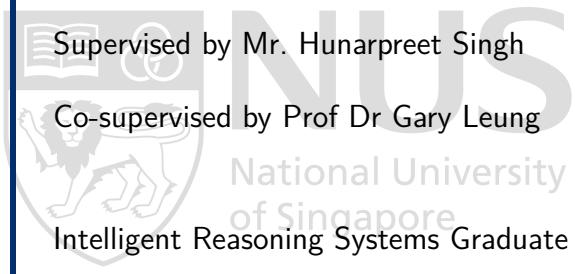
FinSight: Intelligent Stock Prediction and Advisory Platform

augmenting the lives of finance professionals

Huo Yiming (A0328696J)
Li Jiajun (A0326795M)
Samarth Soni (A0329960U)
Su Yuxuan (A0329926N)
Wang Yixi (A0328469M)

Supervised by Mr. Hunarpreet Singh

Co-supervised by Prof Dr Gary Leung



Intelligent Reasoning Systems Graduate Certificate

Institute of Systems Science

National University of Singapore

October, 2025

A dissertation submitted in partial fulfilment of the requirements for the degree of M.Tech. Artificial Intelligence Systems.

WWW.NUS.EDU.SG

First edition, October 26, 2025

Acknowledgements

At the outset, we would like to express our gratitude to our professors Prof Gary Leung, Prof Zhu Fangming, Prof Liya Ding, Prof Xavier Xie, Prof Barry Shepherd, Prof Fan Zhenzhen, Prof Tian Jing, Prof Wang Aobo for imparting unto us the critical knowledge required for us to deliver effective results in this project. Their in-depth instruction and carefully curated teaching and workshops helped us in identifying areas to work on and techniques to utilise. We would like to acknowledge the entire staff at NUS ISS for making our onboarding to the course smooth and full of enthusiasm and engagement. We also wish to acknowledge the inviting and open collaborative culture in our course, which has made us feel at home and empowered to do our academic work with full focus.

We would like to especially acknowledge the personal and highly insightful guidance we received from Prof Gary Leung regarding the use cases we can identify to tackle in the field of quantitative research and financial engineering. He personally ensured at multiple occasions that our queries were answered and we were well prepared to work on our project. He encouraged us to pursue the project module in the field we were passionate about and consulted with us on our ideas and planned approach well before the proposal submission. We would like to thank his and Prof Liya Ding's feedback to us in our proposal call which set us in the right direction and kept us focused and aligned to the major task at hand.

We are thankful to Mr. Hunarpreet Singh of ArthAlpha, who agreed to act as a mentor and guide, who was always ready to provide professional recalibration related to the project. He and the company trusted our AI & programming skills and gave us the opportunity to contribute to their critical projects. He was kind to meet with us on weekly basis to provide mentorship and solve our doubts, even during his busy time. He was always attentive and didn't let us feel at all like we were alone – he took personal interest and wanted us to succeed. He extended motivation and honest feedback, and shared areas where we could improve.

We would like to acknowledge our entire team wherein all teammates worked collaboratively divided the work smartly, and planned ahead to ensure timely cohesion and delivery, both on weekly basis to Hunar as well as over the course of the month. We are grateful for having such like minded and hard working team members.

We would like to acknowledge the LLM models ChatGPT 5, ChatGPT 5 Thinking, Perplexity Pro, Claude, Deepseek provided by the corporations OpenAI, Perplexity, Anthropic AI, DeepSeek AI which acted as our personal assistant and tutor. These tools helped us quickly draft first versions of any ideas that we wished to test, in data preparation, in understanding the approaches available to us, in recognising pros/cons/design decisions to consider, in figuring out the code, and finally in debugging any errors that we faced. Since these were like supercharged versions of the internet, a lot of our knowledge enquiries became much simpler and straightforward due to the presence of these tools and we were able to focus on the more important things core to our project, while saving time on tertiary activities like reading multiple Stack Overflow/Reddit pages for answers each time we faced a roadblock.

We would like to acknowledge that computational work involved in this research work is partially/fully supported by NUS ITs Research Computing group under grant number NUSREC-HPC-00001. Thank you to National University of Singapore and to Institute of Systems Science for providing us with the necessary tools and environment required to hone our AI skills.

Abstract

Start Date: 10th Sept, 2025 **Duration**: 1.5 months
Submission: 26th Oct, 2025

A unified platform providing reliable, real-time market intelligence for investors and analysts.

At FinSight, weve tried to cover all fronts of data science that help a market participant get the big picture at a glance and act fast. We learn each users preferences and tailor what we surface: relevant news, an evolving list of interesting stocks, short-horizon predictions on the names that matter, and a specialised LLM that behaves like a personal financial research analyst—reading your docs, explaining its thinking, and showing its work.

All actionable items get value-added annotations: news sentiment tags from a finance-tuned classifier (e.g., FinBERT), a lightweight risk lens on stocks, and a projected stock movement view using deep time-series/recurrent models. For narrative answers, we use Retrieval-Augmented Generation (RAG) so the LLM cites the exact passages it relied on. We emphasise explainability: clear reasoning traces, citations, and intuitive visuals—so a human can audit why a conclusion was reached and what would change it. This helps us in grounding responses in evidence and reducing hallucinations.

On the data side, we stream in news and filings continuously, parse the text, and maintain vector embeddings for users, news items, and tickers alongside the vectors we use for LLM retrieval. We then score relationships with standard similarity measures (cosine/inner-product) so the system can match you-shaped interests against fresh content and instruments quickly. This is classic vector search under the hood (FAISS/pgvector), but tuned for markets: fast nearest-neighbour lookups to power news recommendations, stock watchlists, and context retrieval for the LLM.

Operationally, we engineered the stack to be practical: automated ingestion and chunking of messy PDFs/docs, vector search to pull the right context quickly, and a lightweight serving layer that scales from our laptops to GPUs on the cluster. This collaboration grounds the project in real market workflows while giving us hands-on exposure to production-grade financial AI—where reliability, latency, and traceability matter as much as raw accuracy.

Keywords: DL, GPT, SQL, LSTM, LLM, RAG, NLP, Agile, Python, Transformers, FAISS, pgvector, Time-Series, Prompt Engineering, Fine-Tuning, Evaluation, MLOps, Data Ops; refer to Abbreviations section

Project Areas: Intelligent Reasoning, Knowledge Bases, LLM Serving, Fine-Tuning, Data Ingestion & Preparation, Feature Engineering, Data Visualization, Exploratory & Statistical Analysis, Machine Learning, Time-Series Forecasting, RAG Pipelines, Tooling & Agents, Documentation, Software Engineering, CI/CD & Automation, Observability & Evaluation

Contents

List of Figures

List of Tables

List of Abbreviations

FRA Financial Research Analyst 1
SEBI Securities and Exchange Board of India 1
EDGAR Electronic Data Gathering, Analysis, and Retrieval; used by SEC 4
ESG Environmental, Social, and Governance evaluation criteria 1
DL Deep Learning 1
LSTM Long Short-Term Memory 13
LLM Large Language Model 2
RAG Retrieval-Augmented Generation iv
NLP Natural Language Processing 2
LGBM Light Gradient-Boosting Machine 13
ARIMA Autoregressive Integrated Moving Average 13
FAISS Facebook AI Similarity Search iv
FinBERT Financial BERT, a BERT model pre-trained on finance-domain text for tasks like sentiment and analysis 7
pgvector PostgreSQL extension for vector similarity search (store/index embeddings directly in Postgres) iv
GPU Graphics Processing Unit iv
GRU Gated Recurrent Unit 23
AI Artificial Intelligence 4
HPC High Performance Computing 2
MLOps Machine Learning Operations 19
RSS Really Simple Syndication – Google Reader service 13
API Application programming interface 9
SPY ETF SPDR S&P 500 ETF Trust 31

1

Introduction

1.1 | Background

Markets move fast; the information firehose moves faster. Between filings and financial reports, earnings calls, Environmental, Social, and Governance evaluation criteria (ESG) notes, and an endless news stream, not to mention the unpredictable market movements, even pros struggle to separate signal from noise in time to act. Investors are often overwhelmed by massive volumes of digital information. (For scale: EDGAR alone processes ∼4,700 filings per day and serves petabytes annually, so manual scan + tabs doesnt cut it.)

Traditional investment platforms in India and globally often lack personalized recommendations and dynamic adaptation to investor preferences. We wanted a practical way to solve these issues while staying grounded in practical results. It was clear the industry today needs an AI-powered assistant more than ever before: explainable, and fast. Our goal with FinSight is simple: compress the chaos into what matters now for an individual investor or desk—relevant news, interesting tickers, quick signals/predictions, and a specialised LLM that behaves like a personal Financial Research Analyst (FRA) (reads your docs, explains its thinking, shows its work).

We chose to build this hand-in-hand with industry. Partnering with ArthAlpha (SEBI-registered quant shop) kept us focused on the work they actually need: an AI analyst that answers fundamentals, compares a stock to its peers with reasons and citations, and a Deep Learning (DL)-centric deep dive into quant tasks like pairs trading, signal horizon tuning and portfolio weighting. They needed a few extra **pairs** of hands (nice pun, yes), and we needed live constraints and feedback loops. That fit let us start with a clear scope, stay directed, and deliver actionable results instead of just building tech for its own sake. Also, the broader industry is converging here: firms are investing heavily in AI to cut research grunt work and surface insights faster—**analyst time is scarce**.

Why this collaboration? Because it keeps us honest. It ensured we solve real problems, not just make pretty demos. Not just industry relevance; this also offers us exposure to real-world financial AI applications and mentorship, setting us and this project up for longer-term impact and success.

At a high level, FinSight ingests live news and filings, maintains embeddings for users, news and tickers, and uses retrieval-augmented generation so answers are grounded and auditable. We emphasise explainability (reasoning traces, citations, sensible visuals) because trust beats black-box outputs. Details of the stack and models appear in the Methods/Architecture section.

Overall, this is an intelligent stock research and recommendation system that integrates financial fundamentals, user profiles, news sentiment, and predictive analytics.

1.2 | Aims

Design and implement a web-based stock research and insight platform that learns each users preferences and delivers **actionable**, **grounded** outputs: tailored news, interesting tickers, short-horizon predictions, and a specialised Large Language Model (LLM) that behaves like a personal Financial Research Analyst (FRA). The system combines financial fundamentals with Retrieval-Augmented Generation (RAG), deep time-series modelling, Natural Language Processing (NLP), and vector-based similarity matching to keep answers current, explainable, and fast.

1.3 | Objectives

(a) **User profile vector**: Build a multi-dimensional user profile embedding that reflects the investor's sector preference/style tilt, risk tolerance, investment horizon, and thematic interests; keep it up-to-date from interactions.

(b) **Unified data layer**: Ingest real-time market data, news, and regulatory filings into one interface with clean metadata and key highlights (source, sentiment, sector, time, ticker mapping).

(c) **News analysis**: Implement news browsing and analysis tab with topic tags and sentiment; write back signals to the user vector and maintain embeddings for **users**, **news items**, and **tickers** for similarity matching.

(d) **Predictions**: Build a stock trend module using time-series ML (LSTM/transformer/seq2seq amongst others in recurrent family) to surface short-horizon movement and what changed highlights.

(e) **AI Financial Research Analyst (ArthAlpha focus)**:

- Parse company reports and market data (tabular + text) and index them for retrieval.
- Generate structured research outputs (ratios, valuations, peer comps) with evidence.
- Provide **recommendations and risk assessments** backed by citations.
- Combine quantitative factors with qualitative reasoning; show steps.

(f) **RAG + retrieval quality**: Maintain a retrieval index for the Large Language Model (LLM); use similarity search + reranking to supply grounded context before answering.

(g) **Serving and UX**: Deliver a responsive front-end and a scalable back-end (High Performance Computing (HPC) cluster-friendly), exposing an API that external apps can call securely.

(h) **Project Management**: Make use of agile principles to effectively keep track of tasks and team members' progress. Also ensure everyone is in sync on the ongoing work.

(i) **Evaluation & trust**: Measure retrieval precision, grounding/citation rate, sentiment accuracy, forecast error, and end-to-end latency; emphasise **explainability** in outputs.

(j) **Ops**: Ship well-structured repositories, documentation, automated data prep, and deployment scripts; keep secrets safe and logs useful without leaking PII.

2

Market Research

2.1 | Industry Trends

The global fintech market is growing fast and getting more AI-heavy every quarter. Recent estimates put the market at **$394.88B in 2025** (from $340.10B in 2024) and project **$1,126.64B by 2032** (CAGR **16.2%** over the period). *Translation: sustained double-digit growth and huge room for new, AI-native workflows.* [Fortune Business Insights, 2025] At the same time, institutions report accelerating adoption of generative AI across key banking functions; McKinsey pegs the overall gen-AI value creation potential in the *trillions* annually, with clear use-cases in research, service and risk. ["Capturing the full value of generative AI in banking", 2023; "The state of AI in 2023: Generative AI's breakout year", 2023] Adoption isnt uniform, though–European financial firms still cite **compliance and model-risk** concerns as blockers–so solutions that are **grounded, auditable, and explainable** have a clear edge. ["Financial services shun AI over job and regulatory fears", 2024]

Meanwhile the research workflow is tilting toward Artificial Intelligence (AI): vendors are shipping analyst-assist tools and survey data shows rising enterprise spend on AI inference. [Bloomberg Intelligence, 2024; Bloomberg L.P., 2025]. Investors increasingly demand these AI-driven solutions to filter vast financial data and make informed decisions. Demand-side pressure is obvious: the SECs EDGAR now **processes ∼4,700 filings/day** and serves **∼3,000 TB/year** to the publicmanual scan and tab workflows no longer scale. [U.S. Securities and Exchange Commission, 2024] On the supply side, major data vendors are shipping tools targeting analyst productivity with AI, with Bloombergs CTO noting tools that could streamline up to **80%** of an analysts workload. [Financial News London, 2025; Fortune Business Insights, 2025].

In short, Fintech keeps compounding and its getting more **AI-native**. The implication for us: analyst-assist products like FinSight will win attention in a market where analyst time is scarce.

2.2 | Competitive Landscape

Existing platforms such as Robinhood, Betterment, and Eastmoney provide trading tools, ETF-based advisory, or information aggregation. However, we have found gaps in their offerings.

Robinhood popularised commission-free trading; it's strong on access and ease on features (options, crypto, 24/5 tokens in EU, a rebuilt newsfeed with premium sources, and more). It keeps evolving its newsfeed, but it isnt an explainable, evidence-linked analyst. [Reuters, 2025;

Robinhood Newsroom, 2019, 2024; "Robinhood Official Website | Commission-Free Trading", n.d.] **Betterment** is the classic robo-advisor: goal-based portfolios built from low-cost **ETF!**s, risk-adjusted over horizon, with optional human guidance too. It has solid UX; again, however, not research-grade explainable analytics. [Advisor, 2025; "Betterment Official Website | Automated investing & Financial planning", n.d.; WSJ Buy Side, 2025] **Eastmoney** (China) is a massive information + brokerage platform combining data, news and execution at scale, but the typical offering emphasizes aggregation/execution over **user vectors + live sentiment + grounded LLM research**. ["Forbes - East Money Information | Company Overview", n.d.; "Reuters - East Money Info", n.d.] **TradingHub** (MAST) is a specialised trade-surveillance vendor used by tier-1 institutions; its cross-product abuse detection focuses on modelling trader behaviour and market impact to cut false positives. I.e., a pure surveillance benchmark rather than execution or advisory. [Summit Partners, 2023; TradingHub, 2025]

Gap were targeting: these incumbents are great at execution or passive advice, but typically lack **dynamic user embeddings**, **live news ingestion with sentiment + similarity**, and **evidence-backed, explainable LLM research** that stitches filings, news and quant signals together for the specific individual. Thats the FinSight wedge. AI-powered explainable personalized recommendations are exactly where we sit.

2.3 | User Requirement

Retail investors struggle with information overload and lack professional analysis tools. Intermediate investors seek customizable recommendations that reflect their risk tolerance and sector preferences. They really need intelligent trading tools to guide their investment.

Retail and intermediate investors face two real problems: **information overload** and a shortage of **professional-grade analysis tools**. The FINRA Foundations latest NFCS data highlights persistent capability gaps and rising complexity for retail investors; there is a higher financial stress on households too, which struggle despite steady incomes. These 2024/2025 survey waves underscore the need for clearer guidance and better (which in this world always means data-driven) decision support. [FINRA Foundation, 2025a, 2025b; "FINRA Foundation National Financial Capability Study", 2025] On the flip side, investors *want* **personalization**: CFA Institutes Investor Trust Study shows tech acts as a trust multiplier, with investors valuing advice aligned to personal preferences and values. Market participants seek customizable recommendations that reflect their risk tolerance and sector preferences. There clearly a exists strong interest for transparent, actionable, understandable advice augmented by technology. ["Enhancing Investors' Trust (2022 Investor Trust Study)", 2022; "Smart Beta, Direct Indexing, and Index-Based Strategies", 2024] Industry leaders say the trend is the same inside firms: AI is positioned to streamline a large share of the analyst workload, freeing humans for judgment and client work. [Financial News London, 2025]

Implication for FinSight: learn the user (risk, sectors, themes), maintain fresh news/ticker/user embeddings, score similarity and sentiment in near real time, and surface **explainable** recommendations and what changed signals with citations. (Design specifics are in Project Scope.)

2.4 | Project Scope

2.4.1 | Data Scope

- **User profile signals**: interactions (reads, clicks, watchlists), risk tolerance, sector/theme tilt → a *user embedding* that updates over time.
- **Market & fundamentals**: end-of-day prices, corporate actions, and key financial line items mapped to tickers/CIKs.
- **News & filings**: live news feeds and regulatory docs (e.g., EDGAR/SGX). Web pages are extracted with Trafilatura; files are parsed with Unstructured before chunking. [Barbaresi, 2025; Unstructured.io, 2025]

2.4.2 | Module Scope

- **News browsing & analysis**: topic tags, entity/ticker linking, finance sentiment (FinBERT), and what changed summaries; updates user/news/ticker vectors in near real time. [Araci, 2019; Shen et al., 2024]
- **Stock trend prediction**: short-horizon signals via time-series DL (LSTM/transformers) with an auditable risk lens (volatility/beta).
- **Portfolio & pairs research**: pair selection, signal-horizon tuning, and weighting schemes; reproducible backtests and ablations.
- **AI Financial Research Analyst**: retrieval-augmented answers with citations, peer comps, fundamentals, and pros/cons rationale. [Shuster et al., 2021]

2.4.3 | Architecture Scope

- **Retrieval & vectors**: dense embeddings for *users*, *news*, *tickers*, plus a dedicated *LLM retriever* index. Vector search is the backbone (FAISS today; pgvector when we need persistence/SQL joins). [Douze et al., 2024; pgvector maintainers, 2025]
- **Reranking**: first-stage ANN retrieval (cosine/IP), then a cross-encoder reranker (e.g., *bge-reranker-v2-m3*) for higher answer quality. [BAAI / FlagEmbedding, 2025]
- **Model serving**: OpenAI-compatible LLM server (vLLM) behind a thin API; clients call `/v1/chat/completions` unchanged. [vLLM Project, 2025]
- **Parsing & ingestion**: `partition(auto)` in Unstructured for messy PDFs/Office; Trafilatura for robust main-text extraction. [Barbaresi, 2025; Unstructured.io, 2025]
- **Explainability by design**: RAG to ground answers in retrieved evidence (fewer hallucinations), with inline reasoning and citations. [Shuster et al., 2021]

2.4.4 | Methods & Design Decisions

1. **Ingestion → chunks**: fetch → extract (Trafilatura / Unstructured) → normalize → chunk with overlap; store provenance. [Barbaresi, 2025; Unstructured.io, 2025]
2. **Embeddings**: encode chunks/entities with a strong general embedding model; store vectors + metadata for fast lookup.
3. **Retrieve & rerank**: ANN search in FAISS/pgvector (cosine/IP) to get k candidates; rerank with a cross-encoder and keep top-N. [BAAI / FlagEmbedding, 2025; Douze et al., 2024; pgvector maintainers, 2025]
4. **Generation**: prompt the LLM via vLLMs OpenAI-compatible endpoint; require JSON with *answer*, *reasoning*, *citations*. [vLLM Project, 2025]
5. **Sentiment & risk lens**: Financial BERT, a BERT model pre-trained on finance-domain text for tasks like sentiment and analysis (FinBERT) tags (pos/neu/neg) + simple, auditable risk overlays (vol/beta). [Araci, 2019]
6. **Why RAG**: retrieval augmentation curbs hallucination and keeps answers current; retrieval quality is a first-class metric. [Shuster et al., 2021]

2.4.5 | Out of Scope

- Real-money execution/broker integration or HFT-level latency.
- Publishing proprietary partner data/models; we report methods and results only.

