcloud database for retrieving or storing required data.

4.2 Design Constraints

Fintrix Design The design of Fintrix was influenced by several practical and technical constraints that determined howsuch a system could feasibly be implemented. But sincethis project was only a proof of concept, not a full banking solution, for most decision the bias was towards simplicity, modularity and testability rather than performance or mass deploy-ability.

4.2.1 Local Development Environment

Fintrix's backend is hosted on a local machine, preventingus from using models that are very large or resource-intensive. For this reason, lightweight language models and a simple routing mechanism are combined in each moduleinstead of having one large orchestrator. Running everything

locally also implied the system needs to be able to start quickly, debug easily andnot rely heavily on cloud compute.

4.2.2 Cloud Database Dependency

Dependence on the cloud database is a major challenge in terms of knowing where the data stored by the CSP resides and therisks related to its storage location$_{C}$*hen*09 [12].

“All the application data is living in a Supabase PostgreSQLdatabase that’s cloud-hosted. This results in a "soft" dependence on the Internet, where thereis a little delay each time modules try to read/update records. That’s fine for a prototype, but in production banking code you often need to get more performance and tunethe hell out of it.

4.2.3 Limited Real-Time Capabilities

Currently The FraudDetection module is more of a demonstration where you as the user enter a transaction by hand. Noreal-time streaming or automatic event triggers are yet in place, mostly due to time and infrastructure constraints. A potential future version of Fintrixwould connect directly to transaction streams or message queues.

4.2.4 Model and Data Constraints

The fraud module is based on an XGBoost model trained withsample or publicly available datasets — not real customer data — so its accuracy and generalisation are constrained. Simi- larly, each languagemodel operates with heavily cleaned prompts and no advanced conversa- tion memory or deep personalisation which a real bank would need.

4.2.5 Security and Privacy Constraints

So long as the backendis hosted locally and only the database in cloud, credentials should be convenient to handle. No personal, sensitiveuser data was accessed at all. A real production systemwould require proper encryption, strict access control, audit logs and certainly far stronger security hygiene.

4.2.6 User Interface Limitations

Every segment hasa miniscule web page. Roads Support and Advisory modules are static front-endpages and the Fraud Detection interface is simply a bad demo. This was by designto make the prototype more manageable in scope. More packaging UI/UX around a singlepane can be supplied later.

4.3 Design Methodology

The design idea behind Fintrix was to keep it modularand extendable later, but also simple enough that you can test the hell out of it without having to set up too much. Our systemis composed of three independent AI we wanted to remain as such; they have been designed with their specificity in mind, and each one has his own reasoning style. We had to find a way letting them be the independent agents but exchange data when it was necessary directly on the backend side. The point wasn’t tobuild a big, complicated piece

of architecture, but rather to provide a proof that, here at least is how you might create modular task specific AI pieces that could coexist within one application.

4.3.1 Modular and Independent Components

It turned out that every one of these modules — Customer Support, Financial Advisory and Fraud Detection solutions —all were developed as separate parts, with separate language models, separate logic and only their own way to process such data. By doing this, it reduced the amount of management I needed to do and it becamemuch more testable; all parts could be asserted to independently of the others.

4.3.2 Simple Routing Instead of Orchestration

We also talked, at an early stage,about having a central orchestrator but we felt for a prototype it was overkill and would be too heavyweight. So here is how we did it: a very simple crappy routing: when you get in therequest arrives to a backend, and then this backend would just say, “okay, well, I know actually where this module belongs to” and then send it. That made things simpler, andno additional layers (which we didn’t really need at the moment) were added.

4.3.3 Using UML for Structure

Toavoid the design from becoming a mess we used various UML diagrams — use case diagrams, com- ponent diagram, sequence diagram, deployment diagram etc. The point wasn’t to create perfect diagrams, but to give us a better pictureof how the whole thing works and which segment talks to what. Actually seeing stuff on paper, helped us to keep in sync andnot lose sight of the structure as it went through stages.

4.3.4 Hybrid AI Reasoning Approach

Fintrix incorporates a hybrid of the language models and conventionalML model. This is quite apparent in the Fraud Detection– where instead of explaining ranked feature using an XGBoost model and thengiving some short explanation as a LLM so that output in more handy. We could thus couple a pattern-recognition strength to the natural-language generationskills.

4.3.5 Iterative, Prototype-Based Development

Since it is a prototype system, we adopted an iterativeapproach. Each module was added incrementally,starting with better input–output behavior, followed by progressively larger improvements in reasoning and data handling with small interface improvements. We were able to catch problems early enoughthat we didn’t have to throw everything away.

4.3.6 Focus on Explainability and Simplicity

While full XAI was not the main goal of this work, everymodule was intended to produce brief and easy interpretable explanations. This feature determined the way that data was used by individual modules andthe manner in which results were presented to end users. Through the inclusion of short, easily understandable rationaleswithin the interaction pattern, a trade-off between transparency and operation facilitation is achieved by the system. Therefore, an intuitive and user-friendly interface was accomplished withoutimposing unwarranted technical or cognitive burden.

4.4 High Level Design

High-leveldesign — Fintrix The high-level architecture of Fintrix is built upon the previous architectural artifacts and provides a better picture of how the things are connected together. The ultimate goal for nowis to show how the parts communicate, data travels within your system, and what it kinda looks like in action. Iillustrated this with several familiar views... conceptual view, process view, physical view, module view and the security view. They together provide enough of a picture, before diving into thespecific implementation!

4.4.1 Conceptual / Logical View

The conceptual view is basically a simple picture of how Fintrix is organised inside and how the main parts are connected. The system is built around three AI modules: customer support, financial advisory and fraud detection. Each one has its own small internal parts and kind of its own way of doing things. They run on their own but still use the same backend, which deals with routing, checking inputs, and talking to the external data.

In the backend there is a small routing bit that looks at what the user sent and then passes it to the right module. All the modules use the data layer to read or update information. The data layer uses a Supabase PostgreSQL database in the cloud for normal structured data, and Pinecone for vector based knowledge, so modules can grab whatever they need.

Figure 4.3 shows the logical component diagram with the frontend, backend, the AI modules and the data layer all connected. Inside each module there are things like the LLM part, behaviour analyser, the ML classifier and the bit that formats the final output before sending back to the user.

The main parts shown in the diagram are:

- **Frontend layer**, which is basically the user interface where customers interact with the system. All the queries start from here.

- **Backend layer**, which has the API handler, input checking, logging, and the small routing bit. This part handles the communication between the frontend, the different modules, and the data services.
- **Customer Support module**, which has the LLM engine, the query processor, a bit of context memory, and the formatter that puts together the final conversational reply.
- **Financial Advisory module**, which includes the advice-generation LLM, the behaviour analyser, trend detector, and the piece that formats the recommendation into something readable.
- **Fraud Detection module**, which has the behaviour analyser, pattern checking, the XGBoost model, a simple risk scorer, and the LLM explainer that turns the analysis into a short explanation for the user.
- **Data layer**, where Supabase PostgreSQL holds the customer and transaction data, and Pinecone works as the vector database for all the embedding-based knowledge used by the advisory and support parts.

4.4.2 Process View

The process view shows how the system behaves while it is running, and how the frontend, backend, routing logic, the AI modules, and the data layer all interact with each other. To explain this clearly, three sequence diagrams are used — one for each main module: customer support, advisory, and fraud detection.

These diagrams show the full request flow, which subcomponents get involved (LLMs, analysers, validators, the classifier, etc.), and how the system talks to the Supabase database and the Pinecone vector store during the process.

Customer Support Workflow

Figure 4.4 shows the runtime behaviour of the Customer Support Module.A user sends a support requestfrom the web interface, this is received by the API Handlerbefore it's processed (and validated) by the Customer Support Module. The module fetches any necessarycustomer or account records from Supabase, issues the query via its LLM engine and context memory, into a formattedresponse.

Financial Advisory Workflow

The advisory process isillustrated in Fig. 4.5 and initiated when a user requests personalised financial guidance. Upon successful authorization,the request is dispatched to the Financial Advisory Module. Themodule retrieves historical financialGI financial data and

other profile data from the Supabase and returns significantembedding knowledge from the Pinecone vector database. The behaviour analyserand trend detector executes this information, the Insight LLM provides a customized recommendationand explanation.

Fraud Detection Workflow

Figure 4.6 illustrates the fraud detection workflow. When a new transaction is received, it is verifiedand forwarded towards the Fraud Detection Module. Historical on-shelf behaviour is pulled fromand engineered from Supabase is then fedinto the XGBoost classifier to produce a fraud-risk score. Then the LLM Explainerproduces a natural-language explanation of the model's decisionis returned to the frontend.

4.4.3 Physical View

The physical viewdetails the way in which Fintrix is deployed on our local setup as well as the cloud services. Although the system is only a prototype, it adheres to a modular architecture where the backend inferences are performed locally on devices, and all stored data as well as vector-based retrieval componentsreside in the cloud. This separation made it easier to experiment during development, but still keep access to scalable cloud storage whenever the modules needed it.

Figure 4.7 shows the deployment diagram, with the three frontend interfaces, the local backend environment, and the external cloud services that Fintrix connects to.

- **Local machine (backend runtime)** holds the main backend application. This includes the API handler, routing logic, validator, logging, and the three AI modules: customer support, advisory, and fraud detection. All the model inference, request handling, and the decision-making for each module happens here on the local setup.
- **Supabase PostgreSQL (cloud database)** stores all the structured data Fintrix uses: customer profiles, transaction histories, behaviour patterns and similar records. Every module reads and updates information through this cloud database.
- **Pinecone vector database** is used for fast vector search and embedding-based knowledge lookup. The customer support and advisory modules rely on Pinecone when they need semantic context during their reasoning.
- **User frontends** include the three small interfaces: customer support, advisory, and the fraud demo page. These are the entry points for the system, where users send their queries or transactions, and everything is then passed to the backend for processing.

4.4.4 Module View

The module view explains how Fintrix is structured on the inside by breaking the system into its main implementation modules. Instead of looking at deployment or how the system behaves at runtime, this view focuses on how the frontend and backend folders are organised, and how each part of the system is actually put together in the codebase. Fintrix revolves around three core modules — customer support, financial advisory and frauddetection. Each had it's own little frontend GUI, their backend logic and would connect to the cloud Data Services via the sharedAPI client. Also this way, keeping them in separated modules make the development easier, debugging it even more and keep us able to extend one of them later without violatingother(s). Forthe sake of demonstrating this layout three separate module diagrams are part of the design.

Customer Support Module

The LLM-driven conversational butler utilized in customer support scenarios is realised throughthe Customer Support Module, for user support questions. This module has its own subpage with infrastructure (i.e., the Fintrix APIclient, prompt templates and backend LLM logic) to access Supabase as well as Pinecone in order to fetch the info necessary for informed responses. Its main features are the SupportPage (Next. js) that's the frontend interface which users use tomake queries fintrix-client. ts file (it connects the Support page and API backend), "the Python side," then presents a hypothetical agent called SupportLLM Logic, which creates conversational responses for us, the Support team; and prompts/base. py file that acts as data regarding thetemplate prompts for the support agent. Furthermore, the module uses Supabase Access to access customer records storedand Pinecone Access to get contextual embeddings for improved responses.

Financial Advisory Module

The Financial Advisory Module powers personalised financial insights by combining historical account data with domain-specific knowledge stored in Pinecone. It includes its own frontend page, advisory agent logic, support tools, and prompt templates.
Key components:

- **Advisory Page (Next.js):** Interface where users request financial advice.
- **fintrix-client.ts:** Sends advisory queries to the backend service.
- **financial_assistant.py:** Main advisory agent responsible for generating insights.
- **financial_assistant_tools.py:** Helper functions for processing financial patterns.
- **financial_assistant_prompt.py:** Prompt templates used by the advisory LLM.

- **Supabase Access:** Retrieves customer transaction histories.
- **Pinecone Access:** Retrieves financial knowledge embeddings.

Fraud Detection Module

The Fraud Detection Module integrates machine learning, behavioural analysis, and rule-based logic into a unified workflow. It includes a dedicated frontend demo page, backend fraud agent, XGBoost model, and schema definitions.
Key components:

- **Fraud Page (Next.js):** Demonstration interface for fraud scoring.
- **fintrix-client.ts:** Sends transaction requests to the backend.
- **agent.py:** Core Python fraud agent orchestrating the detection workflow.
- **model.pkl:** Trained XGBoost model for fraud-risk classification.
- **feature_schema.json:** Defines the feature structure expected by the ML model.
- **Supabase Access:** Used to retrieve behavioural and historical transaction data.

This modular decomposition reflects Fintrix's design philosophy: each subsystem is implemented as a self-contained unit that communicates through simple interfaces, enabling flexibility, maintainability, and clear separation of responsibilities.

4.4.5 Security View

Fintrix: A Systematic Security View on Risk Mitigation Strategies In more detail, this paper presents the security view of Fintrix in terms of how to mitigate risks such as unauthorized access, data leakage, malicious inputs,model misuse and unanticipated AI behavior. As Fintrix deals with sensitive financial data, such as customer information, transaction history, behavior signalsand fraud scores, it uses layered security at the frontend, backend and data layer. Security follows four principles: (1) authentication and access control to authorize only authenticated users and trusted clients to access backend services; (2) input validation and request filtering to reject malformed payloads or injection-style attacks before they reach LLMs or ML models; (3) audit logging and monitoring for compliance, debugging, as well as alerting on anomalous/fraud-related activity; and (4) encrypted, policy-governed data storage where Supabase enforcesencryption at rest and row-level security while Pinecone holds embeddings in a secured index with scoped API keys. All together, these thingskeep us from doing anything to leak our secrets or allow system level attacks / abuse of smart components.

Figure 4.11 shows the full security layout, and how these protection layers sit across the different parts of the system.

4.5 Summary

Fintrixarchitecture: Step back and look into the system This chapter was an attempt to plumb the design depth of Fintrix from different perspectives. The conceptual model explained the main elements — customer support, advisory and fraud detectionamong them — and demonstrated how they interacted via both a backend routing layer and shared cloud data services. The process view thendrilled down into the runtime behavior using sequence diagrams and illustrated how requests flow from the frontend down through the different backend modules, the ML parts, and finally reach the data layer. The physical aspect also described the deployment ofsystems in local and cloud infrastructures, such as Supabase and Pinecone, with distinct compute and storage parts. The module view dissected Fintrix into its frontend, backend and data parts such that we got to see howthe whole thing is organized. The security view covered the most important lines of defense, including authentication, validation, logging and encryptedstorage. All of these views together give a complete picture of Fintrix — how it is structured, how it behaves while running, and how it stays secure. This architecture sets the base for the next chapter, where the system is actually implemented through the code, the models, the interfaces and the workflows that tie everything together.

