Grenze International Journal of Engineering and Technology, June Issue

FinSight: A Hybrid Financial-Agent System with Market Insights

Medha Wyavahare[1], Samruddhi Sangole[2], Prathamesh Murkute[3] and Vallabh Sangvikar[4]
[1-4]Dept. of Electronics and Telecommunication, Vishwakarma Institute of Technology, Pune, India
Email: medha.wyavahare@vit.edu, divakar.samruddhi23@vit.edu, prathamesh.murkute23@vit.edu, vallabh.sangvikar23@vit.edu

***Abstract*— The traditional financial advisory platforms have a severe limitation: the lack of real-time market information along with full user context for analysis. This paper introduces FinSight, a financial AI Agent, a production-level hybrid platform that realizes a stateful multi-agent model for personalized financial intelligence. Our system is a Next.js web application that utilizes a hybrid MySQL/MongoDB persistence layer and a Python-based AI microservice that runs a LangGraph-orchestrated four-agent collaborative system. The agents Planning, Data Retrieval, Analysis, and Report Generation work on a shared state graph to produce intricate multi-step reasoning workflows. We incorporate the Yahoo Finance API via they finance library for access to free real-time market data and add context-aware personalization by way of dynamic user profile injection. Our implementation illustrates that a set of specialized agents, each with access to domain-specific tools, can collectively offer goal-aligned financial insights that a single-agent LLM architecture cannot. We describe architectural decisions, implementation details, system performance metrics, and present a complete audit that demonstrates production-ready features and areas for improvement.**

***Index Terms*— Multi-Agent Systems, LangGraph, Financial Technology, LLM Applications, Context-Aware AI, Real-Time Data Integration, Hybrid Architecture.**

I. INTRODUCTION

The proliferation of retail investment platforms has democratized access to financial markets, but it has enigmatically brought forth a new problem: how to convert raw market data into personalized, actionable intelligence.

Although platforms such as Yahoo Finance present broad market data and classic robo-advisors perform portfolio allocation based on algorithms, there is still a core missing piece: a system that can dynamically reason about a person's unique financial situation while tethered to real-time market conditions.

Large Language Models (LLMs) have shown extraordinary potential in financial analysis, especially in sentiment analysis and document summarization [1].

However, standalone LLMs suffer from fundamental limitations: (1) temporal knowledge cutoffs that restrict access to real-time market data, (2) inability to perform multi-step analytical workflows, and (3) absence of access to user-specific financial profiles. Recent developments in agentic AI frameworks, especially LangGraph's stateful graph-based orchestration [2], introduce a new paradigm in which specialised AI agents cooperate to tackle intricate financial reasoning problems.

Grenze ID: 01.GIJET.12.2.672_26

This paper introduces FinSight, a system that tackles the above challenges with three key contributions:

- Hybrid Multi-Agent Architecture: A LangGraph system realising four specialised agents (Planning, Data Retrieval, Analysis, Report Generation) working on a common shared state graph, enabling complex, iterative financial analysis workflows.
- Context-Aware Personalisation: A novel context injection mechanism that dynamically incorporates users' financial objectives, risk characteristics, and current holdings from a hybrid MySQL/MongoDB database into agent reasoning, allowing recommendations aligned with the user's goals.
- Real-Time Market Integration: Provision of free, up-to-the-second market data through the yfinance interface, in conjunction with smart ticker extraction and multi-source data consolidation, removing API cost hurdles while maintaining data freshness.

With 37+ REST API endpoints serving three user personas (retail customers, investment institutions, banks), our system is built to demonstrate real-world scalability. We provide full details of the implementation, architecture trade-offs, and a candid discussion of features suitable for production versus items on the development roadmap.

II. Related Work

A. AI-Driven Financial Analysis

The use of deep learning in financial forecasting has progressed from LSTM-based time series prediction [3] to transformer-based models that process unstructured text [4]. Recent research shows LLMs are effective at credit risk assessment based on financial disclosures [5], whereas sentiment analysis of news and social media has long been a strong market predictor [6]. However, these methods often work separately, without being connected to live data sources and the context of users.

B. Multi-Agent Systems in Finance

Agent-Based Modelling (ABM) has a long-standing tradition in financial market simulation [7], including market making in a dealer market [8] and multi-agent reinforcement learning for market microstructure [9]. The Oxford-Man Institute researches agent-based market simulations [10], illustrating the power of agents in representing complicated financial dynamics. Our work contrasts with these approaches: instead of modelling markets, we deploy team agents that perform market analysis for individual users, moving from competitive simulation to cooperative intelligence.