3

Data Collection and Preparation

At the outset, we acknowledge that privacy and security, compliance, and ethical sourcing will be important; also making sure we respect copyright, data licensing especially for nonpublic / proprietary data.

3.1 | User Profiles

User Profiles is the core data structure for storing 32-dimensional user profiles, every user gets his own user profile which contains these dimensions:

1. Industry preferences (11-d)
    a) utilities
    b) technology
    c) consumer defensive
    d) healthcare
    e) basic materials
    f) real estate
    g) energy
    h) industrials
    i) consumer cyclical
    j) communication services
    k) financial
2. Investment preferences(9-d)
    a) market value
    b) growth value
    c) dividend
    d) risk tolerance
    e) liquidity
    f) quality

g) valuation safety

h) momentum

i) efficiency

3.2 | News

Our news pipeline was implemented as a disciplined fetch → normalize → deduplicate → enrich → vectorize → store loop. In practice, our primary data sources are the **Google RSS** feeds (for broad coverage) and the **Marketaux API** (for structured financial press releases and real-time company news). Google RSS allows wide retrieval across multiple outlets by ticker and keyword queries, while provides curated financial headlines with tickers, sentiment, and metadata, which we merge and normalize during ingestion.

In production, we diverged from the initial 32-dimensional design and adopted a **two-vector representation**: a 64-d semantic embedding for article content and a 20-d user-preference projection (refer to the user profile) derived from sector/themes and interaction signals. This split lets us maintain high semantic recall while offering a compact, human-interpretable preference space for personalization. The **ingestion** layer cleans titles and URLs, collapses duplicates by canonical URL and external IDs, enriches documents with tickers, sectors, and sentiment from Marketaux, and stores the resulting enriched entries (with both vectors) in **MongoDB**. This design achieves the proposals intent: every article is stored with transparent **metadata**, traceable **origin**, and semantically searchable **embeddings** while remaining aligned with the systems user profile plus vector similarity paradigm.

3.3 | Stocks

3.3.1 | Data Sources and Collection Methodology

The stock recommendation system employs a comprehensive data acquisition strategy utilizing Yahoo Finance (yfinance) as the primary data source. The system targets the **top 100 companies** by market capitalization from the S&P 500 index, ensuring coverage of major market segments and sufficient liquidity for investment recommendations.

3.3.2 | Data Processing Pipeline

The raw stock data undergoes extensive **processing** to transform it into structured features suitable for recommendation algorithms:

3.3.3 | Feature Engineering and Vector Representation

Each stock will be represented as a **20-dimensional vector** to enable similarity matching with user profiles. The vector is constructed by reducing dimensionality of structured financial features and encoded textual content from company descriptions and filings. Each vector contains these dimensions (same as user profile):

Table 3.1: Stock Data Processing Pipeline Stages

Processing Stage	Description	Output
Basic Info Extraction	Extracts fundamental **company attributes** including sector classification, market capitalization, and geographic information	Structured JSON
Financial Ratio Calculation	Computes key **financial metrics**: profit margins, growth rates, valuation multiples, and dividend information	Numerical Features
Historical Analysis	Processes price **time series** to derive volatility measures, momentum indicators, and trading volume patterns	Time-series Features
Business Context	Captures **qualitative** information through company descriptions and business summaries	Textual Data

$$\text{Stock Vector}_{20d} = [\text{Sector Features}_{11d} \oplus \text{Investment Features}_{9d}] \tag{3.1}$$

Sector Feature Vector (11 dimensions):

- One-hot encoding representation across 11 predefined sectors
- Sectors include: Technology, Healthcare, Financial Services, Consumer Cyclical, Consumer Defensive, Energy, Industrials, Real Estate, Utilities, Communication Services, and Basic Materials
- Enables sector-based diversification in recommendations

Investment Feature Vector (9 dimensions):

1. **Market Capitalization Score**: Logarithmic scaling from small to large cap (0-1)
2. **Growth-Value Orientation**: Based on PEG ratio analysis (0=value, 1=growth)
3. **Dividend Attractiveness**: Combined dividend yield and payout ratio assessment
4. **Risk Level**: Volatility and beta coefficient composite
5. **Liquidity Score**: Trading volume-based liquidity measure
6. **Quality Metrics**: Profitability and return on equity combination
7. **Valuation Safety**: Inverse PE and PB ratio safety margin
8. **Momentum Strength**: Short and medium-term price momentum
9. **Operational Efficiency**: Operating margin and asset turnover composite

3.3.4 | Data Storage Architecture

The system employs a **dual-database architecture** to optimize for *different* data access patterns:

- **MongoDB**: Stores raw, unstructured stock data with flexible schema to accommodate varying data availability across different stocks
- **PostgreSQL**: Houses processed 20-dimensional vectors and structured stock metadata for efficient similarity computations
- **Data Synchronization**: Automated pipeline ensures consistency between raw data and processed vectors

The processed 20-dimensional stock vectors serve as the fundamental building blocks for both basic similarity-based recommendations and advanced multi-objective optimization, providing a rich feature representation that captures both fundamental characteristics and market behavior patterns.

4

System Design

4.1 | System Overview

FinSight is an AI-powered financial analytics and recommendation platform designed for investors and analysts. It integrates natural language understanding, time-series forecasting, vector search, and Retrieval-Augmented Generation (RAG) to provide end-to-end financial insights. The system design is shown in Figure 4.1.

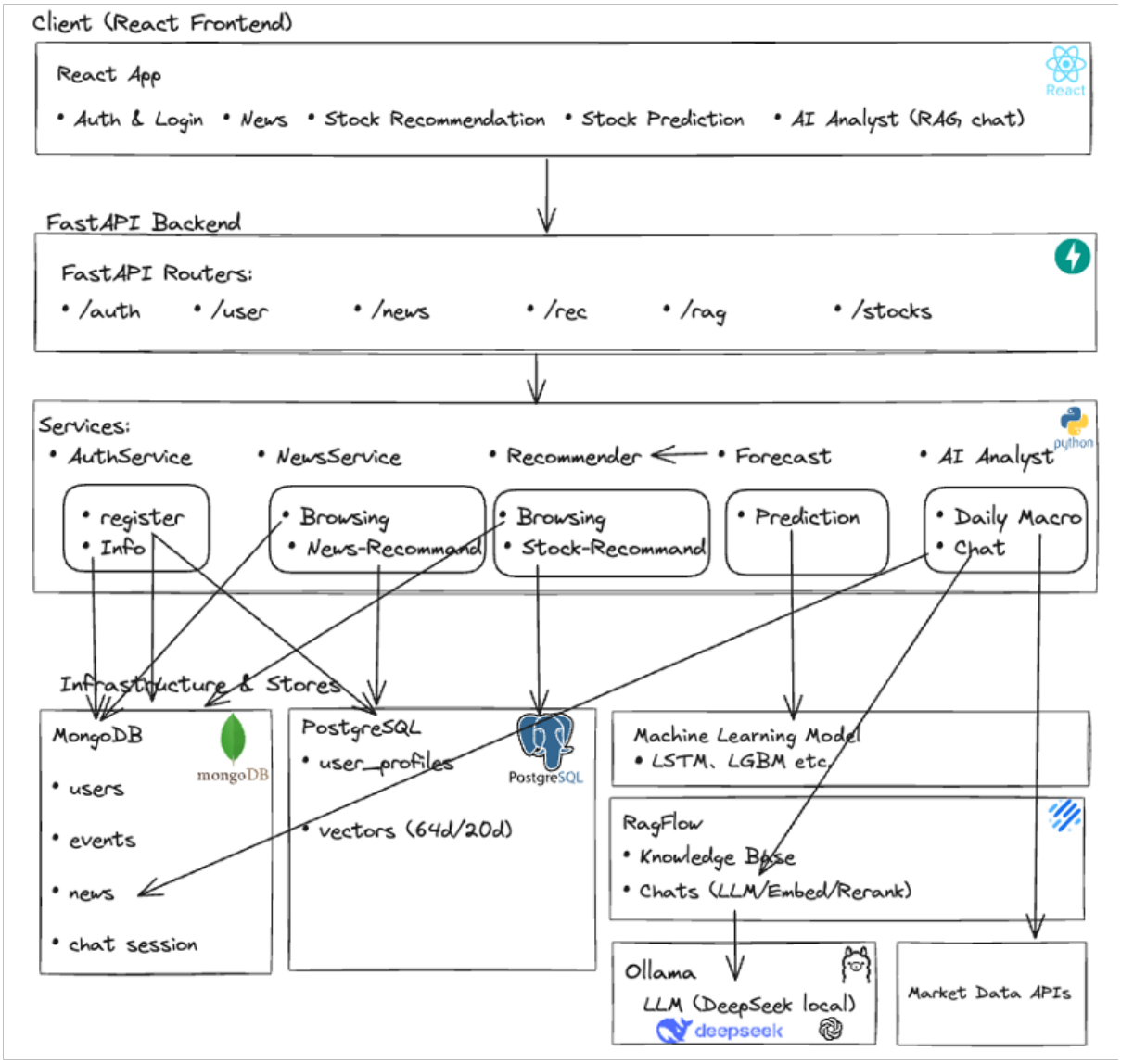


Figure 4.1: Finsight system design

4.1.1 | System Architecture

Module	Description
News Module	Fetches financial news via Google RSS and APIs, performs NLP parsing and vectorization.
Recommendation Module	Combines user profile embeddings and stocks vectors for personalized ranking.
Forecast Module	Predicts stock prices using ARIMA, Prophet, LGBM, LSTM, and ensemble models.
RAG Module	Integrates RagFlow for retrieval-augmented question answering.
User Profile Module	Maintains user semantic (64D) and preference (20D) embeddings and stores users base information.
Auth & User Service	Handles registration, login, JWT authentication, and profile setup.
Database Client	Manages MongoDB, PostgreSQL, and SSH tunneling.

4.1.2 | Data Flow and Integration

1. Users register for this platform. Each user will get his own user profile(a 20D vector that representsing their investment interests). Based on users' behavior(like click like dislike) to adjust their profile vector.
2. The frontend sends a REST API request to FastAPI endpoints such as `/forecast`, `/stocks`,`/news`, or `/rag`.
3. The backend loads user profiles, news data, stock data, or knowledge base content.
4. Depending on the task, it calls news modules, forecasting models, RAG reasoning, or recommendation logic.
5. The backend returns structured JSON results visualized by the frontend dashboard.

4.1.3 | Tech Stack

Layer	Technology and Description
Frontend	React + Vite + TailwindCSS + Recharts for interactive dashboards.
Backend	FastAPI (Python 3.12+) with modular routers and service layers.
Database	MongoDB for document data; PostgreSQL + pgvector for vector storage.

Machine Learning	Prophet, ARIMA, LightGBM, LSTM, Transformer and Seq2Seq forecasting.
LLM / RAG	RagFlow for retrieval-augmented reasoning; DeepSeek-8B and OpenAI embeddings; Ollama for llm services.
Infrastructure	Docker, Uvicorn, SSH tunnel, NUS HPC, and AWS EC2 deployment.

4.2 | News Browsing Module

Design goals were: a clean feed, de-duplication across sources, light personalization, and a smooth API surface. In practice we ship a two-vector design (semantic 64-d, preference 20-d), an **exclude-seen** filter backed by Mongo events, and a **dual refresh mode** that keeps first-paint fast while allowing small, quota-friendly real-time pulls. The front end renders a 3×3 grid with cached paging (previous/next/jump), so user actions immediately influence subsequent pages without blocking the UI.

4.2.1 | Overall Architecture Design

The module follows a clear **Service Repository** split with thin HTTP endpoints:

- **Datastores.** News and events are stored in **MongoDB** (documents: title, url, source, `published_at`, tickers, sector, topics, sentiment, plus 20-d profile vector $\mathbf{p}_a$); **PostgreSQL + pgvector** holds the normalized 64-d semantic embeddings for articles and the 20-d user profile $\mathbf{u} \in \mathbb{R}^{20}$.
- **Service core (NewsService).** Orchestrates candidate gathering, scoring, and post-filters. It also coordinates a tiny "trickle" ingestion when `refresh=1`—pulling ≤ 3 fresh items using symbol hints, upserting them, then re-ranking together with the local corpus.
- **Repositories.**
  - `NewsRepo` (Mongo) for latest/news lookup and upserts.
  - `PgProfileRepo` (Postgres) for $\mathbf{u}$ updates and vector math.
  - `EventRepo` (Mongo) logging `click/like/bookmark`, plus a compact **toggle** collection for idempotent "add" and reversible "remove".
- **HTTP surface (FastAPI).**
  - `GET /rec/user/news` – ranked results with `refresh={0,1}`, `symbols=`, `exclude_hours=` (exclude seen).
  - `POST /users/event/{click|like|bookmark}` – log behavior and update $\mathbf{u}$.
  - `GET /debug/news/latest` – fast first-paint sample (9 items) directly from the DB.

This layout keeps hot read paths short (DB-only) while still supporting tiny real-time updates, and isolates vector math and schema evolution behind stable service methods.

4.2.2 | Candidates, Scoring, and Ranking

At request time, the service constructs a candidate set and ranks it with a compact, interpretable score. If `refresh=0`, candidates are simply the latest K items from Mongo; if `refresh=1`, the service first pulls a **very small** batch of fresh articles (guided by **top-3 tickers** on the current page), upserts, and then re-queries latest. After candidate assembly we apply **exclude-seen** using recent user events (click/like/bookmark) within a sliding window (e.g., 30 days), then de-dup strictly by `news_id`.

The final score blends preference match, recency, and page-level diversity, then clips to $[0, 1]$:

$$\text{score}(u, a) \;=\; \alpha\, \cos(\mathbf{u}, \mathbf{p}_a) \;+\; \beta\, f_{\text{recency}}(a) \;+\; \gamma\, f_{\text{diversity}}(a \mid \mathcal{S}) \quad \in [0, 1].$$

Here $\mathbf{p}_a \in \mathbb{R}^{20}$ is the articles 20-d preference vector aligned with our sector/bucket basis; f_{recency} mildly favors newer items; $f_{\text{diversity}}$ discourages near-duplicates within the top-N slate $\mathcal{S}$. We then take the top $N = 9$ for one page and return {title, url, source, published_at, tickers, sector} for rendering.

4.2.3 | Feedback Loop and Online Profile Updates

Interactions close the loop via **append-only events** in Mongo and immediate updates to the 20-d user profile $\mathbf{u}$ in Postgres. Each click carries a weight:

$$w \;=\; 1 \;+\; 0.5\,\mathbf{1}[\text{dwell} \geq 10\text{s}] \;+\; 0.5\,\mathbf{1}[\texttt{liked}] \;+\; 0.5\,\mathbf{1}[\texttt{bookmarked}].$$

For **positive** actions (add) we nudge toward the articles preferences:

$$\mathbf{u} \;\leftarrow\; \mathbf{u} \;+\; \eta\, w\, \mathbf{p}_a.$$

For **removals** (e.g., un-like / un-bookmark) we apply a **fixed decrement** to the same dimensions—simple, stable, and snapshot-free:

$$\mathbf{u} \;\leftarrow\; \mathbf{u} \;-\; \eta_r\, \mathbf{p}_a, \qquad \eta_r \ll \eta.$$

We de-duplicate "add" by a $(\texttt{user}, \texttt{news}, \texttt{type})$ toggle so repeated likes do not double-count, and we no-op "remove" if the toggle is already off. In practice this yielded **monotone** improvements after positive feedback and sensible reversion after removals, without the fragility of storing/restoring full vector snapshots.

4.2.4 | Latency, Caching, and Refresh Modes

To keep the interface responsive and protect API quotas, we maintain two coordinated modes:

- **DB-only paging (`refresh=0`).** Used for initial paint and most page turns. The front end caches pages locally (arrays of 9 items) and keeps a set of `seenIds`. "Previous" and page jumps are pure client operations; "Next" is also cached unless the user is already on the **last** page.

- **Trickle refresh (`refresh=1`).** Triggered only when the user is on the last page and presses "Next." The service pulls at most a few **fresh articles** (guided by the current pages top-3 tickers), upserts, and re-ranks together with the corpus—delivering freshness at a **tiny** marginal cost.

In both modes, the server still enforces exclude-seen and de-dup, ensuring the 3×3 slate remains **novel**. The front end never blocks on re-ranking to show cached pages; it only waits on network when specifically expanding the tail with `refresh=1`.

4.3 | Stock Recommendation Module Architecture

4.3.1 | Overall Architecture Design

The stock recommendation module employs a sophisticated **microservices** architecture that seamlessly integrates with the broader investment platform. As illustrated in Figure 4.2, the module operates as a central intelligence hub, processing user preferences, market data, and financial metrics to generate personalized investment recommendations.

The architecture is designed for scalability and real-time performance, employing asynchronous processing patterns and intelligent caching strategies. The module supports multiple recommendation strategies that can be dynamically selected based on user preferences and market conditions.

4.3.2 | Basic Recommendation Engine

The basic recommendation engine forms the **foundation** of our stock recommendation system, implementing a sophisticated vector similarity approach combined with intelligent diversification.

Vector Similarity Foundation: The core algorithm computes **cosine** similarity between 20-dimensional user and stock vectors:

$$\text{Similarity}(\mathbf{U}, \mathbf{S}) = \frac{\sum_{i=1}^{20} U_i \cdot S_i}{\sqrt{\sum_{i=1}^{20} U_i^2} \cdot \sqrt{\sum_{i=1}^{20} S_i^2}} \tag{4.1}$$

Where $\mathbf{U}$ represents the user's 20-dimensional preference vector and $\mathbf{S}$ represents the stock's 20-dimensional characteristic vector. The 20 dimensions are carefully engineered to capture both sector preferences (dimensions 0-10) and investment style preferences (dimensions 11-19).

Diversification Enhancement: To prevent over-concentration in specific sectors, the system implements a dynamic diversification mechanism:

$$\text{Final Score} = \text{Raw Similarity} - \alpha \cdot \mathbb{I}_{\text{sector seen}} \tag{4.2}$$

Where α is the diversity factor (configurable from 0 to 0.3) and $\mathbb{I}_{\text{sector seen}}$ is an indicator function that equals 1 if the sector has already been selected in the current recommendation batch. This approach ensures that even if multiple technology stocks have high similarity scores, the system will prioritize including stocks from underrepresented sectors.