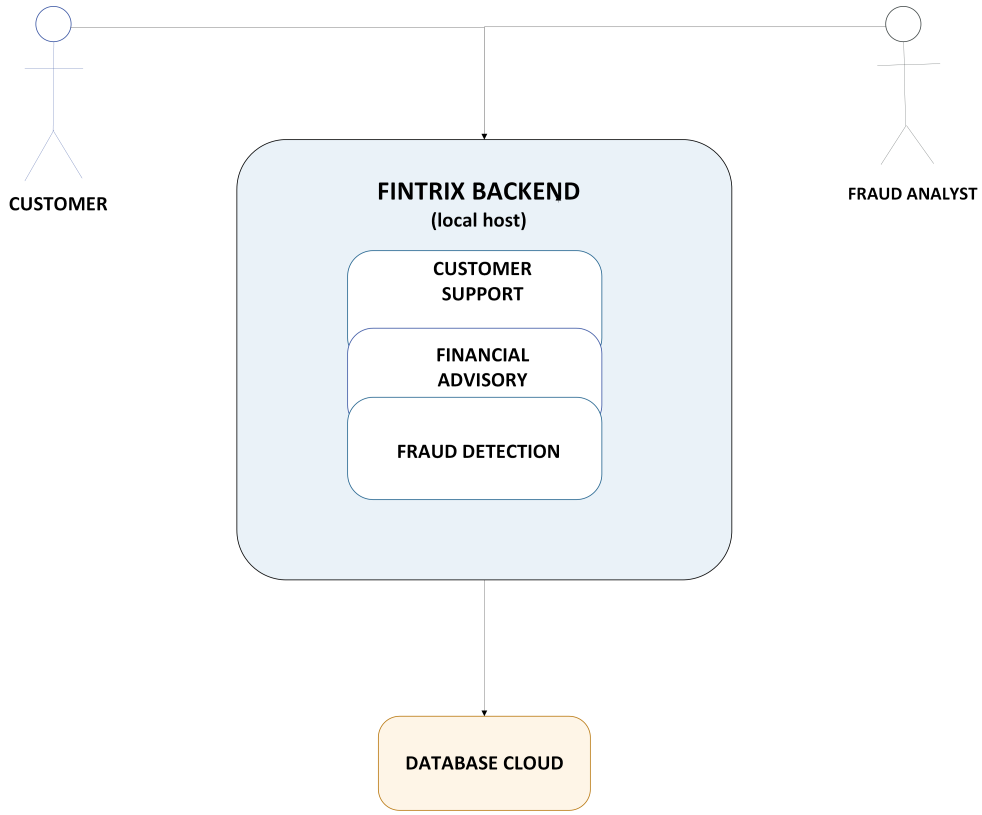


Figure 4.1: High-Level Context Diagram of the Fintrix System

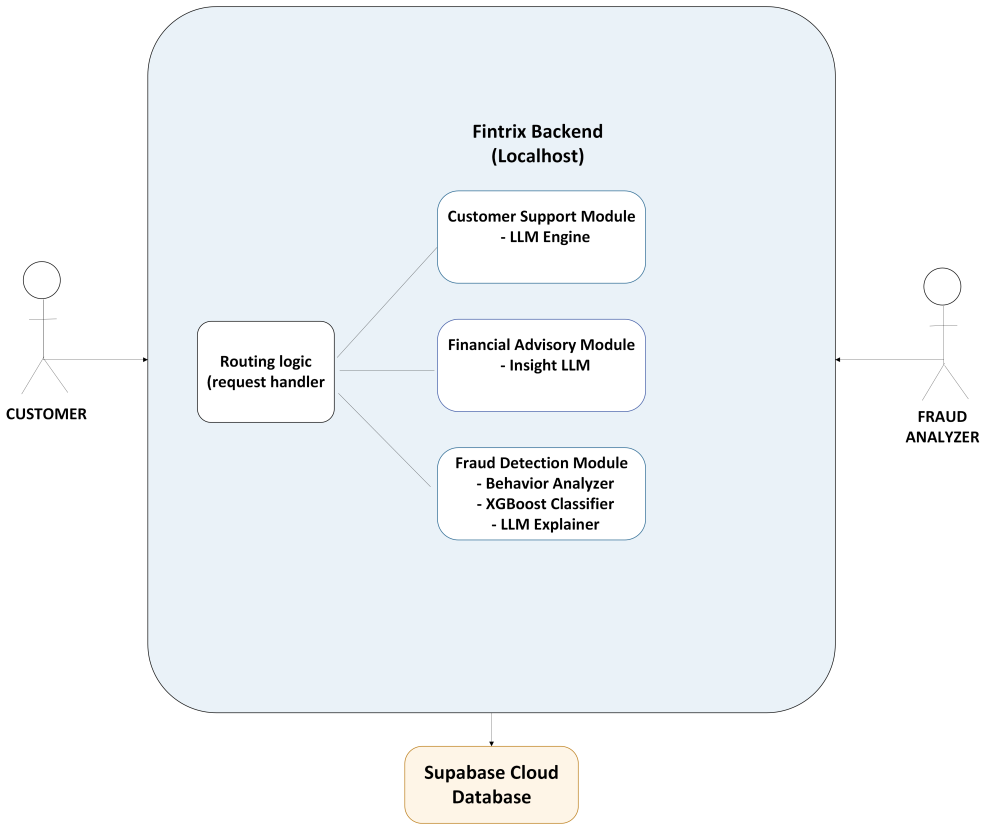


Figure 4.2: High-Level System Architecture of Fintrix Showing Internal Modules and Routing Logic