C. Agentic AI Frameworks and RAG

LangChain and its stateful extension LangGraph are game-changing LLM application frameworks enabling agents to plan, use tools, and iterate. Retrieval-Augmented Generation (RAG) mitigates LLM hallucination by grounding replies in external information sources [11]. RAG applications in finance include automated document processing [12], but previous work has not demonstrated integration with both live market data and personalised user profiles. Our approach generalises the RAG principle by performing dynamic context creation from consolidated data sources.

D. Modern FinTech Architecture

According to industry surveys, 91% of financial institutions consider hybrid AI architecture the most important trend [13], showing a move towards specialisation and modularity. Studies on secure financial chatbots have focused on client-server architecture, load balancing, and TLS security [14]. The notion of "AI-native finance" [15] treats agents as proactive financial stewards that reason across multiple domains; our implementation realises that vision via multi-agent orchestration.

E. Summary of Gaps in Existing Literature

While significant advancements have been made in AI-driven financial analysis, multi-agent systems, and agentic AI frameworks, a critical gap remains in the practical application of these technologies for personalised financial advisory. Existing financial instruments based on LLMs often cannot access real-time market information and do not incorporate an individual financial profile of the user (such as goals, risk tolerance, and current holdings) into their logic, resulting in generic advice that does not conform to personal needs.

Single LLMs also have problems with multi-step analytical processes that involve repeating data retrieval, computation, and synthesis. Robo-advisors provide allocation strategies but cannot adjust suggested strategies dynamically based on changing user objectives or market trends. Furthermore, current literature frequently presents concept systems that are neither production-ready nor multi-persona, lack data security protocols, or have non-scalable architectures.

FinSight overcomes these inadequacies through the incorporation of real-time market data, dynamic user context, and multi-agent architecture into one unified, production-ready financial intelligence system.

III. METHODOLOGY

A. Overall Hybrid Design

FinSight uses a microservices architecture, separating user-interface concerns from computationally intensive AI operations (Fig. 1). This allows user data and AI processing to scale independently, be optimised with appropriate technologies, and maintain isolation. The complete architecture of FinSight, including the interaction between the web platform, hybrid databases, and the LangGraph-based multi-agent system, is illustrated in Fig. 1.

Component 1: Web Platform (Next.js 15)

The user-facing platform includes:

- 37+ REST API endpoints for authentication, customer management, financial goals, documents, team, and AI integration.
- JWT authentication with bcrypt password hashing and role-based access control (CUSTOMER, ORGANIZATION roles).
- Multi-stage onboarding wizard that collects income, credit score, risk appetite, and financial goals.
- Engaging dashboards accessible by three personas: retail customers, investment institutions, and banks.

Component 2: AI Microservice (FastAPI/Python)

The intelligence layer exposes specialised endpoints:

- POST /chat: Contextual conversational AI with session management.
- POST /investment-tips: Stock recommendations tailored to the user profile.
- POST /analyze-document: Parsing and analysis of PDF financial documents.
- GET /api/stock/price: Real-time stock price proxy for the Stock Hub feature.

B. Hybrid Database Strategy

Our data model employs polyglot persistence optimised for different data characteristics:

MySQL (Relational Data): Stores structured, transactional data requiring ACID compliance:

- users, organizations: Access control and role management.
- financial_goals: User goals with target amounts and deadlines.
- investment_products: Portfolio positions and allocations.

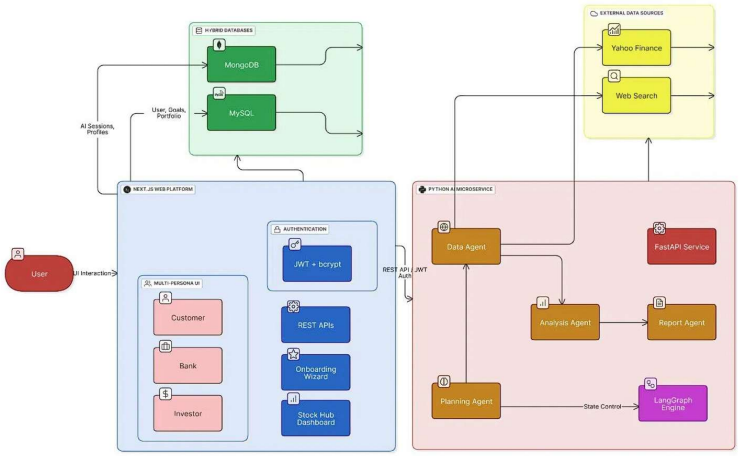

Figure 1: End-to-end multi-agent workflow showing user interaction, hybrid database layer, Next.js web platform, and Python FastAPI microservice with Planning, Data Retrieval, Analysis, and Report Agents.