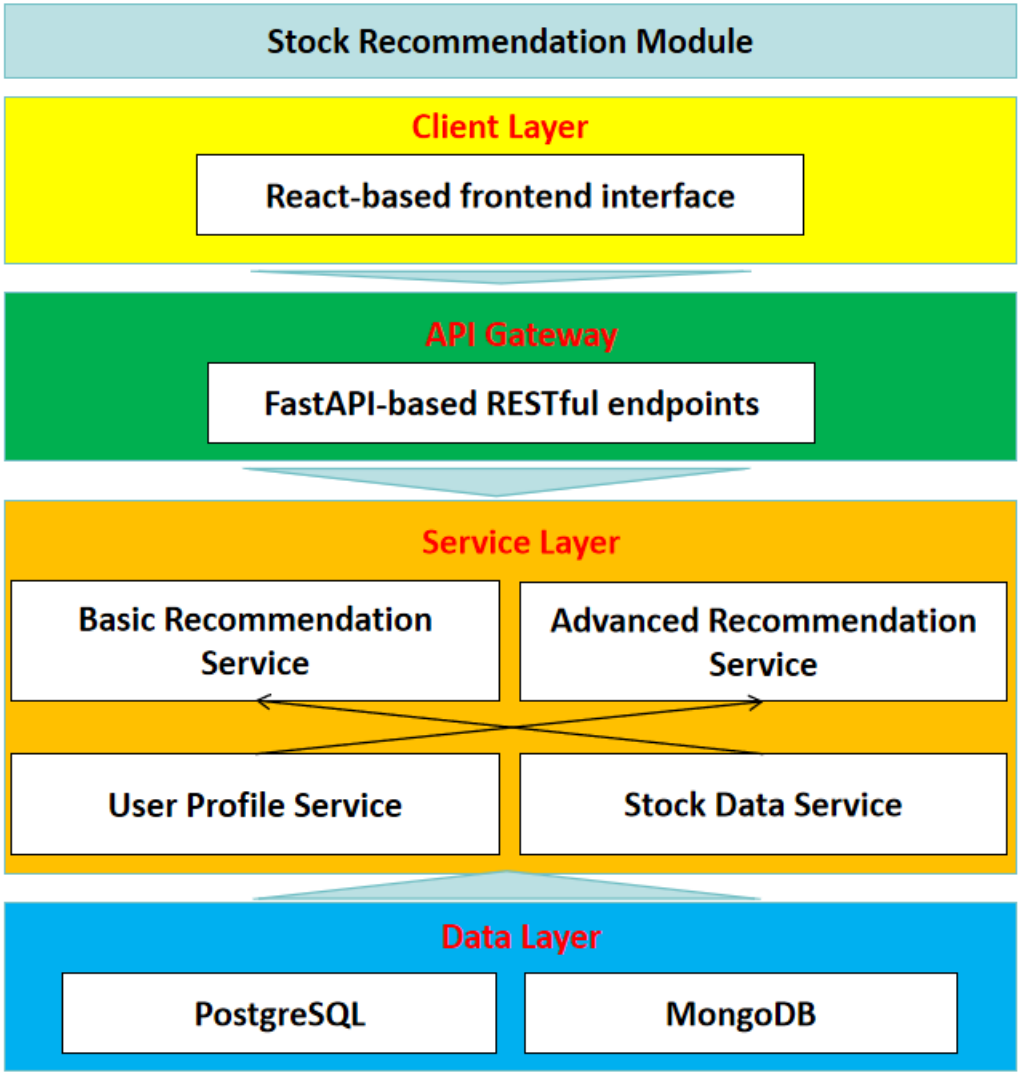


Figure 4.2: Stock Recommendation System Architecture

This approach ensures that recommendations are both personalized to user preferences and strategically diversified across market sectors.

4.3.3 | Advanced Multi-Objective Recommendation Engine

The advanced recommendation engine represents a significant evolution beyond basic similarity matching, implementing a sophisticated multi-objective optimization framework that balances four critical investment dimensions.

Multi-Objective Optimization Framework: The final recommendation score is computed as a weighted combination of four component scores:

$$\text{Final Score} = w_p \cdot S_p + w_r \cdot S_r + w_d \cdot S_d + w_t \cdot S_t \tag{4.3}$$

Each component score represents a distinct investment objective:

Preference Similarity Score (S_p): This component measures alignment between user preferences and stock characteristics using the same 20-dimensional vector similarity as the basic engine, but with additional normalization to ensure compatibility with other components.

Risk-Adjusted Return Score (S_r): This sophisticated component evaluates the risk-reward profile of each stock:

$$S_r = \frac{\text{Expected Return}}{\max(\text{Volatility}, 0.01)} \tag{4.4}$$

Expected return is estimated using a multi-factor model:

$$\text{Expected Return} = 0.3 \cdot R_{\text{historical}} + 0.4 \cdot R_{\text{growth}} + 0.3 \cdot R_{\text{valuation}} \tag{4.5}$$

Where:

- $R_{\text{historical}}$: 3-month price momentum normalized to annual returns
- R_{growth}: Revenue growth projections from fundamental analysis
- $R_{\text{valuation}}$: Valuation reversal effect based on P/E ratios

Volatility is computed from 30-day historical price movements and annualized for consistency.

Diversification Benefit Score (S_d): This component evaluates each stock's contribution to portfolio diversification:

$$S_d = 1 - \text{Sector Concentration} \tag{4.6}$$

Sector concentration is calculated as:

$$\text{Sector Concentration} = \frac{\text{Number of stocks in sector}}{\text{Total stocks in universe}} \tag{4.7}$$

This ensures that stocks from underrepresented sectors receive higher diversification scores, promoting balanced portfolio construction.

Market Timing Score (S_t): This dynamic component adapts recommendations to current market conditions:

$$S_t = \begin{cases} \beta \cdot 0.8 + \text{Dividend Yield} \cdot 5 \cdot 0.2 & \text{if bear market} \\ \beta \cdot 0.8 + \text{Growth Score} \cdot 0.2 & \text{if bull market} \\ (1 - \text{Volatility}) \cdot 0.5 + \text{Valuation Score} \cdot 0.5 & \text{if sideways market} \end{cases} \tag{4.8}$$

Market regime detection analyzes SPY ETF data using 3-month returns and volatility:

- **Bull Market**: Return $>$ 10% and Volatility $<$ 20%
- **Bear Market**: Return $<$ -5% and Volatility $>$ 25%
- **Sideways Market**: All other conditions

Dynamic Weight Adjustment: The system employs risk-profile-based weight configurations:

Table 4.3: Multi-Objective Weight Configuration by Risk Profile

Risk Profile	Preference (w_p)	Risk Return (w_r)	Diversification (w_d)	Timing (w_t)
Conservative	0.2	0.4	0.3	0.1
Balanced	0.3	0.4	0.2	0.1
Aggressive	0.3	0.5	0.1	0.1

4.3.4 | Real-time User Behavior Integration

Both recommendation engines incorporate real-time user feedback to continuously refine preference models:

$$\mathbf{U}_{t+1} = \mathbf{U}_t + \eta \cdot (\mathbf{S} - \mathbf{U}_t) \tag{4.9}$$

Where η is the learning rate that varies by behavior type:

- $\eta = 0.05$ for click interactions
- $\eta = 0.20$ for favorite/dislike actions
- $\eta = -0.20$ for removal of previous interactions

This adaptive learning mechanism ensures that the system evolves with user preferences, providing increasingly relevant recommendations over time.

4.4 | Stock Trend Prediction Module

The Stock Forecasting module forms one of the core intelligent reasoning components of the FinSight system. It provides users with **real-time price forecasting** capabilities powered by machine learning and time-series models. The system integrates a FastAPI-based backend with a React-based frontend, enabling seamless interaction and visualization.

4.4.1 | System Architecture

The stock prediction module forms a core component of the FinSight system, providing short-horizon predictions such as one-day, seven-day, and thirty-day forward prices. We demonstrate familiarity with **Machine Learning Operations (MLOps)** here. It adopts a microservice architecture that **separates responsibilities** between data processing, model inference, and user presentation. The backend, implemented in Python using the FastAPI framework, exposes several endpoints. These endpoints handle data retrieval, model orchestration, and forecast generation. Historical price data are stored in MongoDB database, where each document contains time-series data represented as a list of date-close pairs. The forecasting service reads this data, invokes the appropriate prediction models, and returns structured results through a REST interface.

On the frontend, The user can select a stock ticker from the recommandation tickers' list, and the interface presents both the last seven actual closing prices and the predicted trend for

upcoming horizons. This integration enables a smooth interaction between data analytics and visualization, delivering both reasoning and interpretability to end users.

The **Stock Trend Prediction Module Design** is shown in Figure 4.3

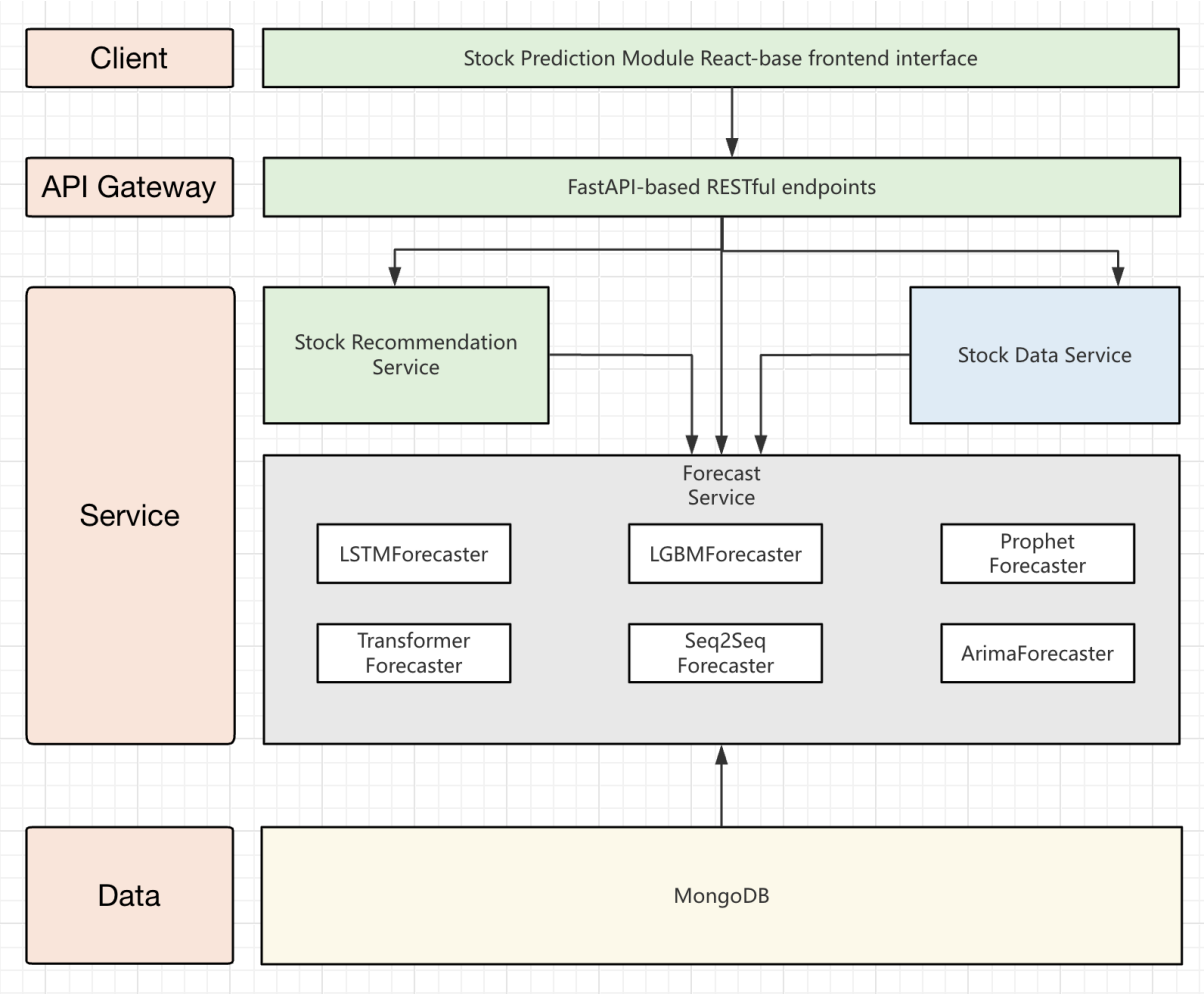


Figure 4.3: Stock Trend Prediction Module Design

4.4.2 | Frontend and Backend Data Processing

The data flow between the frontend and backend follows a clear **five-stage** pipeline. **First**, the user selects a ticker symbol within the interface. The **frontend then** issues two asynchronous requests to retrieve the most recent seven actual closing prices. Upon receiving these requests, the **backend** loads the corresponding historical series from our MongoDB database, performs validation, and automatically determines which forecasting model or ensemble to use based on data length and volatility. The **chosen model** then produces a list of predicted values, each associated with a forecast horizon in days and optionally a confidence score. All results are **normalized** into a unified `ForecastResult` structure, which contains the ticker symbol, current price, model name, and an array of prediction points.

On the frontend side, the two responses are merged into a continuous timeline that combines historical and future values. Dates for predicted points are generated by adding the forecast horizon to the current date, ensuring chronological alignment. The React component then

renders a dual-line chart using the Recharts library, where the solid line represents actual historical prices and the dashed line displays the predicted trajectory. When available, confidence intervals can also be visualized as shaded bands around the prediction curve.

Through this bidirectional communication pipeline, users experience a real-time, interpretable forecasting workflow that blends reasoning and visualization.

4.4.3 | Forecaster Types and Selection

The forecasting service integrates a diverse set of models organized into three methodological families.

The forecasting engine integrates multiple models spanning statistical, machine learning, and deep learning approaches. Each model family contributes a distinct reasoning capability to handle different market conditions and data characteristics. A summary of these forecasters is given below:

- **LSTM (Long Short-Term Memory).**

  LSTM is a **recurrent neural network** architecture capable of learning long-range temporal dependencies. It aims at mitigating the vanishing gradient problem commonly encountered by traditional RNNs. It excels in capturing nonlinear price dynamics and is particularly effective with large, continuous datasets.

  An LSTM unit consists of **a cell and three gates**—input, forget, and output—that regulate the flow of information. The cell preserves information across time, while the gates determine what to discard, store, or output based on the current input and previous state. This selective control enables the network to retain long-term dependencies and make accurate sequential predictions.

  The compact form of the equations for the forward pass of an LSTM cell with a forget gate is given by:

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f), \tag{4.10}$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i), \tag{4.11}$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o), \tag{4.12}$$
$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c), \tag{4.13}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \tag{4.14}$$
$$h_t = o_t \odot \sigma_h(c_t), \tag{4.15}$$

  where the initial values are $c_0 = 0$ and $h_0 = 0$, and the operator $\odot$ denotes the element-wise product. The subscript t indexes the time step.

  Variables Let superscripts d and h refer to the number of input features and hidden units, respectively:

  – $x_t \in \mathbb{R}^d$: input vector to the LSTM unit,

- $f_t \in (0,1)^h$: forget gate's activation vector,
- $i_t \in (0,1)^h$: input/update gate's activation vector,
- $o_t \in (0,1)^h$: output gate's activation vector,
- $\tilde{c}_t \in (-1,1)^h$: cell input activation vector,
- $c_t \in \mathbb{R}^h$: cell state vector,
- $h_t \in (-1,1)^h$: hidden state vector (also known as the output vector),
- $W \in \mathbb{R}^{h \times d}$, $U \in \mathbb{R}^{h \times h}$, $b \in \mathbb{R}^h$: weight matrices and bias vectors learned during training.

Activation Functions

- σ_g: sigmoid activation function,
- σ_c: hyperbolic tangent function,
- σ_h: hyperbolic tangent function, or the identity function $\sigma_h(x) = x$ in the peephole LSTM variant.

- **ARIMA (Auto-Regressive Integrated Moving Average).**

  A classical time-series model that captures **autocorrelation patterns and local trends**. It performs well for stationary or mildly non-stationary data and is computationally efficient for short horizons. We use non-seasonal ARIMA model in the Finsight prediction module.

  Non-seasonal ARIMA models are usually denoted as ARIMA(p, d, q), where the parameters p, d, q are non-negative integers. Here, p is the order (number of time lags) of the *autoregressive* part of the model, d is the degree of differencing (the number of times the data have had past values subtracted), and q is the order of the *moving-average* part.

  Given time series data X_t, where t is an integer index and X_t are real numbers, an ARMA(p', q) model is given by

$$X_t - \alpha_1 X_{t-1} - \cdots - \alpha_{p'} X_{t-p'} = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}, \tag{4.16}$$

  or equivalently,

$$\left(1 - \sum_{i=1}^{p'} \alpha_i L^i\right) X_t = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right) \varepsilon_t, \tag{4.17}$$

  where L is the *lag operator*, the α_i are the parameters of the autoregressive part of the model, the θ_i are the parameters of the moving average part, and the ε_t are the error terms. The error terms ε_t are generally assumed to be independent and identically distributed (i.i.d.) random variables sampled from a normal distribution with zero mean.

- **Transformer.**

  The Transformer forecaster adopts a **self-attention**-based architecture that replaces recurrent computation with parallelizable attention mechanisms. Unlike RNNs, which process

sequences step by step, Transformers model all time-step relationships simultaneously, allowing them to learn long-range dependencies and contextual interactions more efficiently.

Given an input matrix $X \in \mathbb{R}^{L \times d}$ of historical embeddings, the self-attention mechanism computes:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V, \tag{4.18}$$

where $Q = XW_Q$, $K = XW_K$, and $V = XW_V$ are the query, key, and value projections with learnable matrices $W_Q, W_K, W_V \in \mathbb{R}^{d \times d_k}$. This attention operation enables the model to weigh the relative importance of all past observations when producing a new forecast representation.

The Transformer forecaster **used in FinSight** employs multiple attention heads and residual feed-forward layers to capture both local and global temporal structures. Positional encodings are added to preserve sequence order, and layer normalization ensures stable convergence during training. The decoder projects the attention output to the predicted horizon through a linear layer.

- **Seq2Seq.**

  The Sequence-to-Sequence (Seq2Seq) forecaster extends the recurrent neural network framework by **explicitly separating the encoding and decoding** phases of temporal modeling. In the FinSight implementation, the encoder reads an input sequence of historical prices and compresses it into a latent representation h_t, which summarizes the temporal dynamics of the recent window. The decoder then unfolds this hidden context over future horizons, generating multiple predicted values recursively or in parallel.

  Formally, given an input sequence $X = (x_{t-L+1}, \ldots, x_t)$, the encoder produces a sequence of hidden states:

$$h_j = f_{\text{enc}}(x_j, h_{j-1}), \quad j = t - L + 1, \ldots, t, \tag{4.19}$$

  where f_{enc} is typically a Gated Recurrent Unit (GRU) or long short-term memory (LSTM) cell. The decoder then generates forecasted outputs $\hat{y}_{t+k}$ as:

$$s_k = f_{\text{dec}}(s_{k-1}, \hat{y}_{t+k-1}, h_t), \qquad \hat{y}_{t+k} = W_o s_k + b_o, \tag{4.20}$$

  where s_k is the decoder hidden state and h_t provides context from the encoder.

  The Seq2Seq design allows the model to capture asymmetric temporal dependencies between past and future horizons. It can learn patterns where the relevance of earlier observations decays non-linearly, and can adapt to varying forecast lengths without retraining. In FinSight, this forecaster is implemented as a lightweight, single-layer GRU-based encoder decoder pair, optimized for short- to medium-term forecasting horizons. The model selection policy **activates** the Seq2Seq forecaster when the **available training series is sufficiently long** and exhibits **sequential dependencies** that simpler autoregressive models cannot capture effectively.