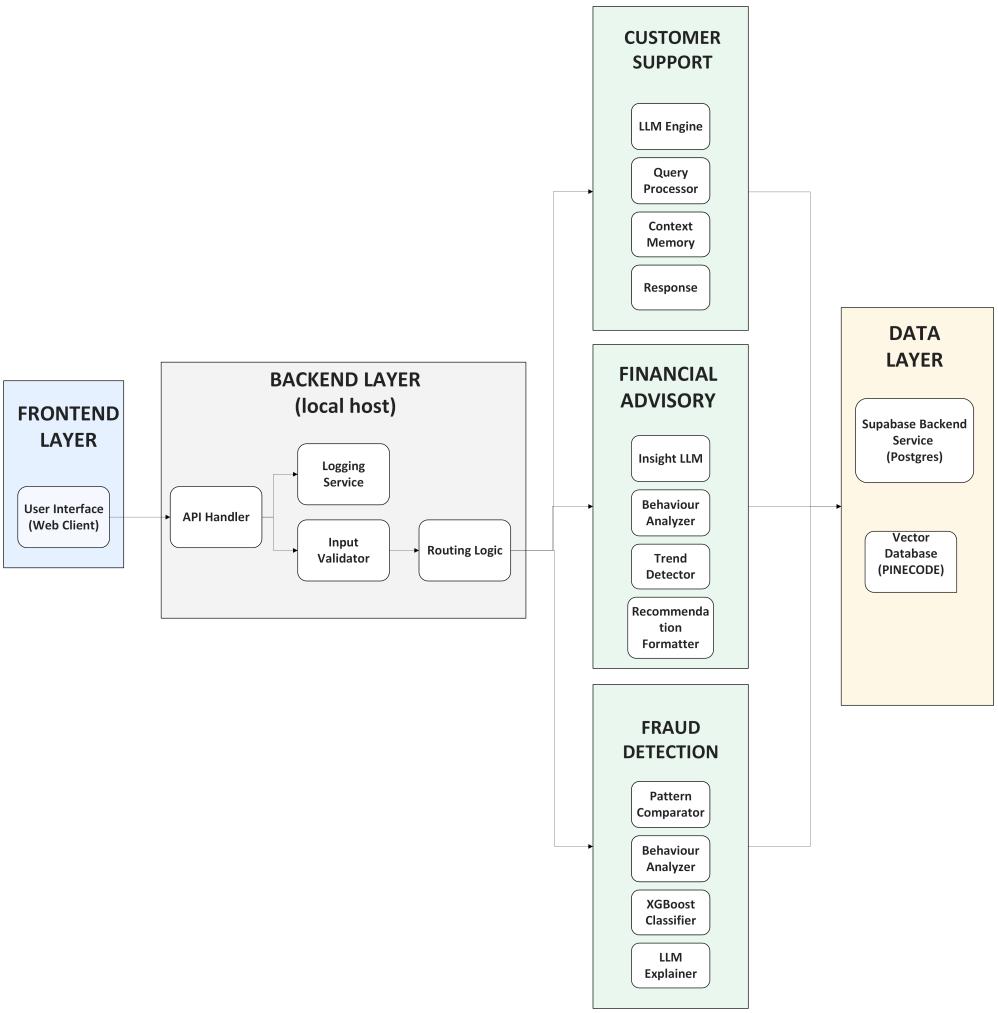


Figure 4.3: Detailed Logical Component Diagram of the Fintrix System

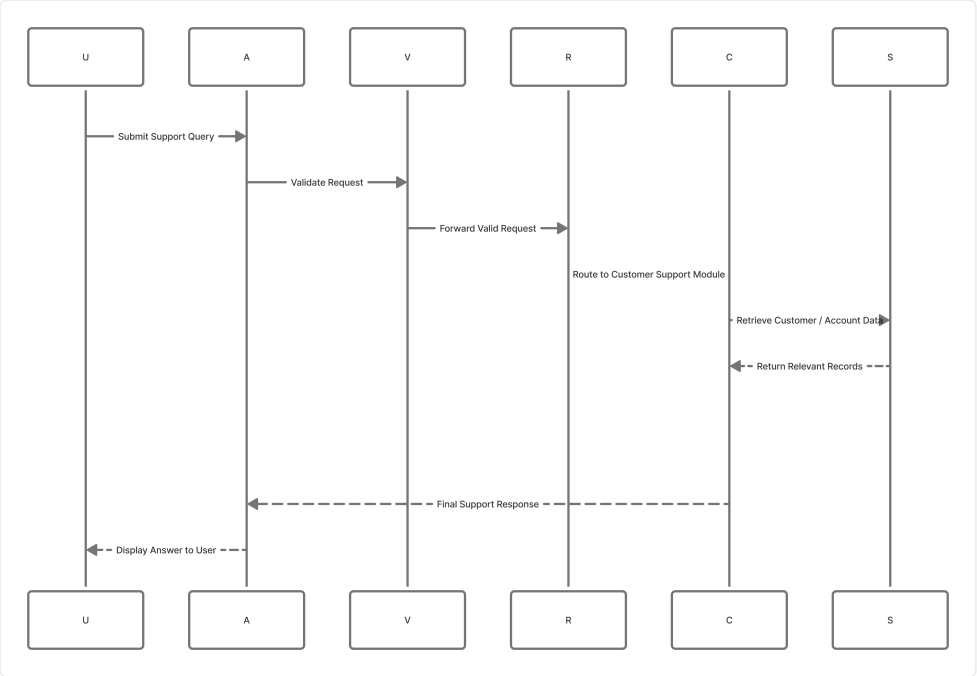


Figure 4.4: Sequence Diagram for Customer Support Workflow

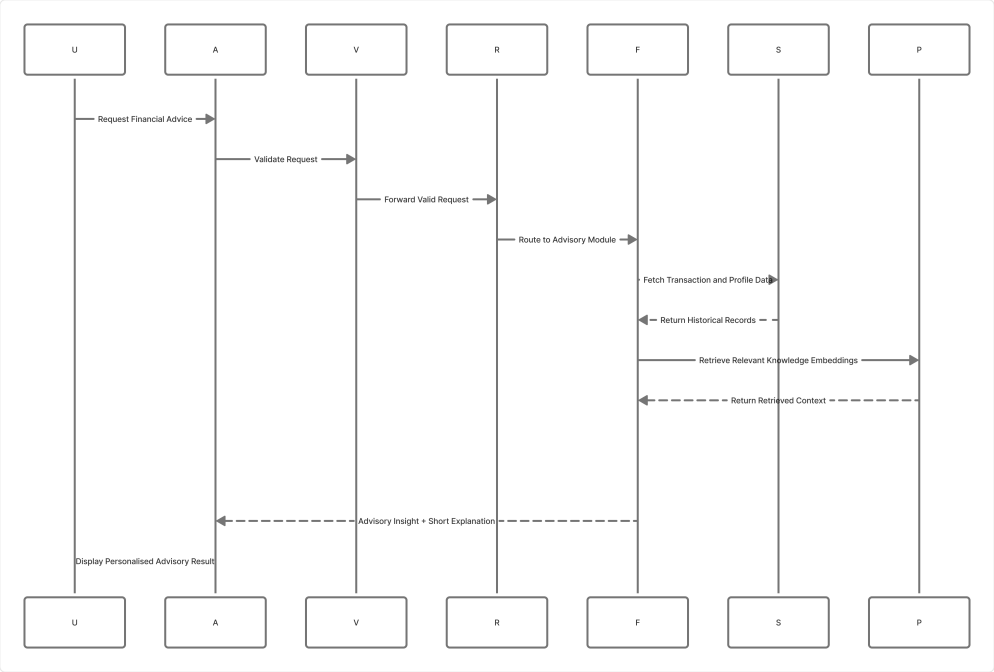


Figure 4.5: Sequence Diagram for Financial Advisory Workflow

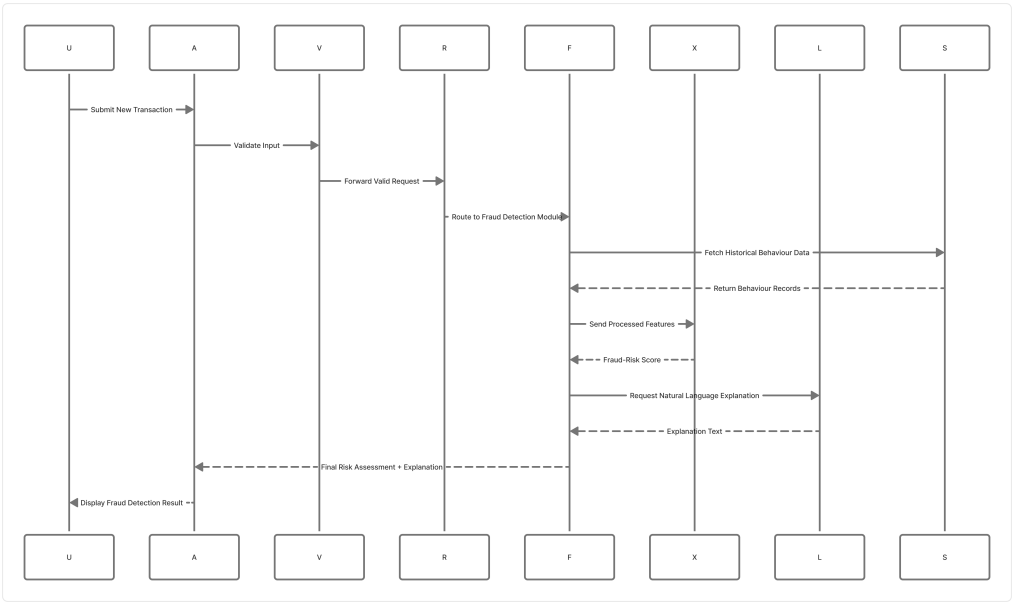


Figure 4.6: Sequence Diagram for Fraud Detection Workflow

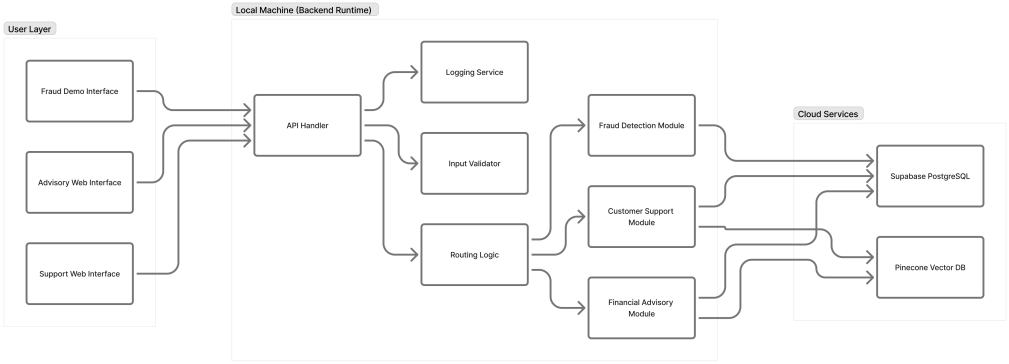


Figure 4.7: Deployment Diagram Showing Local Backend and Cloud Services

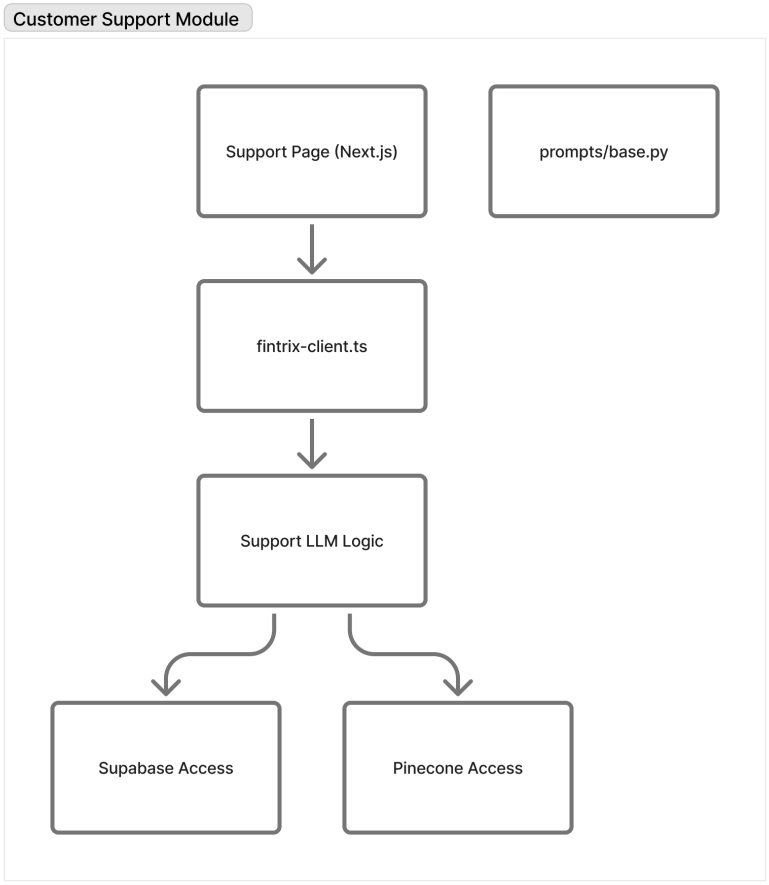


Figure 4.8: Module View of the Customer Support Subsystem

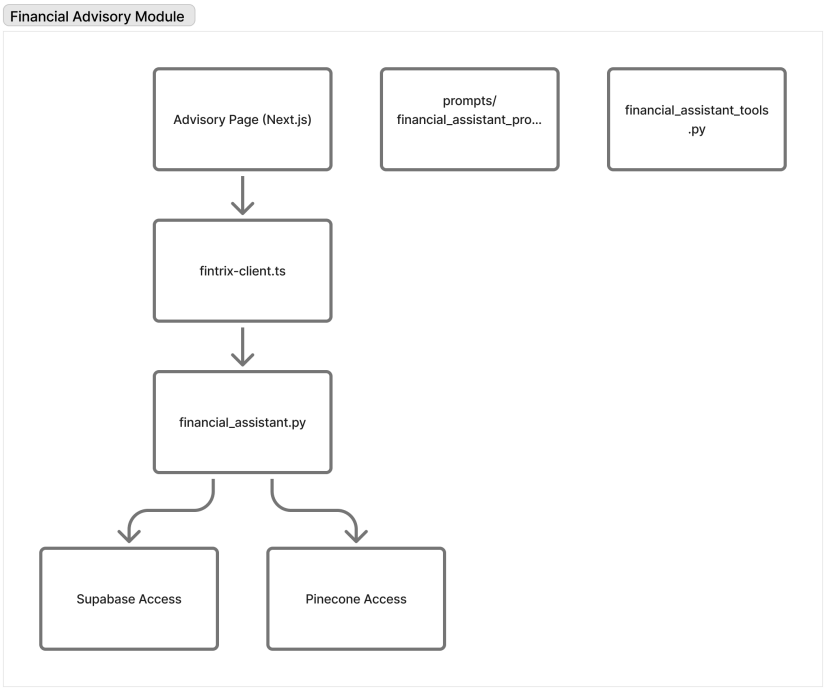


Figure 4.9: Module View of the Financial Advisory Subsystem

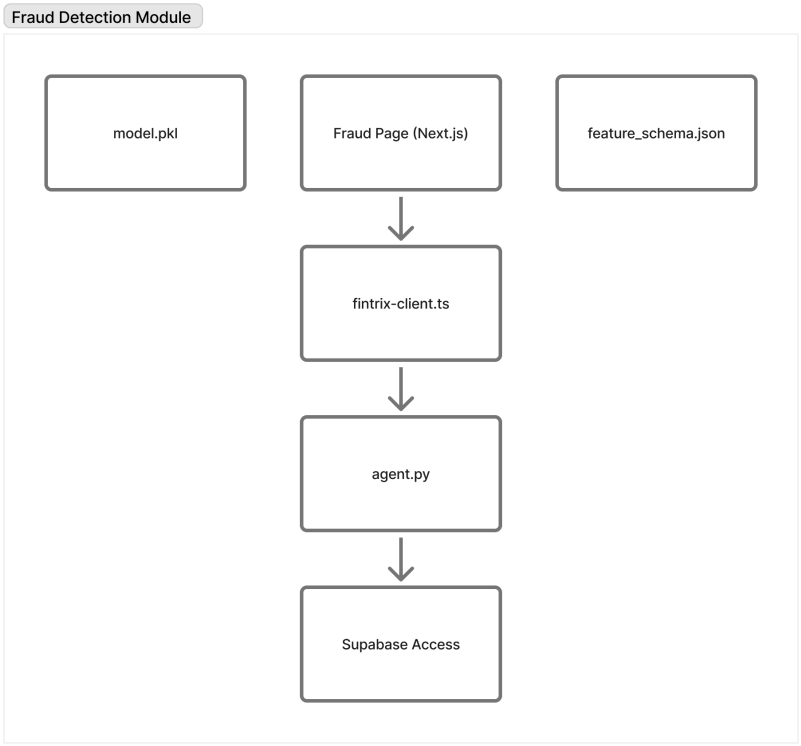


Figure 4.10: Module View of the Fraud Detection Subsystem

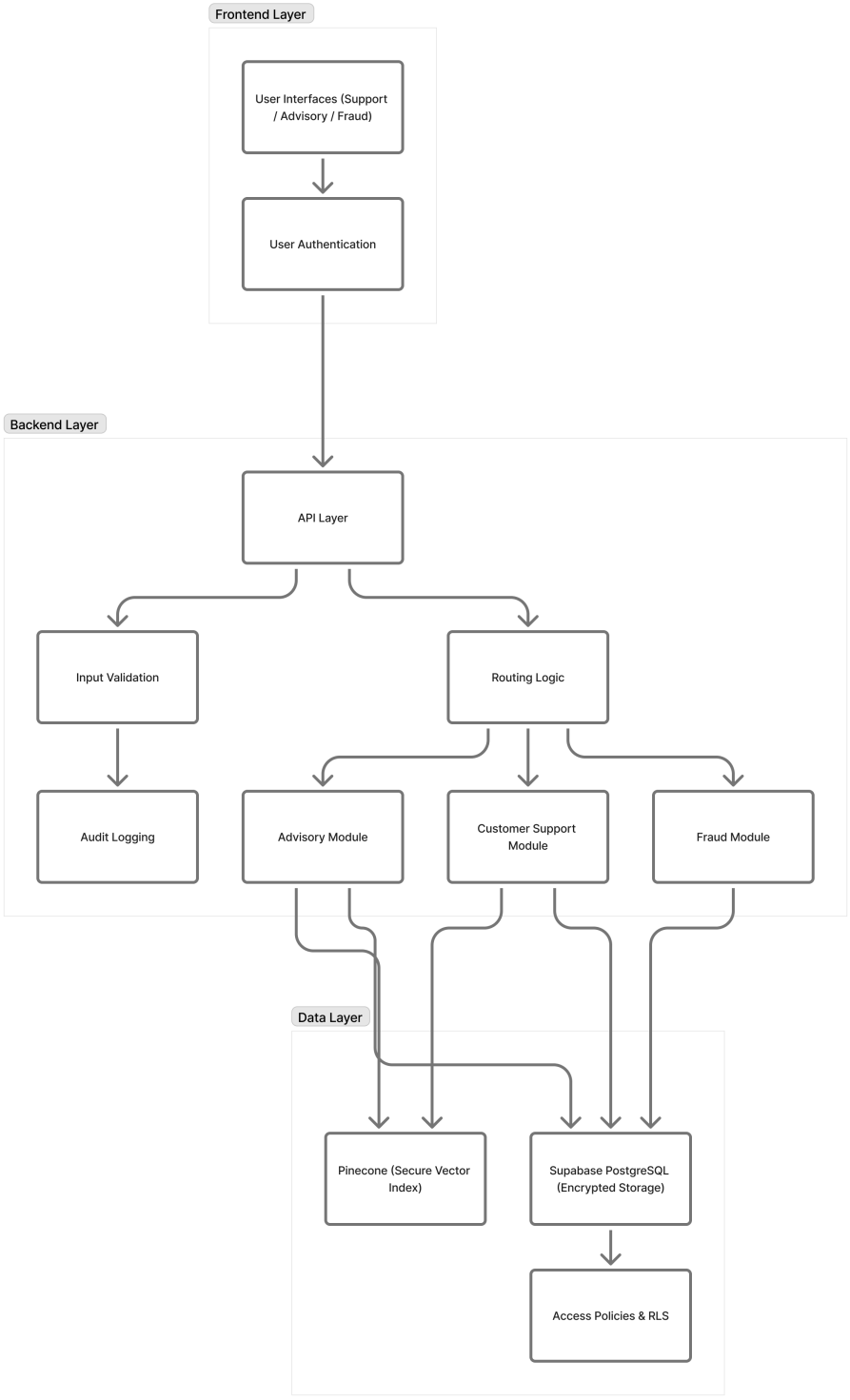


Figure 4.11: Security Architecture of the Fintrix System

Chapter 5

System Implementation

This chapter shows how Fintrixis realised concretely, and how the design of Chapter 4 becomes a practice system. While the last chapter described the system's structure and explained what we wanted to achieve with it, in this one we will talk about how everything is built (in code), how all parts work together and how everythingputs into form. The implementation includes frontend, backend (including agentlogic, ML pipelines, database connections), vector retrievals and user interfaces that are intended for interacting with Fintrix. The tooling, technology and security setup to operate everything in a reliable way are alsodescribed.

5.1 System Architecture

The adopted architecture is modular and layered in design that partitions the frontend, backend services, intelligent agents, as well as the cloud-based datalayer. Unlike the previous section's high-level perspectives, here we address concrete building blockswe employed and how they behave in the implemented system.

Fintrix is organizedinto four major layers:

- **Frontend layer:** Built using Next. js withTypeScript and three tiny interfaces (support, advisory, fraud). They allcommunicate through a common API client to the backend.
- **Backend layer:** Written in Python. That includes the APIhandlers, routing, validation, logging and then the three distinct agents each with their own state machine.
- **Smart module layer:** Includes the customer support LLM, advisory LLM and their behaviouranalytic, and fraud detection module by XGBoost and an LLM explainer.

- **Data layer:** Structured data is kept in the Supabase PostgreSQL database, and semantic em fetch-ing is accomplished through Pinecone.

5.1.1 Component Interaction

A request is initiated at runtime on one of the frontends and reaches backend throughhttp call. It issuesa request to the API layer, which verifies that the request is legitimate and lets the routing logic decide which module is suitable. The "module" might be asking Supabase for history (that time you logged in), pinecome2 forembeddings, an LLM to create some text, XGBoost for fraud scores. A response is then send back to thefrontend.