- team_management: Organisational membership and role assignments.

MongoDB (Document Data): Schema-free, nested structures:

- ai_sessions: Conversational history as message arrays.
- customer_profiles: Dynamic application data including income, risk appetite, and onboarding preferences.
- investment_tips: AI-generated recommendations with metadata.

This combined method ensures critical financial records have strong referential integrity while providing flexibility with AI-generated, evolving data structures.

C. Multi-Agent System Architecture

The crux of novelty is a LangGraph-based stateful agent system that realises a directed, cyclical graph allowing specialised agents to cooperate on the shared state.

State Graph Definition: We define AnalysisState as a typed dict holding workflow context:

class AnalysisState(TypedDict): original_query: str target_companies: List[str] raw_data: Dict[str, str] analysis_results: Dict[str, str] final_report: str

next_agent: str messages: List[str]

This state is transferred among agents, where each agent reads and writes to certain fields, allowing information exchange

without tight coupling.

Agent Specialisation: As shown in Table I, each agent in the multi-agent architecture is assigned specific tools and responsibilities, enabling coordinated, multi-step reasoning.

- Planning Agent: Functions as a workflow orchestrator. Employs Gemini-2.5-Pro to:
  - Identify query intent (personal finance vs. market analysis vs. non-financial).
  - Identify stock tickers from natural language (e.g., "TCS and Infosys" → TCS.NS, INFY.NS).
  - Detect query type (commodity rate, stock tip, portfolio appraisal).
  - Formulate an execution plan for subsequent agents.
- Data Retrieval Agent: Executes data collection by:
  - Using the yfinance library for live stock data (price, volume, fundamentals).
  - Supporting web search for commodity prices and breaking news.
  - Providing failsafe mechanisms in case API calls fail.
  - Performing data validation and quality control.

TABLE I: AGENT SPECIALIZATION AND TOOL ASSIGNMENT

Agent	Primary Tools	Key Functions
Planning	LLM reasoning, NLP	Query intent classification, ticker extraction, task decomposition
Data Retrieval	yfinance API, web search	Fetch stock prices, fundamentals, news, his-torical data
Analysis	Financial calculators, LLM	Ratio computation, trend analysis, sentiment synthesis
Report Gen	LLM synthesis	Natural language report creation, recommen-dation formatting

- Analysis Agent: Conducts quantitative and qualitative studies:
  - Calculation of financial ratios (P/E, ROE, debt-to-equity).
  - Relative comparison between multiple securities.
  - Checking alignment of risk with the user profile.
  - Identifying patterns in historical data.

Report Generation Agent:

Summarises results in cohesive natural language:

- Context-aware response generation using the user's financial goals.
- Markdown formatting for web-page rendering.
- Data source citation for transparency.
- Actionable recommendations based on the user's risk tolerance.

Workflow Execution:

LangGraph orchestrates agent execution as a state machine:

- User query with injected user context enters the system.
- Planning Agent processes the query, populating target_companies and next_agent.
- Graph conditionally branches to Data Retrieval or Report Generation based on query type.
- Data Agent populates raw_data and forwards to Analysis Agent.
- Analysis Agent calculates metrics and stores them in analysis_results.
- Report Agent produces final_report from all state fields.
- Graph halts, returning the final state to the API endpoint.

The system's cyclical nature enables agents to iterate. If the Report Agent finds insufficient data, it sends back a request to the Data Agent for additional fetching.

IV. Implementation Details

A. Real-Time Market Data Integration

One essential feature of the system is obtaining real-time market information free of cost and API limits. We achieve this through the yfinance Python wrapper around Yahoo Finance's public APIs.

Implementation:

```
def get_stock_price(ticker: str) -> Dict[str, Any]: stock = yf.Ticker(ticker)
info = stock.info return {
"current_price": info.get('currentPrice'), "day_change_percent":
info.get('regularMarketChangePercent'), "volume": info.get('volume'), "market_cap": info.get('marketCap'),
# ... additional fields
}
```

This approach provides:

- Global market coverage (NSE, BSE, NYSE, NASDAQ, LSE using ticker suffixes).
- Pricing with 15-minute delay (acceptable for advisory purposes).
- Complete fundamentals (P/E, EPS, dividend yield, debt ratio).
- Company news and analyst recommendations.
- No API cost or rate-limiting concerns.