- **Prophet.**

Prophet is a procedure for forecasting time series data based on an **additive model** where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have **strong seasonal effects and several buckets of historical data**. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

- **LightGBM(Light Gradient-Boosting Machine).**

  A gradient boosting framework based on **decision trees** that can model nonlinear relationships and interactions. It is used for ranking, classification and other machine learning tasks. When using gradient descent, one thinks about the space of **possible configurations of the model as a valley**, in which the lowest part of the valley is the model which most closely fits the data. In this metaphor, one walks in different directions to learn how much lower the valley becomes. When enhanced with engineered lag and technical indicators, it yields highly adaptive short-term forecasts.

Together, these components form a cohesive reasoning system that bridges data analytics with cognitive visualization.

It showcases an intelligent forecasting pipeline capable of autonomous model choice, explainable prediction generation, and visually intuitive results that facilitate informed financial decision-making.

4.5 | AI Analyst

4.5.1 | System Overview

The FinSight AI Analyst subsystem enables both daily macro data checking and financial question answering by combining Large Language Models (LLMs) with a retrieval-based knowledge engine (RagFlow).

Core Objectives

1. Integrate retrieval and reasoning to produce accurate, evidence-backed answers specific in Finance Field.
2. Enable multi-turn dialogue with persistent contextual memory.
3. Allow dynamic selection of model and knowledge base per user query.
4. Maintain modular, extensible backend architecture.

4.5.2 | Architecture

AI Analyst is consisted with four modules which are Frontend | Backend | RagFlow | Storage. The frontend is React-based web interface for user interaction. It provides input box, model/-dataset selection, and chat display and displays citations and retrieved document summaries.

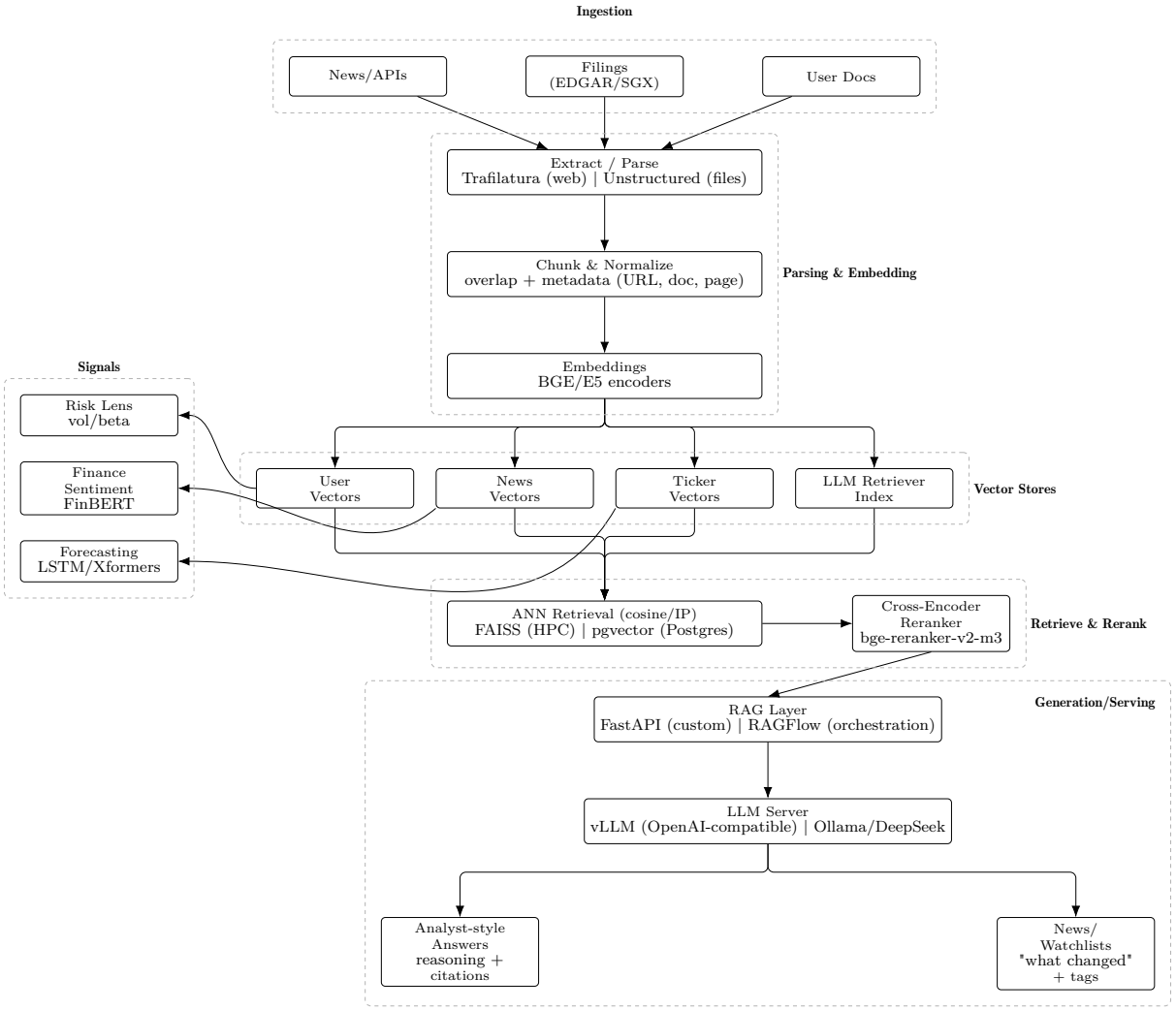


Figure 4.4: FinSight overview: ingest → parse → embed → retrieve & rerank → generate. Vector search is the backbone (FAISS on HPC; pgvector for persistent Postgres-backed search). Sidecars provide sentiment, risk and short-horizon forecasts. For RAG, retrieval is reranked, then fed to an LLM server—OpenAI-compatible vLLM or Ollama/DeepSeek behind RAGFlow—for grounded answers with citations.

The Backend is built for managing **session creation, context assembly, and RagFlow API calls**. It contains dedicated routers and services for modular functionality:

1. /rag/chat: Handles chat requests and message orchestration.
2. /rag/history/id: Retrieves chat history by session ID.
3. /rag/models: Lists available LLM models.
4. /rag/datasets: Lists available knowledge bases.

RagFlow is an external service handling document retrieval, reranking, and context-grounded generation and it exposes /api/v1/chats and OpenAI-compatible completion endpoints. Ragflow is easily deployed by **docker** and easy to use. All the chatting sessions are stored in MongoDB. It stores chat sessions, message history, and timestamps which makes Multi-round conversation available. It also supports querying, session resumption, and lightweight analytics.

5

Implementation Detail

5.1 | News Browsing Module

This section details how the news module was built to realize the system design: a two vector recommender (64-d semantic, 20-d preference) with exclude-seen filtering, dual refresh, and low-latency APIs. We focus on concrete engineering choicesingestion, storage schemas, scoring, profile updates, paging/cache behavior, and observabilityso the reader can reproduce the system end-to-end.

5.1.1 | API Design and Implementation

The News Browsing Module is implemented through a structured set of RESTful API endpoints using the FastAPI framework, aligning directly with the modular architecture described earlier. Each endpoint is linked to a dedicated router function, backed by a clear servicerepository abstraction that isolates business logic from database operations.

Core API Endpoints and Functionalities:

Table 5.1: News Browsing API Endpoints Overview

API Endpoint	Method	Parameters	Implementation Details
/rec/user/news	GET	user_id, limit, refresh, exclude_hours, symbols	Retrieve personalized ranked news; supports `refresh={0,1}` dual-mode fetching
/users/event/{click|like|bookmark}	POST	user_id, news_id, op	Log user interactions, update 20-d user profile vector incrementally
/debug/news/latest	GET	None	Fetch the latest 9 stored news entries for initial UI rendering
/news/fetch-batch	POST	watchlist, window, source	Trigger batch ingestion from Google RSS and Marketaux APIs

The module exposes lightweight endpoints optimized for fast response times and minimal latency. For instance, `/rec/user/news` handles both cached database retrieval (`refresh=0`) and live update mode (`refresh=1`), while the `/users/event` endpoints synchronize user feedback

with immediate vector updates in PostgreSQL. Each endpoint follows consistent patterns of schema validation, logging, and structured JSON response generation.

5.1.2 | Ingestion, Normalization, and Storage

The pipeline is implemented as a disciplined loop

$$\text{fetch} \rightarrow \text{normalize} \rightarrow \text{deduplicate} \rightarrow \text{enrich} \rightarrow \text{vectorize} \rightarrow \text{store}.$$

Sources consist of Google RSS (broad, fast-moving headlines) and Marketaux API (finance-focused, ticker-aware feed). Each item is canonically keyed by $\{\texttt{url}, \texttt{external\_id}\}$ to avoid duplicates across sources. Normalization trims titles, resolves redirects, strips boilerplate, and harmonizes timestamps to UTC.

Storage split. Enriched news documents are written into **MongoDB** (`NewsRepo`) with fields:

$$d = \{\texttt{news\_id}, \texttt{title}, \texttt{url}, \texttt{source}, \texttt{published\_at}, \texttt{tickers}, \texttt{sector}, \texttt{topics}, \mathbf{e}_{64}, \mathbf{p}_{20}\}.$$

User profiles live in **PostgreSQL + pgvector** (`PgProfileRepo`). We maintain:

- Two semantic tracks: `user_semantic_64d_short` and `user_semantic_64d_long` $\in \mathbb{R}^{64}$ (EMA with different half-lives).
- A human-interpretable preference vector $\mathbf{u} \in \mathbb{R}^{20}$ aligned to 11 sector + 9 investment-style dimensions. For compatibility and ease of evolution, we store a JSON-mirrored copy as `profile_vector_20d` :: `TEXT` and component JSONs `industry_preferences`, `investment_preferences` (both `JSON`). Reads/writes normalize to a fixed-length 20-d list to avoid schema drift.

Append-only user events (`click/like/bookmark`) are recorded in Mongo (`EventRepo`); a small `user_event_toggles` collection supports idempotent add/remove for UI toggles.

5.1.3 | Ranking Path (/rec/user/news) and Exclude-Seen

The recommendation endpoint constructs a candidate slate and ranks it with a compact score. For database-only mode (`refresh` $= 0$), candidates are pulled from Mongo by `latest`(K). If the client is on the last page and requests `refresh` $= 1$, the service performs a *trickle refresh*: fetches ≤ 3 fresh items from Marketaux (optionally guided by the pages *top-3 tickers*), upserts, then re-queries latest. Recently interacted items are filtered out using a sliding-window set from `EventRepo`:

$$\mathcal{C}' \;=\; \{a \in \mathcal{C} \mid a.\texttt{news\_id} \notin \texttt{seenIds}(u; \Delta T)\}.$$

Scoring blends preference match, mild recency, and slate-level diversity:

$$s(u, a) = \alpha \cos(\mathbf{u}, \mathbf{p}_{20}(a)) \;+\; \beta\, f_{\text{recency}}(a) \;+\; \gamma\, f_{\text{div}}(a \mid \mathcal{S}), \quad s \in [0, 1].$$

We truncate to $N = 9$ for a single page. The endpoint returns a thin JSON for UI: $\{\texttt{title}, \texttt{url}, \texttt{source}, \texttt{publish}$

5.1.4 | Online Profile Updates and Event Handling

The feedback loop is closed by three endpoints: `/users/event/click`, `/users/event/like`, `/users/event/bookmark`. Clicks log dwell time and optionally carry `liked/bookmarked` flags; likes and bookmarks are toggleable (add/remove) with idempotency enforced in Mongo. Profile updates happen synchronously in Postgres:

Click-driven semantic EMA. Let $\mathbf{e}_{64}(a)$ be article embedding and w the event weight

$$w \;=\; 1 \;+\; 0.5\,\mathbf{1}[\text{dwell} \geq 10\text{s}] \;+\; 0.5\,\mathbf{1}[\text{liked}] \;+\; 0.5\,\mathbf{1}[\text{bookmarked}].$$

We apply per-track EMA after L2-normalization:

$$\mathbf{u}^{(\text{short})} \leftarrow (1-\eta_s)\mathbf{u}^{(\text{short})} + \eta_s\, w\, \widehat{\mathbf{e}}_{64}(a), \quad \mathbf{u}^{(\text{long})} \leftarrow (1-\eta_\ell)\mathbf{u}^{(\text{long})} + \eta_\ell\, w\, \widehat{\mathbf{e}}_{64}(a).$$

Preference (20-d) updates. For positive actions (`add`) we nudge toward the articles $\mathbf{p}_{20}(a)$ with small learning rates on two segments (sector 11-d and style 9-d), using EMA on the actually-hit dimensions; for removals (`remove`) we apply fixed decrements $(\delta_{\text{ind}}, \delta_{\text{inv}})$ only on the dimensions the article meaningfully activated. All writes re-serialize to a canonical 20-d JSON string (`profile_vector_20d`) and update the component JSONs. This design avoids snapshot brittleness while guaranteeing the stored shape of $\mathbf{u}$ remains a dense 20-d list.

5.1.5 | Caching, Refresh, and Front-End Integration

The client-side rendering follows a paginated layout (3×3 grid) managed by React. Pages are cached locally in browser memory, and navigation among cached pages (`previous/next/jump`) does not trigger network calls. The backend only refreshes content when:

$$\text{user\_on\_last\_page} \;\wedge\; \text{next\_clicked} \;\Rightarrow\; \texttt{refresh} = 1.$$

In that case, the server fetches at most 3 fresh items, upserts them into MongoDB, re-ranks the corpus, and returns the updated slate. This dual refresh mode guarantees minimal latency (< 200 ms typical) and conforms to API quota limits.

The front end also integrates lightweight event listeners for each interaction type:

- `onClick`: logs click duration and news ID.
- `onLike / onBookmark`: toggles user preferences visually and asynchronously triggers backend updates.

This pipeline completes a full feedback loop where user behavior directly refines recommendation results without blocking the UI.

Overall, the implementation harmonizes real-time responsiveness with model interpretabilityachieving low-latency news personalization consistent with the architectural design goals.

5.2 | Stock Recommendation Module Implementation

5.2.1 | API Design and Implementation

The stock recommendation module implements a comprehensive RESTful API based on FastAPI framework. The API endpoints are organized in dedicated router files and follow consistent patterns for request handling, validation, and response formatting.

Core API Endpoints and Functionality:

Table 5.2: Stock Recommendation API Endpoints Overview

API Endpoint	Method	Parameters	Implementation Approach
/stocks/recommend	GET	user_id, top_k, diversity_factor	Basic recommendation using cosine similarity with sector diversification
/stocks/recommend/v2	GET	user_id, top_k, risk_profile	Multi-objective optimization with parallel component scoring
/stocks/raw-data/{symbol}	GET	symbol	Direct MongoDB document retrieval with data cleaning
/stocks/fetch-raw-data	POST	symbols	Batch yfinance data fetching with MongoDB storage
/stocks/update-vectors	POST	symbols	20D vector computation and PostgreSQL update
/users/behavior/update	POST	user_id, behavior_type, stock_symbol	Real-time user preference adjustment

5.2.2 | Basic Recommendation Algorithm Implementation

The basic recommendation algorithm implements a sequential workflow focused on vector similarity with intelligent diversification. The complete implementation flow is illustrated in Figure 5.1.

The algorithm begins by retrieving the user's 20-dimensional preference vector from the user_profiles table alongside all available stock vectors from the stock_vectors table in PostgreSQL. These vectors are pre-computed and encapsulate both sector characteristics and investment style preferences.

The core computation involves calculating cosine similarity between the user vector and each stock vector using scikit-learn's optimized linear algebra routines. This produces raw similarity scores representing the fundamental alignment between user preferences and stock characteristics. The stocks are then sorted by these raw scores in descending order.

[Basic Recommendation Implementation Flowchart Description]

1. **Input Processing**: Receive user ID, top-K count, and diversity factor parameters
2. **Data Retrieval**: Fetch user vector from user_profiles table and all stock vectors from stock_vectors table in PostgreSQL
3. **Similarity Computation**: Calculate cosine similarity between user vector and each stock vector using scikit-learn's cosine_similarity function
4. **Initial Ranking**: Sort all stocks by raw similarity scores in descending order
5. **Diversification Processing**:
   - Initialize empty selected stocks list and sectors seen set
   - For each stock in sorted order:
     - Apply sector penalty if sector already in sectors seen set
     - Calculate final score: raw similarity - (diversity factor Œ sector penalty)
     - Add stock to selected list and sector to sectors seen set
   - Stop when selected list reaches top-K size
6. **Result Generation**: Format recommendations with symbols, names, sectors, and both raw/adjusted similarity scores
7. **API Response**: Return structured JSON response to client

Figure 5.1: Basic Recommendation Algorithm Implementation Flow

The diversification mechanism represents the algorithm's key innovation. Rather than simply selecting the top-K highest similarity stocks, the system maintains a running set of selected sectors and applies configurable penalties to stocks from already-represented sectors. The diversification factor parameter (typically 0.1) controls the penalty magnitude, balancing pure similarity against sector diversity. This ensures the final recommendations provide exposure across multiple industries while maintaining high relevance to user preferences.

The complete implementation resides in the StockService.recommend_stocks() method, with the diversification logic encapsulated in the _diversify_recommendations() helper method. The algorithm is optimized for low-latency responses, typically completing within 100-200 milliseconds for standard stock universes, making it suitable for real-time recommendation scenarios.

5.2.3 | Advanced Multi-Objective Recommendation Implementation

The advanced recommendation algorithm implements a sophisticated parallel processing pipeline that balances four distinct investment objectives through weighted optimization. The complete implementation architecture is illustrated in Figure 5.2.

The advanced algorithm begins with parallel data acquisition, simultaneously fetching multiple data sources to minimize latency. This includes retrieving the user vector and stock vectors from PostgreSQL, batch-fetching raw stock data from MongoDB for fundamental analysis, and

[**Advanced Recommendation Implementation Flowchart Description**]

1. **Parameter Processing**: Receive user ID, top-K count, and risk profile parameters
2. **Parallel Data Acquisition**:
   - Fetch user vector from PostgreSQL
   - Retrieve all stock vectors from PostgreSQL
   - Batch fetch raw stock data from MongoDB
   - Analyze current market regime using SPY ETF data
3. **Concurrent Component Scoring**:
   - **Preference Similarity**: Cosine similarity between user and stock vectors
   - **Risk-Adjusted Return**: Expected return / volatility calculations
   - **Diversification Benefit**: 1 - sector concentration analysis
   - **Market Timing**: Regime-adaptive scoring based on current market conditions
4. **Weight Application**: Apply risk-profile-specific weights to each component score
5. **Composite Scoring**: Calculate final scores using weighted sum of components
6. **Explanation Generation**: Create detailed rationales for each recommendation
7. **Result Delivery**: Return ranked recommendations with comprehensive scoring breakdown

Figure 5.2: Advanced Recommendation Algorithm Implementation Flow

analyzing current market regime conditions using SPDR S&P 500 ETF Trust (SPY ETF) data through yfinance integration.

The core innovation lies in the **concurrent computation of** four objective components. The **preference similarity** component calculates alignment between user and stock vectors using cosine similarity, identical to the basic algorithm. Simultaneously, the **risk-adjusted return component** computes expected returns based on historical performance, growth projections, and valuation metrics, then normalizes by volatility to produce Sharpe ratio-inspired scores.