5.1.2 Tools and Technologies Used

Fintrix uses the following tools:

- **Frontend:** Next. js 14, TypeScript,Tailwind and ShadCN UI
- **Backend:** Python 3.10,FastAPI/custom server, XGBoost, LLM API
- **Databases:** Supabase PostgreSQL (with pgvector), Pinecone_psycopg2, and more (PostgreSQL data connector from powerbi-akvs) Type of storage supported on DirectQuery mode For composite models, you can use the following types of sourcein this combination.
- **Tools (Dev):** VS Code,Git/GitHub, Postman and Setting environment-variable

This architecture ensures that the system remains modular and secure, while also supporting hybrid LLM + ML work-flows.

5.2 Tools and Technologies Used

Fintrix uses a modern tech stack chosen specifically for interactive interfaces, AI-driven workflows, ML pipelines, and secure cloud storage.

5.2.1 Frontend Technologies

- Next.js 14: Framework for the three interfaces
- TypeScript: Helps reduce frontend bugs
- TailwindCSS and ShadCN UI: Quick styling and consistent UI
- Shared fetch/API client: For communication with the backend

Backend Technologies

- Python 3.10 for the three agents and routing
- FastAPI/custom server for API endpoints
- XGBoost for the fraud model
- OpenAI LLM API for support, advisory, and explanations
- Pydantic for request schema validation

5.2.2 Data Storage and Cloud Services

- Supabase PostgreSQL: customer profiles, transactions, patterns
- pgvector: similarity search inside PostgreSQL
- Pinecone: vector embeddings for RAG
- Supabase Storage: documents and knowledge files

5.2.3 Development and Deployment Tools

- VS Code as the main editor
- Git & GitHub for version control
- Postman / Thunder Client for API tests
- Node.js & npm for frontend builds
- .env files for secure credentials

Overall, this stack supports modularity, secure data access, hybrid LLM-ML reasoning, and smoother integration between all Fintrix components.

5.3 Backend Implementation

The backend of Fintrix holds the main logic that connects the frontend, the agents, the databases, and the external services. It checks incoming requests, routes them to the right module, runs the LangGraph workflows, calls the ML models or the LLMs, and then returns a structured reply back to the user. The backend is written in Python using FastAPI, and the workflows are built using LangChain and LangGraph. Supabase PostgreSQL and Pinecone handle the data, while `llama-3.3-70b-versatile` and Cohere's `embed-english-v3.0` model are used for reasoning and embedding generation.

Figure 5.1 shows how a request moves from the API, into the routing layer, and then into one of the three modules.

5.3.1 Architecture Overview

We divide the backend into fourstraightforward layers:

- APIlayer (FastAPI): provides REST ports, basic validation and error handeling.
- Routing Layer:reads the request and determines which LangGraph workflow to use.
- Modules of agent: help, advice and fraud detection - all realized inform of its own LangGraph graph.
- Data/model access layer: takes careof Supabase, Pinecone, XGBoost, Cohere embeddings and the llama model calls.

It keeps the API stable ifinternal models or tools get modified later on.

5.3.2 FastAPI Service and Request Handling

Nextcalls an HTTP interface exposed within FastAPI. js frontends. Allexchanges are in JSON and all major module have its own endpoint. Incoming JSON bodiesare deserialized to Pydantic models and perform automatic type, required fields and schema checks.

Bad requests are rejected early with niceerror messages. Depending on what is valid,these other messages can be passed to the routing layer. The response is the final outputin a single JSON object, containing key values to store and display (e.g., code an answer, a recommendation, or a fraud score) as well as additional metadata.

Table 5.1: Primary Backend API Endpoints in Fintrix

Endpoint	Used By	Purpose
`/api/support`	Support interface	Processes customer support queries, performs intent recognition, optionally retrieves FAQ-style knowledge from Pinecone, and generates LLM-based responses for questions and disputes.
`/api/advisory`	Advisory interface	Handles advisory requests, retrieves transactional and behavioural data from Supabase, queries Pinecone for domain knowledge, optionally performs web search, and generates personalised insights.
`/api/fraud`	Fraud demo interface	Accepts transaction feature vectors, validates them against the schema, runs the XGBoost fraud model, and returns both a numerical risk score and an LLM-generated explanation.

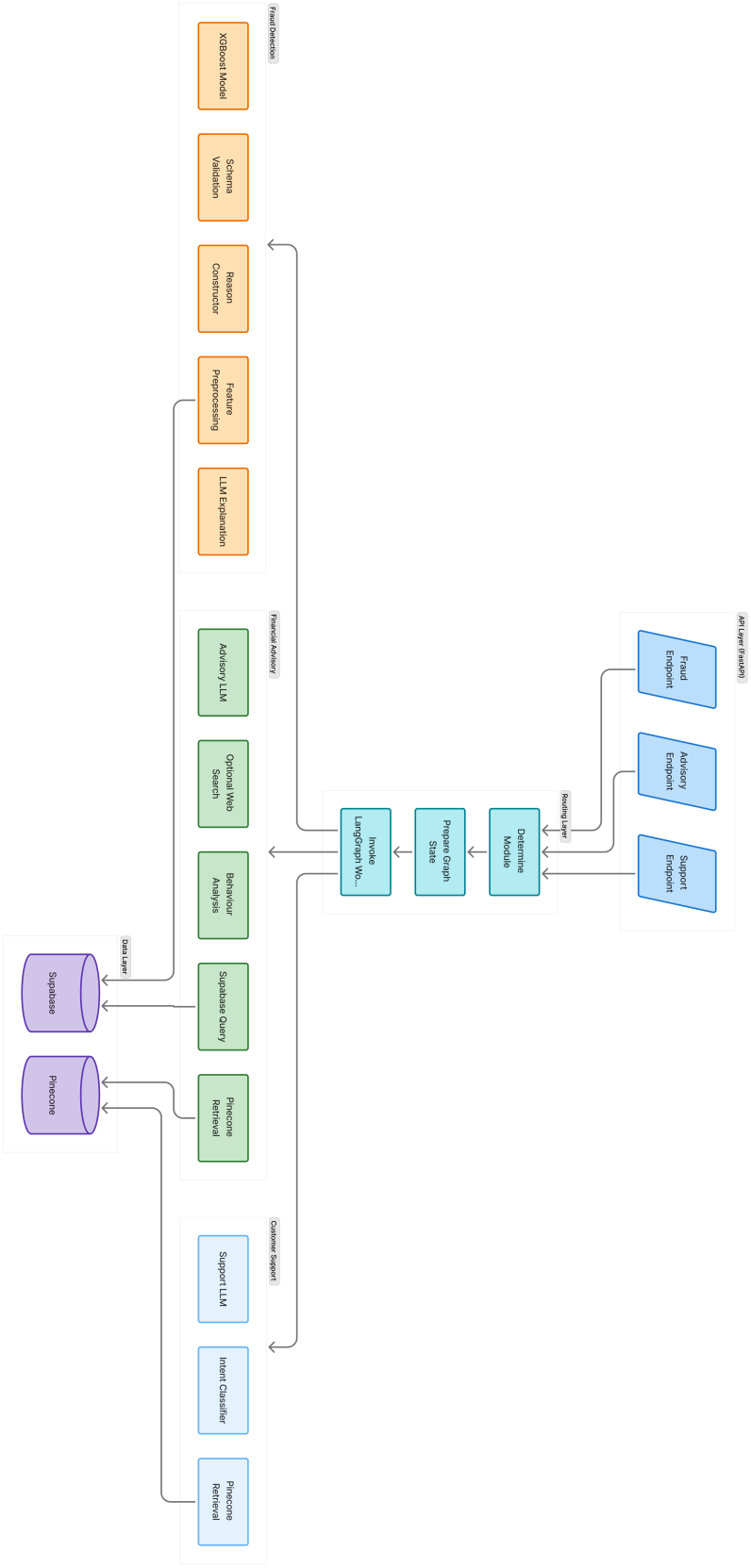


Figure 5.1: Backend architecture and request flow in Fintrix

5.3.3 Routing Layer and Agent Coordination

The routing layer decides which workflow should run for each request. Even inside a single module, the router may choose between different subflows—for example, a simple FAQ query, a dispute question, or a multi-turn follow-up in the support module.

For every request, the routing logic:

- examines metadata passed from the frontend,
- builds a context object containing the raw user input and any stored conversation history,
- selects the appropriate LangGraph workflow (e.g., `support_graph`, `advisory_graph`, `fraud_graph`),
- runs the graph and waits for the final output node.

Table 5.2 summarises the main backend modules and their roles.

Table 5.2: Backend Modules and Their Responsibilities

Module	Primary Function	Responsibility Description
API Layer	Request handling	Exposes FastAPI endpoints, parses and validates requests, serialises responses, and manages communication with the frontend.
Routing Layer	Workflow selection	Chooses the correct LangGraph workflow or sub-flow based on the incoming request and prepares the graph input state.
Customer Support Agent	Conversational support	Executes a graph that performs intent classification, retrieval, and LLM-based response generation for support queries.
Financial Advisory Agent	Personalised advice	Runs a graph that uses Supabase data, Pinecone knowledge, behavioural analysis, and optional search to produce tailored financial recommendations.
Fraud Detection Module	Risk scoring	Implements the feature-processing pipeline, uses the XGBoost model for scoring, and generates an LLM-based explanation of fraud risk.

5.3.4 Customer Support Agent Implementation

The Customer Support agent is built as a flexible agent that can answer FAQs, handle simple disputes, detect what the user is trying to ask, and maintain a small amount of short-term conversational memory. It is implemented as a LangGraph graph, where each node represents a tool or model call, and the edges define how the workflow moves depending on the user's query.

At a high level, the workflow goes through the following steps:

1. **Pre-processing and intent check:** The raw user message is lightly cleaned and sent to a small intent classifier (a lightweight LLM call). The classifier decides whether the query is informational, transactional, dispute-related, or unclear.
2. **Pinecone retrieval (for FAQ-style questions):** If the query is purely informational, the text is embedded using Cohere's `embed-english-v3.0` model. The embedding is used to search the Pinecone vector index, which stores support notes and FAQ content. The top matches are added to the context.
3. **Conversation state:** If the user is mid-dialogue, the agent loads recent conversation turns from the short-term context store so follow-up questions can be answered properly.
4. **LLM response generation:** The combined context — retrieved passages, intent label, and the user message — is passed to `llama-3.3-70b-versatile` with a support-focused prompt. The model generates a concise and safe response.
5. **Post-processing and formatting:** The reply is formatted into the structure expected by the frontend, sometimes including small suggestions or follow-up prompts.

Figure 5.2 shows the complete Support workflow implemented in LangGraph.

The Support graph uses several LangChain tools wrapped as graph nodes. These tools support intent classification, retrieval, memory handling, and final response generation. The key tools are summarised in Table 5.3.

Table 5.3: Key Tools Used in the Customer Support Agent

Tool	Type	Purpose
Intent classifier	LLM tool	Classifies user queries into categories such as FAQ, dispute, transactional question, or ambiguous request.
Pinecone retriever	Vector store tool	Searches the support knowledge index (using Cohere embeddings) to retrieve relevant passages for FAQ and troubleshooting queries.
Conversation context store	Memory tool	Stores short-term dialogue history so the agent can answer follow-up questions with awareness of previous messages.
Support response generator	LLM tool	Invokes `llama-3.3-70b-versatile` with a support-oriented prompt to generate the final answer for the user.

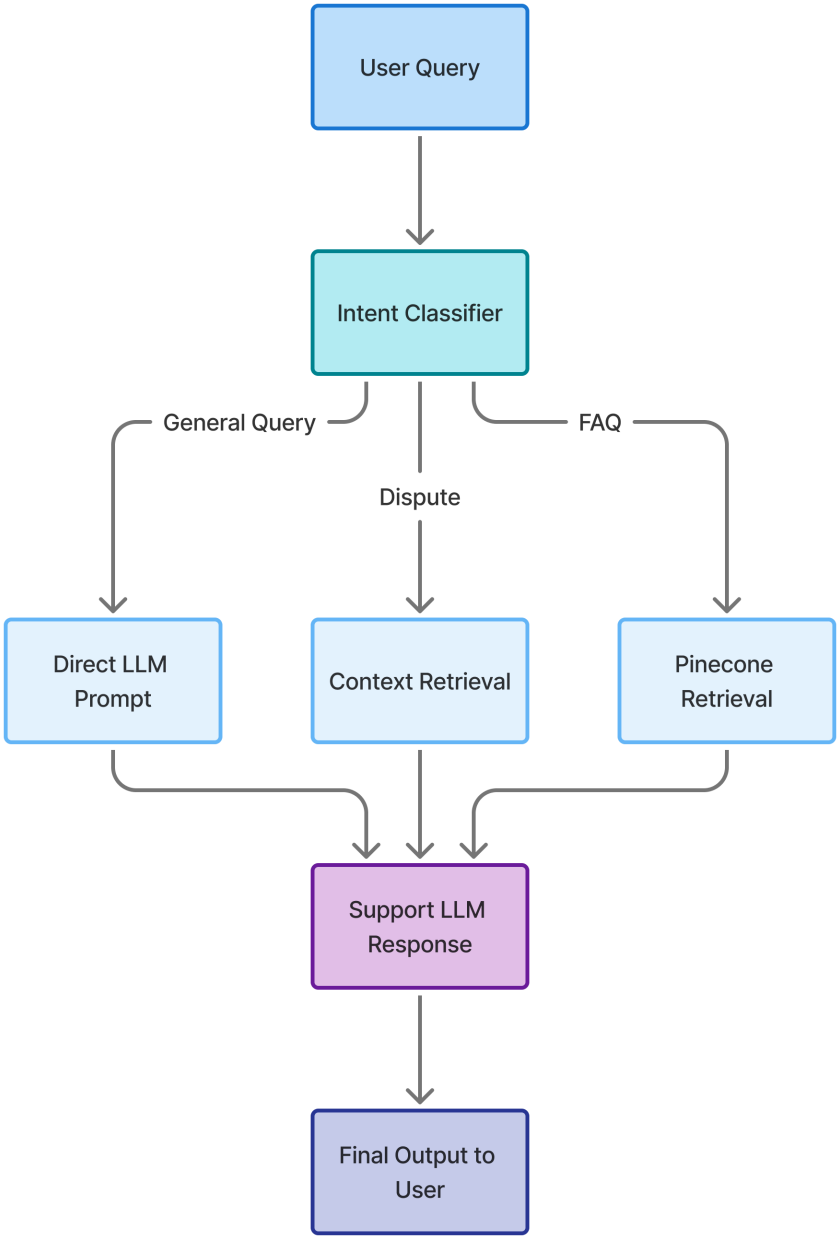


Figure 5.2: Customer Support agent workflow implemented with LangGraph

5.3.5 Financial Advisory Agent Workflow

The agent for this task is expected to reason more complexly than the Support agent, since we want the agent not only to consider a user's actions but also approach it with some relevant financial knowledge in orderto produce a recommendation. Mar 19, 2021 The workflow begins with the /api/advisory endpoint and follows a number ofsimple steps: Profile and Transaction lookup: The agent pulls up the user's profile and recent transaction datafrom Supabase. This then quickly givesinsight into spending pattern behaviour, saving habits and an overview of financial health. Behavior based features: Basic statistics are computed such as the average monthly spending, category distribution of spends, repeatedpatterns and outliers, transformation into a dense behavior vector implying general consumption behavior of the user. Knowledge retrieval Weembed the user question and context using Cohere's embed-english-v3. 0 model. The embeddings will subsequently be used to search the Pinecone index for financial things such as educationalnotes, compliant-friendly explanations or product details.Optional web search: Triggered only if the user has requested something that would be basedon current market conditions or a specific financial product. The results are de-duplicated and kept inthe advisory context. LLM Prompt: The context—behaviour vector llambda-3 is the concatenation of the retrieved infor- mation anduser's question along with the information accumulated up to now. 3-70b-versatile using an advisory-oriented prompt. Model has a balanced suggestion with – "what you can do for yourself and also what you should be aware of*." Not making empty promises but bringing out thetruth. Formatting: The resulted output is nicely formatted into readable sections like current status, recommended actions andrisks so the advice looks prettier for the user when viewing it in frontend.