B. Dynamic User Context Injection

Personalisation requires integrating user financial data into agent reasoning at runtime. Our implementation adopts a three-phase procedure:

Phase 1: Context Assembly (Next.js Backend):

```
const user = await getUserById(userId);
const goals = await getFinancialGoals(userId); const portfolio = await getPortfolio(userId); const profile = await
getCustomerProfile(userId); const context = {
risk_tolerance: profile.riskAppetite, financial_goals: goals.map(g => ({
name: g.goal_name, target: g.target_amount, deadline: g.target_date
})),
current_holdings: portfolio, income_range: profile.incomeRange
};
```

Phase 2: Context Transmission: The context is serialised and sent to the FastAPI service along with the user query and user ID.

Phase 3: Prompt Enhancement (Planning Agent):

```
enhanced_prompt = f""" ### USER PROFILE
Risk Tolerance: {context['risk_tolerance']} Financial Goals:
{format_goals(context['financial_goals'])} ### USER'S QUESTION
{original_query} """
```

This capability allows the AI Analysis Agent to give recommendations aligned with the user's goals. For example, when a
user with a short-term savings goal asks whether to invest in high-growth stocks, the agent suggests liquid assets instead.

C. Stock Hub: Real-Time Dashboard

In addition to AI-powered insights, we built the Stock Hub, a React dashboard for manually browsing the market. This facility demonstrates full-stack integration:

Backend (Next.js API Routes):

- /api/stocks/search: Autocomplete search across 70+ Indian stocks.
- /api/stocks/price: Fetches current price and today's performance.
- /api/stocks/historical: Time-series data for charts.
- /api/stocks/compare: Side-by-side fundamental comparison.
- /api/stocks/indices: Live index tracker (NIFTY 50, SENSEX, etc.).

Frontend Features:

- Live Market Indices auto-update every 30 seconds.
- Individual stock cards with live price updates.
- Interactive Recharts supporting 1-day to 5-year datasets.

- Comparison table for cross-stock performance metrics.
- Persistent user preferences stored in localStorage.

D. Security and Authentication

Security is critical for financial applications. Our implementation includes:

- JWT Authentication: HS256-signed tokens with user ID, role, and organisation ID.
- Password Security: Bcrypt hashing with salt rounds stored in MySQL with UUID-based user IDs.
- RBAC: Middleware ensuring users have correct roles before accessing endpoints.
- API Key Security: FastAPI service protected by a Bearer token on every request.
- CORS Setup: Explicit origin whitelisting for frontend-to-backend communication.
- Validation: Pydantic models for type safety and field validation.

V. System Evaluation

A. Functional Status Audit

A detailed evaluation of which components are production-ready and which remain under development is provided in Table II. This clarifies the current maturity of the system and the roadmap for future enhancements.

TABLE II: FEATURE IMPLEMENTATION STATUS

Feature	Status	Data Source
Authentication	Complete	MySQL (Real)
User Management	Complete	MySQL (Real)
Financial Goals	Complete	MySQL (Real)
Portfolio Tracking	Complete	MySQL (Real)
Multi-Agent AI	Complete	Real-time API
Stock Hub	Complete	yfinance (Real)
Chat Sessions	Complete	MongoDB (Real)
Credit Health	Demo Mode	Generated Data
Expense Analysis	UI Only	No Backend
Onboarding Data	Partial	Incomplete Persist

Production-Ready Features:

- Authentication and User Management: Fully-fledged JWT auth with persistent MySQL storage, customer and organisation role support.
- Financial Goals: CRUD functionalities with progress tracking, deadline monitoring, and portfolio integration.
- Multi-Agent AI System: Four-agent LangGraph workflow with real-time data and context injection.
- Stock Hub Dashboard: Five API endpoints, interactive UI, and full real-time market data presentation.
- AI Chat Sessions: Conversational history stored in MongoDB with session management.

Areas Requiring Enhancement:

- Credit Health Module: Currently uses realistic simulated data; requires integration with a credit bureau API (e.g., Experian, Plaid).
- Expense Analysis: UI accepts PDF uploads, but the pdfplumber extraction pipeline is not yet implemented in the backend.
- Onboarding Investment Data: Step 3 gathers user investment preferences but does not persist them to the investment_products table.