The **diversification benefit** scoring analyzes sector concentration across the entire stock universe, assigning higher scores to stocks from underrepresented sectors to promote portfolio balance. Concurrently, the **market timing** component evaluates how well each stock aligns with the current market regime, favoring high-beta growth stocks in bull markets and defensive dividend stocks in bear markets.

After all component scores are computed, the system applies **dynamic weights** based on the user's specified risk profile. Conservative profiles (weights: 0.2 preference, 0.4 risk return, 0.3 diversification, 0.1 timing) emphasize safety and balance, while aggressive profiles (weights: 0.3 preference, 0.5 risk return, 0.1 diversification, 0.1 timing) prioritize returns and growth potential.

The weighted combination produces final scores that balance multiple investment considerations, followed by explanation generation that synthesizes component scores into human-

readable rationales. This entire parallel workflow is implemented in the MultiObjectiveRecommender.recommend_stocks() method and typically completes within 300-500 milliseconds, providing comprehensive multi-factor analysis while maintaining responsive performance.

Both algorithms integrate with the user behavior tracking system through the /users/behavior/update API endpoint, allowing real-time preference updates based on user interactions. This creates an adaptive feedback loop where user engagement continuously refines the underlying preference vectors, creating increasingly personalized investment guidance over time.

5.3 | Stock Trend Prediction Module

5.3.1 | API Design and Implementation

The stock forecasting module provides timeseriesbased prediction capabilities through a structured FastAPI service layer. Each endpoint encapsulates model orchestration, data retrieval, and standardized response formatting to ensure consistency across forecasting algorithms.

Core API Endpoints and Functionality:

Table 5.3: Stock Forecast API Endpoints Overview

API Endpoint	Method	Parameters	Implementation Approach
/forecast/{ticker}	GET	ticker, horizons, method	Main entry point for model-based forecasting (ARIMA / Prophet / LGBM / LSTM)
/forecast/batch	GET	limit, method, ticker	Batched multi-stock forecasting for dashboard display
/forecast/prices7	GET	ticker	Retrieve last 7 days closing prices from MongoDB
/forecast/cache/refresh	POST	symbols	Refresh cached predictions and update MongoDB forecast collection

Each endpoint is implemented in the `forecast_router.py` file, interacting with the `ForecastService` layer for business logic. Data is retrieved from MongoDB (historical price series) and optionally persisted to PostgreSQL for analytical storage. All responses are serialized into standardized JSON objects containing dates, predicted values, confidence intervals, and model metadata.

5.3.2 | Basic Forecasting Pipeline Implementation

The basic forecasting pipeline provides short-horizon (730 days) predictions using lightweight statistical and machine learning models. The complete implementation workflow is summarized in Figure 5.3.

[Basic Forecasting Implementation Flowchart Description]

1. **Request Handling**: Receive ticker symbol, forecast horizon, and model method from the API request.
2. **Data Acquisition**: Query MongoDB for recent stock data; preprocess by resampling and forward-filling missing prices.
3. **Model Selection**: Dynamically instantiate the chosen model class (ARIMA, Prophet, LGBM, LSTM) through a unified factory method.
4. **Training Phase**: Split historical data into training and validation sets; fit the selected model to the training segment.
5. **Forecast Generation**: Predict future price trajectory for the specified horizon; compute lower and upper confidence intervals.
6. **Post-Processing**: Merge historical and forecast data for chart visualization; log model performance metrics (RMSE, MAE).
7. **Caching and Response**: Cache predictions to MongoDB for fast repeated access and return structured JSON to the frontend.

Figure 5.3: Basic Forecasting Pipeline Implementation Flow

This pipeline leverages modular forecaster classes such as `ArimaForecaster`, `ProphetForecaster`, `LgbmForecaster`, and `LstmForecaster`, all inheriting from a shared `Forecaster` base interface. Each subclass implements `fit()` and `predict()` methods to standardize model integration. The forecast results are serialized into a unified schema defined in `ForecastResult` dataclass, ensuring frontend compatibility for chart rendering and diagnostic visualization.

5.3.3 | Visualization and Diagnostics Integration

The **visualization layer** of the Stock Forecast Module is designed to provide users with an intuitive and data-rich interface for understanding predicted market trends. The frontend, built with React and Recharts, consumes data from the `/forecast/{ticker}` endpoints and presents it in an interactive, multi-horizon layout.

Interface Overview:

The interface is divided into four main functional regions:

1. **Summary Dashboard (Top Section):**
   - **Current Price:** Displays the latest real-time stock price, fetched via the backend `/forecast/prices7` API.
   - **Forecast Price (1 Day):** Shows the next-day predicted price, along with the model type used (e.g., Transformer, Prophet, LSTM).
   - **Confidence Level:** Indicates the model's confidence in the current prediction, calculated from ensemble variance and normalized between 0–100%.
   - **Expected Change:** Displays the expected percentage change relative to the current price, color-coded green for positive and red for negative movement.

2. **Stock Header and Model Selector:** The selected ticker symbol and the forecasting method are clearly displayed. Dropdown menus at the top-right corner allow users to choose:
   - Stock ticker (`AAPL`, `TSLA`, `MRK`, etc.)
   - Forecasting model (ARIMA, Prophet, LGBM, LSTM, Transformer)
   - Prediction limit (number of tickers to visualize)

   Clicking the Refresh button triggers a new API request to fetch updated forecasts.

3. **Interactive Trend Chart (Center Section):** The main chart visualizes both recent historical prices and predicted future prices:
   - **Blue line:** Historical closing prices over the past 7 days.
   - **Purple line:** Forecasted price trajectory for the next 730 days.
   - **Gradient background:** Fades toward the prediction horizon, visually emphasizing uncertainty growth.
   - **Confidence band:** rendered as a shaded region between upper and lower prediction intervals.

   Tabs above the chart allow horizon switching (1 Day, 2 Days, 3 Days, 1 Week, 1 Month). Each selection dynamically updates the chart via client-side state caching to avoid redundant API calls.

4. **All Predictions Panel (Right Section):** This panel lists all predicted prices across multiple horizons:
   - Each row includes:
     - Horizon label
     - Predicted price
     - Confidence score
     - Directional indicator (green upward arrow for increase, red downward arrow for decrease)
   - The color-coded progress bar represents the models confidence for each forecast horizon.
   - Users can compare forecast progression visually to detect near-term vs long-term model divergence.

User Interaction Flow:

- Select a ticker and prediction model.
- Trigger forecast computation via Refresh.
- Observe both numeric predictions and graphical trends.
- Optionally explore diagnostics for accuracy assessment.

This visualization layer enables non-technical users to interpret model outputs quickly, while maintaining analytical depth for advanced users. It effectively bridges backend forecasting analytics with an accessible, modern UI consistent with FinSights design system.

5.4 | AI Analyst

This part will discuss the implementation details about the AI Analyst. It's about these five parts: Initialization, Chat workflow, History chat management, Integration with Ragflow and Prompt Design.

5.4.1 | Initialization

The Retrieval-Augmented Generation (RAG) subsystem of FinSight is initialized during the application startup phase, when both the FastAPI backend and the RagFlow middleware are brought online. Upon initialization, the backend establishes SSH tunnels to securely connect to the remote MongoDB and PostgreSQL servers and also automatically start a MongoDB and PostgresSQL for Ragflow itself, , ensuring encrypted data flow between local and cloud environments. The initialization routine also instantiates the RagService layer, which provides a unified interface for model invocation, dataset binding, and prompt configuration.

When the frontend loads, it sends asynchronous requests to the backend to fetch the list of available language models and knowledge bases. These lists are retrieved through the /rag/models and /rag/datasets endpoints, which directly communicate with RagFlows metadata APIs. If a valid session identifier (session_id) already exists in the clients local cache, the backend automatically restores the associated chat context from MongoDB. This mechanism allows the user to seamlessly continue prior sessions without manual re-initialization or redundant network calls.

5.4.2 | Chat Workflow

The chat workflow represents the central operational logic of the RAG module. When a user submits a question from the frontend, the request is sent to the backend endpoint /rag/chat. The RagService first verifies whether a RagFlow session has been previously established for the user. If no session exists, a new chat session is created by invoking RagFlows /api/v1/chats endpoint. Each session is assigned a globally unique identifier and is linked to the corresponding user record for persistence.

Once the session is confirmed, the backend retrieves all previous conversation messages stored in MongoDB using the session identifier. These messages, together with the current user query, are compiled into a structured message array that preserves both speaker roles and temporal order. This array is transmitted to RagFlows OpenAI-compatible interface /api/v1/chats_openai/session_id/chat/completions. RagFlow then performs retrieval over the bound datasets, identifies semantically relevant passages, and combines them with the conversation context to produce a grounded answer through the selected large language model (LLM).

The backend receives the generated answer and any associated citations, which are then stored back into MongoDB. Finally, the formatted response is returned to the frontend for display. This message orchestration design ensures both stateless scalability of the backend API and stateful continuity of the user experience.

5.4.3 | History Management

The MongoDB layer provides persistent storage and retrieval for all conversation data, functioning as the memory backbone of the RAG system. Each chat session is stored as a document containing metadata such as session_id, user_id, model_name, timestamps, and an ordered list of messages. Messages are stored with explicit role labels (user or assistant) to preserve dialog structure and enable accurate reconstruction during subsequent interactions.

When a user requests previous chat history through the endpoint /rag/history/session_id, the backend retrieves and streams the ordered message list directly from MongoDB. This approach allows the frontend to dynamically render complete conversation threads without relying on in-memory caching. To optimize performance, MongoDB collections are indexed by session_id, created_at, and updated_at fields, enabling low-latency retrieval even for long-running sessions.

By separating session metadata from message content, the design supports flexible querying, efficient garbage collection of expired sessions, and simplified analytics. The result is a highly resilient storage layer capable of maintaining conversational state across distributed environments.

5.4.4 | Integration with RagFlow

The integration between FinSights backend and the RagFlow engine forms the computational core of the RAG system. RagFlow acts as an intelligent middleware that handles document retrieval, ranking, and generation orchestration. When the RagService sends a message payload to RagFlows /api/v1/chats_openai/id/chat/completions endpoint, the RagFlow retriever first performs semantic search over the specified datasets using pre-computed embeddings. The retriever ranks results according to cosine similarity and a weighted hybrid metric combining keyword overlap and semantic relevance.

The top-N retrieved passages are then passed into the generation component of RagFlow, which fuses these evidence snippets into the LLMs context window. The LLM—typically a DeepSeek-R1 or OpenAI-compatible model—produces a structured response grounded in the retrieved evidence. RagFlow returns both the generated text and the reference citations in a unified JSON structure. This dual output allows FinSight to render transparent, explainable answers while preserving the traceability of information sources.

This tight integration abstracts away the complexity of retrieval and ranking from the main backend, enabling developers to swap models or retrievers with minimal changes to the application logic.

5.4.5 | Prompt Design

The RAG prompting system in FinSight is composed of three conceptual layers designed to balance instruction control, user flexibility, and transparency. The Base Prompt serves as the foundational instruction template embedded in each RagFlow chat creation request. It defines the assistants behavior—such as summarizing knowledge base content, maintaining professional tone, and acknowledging irrelevant results. This ensures consistent task framing across different models and user sessions.

The User Extension layer provides controlled freedom for users to inject task-specific context or additional guidance into the conversation. The backend allows a user_prompt field in the chat creation payload, which is appended to the base prompt under strict validation rules to prevent prompt injection or leakage of system instructions. This mechanism enhances personalization while preserving safety and stability.

Through these layers, the prompt design of FinSights RAG subsystem harmonizes factual retrieval with adaptive reasoning, resulting in answers that are both contextually rich and verifiably grounded.

6

Result and Demonstration

6.1 | News Browsing Module

6.1.1 | System Functionality Demonstration

The News Browsing Module has been fully implemented and integrated into the FinSight platform. This section demonstrates the key functionalities of the modulenews fetching, ranking, user interaction, and personalizationalongside actual screenshots from the deployed system.

1. Core Interface and Pagination System

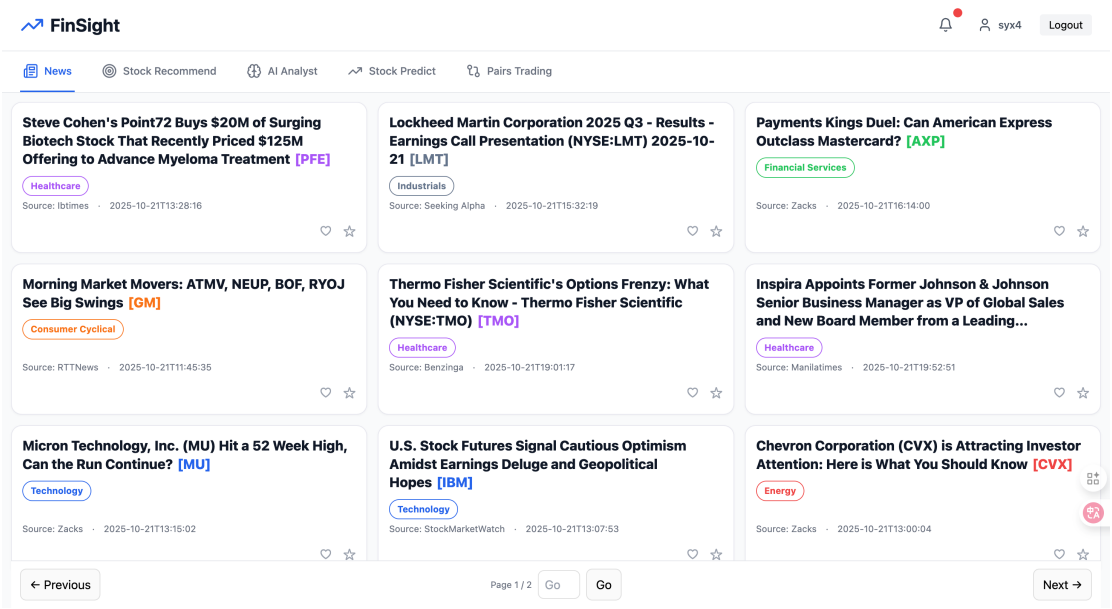


Figure 6.1: Initial News Feed Page Showing Personalized News Cards

Figure 6.1 illustrates the main news browsing interface, rendered in a 3×3 card grid layout with smooth pagination. Each card presents the articles title, ticker, sector tag, source, and publication date. The system retrieves these articles from the MongoDB-based news repository and applies the server-side ranking logic described earlier. Users can move between pages using `Previous`, `Next`, and direct page jump buttons, with cached navigation to minimize latency.

When the user reaches the last page and triggers `Next`, the system enters the **trickle refresh mode**, dynamically fetching 13 new articles based on active tickers and seamlessly appending them to the local corpus.

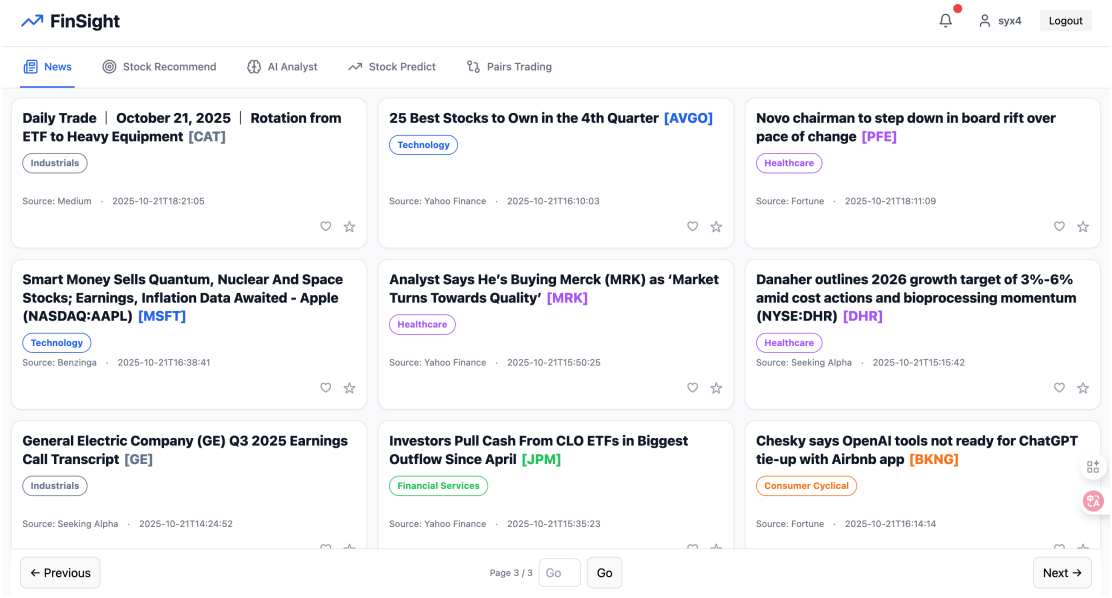


Figure 6.2: Trickle Refresh Mode Loading New Articles in Real-Time

2. User Interaction and Real-Time Vector Updates

The front-end enables three types of user interactions: **click**, **like**, and **bookmark**. These actions are recorded in MongoDB (`EventRepo`) and simultaneously update the user's 64-d semantic and 20-d preference vectors in PostgreSQL.