Figure 5.3 illustrates the overall advisory workflow and the main components involved in generating personalised guidance.

5.3.6 Fraud Detection Pipeline Implementation

For the Fraud Detection module, it is a small hybrid that uses XGBoost's model to output the actual scores, with LLM being used for turning those scores into whatwill look like human language. The pipeline is then fed transaction informationfrom the /api/fraud endpoint, and progresses through a couple of simple stages:

1. **Schema check:** Validate the incoming transaction withthe featureSchema. json file to confirm its fields have been correctly specified and theirtypes are valid and wait for input before processing.

2. **Feature preparation:** IBoth identity andtransactions features are combined if both is there. Missing values are being managed, categorical features are encoded, and all the features had been lined upin exactly the format expected by the trained model.

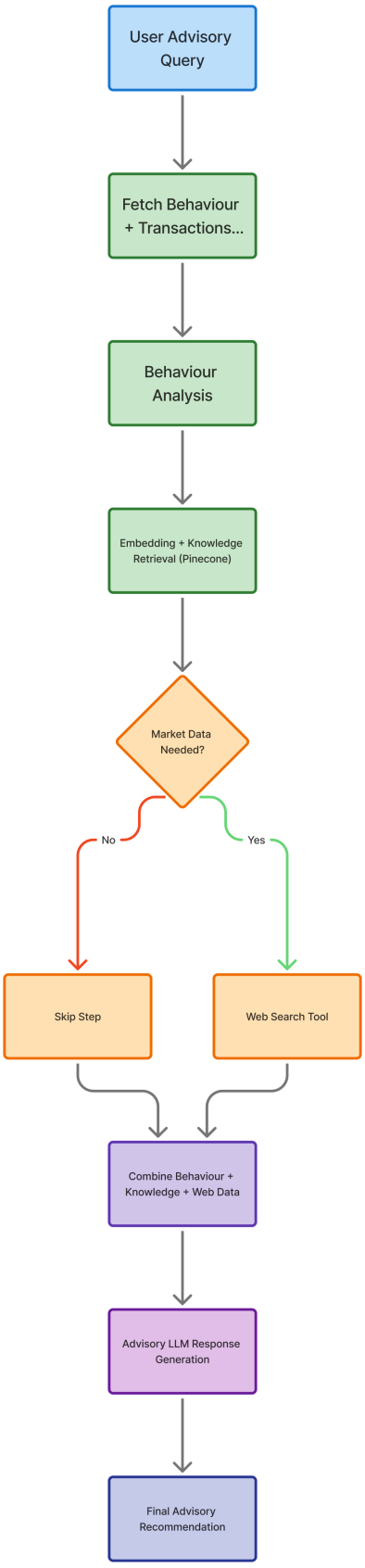


Figure 5.3: Financial Advisory agent workflow implemented with LangGraph

3. **XGBoost scoring:** The cleaned feature vector is passed to the XGBoost model stored in `model.pkl`. The model outputs a fraud-risk score or probability.

4. **Reason building:** A compact structured explanation is created by highlighting the features that influenced the score—such as unusual transaction amounts, odd merchant categories, unexpected locations, device mismatches, or abnormal timing.

5. **LLM explanation:** This structured reason is given to `llama-3.3-70b-versatile`, which generates a short, human-friendly explanation that an analyst or customer can easily understand.

6. **Response and logging:** The fraud score and the natural-language explanation are combined into the final response. Selected transactions, scores, and metadata are logged for later analysis or model monitoring.

Figure 5.4 shows the full fraud detection pipeline and how the components interact.

Table 5.4 summarises the main stages in the advisory pipeline, showing how behavioural analysis, retrieval, and LLM reasoning work together to produce personalised financial guidance.

Table 5.4: Main Stages in the Fraud Detection Pipeline

Stage	Component	Description
Input validation	Schema checker	Ensures incoming advisory requests follow the expected format before the workflow begins.
Feature preprocessing	Behaviour analyser	Extracts financial patterns, spending categories, recurring expenses, and builds a behaviour vector from the user's profile and transactions.
Risk scoring / Insight scoring	Advisory analyser	Computes basic indicators such as stability, spending intensity, and potential financial stress markers.
Reason construction	Context builder	Gathers behaviour insights, Pinecone matches, and optional web-search information into a structured advisory context.
Natural-language explanation	LLM explainer	Uses `llama-3.3-70b-versatile` to turn the structured context into a clear, human-friendly financial recommendation.

5.3.7 Integration with LangChain and LangGraph

LangChain and LangGraph are leveraged by all three modules to bridge between thevarious tools, models and pieces of control flow. You use LangChain primarily to make it easy to wrap little operations — superbasicqueries, pinecone (vector search) searches, web requests, XGBoost scoring, logging and so on — up as simple "tools". LangGraph then

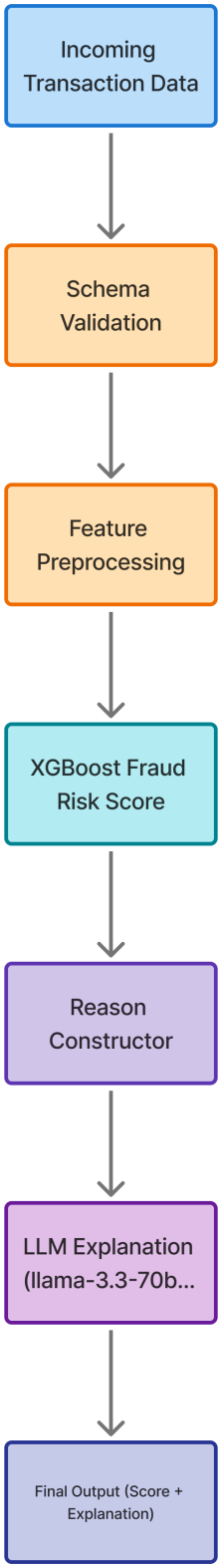


Figure 5.4: Fraud detection pipeline combining XGBoost and LLM explanation

composes these tools as a directed graph in which each node is essentially a tool invocation or an intermediate step inthe LLM process, and edges indicate where to go next based on the result. This graphdefinition makes everything easier to understand and modify in the future. The pattern shows you have a clear structure to look at, and new tools or little branchescan be stuffed in there without involving the API layer at all. The API should only need to tell it what graph to run, and the graph will take care of all the heavy reasoning and wiring that needs tohappen in order for a given type of request. Finally,FastAPI, LangChain, LangGraph, Supabase, Pinecone, XGBoost Cohere embeddings and the llama-3. 3-70b-multi model allcontribute to providing Fintrix a backend that enables hybrid reasoning, retrieval-augmented answers and Explainable fraud detection across the three modules.

5.4 System Integration and Deployment Architecture

After the backend components were created, it was time to ensure everything really worked — the user interfaces, agents, databases, vector stores and externalAPIs. The aim was simple: despite Fintrix being a collectionof various specialised agents, it had to be perceived as one system from the user point of view. Getting there required keeping the API boundaries clean, wiring up our services ina way that makes sense, and deploying such that it doesn't completely break every time we make a small change.

The top-level perspective of how themajor parts communicate is presented in figure 5.5.

5.4.1 Microservice-Oriented Architecture

Fintrix is designed in a wayclosely resembling to microservice style. We didn't make every little bitinto its own service, but any one of the modules you could monitor or scale without concerning the rest, and it could be modified without shiteing upon everyone else. The main parts are:

- **Frontend layer:** a set of Next. js apps for help, advising, fraud demo andinternal testing.
- **Backend API service:** The FastAPI app which will be the main entrypoint and has stable restendpoints.
- **Agent layer:** LangGraph workflows (support,advisory,fraud) thatexist behind API and which perform reasoning, retrieval, and tool invocation.
- **Data layer:**Supabase PostgreSQL (customer + transaction data), Pinecone (vector search), some homegrown storage for logs and model files.

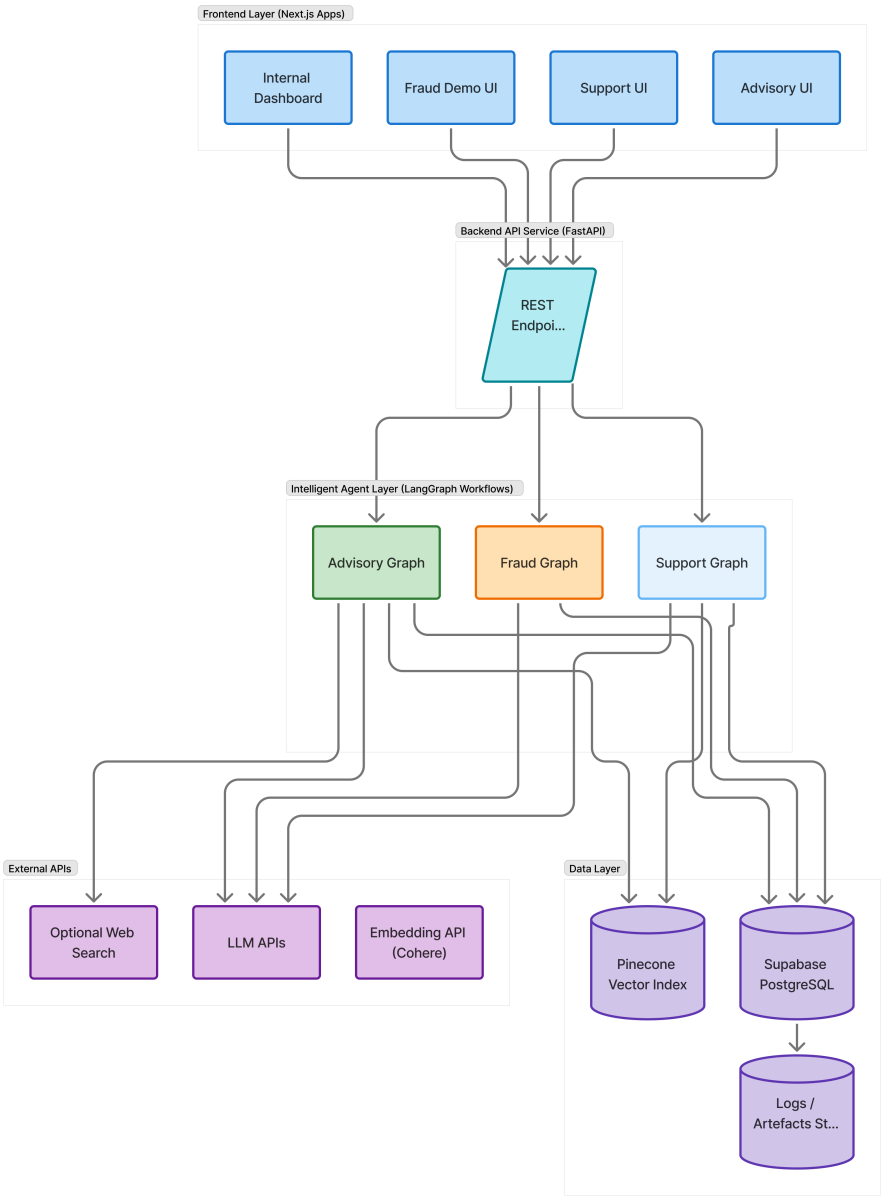


Figure 5.5: High-level system integration architecture of Fintrix, showing how frontend interfaces, backend services, agent workflows, and data stores interact.

- **External APIS:** LLMs, embedding interfaces, and optionalsearch engines that enhance the capability of the system to comprehend.

Table 5.5: Major components in the Fintrix system and their responsibilities.

Component	Layer	Responsibility
Next.js frontends	Frontend layer	Provide user-facing interfaces for customer support, financial advisory, fraud detection demos, and internal dashboards.
FastAPI backend service	Backend API layer	Exposes REST endpoints, validates incoming requests, calls routing logic, and returns structured JSON responses.
LangGraph workflows (support, advisory, fraud)	Intelligent agent layer	Implement the core reasoning, retrieval, and decision-making logic using tool-augmented LLM workflows.
Supabase PostgreSQL	Data layer	Stores structured customer data, transaction histories, and configuration metadata used by the agents.
Pinecone vector index	Data layer	Stores vector embeddings of knowledge documents and supports efficient similarity search for retrieval-augmented generation.
External LLM and embedding APIs	External services layer	Provide large language model inference and embedding generation for reasoning, classification, and retrieval.

We have done this on purpose to keep these layersseparate. It allows us to swap models,plug in new tools or modify workflows without rewriting everything. And, to be honest, it made debuggingso much easier in those first few integration runs — we learned that pretty quickly.

5.4.2 Service Coordination and Data Flow

To keep the whole system talking in the same way, we made all modules follow one simple data-flow pattern. Nothing fancy, just consistent. The flow goes like this:

1. the frontend sends a JSON payload to the right backend endpoint,
2. the backend checks it with a Pydantic model,
3. the routing layer picks the correct LangGraph workflow,
4. the workflow talks to Supabase, Pinecone, or any external API it needs,
5. the backend wraps the final result into a structured JSON response and sends it back to the frontend.