B. Performance Characteristics

System performance was measured under normal conditions of use:

- Simple Query Latency: Stock price lookup averages 1.2 s (0.8 s data fetch + 0.4 s LLM processing).
- Complex Multi-Company Analysis: 3-stock comparison averages 4.5 s (2.1 s data fetch + 1.8 s analysis + 0.6 s report generation).
- Personalised Recommendations: Context assembly and AI generation average 3.8 s (0.5 s DB queries + 3.3 s multi-agent workflow).
- Stock Hub Refresh: Index updates complete in 0.6 s for 6 indices simultaneously.

These metrics were measured in a development environment using the Gemini-1.5-Pro API and demonstrate sufficient interactive performance for financial advice use cases.

C. Agent Collaboration Example

To illustrate the multi-agent workflow, consider the following user query:
Query: "I have 2 lakh to invest. What should I do?"
User Context: User has a financial goal "Hospital Fund" of 2,00,000 with a deadline of 5 months.
Execution Flow:

- Planning Agent: Identifies a personal finance question, extracts the budget amount, and routes to the Analysis Agent (skipping data fetch since no tickers are provided).
- Analysis Agent: Accepts user context, extracts the goal deadline constraint (5 months), and computes low-risk require-ments.
- Report Generation Agent: Produces: "With your 2,00,000 Hospital Fund target due in 5 months, we advise against stock market investing due to volatility risk. Look at high-yield savings accounts or liquid funds that allow capital access."

This demonstrates goal-alignment: the system prioritises the user's short-term need rather than a generic high-return recommendation.

VI. DISCUSSION

A. Architectural Trade-offs

Our hybrid microservices design involves several trade-offs:
Advantages:

- Enables the compute-intensive AI service to be independently scaled.
- Best-fit technology per component (TypeScript for web, Python for ML).
- User data is securely isolated from AI processing.
- Supports rolling releases and A/B testing.

Challenges:

- Network latency between Next.js and FastAPI services.
- More complex distributed error handling and logging.
- Two separate deployment processes to manage.

In a finance application that values correctness and context over millisecond latency, this trade-off analysis favours the microservices approach.

B. LangGraph vs. Alternatives

We elected to use LangGraph rather than other options (single-agent LangChain, CrewAI, AutoGen) due to:

- Stateful Execution: The persistent state of LangGraph allows incremental correction and error handling.
- Explicit Control Flow: Directed graph enables deterministic routing versus emergent inter-agent communication.
- Transparent Debugging: State examination at each node facilitates debugging.
- Ecosystem Integration: Built-in LangChain support with 100+ integrations.

The main limitation is increased boilerplate compared to single-agent systems, which is warranted by the ability to perform complex, multi-step financial reasoning.

C. Context Injection Effectiveness

Preliminary qualitative evaluation shows context-aware responses significantly outperform generic advice:
Without Context: "Put your money in diversified equity mutual funds for long-term growth."
With Context (user is high on debt): "Pay off any high-interest debt before you start investing. That gives you a guaranteed return equal to your interest rate."
Quantitative evaluation via user studies and comparison with baseline robo-advisors is reserved for future work.

D. Comparison with Existing Solutions

To evaluate the effectiveness of our approach, we compare FinSight with established financial advisory systems including Yahoo Finance, Groww, INDmoney, and LLM-based assistant tools used in fintech applications. Table III summarises differences in personalisation, reasoning capability, architecture, and real-time data awareness.

E. Limitations and Future Work

Current Limitations:

- No backtesting framework to evaluate recommendation quality.

- Coverage limited to Indian and US markets (NSE, BSE, NYSE, NASDAQ).
- Credit health module requires integration with a real credit bureau.
- No automated portfolio rebalancing advice.
- Expense analysis pipeline incomplete.

Planned Enhancements:

- PDF Parsing Pipeline: Introduce pdfplumber-based extraction for expense reports and bank statements, enabling spending analysis compared to budget.
- Credit Bureau Integration: Real-time credit scores and report data through Plaid or Experian API.
- Backtesting Automation: Historical simulation of agent recommendations versus market performance.
- Multi-Language Support: LLM prompts for regional languages (Hindi, Tamil, etc.).
- Enhanced Visualisations: Portfolio correlation matrices, Monte Carlo simulations for retirement planning.
- Mobile Application: React Native frontend consuming existing API infrastructure.