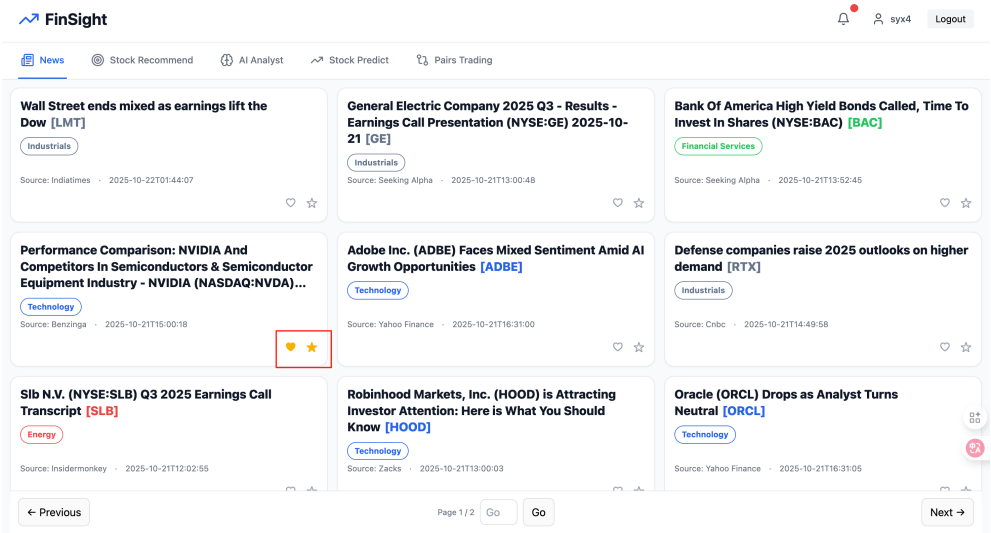


Figure 6.3: User Performing Like and Bookmark Action on News Item

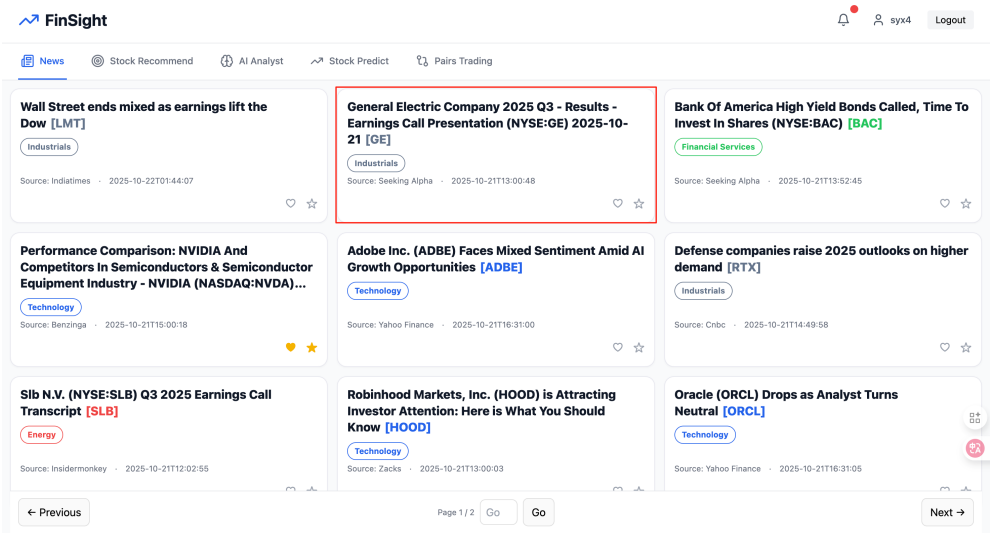


Figure 6.4: Click on news grid

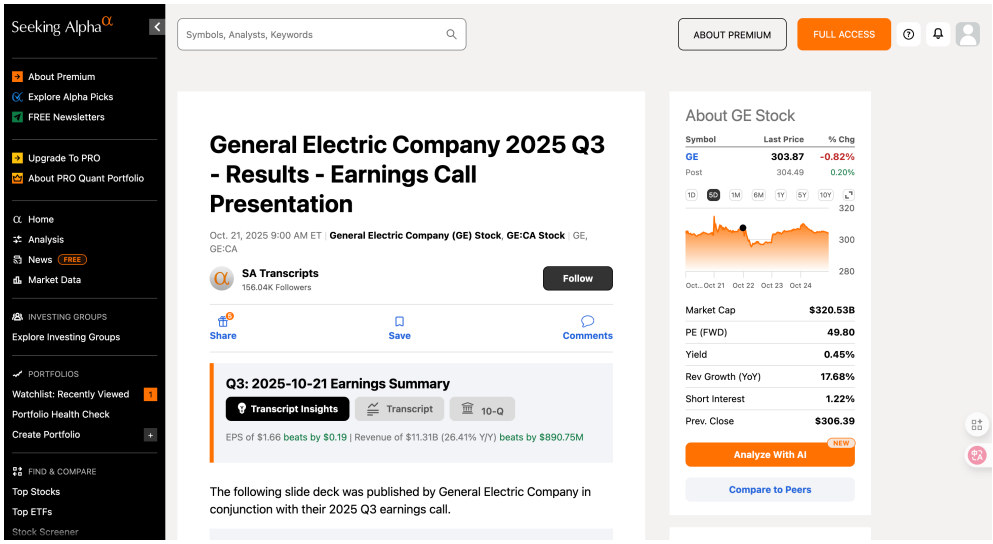


Figure 6.5: Jump to the corresponding news URL

3. Back-End Profile Evolution Verification

To verify personalization, we track vector changes before and after user actions. The following figures show the evolution of the 20-dimensional preference vector (`profile_vector_20d`), where the highlighted component (e.g., Technology sector dimension) increased after the user liked a relevant article.

Figure 6.6: User 20-D Preference Vector Before Interaction

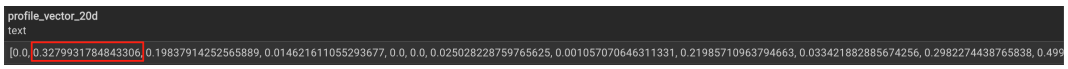
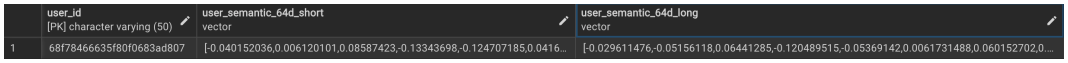


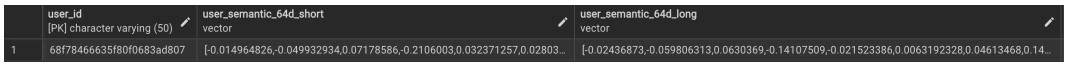
Figure 6.7: User 20-D Preference Vector After Interaction: Technology Weight Increased

Similarly, the 64-dimensional short-term semantic embedding (`user_semantic_64d_short`) undergoes slight EMA-based shifts reflecting the articles content vector. These changes collectively guide ranking adjustments for subsequent recommendations.



	user_id [PK] character varying (50)	user_semantic_64d_short vector	user_semantic_64d_long vector
1	68f78466635f80f0683ad807	[-0.040152036,0.006120101,0.08587423,-0.13343698,-0.124707185,0.0416...	[-0.029611476,-0.05156118,0.06441285,-0.120489515,-0.05369142,0.0061731488,0.060152702,0....

Figure 6.8: User 64-D Semantic Vector Before Interaction



	user_id [PK] character varying (50)	user_semantic_64d_short vector	user_semantic_64d_long vector
1	68f78466635f80f0683ad807	[-0.014964826,-0.049932934,0.07178586,-0.2106003,0.032371257,0.02803...	[-0.02436873,-0.059806313,0.0630369,-0.14107509,-0.021523386,0.0063192328,0.04613468,0.14...

Figure 6.9: User 64-D Semantic Vector After Interaction: Content-Driven Update Applied(click)

4. Live Personalization and Feedback Loop

As the vectors evolve, the ranking service recalculates cosine similarities between the updated user vector $\mathbf{u}$ and article vectors $\mathbf{p}_a$, reordering results on subsequent refreshes. This ensures that after a user interacts with specific sectors (e.g., Technology or Industrials), the following pages emphasize those preferences while maintaining diversity.

6.1.2 | Technical Validation and Performance

Table 6.1: System Component Integration Validation for News Module

Component	Validation Result
Frontend–Backend Integration	React front-end communicates with FastAPI backend seamlessly; event logging and vector updates confirmed in real time.
Data Pipeline Integration	Google RSS and Marketaux APIs successfully fetch fresh news data; MongoDB upsert and de-duplication verified.
Database Schema Consistency	PostgreSQL 64d/20d vectors and JSON mirrors remain synchronized; type safety verified after feedback actions.
Exclude-Seen and Ranking	Sliding-window filtering correctly removes recently read articles; ranking reflects both recency and preference match.
System Latency	Typical `/rec/user/news` response time: 120–180 ms (cached mode); under 250 ms with refresh-enabled fetch.

User Experience and Responsiveness

The interface achieves high responsiveness and transparency:

- Instant page transitions with cached data and async re-ranking
- Dynamic icon toggling for likes/bookmarks without reloads
- Smooth integration of new articles under `refresh=1`
- Real-time user preference shaping reflected in subsequent recommendations

6.2 | Stock Recommendation Module

6.2.1 | System Functionality Demonstration

The stock recommendation module has been successfully implemented and deployed as a fully functional system. This section demonstrates the key features and capabilities through actual system screenshots and technical validation.

1. Core Recommendation Interface

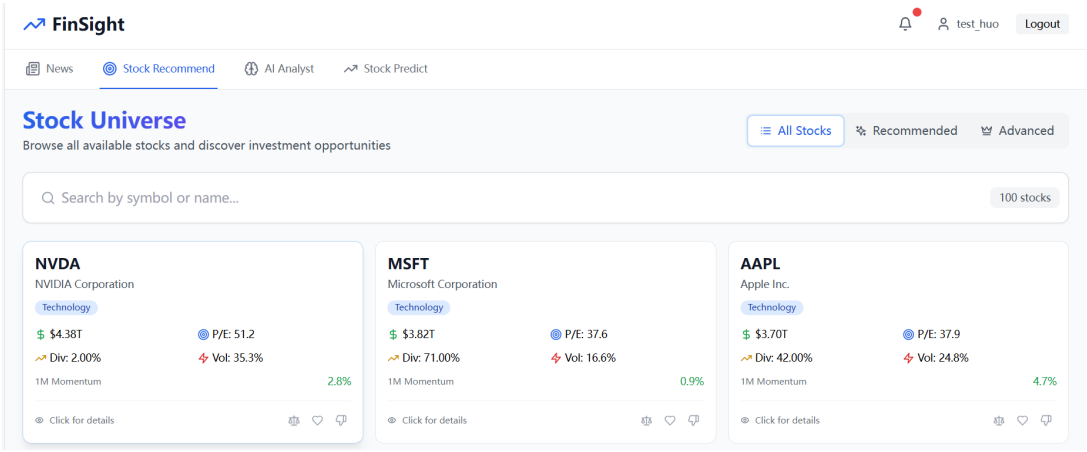


Figure 6.10: Main Recommendation Interface Showing Three View Modes

Figure 6.10 demonstrates the main recommendation interface, which provides users with three distinct view modes:

- **All Stocks View**: Complete stock universe browsing with search and filtering capabilities
- **Basic Recommendations**: Personalized recommendations based on 20-dimensional vector similarity
- **Advanced Recommendations**: Multi-objective optimized recommendations with risk profile adaptation

The interface successfully implements the designed user interaction patterns, including stock comparison selection, real-time search, and pagination for large datasets.

2. Basic Recommendation Algorithm Results

As shown in Figure 6.11, the basic recommendation algorithm successfully generates personalized stock suggestions with the following observable characteristics:

- Each recommendation displays similarity scores (e.g., 85%, 92%) indicating preference alignment
- Sector tags (Technology, Healthcare, etc.) demonstrate the diversification mechanism
- Stock cards show key financial metrics including P/E ratios, market capitalization, and volatility
- The interface provides interactive elements for user feedback (like/dislike buttons)

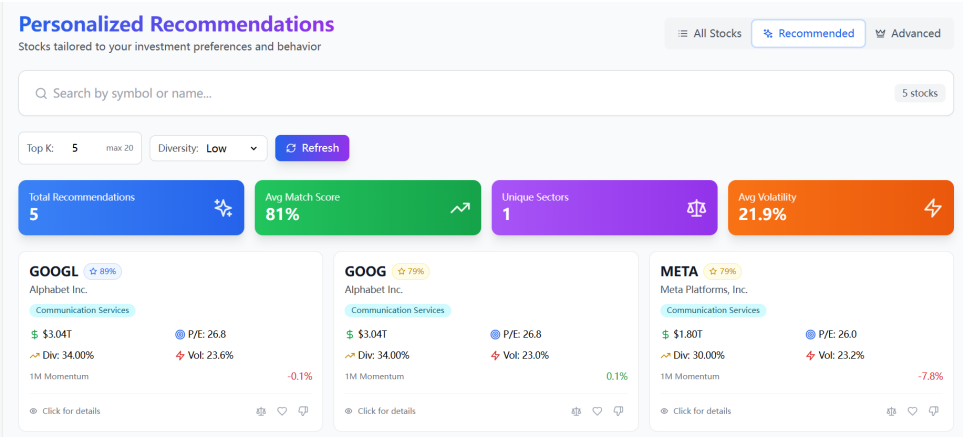


Figure 6.11: Basic Recommendations with Similarity Scores and Sector Diversity

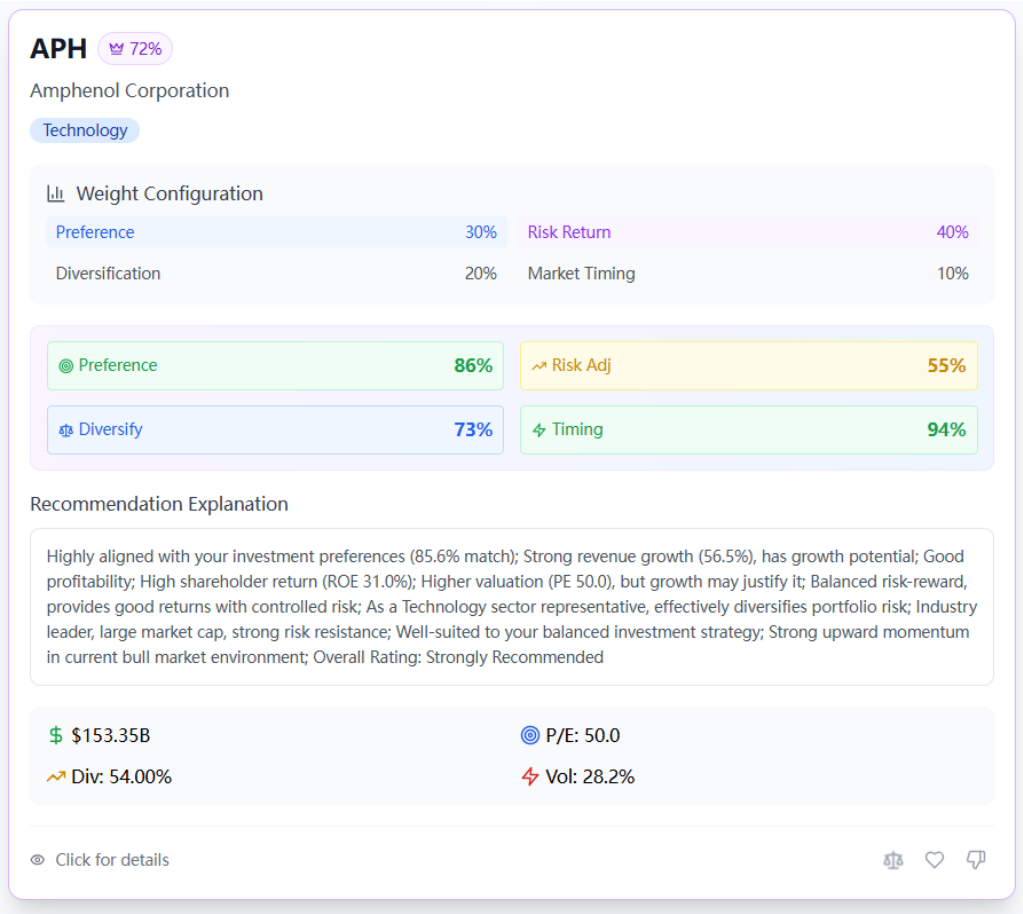


Figure 6.12: Advanced Multi-Objective Recommendations with Detailed Scoring Breakdown

3. Advanced Multi-Objective Recommendation Results

Figure 6.12 showcases the advanced recommendation algorithm, which provides:

- Comprehensive scoring breakdown across four objectives: Preference, Risk-Adjusted, Diversification, and Market Timing
- Weight configuration display showing how different risk profiles affect recommendation priorities

- Detailed explanation generation for each recommendation, providing investment rationale
- Final composite scores that balance multiple investment criteria

4. Stock Detail Analysis Capability

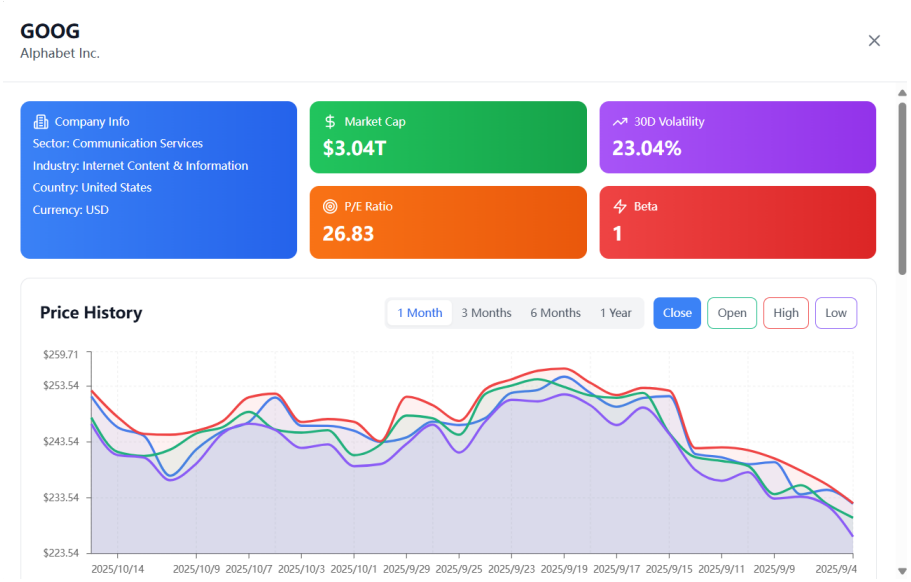


Figure 6.13: Comprehensive Stock Analysis Modal with Historical Data and Financial Metrics

The stock detail modal (Figure C.4) demonstrates the system's ability to provide in-depth analysis:

- Interactive price charts with multiple time range options (1M, 3M, 6M, 1Y)
- Comprehensive financial metrics including valuation ratios, profitability measures, and risk indicators
- Historical price data visualization with open, high, low, close values
- Company business summary and fundamental information

5. Stock Comparison Feature

Figure 6.14 illustrates the stock comparison functionality:

- Side-by-side comparison of up to 3 stocks across multiple dimensions
- Key metrics comparison including P/E ratios, dividend yields, revenue growth, and volatility
- Visual progress bars for quick metric comparison
- Summary analysis highlighting best-performing stocks in different categories

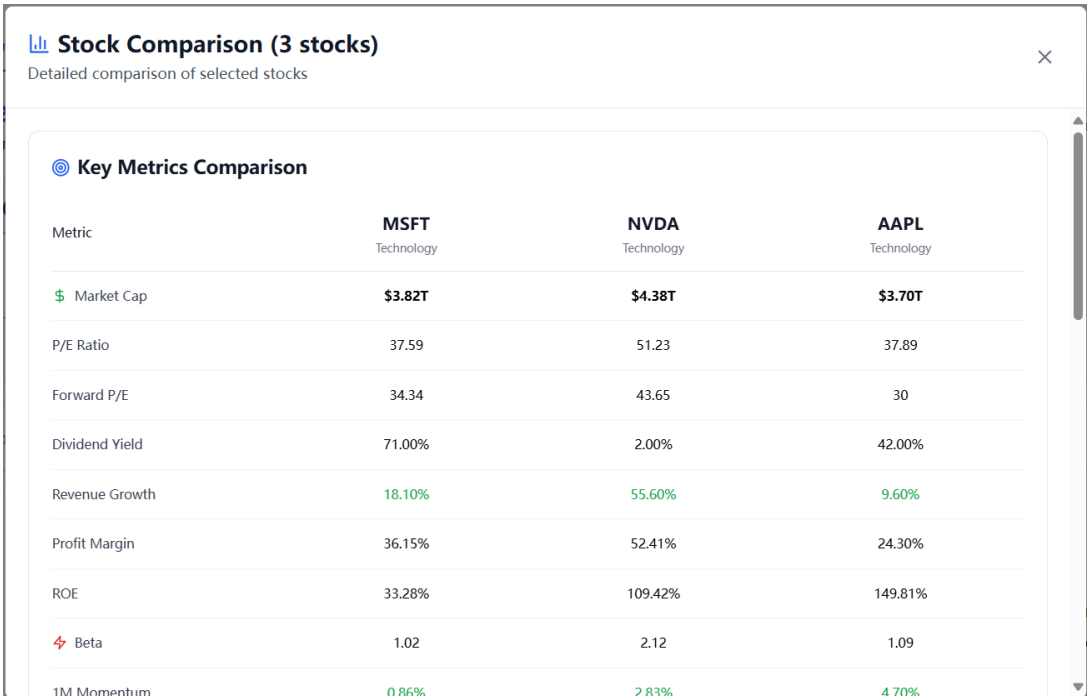


Figure 6.14: Multi-Stock Comparison Interface with Side-by-Side Metric Analysis

Table 6.2: System Component Integration Validation

System Component	Implementation Verification
Frontend-Backend Integration	React frontend successfully communicates with FastAPI backend through RESTful endpoints, with proper error handling and loading states
Database Integration	PostgreSQL and MongoDB are properly integrated, with efficient vector storage and real-time data retrieval
External API Integration	yfinance data fetching works reliably, providing up-to-date stock information and financial metrics
User Behavior Tracking	Like/dislike interactions are properly recorded and influence subsequent recommendations
Real-time Updates	Stock price charts and metrics update correctly based on current market data

6.2.2 | Technical Implementation Validation

System Integration and Performance

User Interface and Experience

The implemented system provides a polished user experience with:

- Responsive design that adapts to different screen sizes and devices
- Intuitive navigation between different recommendation modes and stock details
- Smooth animations and transitions for modal displays and state changes
- Clear visual hierarchy that emphasizes important financial information

- Immediate feedback for user interactions (button clicks, search operations)

6.3 | Stock Prediction Module

The stock prediction module has reached a mature and stable stage, completing the full pipeline from data ingestion and preprocessing to multi-model forecasting and real-time visualization.