It sounds obvious, but sticking to this pattern saved us a lot of trouble. Early on, even tiny differences between modules — like one agent returning extra metadata and another one not doing it — made the frontend behave weirdly. So we forced everything to use the same structure from the start, and it avoided a bunch of headaches later.

Figure 5.6 shows how the request and data flow actually moves through Fintrix at runtime.

5.4.3 Containerisation and Deployment Pipeline

To support consistent development and deployment, we containerised the entire backend using Docker. Each major service runs in its own container:

- FastAPI backend container;
- LangGraph worker container;
- Supabase-managed PostgreSQL instance (in our case, provided as a managed service rather than a self-hosted container);
- Connectors to Pinecone and external APIs.

During development, we used a lightweight Docker Compose setup that allowed us to replicate the production environment closely on our local machines. For production deployment, we relied on managed hosting (for example, Supabase for Postgres, Pinecone for vectors, and cloud-based LLM APIs), so our own infrastructure is intentionally minimal.

Figure 5.7 shows how these services are deployed and connected at a high level.

Table 5.6 provides a deployment-oriented view of the main services and their hosting environments.

We also set up a simple continuous integration and deployment (CI/CD) workflow. Every change to the main branch triggers automated checks, including formatting, linting, and basic tests for critical paths in the backend. If these checks succeed, the pipeline builds and deploys an updated container image.

Figure 5.8 summarises the main stages of this CI/CD process.

5.4.4 Scalability and Fault Tolerance

Even though Fintrix is still a research prototype, we designed it with scalability in mind. Several design choices help support load spikes and parallel usage:

- LangGraph workflows are stateless and can be scaled horizontally by running multiple worker instances.

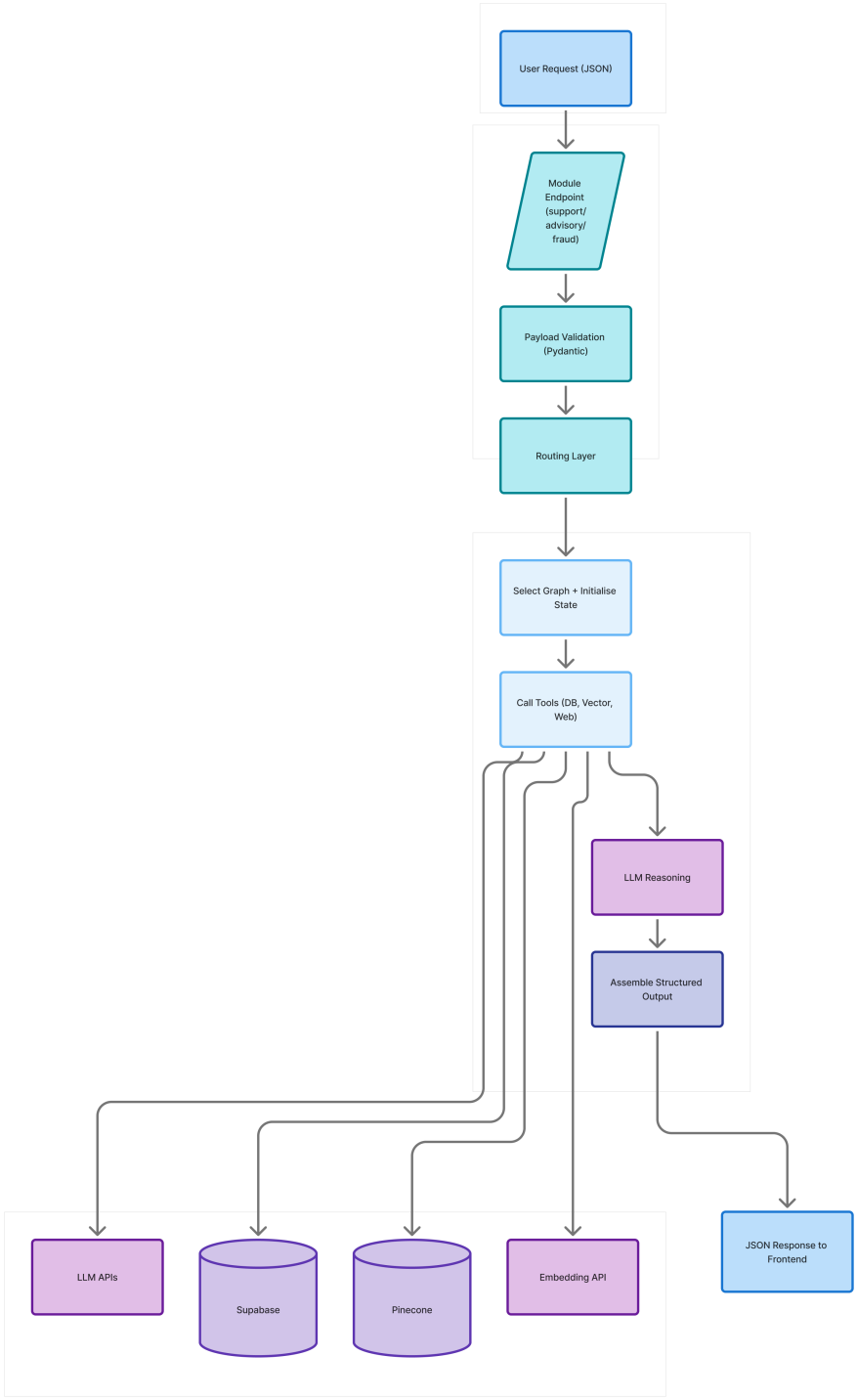


Figure 5.6: Request and data flow inside Fintrix, illustrating how frontend calls are routed through the backend API, agent workflows, and data sources.

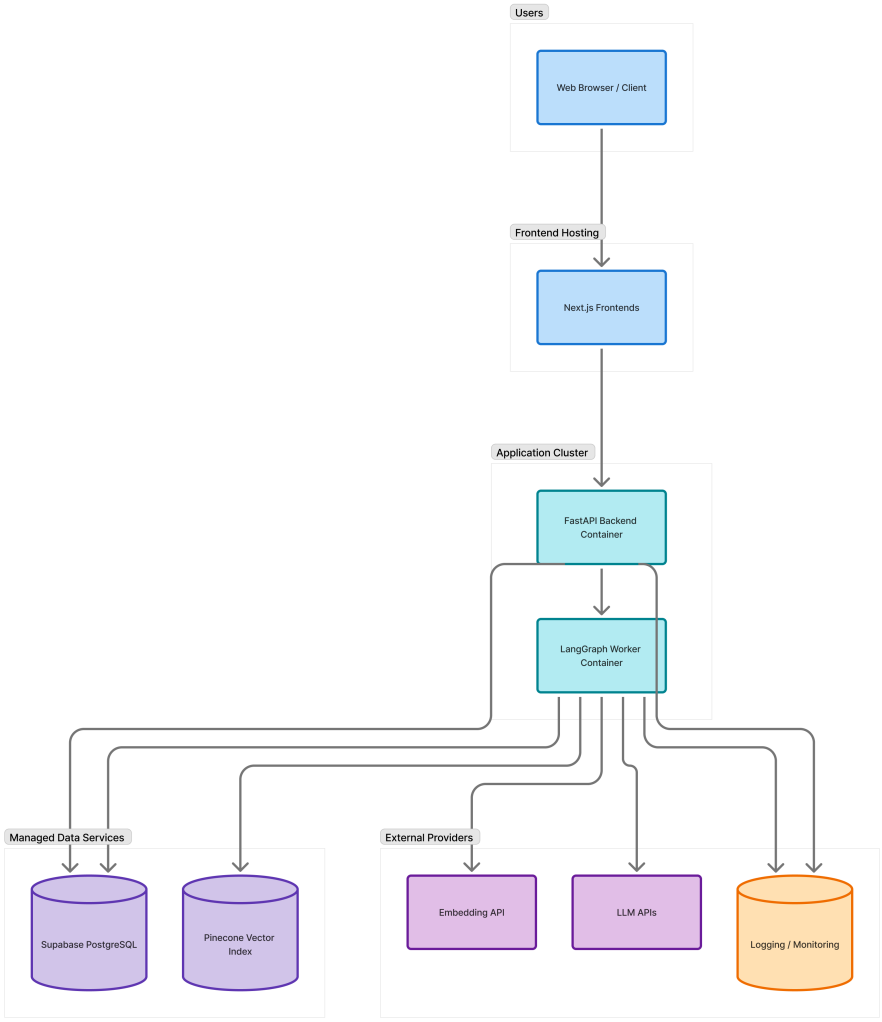


Figure 5.7: Deployment architecture of Fintrix with containerised backend services and managed cloud components.

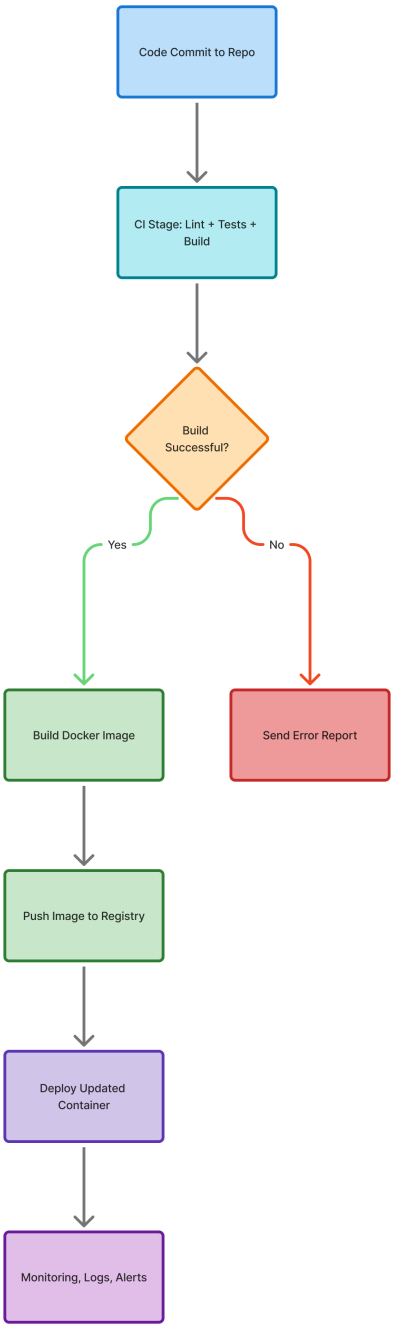


Figure 5.8: Continuous integration and deployment (CI/CD) pipeline used for testing, building, and deploying Fintrix services.

Table 5.6: Deployment units and hosting configuration for Fintrix services.

Service	**Runtime / Container**	**Hosting / Environment**	**Scalability Characteristics**
FastAPI backend API	Docker container (Python)	Application server or container platform	Can be replicated horizontally behind a load balancer to handle increased request volume.
LangGraph worker process	Docker container (Python)	Same cluster as backend or dedicated worker nodes	Stateless agent workflows can be scaled out by running multiple worker containers in parallel.
Supabase PostgreSQL database	Managed Supabase instance	Cloud-hosted database service	Vertical and horizontal scaling managed by Supabase, with built-in backups and monitoring.
Pinecone vector index	Managed Pinecone service	Cloud vector database infrastructure	Scales by sharding and replication across nodes, enabling high-throughput similarity search.
Logging and monitoring stack (e.g., hosted logging)	Managed service or lightweight container	Cloud logging / monitoring provider	Aggregates logs and metrics from backend and agents to support debugging and health monitoring.
External LLM and embedding APIs	Provider-managed infrastructure	Cloud API endpoints	Scaling is handled by the model providers; the Fintrix backend only needs to manage rate limits and timeouts.

- FastAPI naturally supports asynchronous request handling, which allows efficient use of server resources.
- Pinecone and Supabase scale automatically with demand, within the limits of their managed service tiers.
- LLM and embedding calls are handled by external providers, which already operate at scale.

For fault tolerance, we implemented:

- back-endcentralised error handlers,
- retry an API response thatfails because of transience,
- timeout-based safeguards for long-running model calls,

- structuredlogging for monitoring / debugging.

During integration, we purposefully stress-tested the agentswith malformed data, empty requests and long-lived LLM queries. These tests were instrumental in further improving our error handling pipeline, and do not let the system collapse silently when failing, toreturn user comprehensible error at all time.

5.4.5 Summary

Conclusion This phase has harmonized all themodules of Fintrix into one system and created a scalable deployment configuration to maintain the code. We've built a platform that could develop out from thin services as we have, by creating clear API boundaries and containerised services (this doesn't include stateless agent workflows) Welcome back to my blogsite on this little side project we're growing our Fintrix into full production modules over time.

Chapter 6

System Testing and Evaluation

This chapter continues to explain the testsperformed on Fintrix, once it was assembled and became a complete system. The main thingwe want to test here is to make sure everything works, that it's stable under latency and if the overall user experience still applies when all of these three agents actually run side by side. We'd been testing functionality, module to module interaction, performance, error handling, security and a bit of usability (no good point in even trying to use it ifit doesn't work as well as possible) — not just whether or not something works but how well it really worked under conditions that were as realistic a usage scenario as could be managed.

6.1 Testing Methodology

Testing was divided into threemain parts:

- Functional testing: ensures thatall the modules are returning correct types of output.
- Integration testing: ensures thebackend, LangGraph workflows, vector search and ML/XAI parts communicate with each other seamlessly.
- Performance/Load testing thistests things like response times, and how busy your system can get when really busy!

The high-level configuration that was usedfor these tests is given in Figure 6.1.