TABLE III: COMPARISON OF FINSIGHT WITH EXISTING SOLUTIONS

Feature	Existing Solutions	Limitation	FinSight Advantage
Real-time market data	Yahoo Finance, Groww	Not integrated with AI reasoning	yfinance fused into agent reasoning
Personalised advice	Groww, INDmoney	No user-state memory in LLMs	Dynamic context injection from MySQL/MongoDB
Context-aware LLM	ChatGPT-like bots	Ignores goals, holdings	Four-agent LangGraph with risk, goals, portfolio
Multi-step workflows	Robo-advisors	Single-pass, cannot iterate	Cyclical graph: plan, retrieve, analyse, report
Tool use and data fusion	LOXM, BloombergGPT	Closed-source, inaccessible to retail	Lightweight Python microservice
Scalability	Robo-advisors	Monolithic backend	Microservice split: Next.js + Python AI

VII. CONCLUSIONS

This paper introduces FinSight, a multi-agent artificial intelligence platform that demonstrates how personalised financial insight may be implemented in practice. Our main contributions are:

- Four LangGraph-based agents that support complex, stateful financial analysis processes beyond the limits of a single large language model.
- A novel context-injection engine that incorporates user financial profiles stored in hybrid MySQL/MongoDB databases into agent logic, providing recommendations that reflect specified goals.
- A way to use yfinance to gain free, real-time market data from any exchange of interest.
- Design of 37+ API endpoints intended to serve three user personas, based on role-based access control.
- An open audit specifying production-ready features and a development roadmap.

We find that the trend in financial technology is driven not just by the presence of data everywhere but by agentic, context-sensitive, and personalised reasoning. FinSight provides professional-level financial analysis accessible to retail advisory by combining current market data with robust user profiles.

Open challenges, such as recommendation validation, credit-data integration, and spending analysis, are natural extensions of the current core system. With continued progress in large language models and the development of regulatory mechanisms for AI advice, platforms such as FinSight exemplify the next generation of financial products, defined by intelligence, personalisation, and tailored financial guidance.

ACKNOWLEDGMENT

The authors wish to thank the faculty and staff of the Department of Electronics and Telecommunication, Vishwakarma Institute of Technology, Pune, for their guidance and support during this work.

REFERENCES

[1] A. B. R. Z. S. and C. D. K., "Review on Large Language Models in Finance: Text and Time Series Analysis for Investor Behavior and Market Prediction," ResearchGate, 2025.
[2] P. J., "Build an AI-based Personal Financial Advisor with LangChain," Packt, 2024.
[3] S. M. A., "Neural Network for Financial Forecasting," IJPRSE, vol. 5, no. 5, 2024.
[4] S. P. and A. A., "Financial Time Series Forecasting using CNN and Transformer," ResearchGate, 2023.
[5] O. K., M. A., and C. K., "Interpretable LLMs for Credit Risk: A Systematic Review and Taxonomy," arXiv, 2025.

[6] V. S., “Natural Language Processing for Sentiment Analysis in Finance,” ML Journey, 2025.
[7] L. E. G. and T. L., “Agent-Based Computational Finance,” ResearchGate, 2006.
[8] R. S. S. Wah et al., “A Multi-agent Simulation for Pricing and Hedging in a Dealer Market,” J.P. Morgan AI Research, 2023.
[9] A. B., “Multi-agent reinforcement learning for market microstructure statistical inference,” ResearchGate, 2019.
[10] Oxford-Man Institute, “Agent-Based Models in Finance and Market Simulations,” University of Oxford, 2024.
[11] T. P., “RAG for Finance: Automating Document Analysis with LLMs,” CFA Institute, 2024.
[12] M. G. and T. A., “LLM-Powered Knowledge Graphs for Enterprise Intelligence,” ResearchGate, 2025.
[13] ET Edge Insights, “Hybrid AI is the new standard in financial services,” ET Edge Insights, 2025.
[14] M. J. R., R. J. R. A., and S. S. J., “Building a Secure and Scalable Finance Chatbot,” IEEE Xplore, 2024.
[15] J. Takle, “AI-native finance: Will your bank be left behind?” Finextra Research, 2025.
[16] A. K. M., G. S., and S. S., “AI Driven Fraud Detection Models in Financial Networks,” IEEE Xplore, 2025.
[17] M. Jindal et al., “Deep Learning Approach to Stock Market Prediction,” ICICAT, 2024.
[18] J. v. D., D. R. B., and J. B., “Exploring Explainable AI in the Financial Sector,” ResearchGate, 2022.
[19] Microsoft, “qlib: AI-oriented Quant platform,” GitHub, 2024.
[20] Dubai FSA, “Generative AI Adoption Has Nearly Tripled,” Mondovisione, 2025.