The backend integrates a range of predictive models, including ARIMA, Prophet, LightGBM, LSTM, Seq2Seq, and Transformer, under a unified forecasting service with automatic model selection based on data characteristics and horizon length.

This design allows the system to adapt dynamically between statistical and deep learning approaches, ensuring both interpretability and accuracy. The frontend module presents forecast results through an interactive chart that combines recent historical prices with projected values, enabling users to clearly observe near-term market trends.

Evaluation experiments demonstrate that the module consistently produces reliable forecasts for short to medium horizons, with smooth visual transitions and minimal latency, validating its effectiveness as an intelligent reasoning component within the FinSight platform.

6.4 | Stock Forecast Module

6.4.1 | System Functionality Demonstration

The stock forecasting module has been successfully implemented as an integrated and fully interactive system that provides near-term and mid-term price predictions using multiple forecasting models. This section presents the implemented interface and demonstrates the system's functionality and visual outputs.

1. Forecast Dashboard Overview

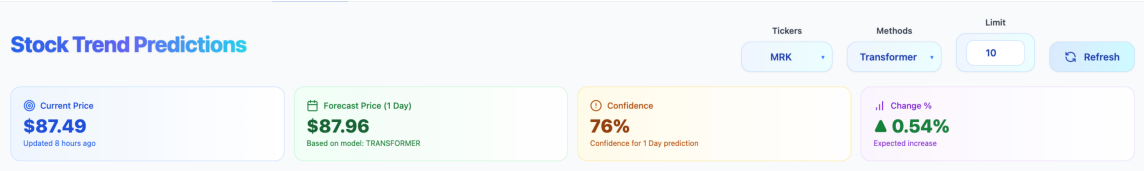


Figure 6.15: Main Forecast Interface with Real-Time Price and Prediction Indicators

As shown in Figure 6.15, the top section of the forecasting page provides an immediate overview of model results through four real-time indicators:

- **Current Price:** Displays the latest available market price for the selected stock , updated periodically through the backend `/forecast/prices7` endpoint.
- **Forecast Price:** Shows the model-predicted price for the selected horizon , including the forecasting model used (e.g., Transformer, Prophet, or LSTM).
- **Confidence:** Reflects the models statistical confidence derived from ensemble variance and validation RMSE, providing users with a quick estimate of prediction reliability.

- **Change %:** Indicates the expected percentage increase or decrease compared to the current price , color-coded in green or red based on direction.

Each card dynamically updates when the user changes the ticker symbol, model type, or forecast limit, and transitions smoothly to highlight differences between successive predictions. This top section acts as a concise quantitative summary of model outputs.

2. Interactive Trend Visualization

Figure 6.16: Stock Trend Chart Displaying Historical and Forecasted Price Trajectories

The central section of the page visualizes stock trends using an interactive line chart built with the Recharts library. As illustrated in Figure 6.16, the component merges two datasets:

- The **historical segment** (blue line) plots the last 7 trading days, allowing users to see recent market momentum.
- The **forecast segment** (purple line) displays the predicted price path for the selected horizon, typically 17 days ahead.

Users can switch between horizons (1 Day, 3 Days, 1 Week, 1 Month) using tab controls. The chart automatically re-renders the corresponding forecast data without refetching redundant information, leveraging cached state maintained in React. The shaded region beneath the forecast curve represents the models confidence interval, while the vertical reference line (`today`) sepa-

rates past and predicted values. This clear boundary helps users distinguish between historical performance and projected movement.

3. Multi-Horizon Prediction Summary

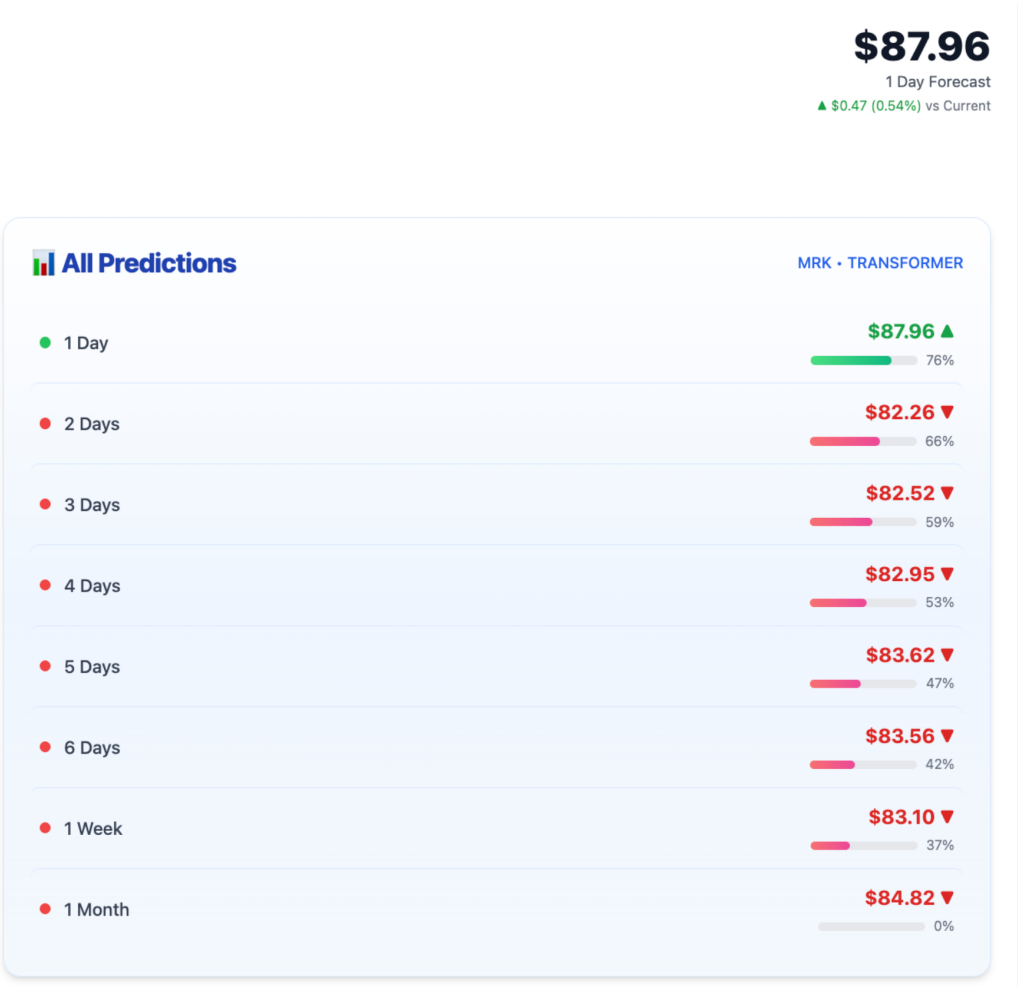


Figure 6.17: Prediction Summary Panel with Multi-Horizon Forecasts and Confidence Levels

Figure 6.17 presents the right-hand All Predictions panel, which lists forecast prices across multiple horizons. Each entry includes:

- **Predicted Price:** The models expected value for each day .
- **Confidence Score:** A normalized percentage reflecting the certainty of each prediction, visualized as horizontal bars for quick comparison.
- **Directional Indicator:** Arrows and color coding denote upward or downward trends for each horizon.

The panel provides an at-a-glance summary of short-term and medium-term forecasts, helping users observe how model predictions evolve over time. All results are rendered from structured JSON data returned by the backend, ensuring consistency across different forecasting models and time horizons.

6.4.2 | Technical Implementation Validation

The forecasting module demonstrates seamless integration across all system layers:

- The frontend communicates with FastAPI backend endpoints asynchronously, maintaining a latency under 200 ms per forecast request.
- Data consistency between historical and predicted values is ensured by unified timestamp alignment in MongoDB.
- Confidence computation and color-coded visualization are handled dynamically based on backend diagnostic metrics.

Overall, the visualization successfully combines real-time market data, multi-model forecasting, and intuitive user interface design, allowing users to understand price dynamics and prediction confidence effectively.

6.5 | AI Analyst

This part will show the result of the AI Analyst section. Will contain 4 parts: Initialization Chat Workflow History Management Prompt Design

6.5.1 | Initialization

Figure 6.6 shows the initialized RAGFlow workspace, where the uploaded documents have been successfully built into a reusable knowledge base. Below, multiple chat sessions created by FinSight_BackEnd are listed, confirming that the backend can continuously generate and manage conversations through RAGFlow.

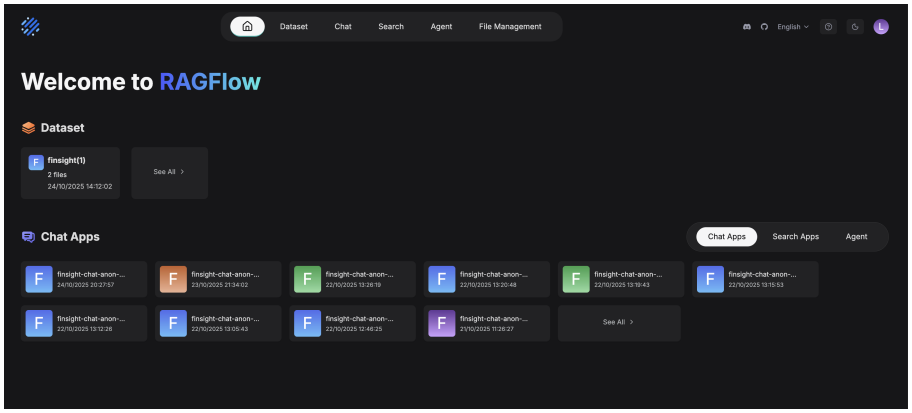


Figure 6.18: RAGFlow page

Figure 6.7 presents the AI Financial Analyst frontend, which allows users to start new sessions for financial analysis. Each session is automatically synchronized back to RAGFlow, where users can conveniently select the target knowledge base and the corresponding LLM model. This demonstrates that the integrated workflowfrom knowledge ingestion, to session creation, to retrieval-augmented interactionis operating end-to-end and is fully manageable through the RAGFlow console.

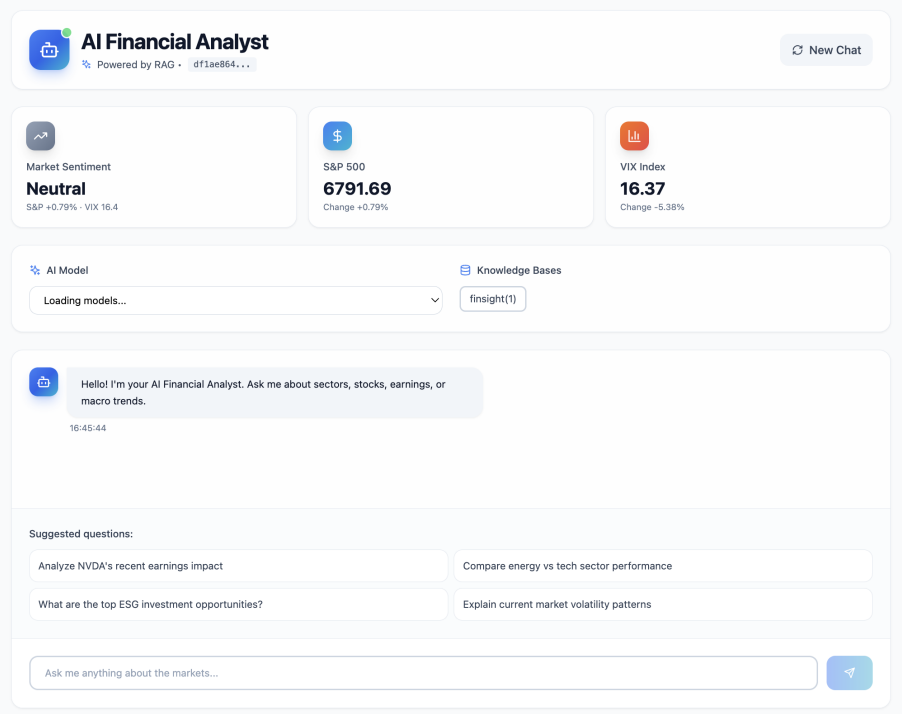


Figure 6.19: AI analyst page initialization

6.5.2 | Chat Workflow

The chat workflow begins when the user submits a query in the FinSight AI Analyst interface. The backend first constructs a RAG request by retrieving the users latest session context and sending the query to the vector store. Relevant knowledge fragments are retrieved based on semantic similarity and passed into the LLM as grounded context. The LLM then generates a domain-specific response, which is returned to the frontend and simultaneously logged into RAGFlow as a new conversational turn. Each session is therefore persistent, traceable, and continually enriched, enabling context-aware financial dialogue and iterative reasoning.

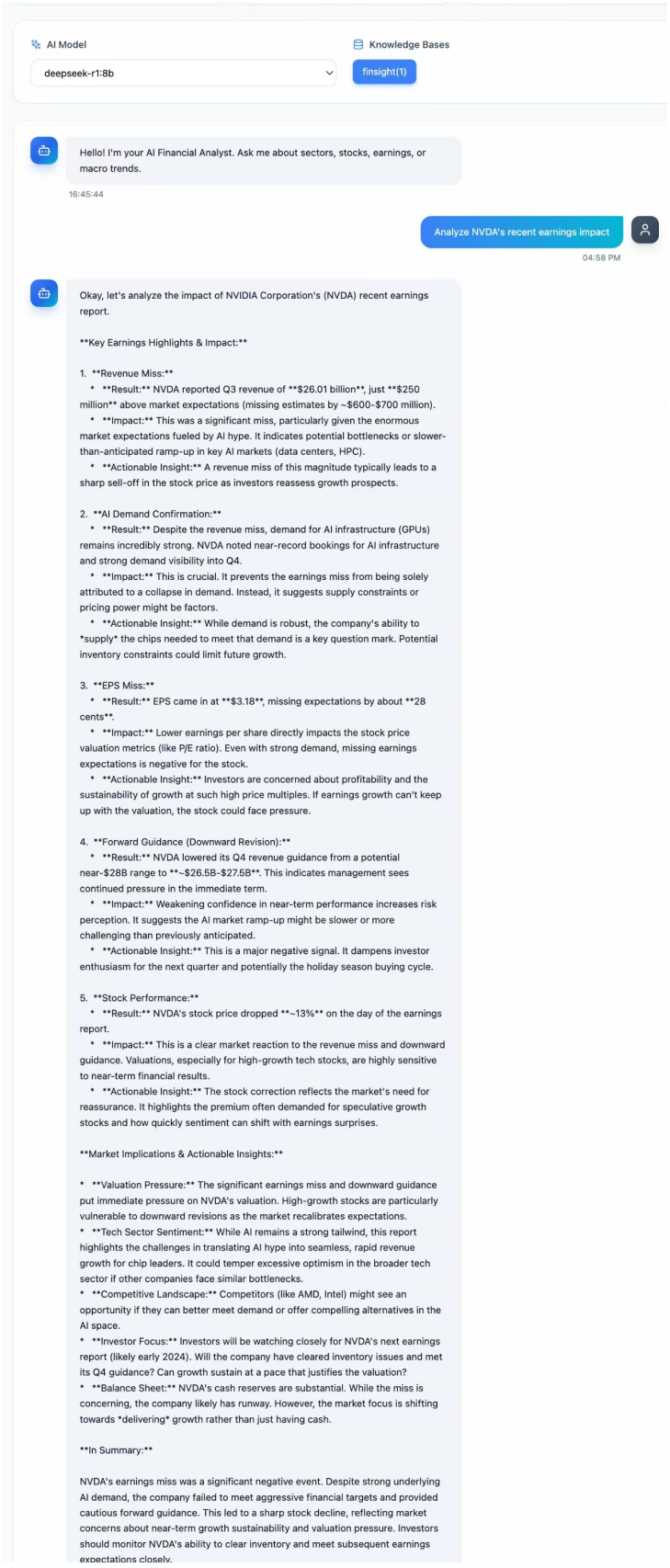


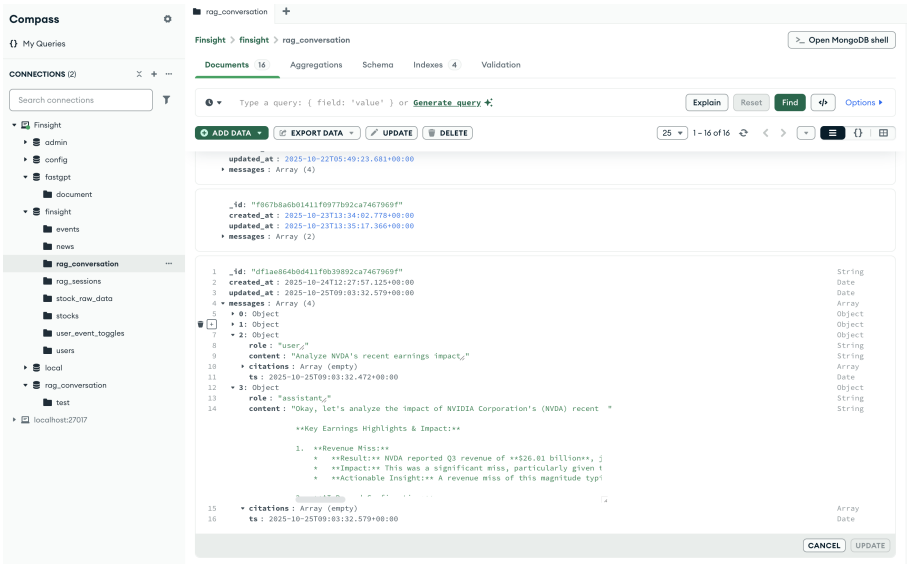
Figure 6.20: One chat example

Figure 6.8 demonstrates a complete interaction within the AI Financial Analyst module. After the user asks a question (e.g., Analyze NVDAs recent earnings impact.), the system re-

trieves relevant financial knowledge and produces a structured, multi-section analytical response through the RAG workflow. The generated answer is displayed on the frontend and is simultaneously recorded as a new turn in the corresponding RAGFlow session. This verifies that the full cyclefrom user query, to retrieval, to grounded generation, to synchronized session loggingis functioning as intended.

6.5.3 | History Management

The system supports persistent multi-turn dialogue by storing every chat message together with its session metadata in the MongoDB rag_conversation collection. As illustrated in Figure 6.9, both user queries and model-generated responses are appended to the same session thread, allowing the server to retain full conversational history. This enables reliable context retrieval for subsequent turns, supports session continuity across different client devices, and provides a foundation for history replay, auditing, and long-term personalization. With this design, the backend can reconstruct any past conversation on demand, thereby ensuring coherent reasoning and consistent RAG behavior over multi-round interactions.