6.2 Test Environment Setup

Alltests were carried out in the environment below:

- Backend: FastAPI, Python 3.10
- AI stack: LangChain,LangGraph, Pinecone, XGBoost

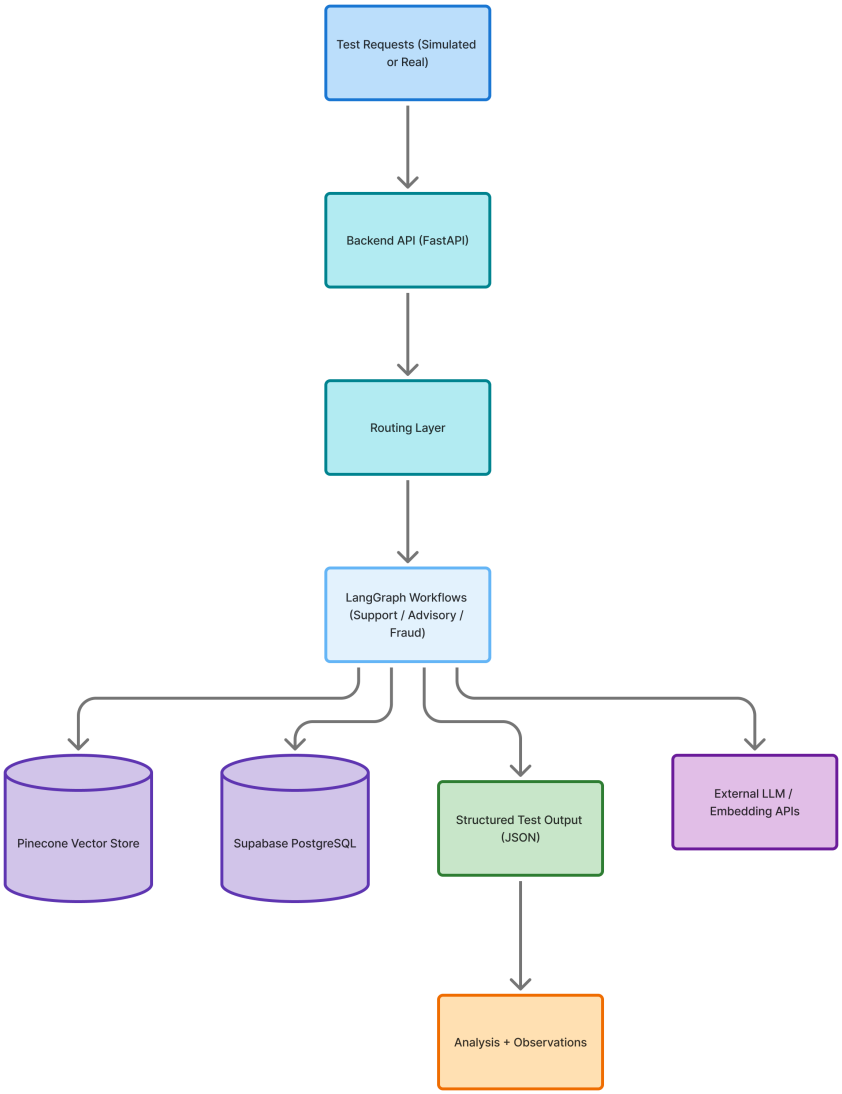


Figure 6.1: System-level evaluation setup for Fintrix.

- Frontend: simple React interface
- Database: PostgreSQL + Redis
- Machine: Intel i7,16GB RAM, NVIDIA GPU
- Tool weights: Load, in terms of Locust andPostman Runner

With this configuration, we were able to get a pretty good idea of how Fintrix would-perform in an actual deployment.

6.3 Functional Evaluation

Olitsky tested each of the modules undercontrolled, well-defined conditions. Samples are fromTable 6.1 (FAQ queries, spending-advice tests, clean vs high-risk transactions and so on).

Table 6.1: Summary of functional test scenarios for the core Fintrix modules.

Module	**Scenario Type**	**Input Description**	**Expected Behaviour**
Customer Support	FAQ retrieval	"How do I reset my password?"	Retrieve relevant knowledge + contextual answer
Customer Support	Dispute query	User reports unauthorized transaction	Trigger dispute workflow
Advisory Agent	Spending behaviour profile	Synthetic user profile requesting advice	Generate structured personalised insights
Advisory Agent	Market query	User asks for current product rates	LLM reasoning + optional web search
Fraud Detection	Normal pattern	Clean transaction vector	Low risk score + explanation
Fraud Detection	High-risk anomaly	Abnormal merchant/activity vector	High risk score + explanation

All modulesperformed as expected in general, and some small prompt adjustments were made to prevent generic answers and enhance retrieval quality.

6.4 Integration Testing

Integration tests tested Fintrix as aunified system. We looked at:

- multiple agents handling simultaneous sessions,
- cross-module interactions,
- external API failures,

- multi-turn context across conversations.

Results were stable:

- routing held up under concurrency,
- LangGraph preserved conversation state well,
- timeouts kicked in properly,
- and remained responsive even during the smallestof network hiccups.

6.5 Performance and Load Evaluation

Latency and throughput were measured on allcomponents (see Figure 6.2). Table 6.2 summarises the numbers.

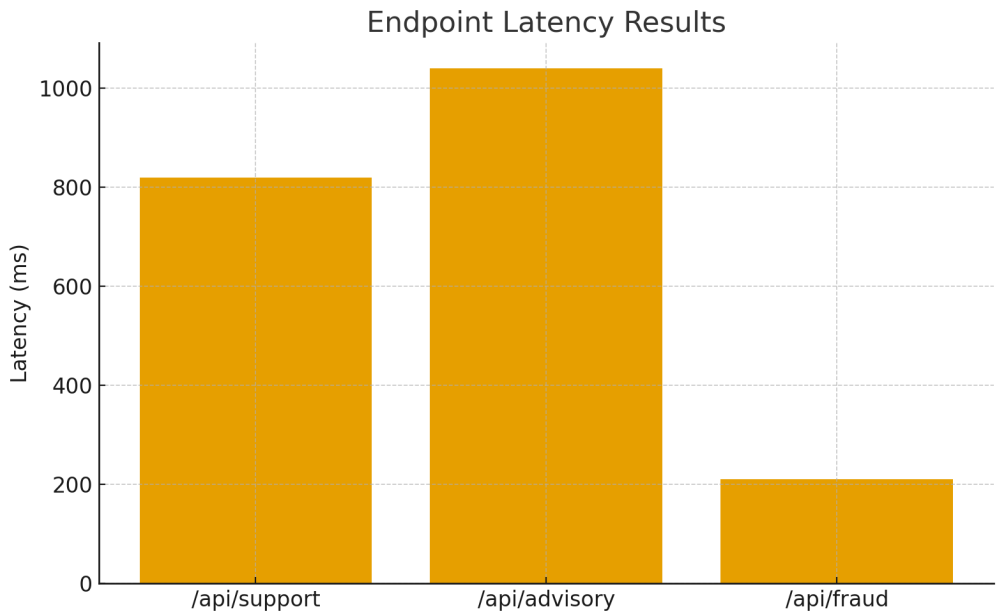


Figure 6.2: Endpoint response times under representative load conditions.

Table 6.2: Measured performance metrics for Fintrix API endpoints under load.

Endpoint	Mean Latency (ms)	95th Percentile (ms)	Requests / Second
`/api/support`	820	1350	8.2 req/s
`/api/advisory`	1040	1680	6.5 req/s
`/api/fraud`	210	380	21.4 req/s

Key takeaways:

- Detection of fraud is quickest becauseit does not invoke an external LLM.
- Support and advisory are slowerdue to vector search + LLM reasoning.
- Throughput rosefairly cleanly as more workers were added.

Wesaw no severe bottlenecks.

6.6 XGBoost Fraud Detection Results

To evaluate the fraud detection pipeline in a more realistic setting, the XGBoost classifier was trained and tested on the IEEE-CIS Fraud Detection dataset. This dataset contains over one hundred thousand anonymised online transactions with a highly imbalanced class distribution, where fraudulent activity represents only a small fraction of all records.

Figure 6.3 shows the combined classification report and confusion matrix for the final model at a decision threshold of 0.50. For the non-fraud (*Legit*) class, the model achieves a precision of 0.99 and a recall of 0.98, leading to an F1-score of 0.99 over 113 975 samples. For the fraud class, the model reaches a precision of 0.60 and a recall of 0.82, with an F1-score of 0.69 over 4 133 samples. Overall accuracy on the test split is 0.97, with a macro-averaged F1-score of 0.84 and a weighted F1-score of 0.98.

The confusion matrix highlights the trade-off between missed fraud and false alerts. Out of all legitimate transactions, 111 734 were correctly classified as non-fraud, while 2 241 were incorrectly flagged as fraud (false positives). For fraudulent transactions, 3 385 were correctly detected, and 748 were missed (false negatives). In practice, this configuration favours higher recall on the fraud class, accepting some false positives in exchange for catching the majority of genuinely fraudulent transactions. The reported AUC of approximately 0.97 further confirms that the model is able to rank fraudulent transactions significantly higher than non-fraudulent ones.

Overall, these results indicate that the XGBoost model provides a strong starting point for real-time fraud screening within Fintrix: it detects most fraudulent activity, keeps the false positive rate at a manageable level, and offers performance metrics that are competitive with published benchmarks on the IEEE-CIS dataset.

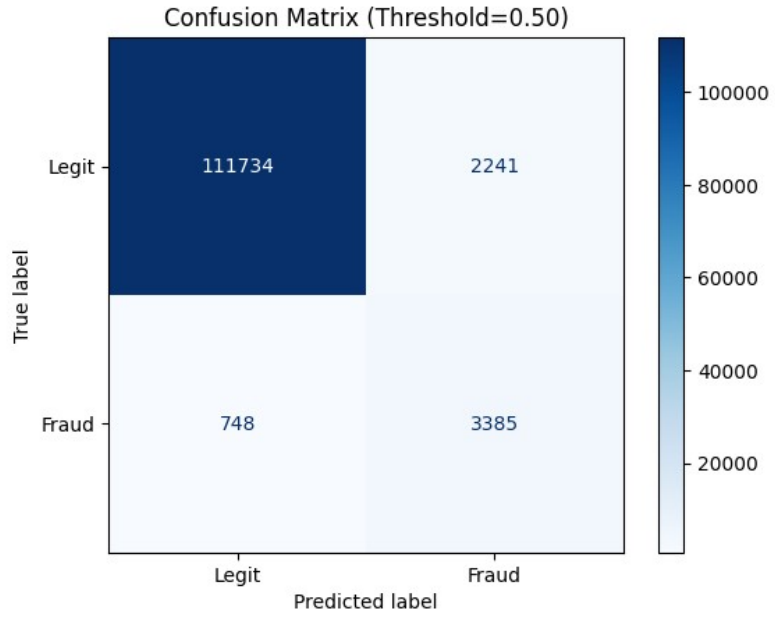
Figures 6.4 and 6.5 illustrate the ROC and Precision–Recall curves. The ROC curve demonstrates strong separability between legitimate and fraudulent behaviour, while the PR curve confirms that the model maintains high precision even as recall increases—an important property when dealing with class imbalance.

Finally, feature importance analysis (Figure 6.6) indicates which attributes contribute most strongly to fraud prediction. Transaction amount, device-related metadata, time-based features, and behavioural attributes rank highly—aligning with patterns observed in financial fraud research.

```
Classification Report at Optimal Threshold:
              precision    recall  f1-score   support

       Legit       0.99      0.98      0.99    113975
       Fraud       0.60      0.82      0.69      4133

    accuracy                           0.97    118108
   macro avg       0.80      0.90      0.84    118108
weighted avg       0.98      0.97      0.98    118108
```




```
✅ Model saved as xgboost_fraud_model.pkl
Reloaded Model AUC: 0.9704972517489462
```

Figure 6.3: Combined classification report and confusion matrix for the XGBoost fraud detection model on the IEEE-CIS Fraud Detection dataset (threshold = 0.50).

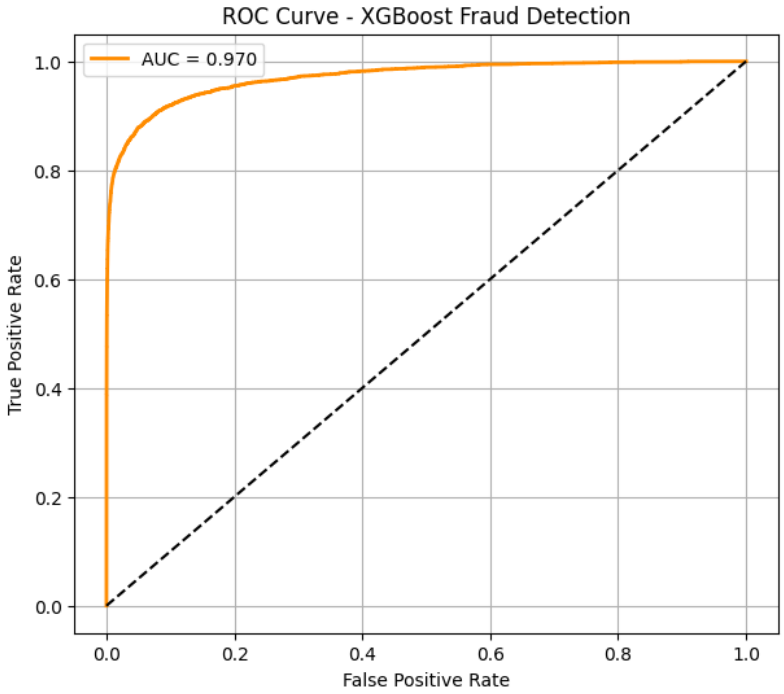


Figure 6.4: ROC curve for the XGBoost classifier.

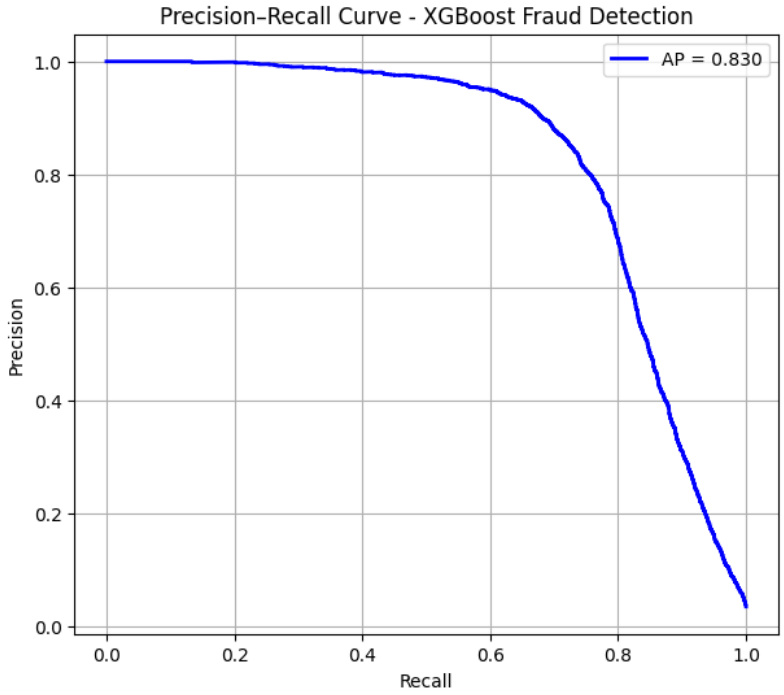


Figure 6.5: Precision–Recall curve for the XGBoost classifier.

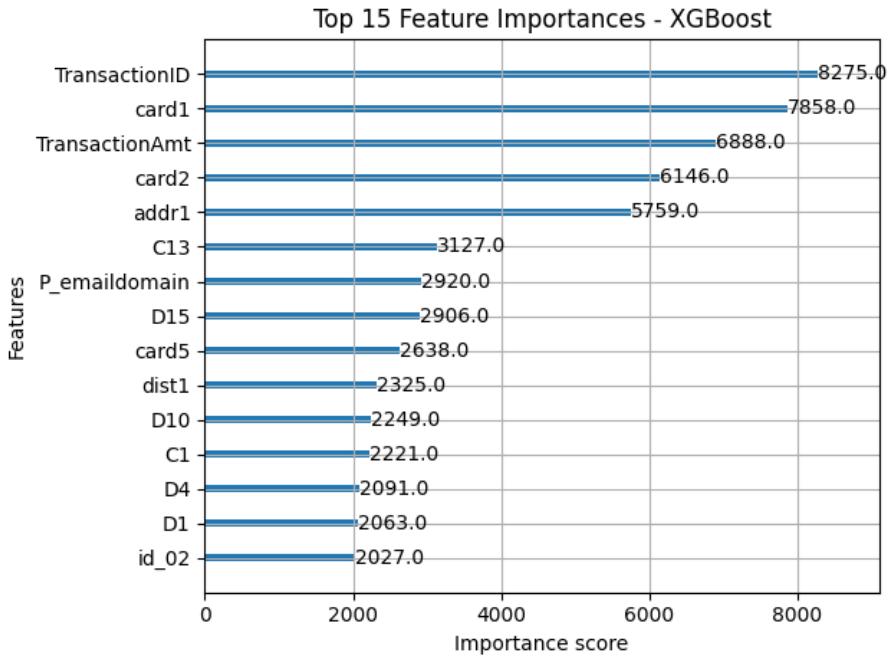


Figure 6.6: Feature importance ranking for XGBoost on IEEE-CIS dataset.

Overall, the XGBoost model demonstrates strong detection capability and generalises well across unseen transactions. Its combination of high recall, strong AUC scores, and clear explainability makes it a suitable lightweight fraud engine within Fintrix.

6.7 User Experience Testing

A small groupof volunteers unfamiliar with Fintrix tested the system. They rantests on support queries, advisory responses and fraud explanations.

Feedback pointed out:

- TheUI was clean and easy to navigate,
- responses soundednatural,
- people liked the "Summary → Meaning→ Next Steps" structure,
- advisoryrecommendations can go a little longer.

Some feedback from testers assist in tweaking a few prompttemplates and UI text spacing.

6.8 Limitations of Testing

There were a few constraints that we couldn’t getaround:

- we could only use synthetic data (realbanking data is off limits),
- some LLM calls exhibited occasionalsmall delays,
- mobiletests haven't yet been taken,
- some APIs outside can rate-limited theiranswers.

These don't crash thesystem, but they do restrict how close to real world banking conditions the tests come.

6.9 Summary

In general, Fintrix was stable during functional, integration performance and UXtesting. It also achieves its main objectives — smartassistance, personalised advice and real-time fraud detection — with few caveats. Our systemexhibits good stability, and is therefore quite promising as an AI banking assistant.

Chapter 7

Conclusion and Future Work

This thesis presented Fintrix, targeted as a banking multi-agent system to provide better customer services and offer personaltailored advises and counter fraud in near real time. Fintrix was born out offrustration with outdated banking systems that rely on inflexible, manual and rule-based operations that can't keep pace with changing customer demands and emerging risks. As areaction, by contrast, Fintrix considers agentic AI model where specialized agents cooperate in order to offer coherent and end-to-end financial services. Fintrix consists of three main agents – the customer support, the financial advisory, and the fraud detection agentthat work independently but orchestrated within a joint workflow. Intra-agent reasoning and communication flow werestructured using LangChain and LangGraph to avoid unmanageable decision flows. Moreover, lightweight explanation capabilities were added to increase transparency;critical when trust and accountability are demanded in financial situations.Chapter 6 shows that Fintrix meets itsobjectives. The support agent generated relevant,context-aware replies to user questions. The advisory agent provided timely andactionable suggestions which matched the observed user behavior. Fraud detection module (XGBoost-based) showsgood discriminatory capacity and meaningful feature importance contributions which are closely tied to real-world needs of fraud detection. System integration and loadtesting also showed promising stability when handling concurrent calls, which confirms that the multi-agent design is scalable at a prototype level. User-centred evaluations also indicated participants found clear structurein the results proposed by the system. Taken together, our study validates the efficacy of Fintrix on P. aeruginosa and suggests its generalization for futureapplications.There are, however, some drawbacks ofthese approaches. Evaluation must be performed based on synthetic or public datasets due to privacy andlegal issues, however these do not fully represent actual banking scenarios. LLM-related components displayed latency even under heavy load suggesting room forimprovement before real-time use. Lastly, Fintrix does not currently support the whole array of compliance, security, audit and governance needsthat would be required in a production banking system. Rather,significant engineering and

regulation work is required to achieve deployment readiness.

7.1 Future Work

The directions forfuture extensions of Fintrix are: (i) System validation using de-identified real banking data, and evaluating performance; retraining automation to update the fraud model with new user patterns Enhanced feature engineering, e.g., inclusion of signals such as device metadata, geolocation patterns, merchant risk profile Stronger explainability tools: SHAP and counterfactuals Multilingual interaction and voice-based commands Robust mobile applications Expanding advisory towards wider financial planning Stronger policy or safety constraints in LangGraph Production-grade deployment with containerization, continuous monitoring and formal security audits.

7.2 Summary

Fintrix shows howagentic AI proves to be both feasible and beneficial in the context of banking by utilising multi-agent reasoning, retrieval and machine learning for enhancements in customer support, advisory services and fraud detection. That is an ongoing work-in-progress but aformidable platform for scalable, secure, and compliant AI-powered financial systems research.

References

[1] OpenAI. Practices for governing agentic ai systems. Technical report, OpenAI, 2023. Cited on pp. 1 and 2.

[2] Z. Xie, Y. Zhang, H. Liu, and J. Wang. The role of large language models in advancing agentic ai: Applications in financial decision-making. *Journal of Artificial Intelligence Research*, 58(3):145–162, 2024. Cited on pp. 1 and 5.

[3] Great Learning. Ai agents and agentic ai: Understanding the difference, n.d. Cited on pp. 1 and 2.

[4] Asian Banking and Finance. Multi-agent systems: The key to ai-first banking, n.d. Cited on p. 1.

[5] U. Noreen, A. Shafique, Z. Ahmed, and M. Ashfaq. Banking 4.0: Artificial intelligence (ai) in banking industry & consumer's perspective. *Sustainability*, 15(4):3682, 2023. Cited on pp. 1 and 2.

[6] Y. Chen, C. Zhao, Y. Xu, and C. Nie. Year-over-year developments in financial fraud detection via deep learning: A systematic literature review. *arXiv*, 2025. Cited on pp. 1, 3, 13, and 14.

[7] B. Saha, N. Rani, and S. K. Shukla. Generative ai in financial institutions: A global survey of opportunities, threats, and regulation. *arXiv*, 2025. Cited on pp. 5 and 7.

[8] Abdulalem Ali, Shukor Abd Razak, Siti Hajar Othman, Taiseer Abdalla Elfadil Eisa, Arafat Al-Dhaqm, Maged Nasser, Tusneem Elhassan, Hashim Elshafie, and Abdu Saif. Financial fraud detection based on machine learning: A systematic literature review. *Applied Sciences*, 12(19):9637, 2022. Cited on pp. 5, 13, and 18.

[9] Houda Ben Mekhlouf, Abdellatif Moussaid, and Fadoua Ghanimi. Financial fraud detection using machine learning: A review of literature. In *Technology and the Environment: Implementing Smart and Sustainable Solutions into Our Cities*, Advances in Science, Technology & Innovation, pages 49–54. Springer, Cham, 2025. Cited on pp. 5, 13, and 18.

[10] Dawei Cheng, Yao Zou, Sheng Xiang, and Changjun Jiang. Graph neural networks for financial fraud detection: A review. *Frontiers of Computer Science*, 19:199609, 2025. Cited on pp. 7, 14, and 18.

[11] Ying Zhou, Haoran Li, Zhi Xiao, and Jing Qiu. A user-centered explainable artificial intelligence approach for financial fraud detection. *Finance Research Letters*, 58:104309, 2023. Cited on pp. 7, 14, and 18.

[12] Saif Khalifa Aljunaid, Saif Jasim Almheiri, Hussain Dawood, and Muhammad Adnan Khan. Secure and transparent banking: Explainable ai-driven federated learning model for financial fraud detection. *Journal of Risk and Financial Management*, 18(4):179, 2025. Cited on pp. 7, 14, 18, and 31.

[13] Ashish Jain and Rishabh Kumar. A review of automated financial advisory systems and personalised banking. *Journal of Financial Services Research*, 64(2):215–234, 2023. Cited on pp. 7, 14, and 19.

[14] Laura Fernandez and Ajay Singh. Machine learning approaches for customer profiling and personalised financial recommendations. *Expert Systems with Applications*, 219:119657, 2023. Cited on pp. 7, 14, and 19.

[15] Longbing Cao. Ai in finance: Challenges, techniques, and opportunities. *ACM Computing Surveys*, 55(3):1–38, 2022. Cited on p. 8.

[16] Ahmet Murat Ozbayoglu, Mehmet Ugur Gudelek, and Omer Berat Sezer. Deep learning for financial applications: A survey. *Applied Soft Computing*, 93:106384, 2020. Cited on p. 8.

[17] Omar H. Fares, Irfan Butt, and Seung Hwan Mark Lee. Utilization of artificial intelligence in the banking sector: A systematic literature review. *Journal of Financial Services Marketing*, 28(4):835–852, 2023. Cited on pp. 8 and 9.

[18] Majlinda Godolja and Laureta Domi. Evolution of artificial intelligence in the banking sector: A systematic literature review. In *EMAN 2024: Economics and Management: How to Cope With Disrupted Times*, pages 131–139. Association of Economists and Managers of the Balkans, 2024. Cited on pp. 8 and 9.

[19] Jurgita Černevičienė and Audrius Kabašinskas. Explainable artificial intelligence (xai) in finance: A systematic literature review. *Artificial Intelligence Review*, 57, 2024. Cited on pp. 8 and 9.

[20] NVIDIA Corporation. State of AI in financial services: 2024 trends. Technical report, NVIDIA, 2024. Cited on p. 8.

[21] McKinsey & Company. Capturing the full value of generative AI in banking. *McKinsey Insights*, December 2023. Cited on pp. 8 and 9.

[22] EY-Parthenon. Generative AI in banking: 2025 survey insights. Technical report, Ernst & Young, 2025. Cited on p. 8.

[23] PwC Global. Generative ai for personalised financial guidance: Industry insights 2024. Technical report, PricewaterhouseCoopers, 2024. Cited on pp. 8, 9, and 19.

[24] Yuna Kim and Soo-Young Park. Deep learning models for behavioural finance and personalised advisory. *Information Processing and Management*, 61(1):103173, 2024. Cited on pp. 8, 9, and 19.

[25] OpenAI. Ai agents: Capabilities, safety, and design principles, 2024. Accessed 2025. Cited on pp. 9 and 12.

[26] R. Wang, X. Liu, and P. Chen. A survey on agentic ai: Reasoning, planning, and tool use in llm-based agents. *arXiv preprint arXiv:2404.01234*, 2024. Cited on pp. 10 and 12.

[27] J. Li, Y. Zhou, and X. Ren. Llm-augmented agents: Capabilities, benchmarks, and applications. *Transactions on Machine Learning Research*, 2024. Cited on pp. 10 and 12.

[28] Tom Xi, Michael Lee, and B. Zhang. Planning with large language models: A comparative study. *NeurIPS*, 2023. Cited on pp. 10 and 12.

[29] Accenture Research. Agentic ai in financial services: Industry applications and trends. Technical report, Accenture, 2024. Cited on pp. 10 and 12.

[30] Lilian Weng. The rise of autonomous agents and the need for alignment. Blog, 2023. Cited on pp. 10 and 12.

[31] Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, 2 edition, 2021. Cited on pp. 10 and 17.

[32] Feng Yang, Zhen Li, and R. Kumar. Multi-agent ai frameworks for modern banking systems. *Journal of Intelligent Information Systems*, 62(1):45–67, 2024. Cited on pp. 11 and 17.

[33] Khalid Barkat, Sarah Rahimi, and Imran Khan. A survey of multi-agent techniques in financial fraud detection. *Expert Systems with Applications*, 223:119899, 2023. Cited on pp. 11 and 17.

[34] Y. Qi, T. Liu, and P. Wang. Llm-assisted multi-agent collaboration: A new paradigm for autonomous systems. *arXiv preprint arXiv:2401.10522*, 2024. Cited on pp. 11 and 17.

[35] Daniel Abraham, Marco Conti, and Floriana Esposito. Explainable ai for multi-agent decision systems in finance. *AI in Finance Review*, 5(2):100–118, 2024. Cited on pp. 11 and 17.

[36] Zihan Zhang, Xinyu Chen, and Wei Xu. A survey on large language models for natural language processing. *arXiv preprint arXiv:2303.18223*, 2023. Cited on p. 11.

[37] Grégoire Mialon et al. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*, 2023. Cited on p. 13.

[38] Deloitte Insights. Generative ai in financial services. Technical report, Deloitte, 2024. Cited on p. 13.

[39] Alejandro Bernabeu Arrieta et al. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges. *Information Fusion*, 58:82–115, 2020. Cited on pp. 13 and 15.

[40] Harish Sundar and Diya Patel. Explainable recommendation systems in retail and financial services. *Decision Support Systems*, 167:113887, 2023. Cited on pp. 15 and 19.

[41] Christoph Molnar. *Interpretable Machine Learning*. 2022. Online book. Cited on p. 15.

[42] David Gunning and David Aha. Darpa's explainable artificial intelligence (xai) program. *AI Magazine*, 42(2):52–59, 2021. Cited on p. 15.

[43] KPMG International. Intelligent banking: A blueprint for value creation through ai. Technical report, KPMG, 2025. Cited on p. 20.