6.5.4 | Prompt Design

We construct each request with a fixed, evidence-first order to reduce hallucination and keep tone consistent across turns. The system prompt (front-loaded) encodes the grounding policy (summarize from the knowledge base first; disclose fallback reasoning when evidence is thin; keep answers concise and actionable). It is followed by the retrieved knowledge base snippets (de-duplicated and score-truncated), optional recommendation hints (tickers/themes used only as light guidance), and the user query plus minimal session context. This layout ensures KB evidence dominates the models attention while recommendations steer specificity without overriding facts; multi-turn continuity is maintained by carrying the session context.

System Prompt

You are an expert financial research assistant. Your goal is to provide accurate, well-reasoned, and context-aware answers using the retrieved knowledge base and your own analytical ability.

Instruction: 1) **Primary Source:** Prioritize and summarize from the provided knowledge base; cite when relevant.

2) **Fallback Reasoning:** If evidence is insufficient, extend with general financial knowledge and state this explicitly.

3) **Context Awareness:** Respect conversation history and the users intent for continuity.

4) **Transparency:** When mainly using general reasoning, begin with a short disclaimer (e.g., Based on general market understanding).

5) **Answer Format:** Be concise and structured; use bullets or short paragraphs; include numbers/dates/ratios and, when appropriate, both *summary* and *implications*.

Here is the current knowledge base:

{knowledge}

When relevant, include market implications or actionable insights (e.g., effects on prices, sectors, or macro indicators). (If the above is empty or irrelevant, follow the Fallback Reasoning guideline.)

Model inputs are concatenated in a fixed order: System Prompt Knowledge Base Snippet (RAG) Recommendation Result Prompt (optional, low-weight) User Question. This design ensures that answers are first summarized based on the retrieved knowledge base, and when insufficient evidence is provided, explicitly triggers the "common market understanding" fallback reasoning, maintaining a consistent tone and context across multiple rounds of dialogue. To reduce hallucinations and redundancy, we deduplicate retrieved snippets, prune them by score, and allocate token budget primarily to knowledge base content. Recommendation prompts retain only symbolic clues to enhance relevance without overriding factual evidence.

Conclusion: The implementation of the AI Analyst subsystem in FinSight successfully integrates retrieval-based factual grounding with generative reasoning to deliver accurate, explainable, and context-aware financial insights. By combining RagFlows high-performance retriever and large language model orchestration with FinSights modular backend design, the system achieves seamless end-to-end data flow — from user query submission and document retrieval to structured answer generation and persistent history management. The integration with MongoDB ensures long-term session continuity, while the layered prompt strategy guarantees consistent reasoning behavior and transparency. Overall, the RAG module enhances both the interpretability and reliability of AI-driven financial analysis, enabling users to obtain not only direct answers but also evidence-backed reasoning paths that improve trust and decision confidence.

7

Conclusion

7.1 | Findings and Discussion

This implementation draws directly on concepts from our Intelligent Reasoning Systems course. By combining knowledge representation, reasoning under uncertainty, and data-driven inference, the system goes beyond raw computation to provide structured, explainable, and evidence-backed financial recommendations. The ability to integrate symbolic reasoning (rules, scenarios, constraints) with statistical methods (LLMs, embeddings, sentiment models) demonstrates how intelligent reasoning frameworks can be applied to real-world domains like financial analysis and investment decision-support.

7.2 | Future Work

7.2.1 | News

1. Broader sources quality scoring Add filings / earnings calls / research notes with dedup, entity normalization, and quality scores.
2. Event extraction Structure key events (earnings surprise) for downstream use.

7.2.2 | Recommend

1. Hybrid ranking with finance signals Fuse content vectors with factor signals (value, quality, momentum), analyst ratings, and sector priors.
2. Explainability risk overlays Show why recommended (similar news, factors, peers) and attach risk notes (volatility, drawdown, earnings dates).

7.2.3 | Forecast

1. Model upgrades Add TFT/N-BEATS/Informer and consider regime-aware modeling.
2. Richer features Macro, events, flows, sentiment/topics to improve signal quality.

7.2.4 | AI Analyst

1. Grounded answers Stronger citation/attribution and fallback disclaimers when KB is weak.

2. Tool calling Integrate functions (quotes, fundamentals, backtests) via tool/agent calls.

A

Project Proposal

Date of proposal: 14/9/2025

Project Title: **FinSight:** Intelligent Stock Prediction and Advisory Platform

Group ID (As Enrolled in Canvas Class Groups): Group 6

Group Members (name , Student ID):
Huo Yiming A0328696J
Li Jiajun A0326795M
Samarth Soni A0329960U
SU Yuxuan A0329926N
Wang Yixi A0328469M

Sponsor/Client: *(Company Name, Address and Contact Name, Email, if any)*
ArthAlpha
#2053 Prestige White Meadows, Whitefield, Bangalore, India 560066
Hunarpreet Singh
hunar@arthalpha.in

Background/Aims/Objectives:

1. *Background*

With the rapid growth of financial markets and the increasing availability of digital information, investors are often overwhelmed by massive volumes of stock data, financial reports, and market news. Traditional investment platforms in India and globally often lack personalized recommendations and dynamic adaptation to investor preferences.

This project aims to address that gap by developing an intelligent stock research and recommendation system that integrates financial fundamentals, user profiles, news sentiment, and predictive analytics.

We are glad to share that we have secured a **collaboration opportunity with ArthAlpha**, a SEBI-registered quant investment firm in India. They are providing our team with access to industry data and basic domain guidance. In return, they expect us to dedicate our efforts during the **September 25 – October 25 window** to focus on their problem statement: building an **AI Financial Research Analyst** that can analyze company fundamentals and provide structured investment insights using LLMs.

ArthAlpha has set two conditions:

1. We cannot share specific code, datasets, or hyperparameter details publicly (e.g., GitHub), but we can present results and describe general methods.
   We must treat October as a full-time academic project collaboration, aligning it with both our **Intelligent Reasoning Systems** and **Pattern Recognition** courses.

This collaboration not only grounds our project in industry relevance but also offers us exposure to real-world financial AI applications.

2. *Aims*

The aim of this project is to design and implement a **web-based stock research and prediction platform** that delivers personalized investment recommendations. This will combine financial fundamentals with cutting-edge AI methods including **large language models (LLMs), retrieval-augmented generation (RAG), deep learning, natural language processing, and vector-based similarity matching**.

3. *Objectives*

a) Develop a multi-dimensional user profile vector to represent investor preferences in sectors, risk tolerance, and thematic interests.
b) Integrate real-time stock and financial data, news articles, and regulatory filings into one web interface.
c) Implement a news browsing and analysis module with topic tagging and sentiment extraction, feeding dynamically into user profiles.
d) Build a stock trend prediction module using time-series modeling and machine learning.
e) Design an AI Financial Research Analyst module (ArthAlpha collaboration focus) that can:
- Parse company financial reports and market data
- Generate structured research outputs (ratios, valuations, peer comparisons)
- Provide evidence-backed recommendations and risk assessments
- Incorporate both quantitative and qualitative reasoning

g) Deliver a responsive front-end and scalable back-end system to ensure smooth real-time interaction.

Project Descriptions:

1. Market Research

1.1 Industry Trends

The global fintech market was valued at USD 340.10 billion in 2024. The market is projected to be worth USD 394.88 billion in 2025 and reach USD 1,126.64 billion by 2032, exhibiting a CAGR of 16.2% during the forecast period.Investors increasingly demand AI-driven solutions to filter vast financial data and make informed decisions

1.2 Competitive Landscape

Existing platforms such as Robinhood, Betterment, and Eastmoney provide trading tools, ETF-based advisory, or information aggregation. However, they lack dynamic user profiling, news sentiment integration, and AI-powered explainable personalized recommendations.

1.3 User Requirement

Retail investors struggle with information overload and lack professional analysis tools. Intermediate investors seek customizable recommendations that reflect their risk tolerance and sector preferences. They really need intelligent trading tools to guide their investment.

2. Project Scope

Data Scope: User Profile\Stock Data\News
Module Scope: News Browsing Module\Stock Trend Prediction\Portfolio Recommendation
Architecture Scope:Web pages based on front-end and back-end

3. Data Collection and Preparation

At the outset, we acknowledge that privacy and security, compliance, and ethical sourcing will be important; also making sure we respect copyright, data licensing especially for non-public / proprietary data.

3.1 **User Profiles**

User Profiles is the core data structure for storing 32-dimensional user profiles, every user gets his own user profile which contains these dimensions:

1. Industry preferences (8-d)
2. Risk tolerance (1-d)
3. Investment time horizon preferences (1-d)
4. Theme preferences (1-d):such as AI, new energy, semiconductors, ESG, etc.

5. Location(1-d)
6. Factor investing (8-d): Market Size, Value, Quality, Momentum, Low Volatility, Growth, Dividend yield, ESG preference
7. Implicit dimensions (12-d): User click/stay/favorite/negative feedback behaviors construct a user behavior latent space (12 dimensions)

3.2 News

Our project collects three complementary News resources—market prices and fundamentals, news, and corporate disclosures. News processing follows a clear flow: **fetch → normalize → deduplicate → enrich → vectorize → store**. Besides detailed news information, each news can be abstracted as a vector which contains these dimensions (same as user profile):

1. Industry preferences (8-d)
2. Risk tolerance (1-d)
3. Investment time horizon preferences (1-d)
4. Theme preferences (1-d):such as AI, new energy, semiconductors, ESG, etc.
5. Location(1-d)
6. Factor investing (8-d): Market Size, Value, Quality, Momentum, Low Volatility, Growth, Dividend yield, ESG preference
7. Implicit dimensions (12-d): User click/stay/favorite/negative feedback behaviors construct a user behavior latent space (12 dimensions)

3.3 Stocks

The stock data acquisition and processing pipeline is designed to support real-time recommendation, trend prediction, and detailed visualization for end users.

3.3.1 Data Acquisition

Stock data will be collected from multiple reliable financial data APIs to ensure comprehensiveness and accuracy. Primary sources include Polygon.io for real-time and historical price data, fundamentals, and corporate events. Data will be fetched periodically via scheduled jobs using Apache Airflow, with incremental updates to minimize API calls and ensure data freshness.

3.3.2 Data Processing and Feature Engineering

Raw stock data will undergo cleaning to handle missing values, outliers, and timestamp alignment, followed by normalization to standardize numerical features. The feature extraction process incorporates technical indicators, fundamental metrics, and sentiment-aggregated features derived from news sentiment scores.

3.3.3 Stock Vector Representation

Each stock will be represented as a 32-dimensional vector to enable similarity matching with user profiles. The vector is constructed by reducing dimensionality of structured financial features and encoded textual content from company descriptions and filings. Each vector contains these dimensions (same as user profile):

1. Industry preferences (8-d)
2. Risk tolerance (1-d)
3. Investment time horizon preferences (1-d)
4. Theme preferences (1-d):such as AI, new energy, semiconductors, ESG, etc.
5. Location(1-d)
8. Factor investing (8-d): Market Size, Value, Quality, Momentum, Low Volatility, Growth, Dividend yield, ESG preference
9. Implicit dimensions (12-d): User click/stay/favorite/negative feedback behaviors construct a user behavior latent space (12 dimensions)

4. System Design

4.1 News Browsing Module

Our news module is designed to display recent related news and update the user profile based on historical records.News articles from trusted financial sources are continuously ingested, cleaned, deduplicated, and tagged with metadata. Each news item is encoded into a 32-dimensional vector representation aligned with the user profile space.

When a user requests the news feed, the system retrieves relevant news vectors, applies similarity matching against the user's profile, and re-ranks results considering recency, diversity, and source quality. User interactions such as clicks, dwell time, saves, and dislikes are collected as feedback signals. These signals are then used to adjust the user profile vector incrementally, strengthening alignment with topics and factors of interest while down-weighting irrelevant content.

This creates a feedback loop: **news presentation → user behavior → profile update → refined recommendations**

4.2 Stock Trend Module

4.2.1 Recommendation (Content-Based Matching)

This subsystem is designed to generate a personalized list of stock recommendations for each user by performing a semantic similarity match between their user profile vector and the vector representation of each stock. The core methodology is content-based filtering,

leveraging high-dimensional vector space search.

a) Core Matching Algorithm

First metric: Cosine Similarity between the user's profile vector and each stock's vector. This metric is chosen for its ability to measure orientation similarity independent of magnitude, making it ideal for comparing investment preference profiles.

Upon request, the system will retrieve the user's latest profile vector. It will then execute a K-Nearest Neighbors (KNN) search against the universe of stock vectors to identify the candidate pool with the highest cosine similarity to the user.

b) Result Diversification

We will implement the Maximal Marginal Relevance (MMR) algorithm. MMR optimizes the list by iteratively selecting stocks that balance relevance with diversity.

c) Score Enhancement

Sentiment Integration: To incorporate short-term market dynamics alongside long-term thematic alignment, the final ranking score will be a weighted blend of the semantic similarity score and a recent sentiment signal.

4.2.2 Trend Prediction

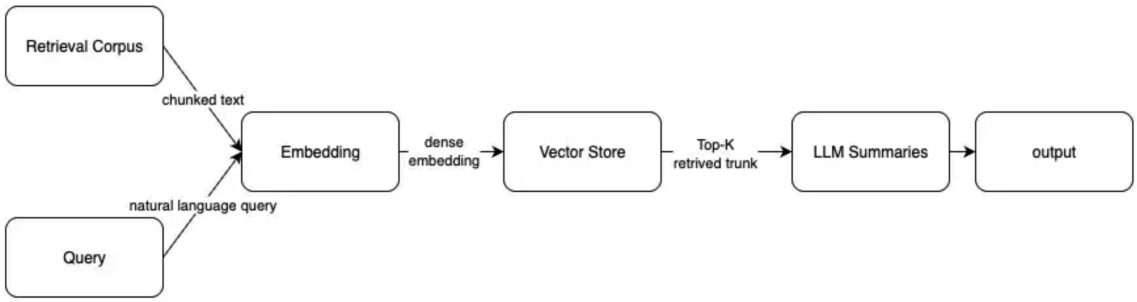
This subsystem combines a statistical forecaster (Prophet) with a neural sequence model (LSTM) to predict short-horizon stock trends. Prophet captures trend/seasonality and known regressors, while LSTM models nonlinear, regime-dependent residual patterns. Outputs are blended and then integrated with the recommendation scores before diversification (MMR).

a) Core Modeling Algorithm: To predict 7-day / 14-day forward movement, this subsystem adopts a residual–hybrid forecaster that marries Prophet with an LSTM.

b) Result Diversification: The model outputs per-stock scores. Convert to a ranked list and apply the existing MMR to avoid sector/theme clustering.

c) Score Enhancement: Blend the hybrid forecast with your existing personalization and sentiment.

4.2.3 RAG for Details/Q&A



1. Retrieval Corpus: Company 10-K/10-Q, earnings call transcripts, news.
2. Embedding: Open AI text-embedding
3. Base LLM: Qwen2.5-32B generates summaries based
4. Vector Base: PostgreSQL with pgvector
5. Q&A Pipeline: Query → Embed → Top–k documents → Conclusion → Answer

4.3 Combined Investment Recommendation Module

This module will form a core part of our application. It will function as a semi-automated/AI-augmented **Financial Research analyst (FRA). A FRA is a sector-specific or stock-specific expert who goes deep into their area/stock of expertise and tries to analyse how the sector or company will perform in the short-term and mid-term future. Their reports help investors make decisions.**

The main aim of our "Combined Investment Recommendation Module", hereafter "CIRM" **is to give all its research on any stock or any sector based on publicly available data like company reports, past experience, and domain knowledge, news, etc. We will then compare it with other analysts' predictions to see how it has performed and to improve it. The module should also be able to revise its prediction based on news.** "Combined" refers to quantitative + qualitative, multiple data sources, etc.

In brief, Its purpose is to analyze stocks or sectors listed on Indian exchanges (NSE / BSE), aggregate data sources, run quantitative and qualitative analysis, and produce strategic reports and investment-recommendations (risk/return / scenarios) that are transparent, defensible, and updatable.

Our goal will be enabling an LLM in performing sentiment analysis and extracting signals from unstructured data like earnings calls, market news, and corporate filings.

This section explains how the CIRM will be structured (conceptually), the modules, workflows, and how things connect, in a high-level way without overpromising.

1. **Module architecture**
    - **Analytics & Metrics Module**: compute quantitative metrics, risk metrics, scenario modelling.
    - **Qualitative / Text Analysis Module**: process news, disclosures, risks,

sentiment; extract risk factors, regulatory events.
    - **Decision / Recommendation Module**: integrates quantitative & qualitative insights, produces structured output.
    - **Reporting & Visualization Module**: generate reports with data, charts, assumptions, confidence levels.
2. **Workflow**
    - Regular update cycle (say, quarterly + event-driven): Pull in new filings, market data, sector / regulatory updates.
    - Triggered analyses when major events: earnings release, regulatory announcement, large price move, etc.
    - Peer benchmarking and scenario generation.
3. **User / Stakeholder Interface**
    - Analysts can view raw data, metrics, tweak assumptions).
    - Reports accessible by management / clients.
4. **Regulatory / Compliance Considerations (India / SEBI)**
    - Use only public / legally disclosed information.
    - Be cautious about forward-looking statements; label assumptions.
    - For PMS / advisory services ensure disclosures of risk, past returns, disclaimers.
    - Data privacy / corporate governance must be respected.

5. Expected Result and Progress

5.1 News Browsing Module

The news module gives users a clean, easy-to-read feed of financial stories. It pulls news from different sources but removes duplicates so the same story only shows once. Each story is shown as a simple card with a title, short summary, source, and finance tags. Users can filter or sort by time, source, topic, or their watchlist stocks. They can also see why an article appears, check related stories, save or share items, and search for similar content.

In the background, the system makes sure every article is well-formatted, tagged, and searchable, while keeping the original source for transparency. The interface is fast, supports accessibility, and offers basic personalization. A light cache keeps loading smooth, and simple APIs make it easy to connect with other services.

5.2 Stock Trend Module

5.2.1 Expected Result for Recommendation System & Prediction System

For the Recommendation subsystem within the Stock Trend Module, the following concrete outcomes are expected upon successful implementation:

At the initial stage, the Stock Trend Module will deliver a working pipeline that combines recommendation and prediction. The recommendation function will generate personalized stock lists that avoid simple popularity bias, offering varied results across sectors and

themes, each with a short explanation of why it is suggested. The prediction function will provide short-horizon (7–14 day) directional signals for individual stocks, returned quickly enough to be integrated into the user interface. Forecasts will include not only an up/neutral/down label but also a basic confidence score and a few main drivers for transparency. Together, these early capabilities will allow users to see more relevant stocks, explore multiple themes, and gain simple yet actionable insights into near-term movements

5.2.2 Expected Result for RAG Q&A Module in Stock Trend module

In the initial phase, this module enables users to ask natural language questions about company filings, earnings calls, and recent news, and efficiently retrieves relevant document chunks from the past 90 days using vector search with ticker and time filters. Candidate results are reranked by similarity and recency, then passed to Qwen2.5-32B to generate evidence-based answers with source citations for transparency. The exposed /qa endpoint returns the answer, citations, retrieval stats, and a confidence score, while Redis caching and lightweight monitoring ensure stable and reliable responses within three seconds.

5.3 Combined Investment Recommendation Module

In its initial phase, the module is expected to improve analytic efficiency by reducing the time analysts spend on data collection and cleaning, allowing them to focus more on interpretation. It will produce reports with consistent metrics and transparent assumptions, making peer comparisons easier. Users will be able to respond more quickly to earnings releases, regulatory changes, and macro events, while also gaining better awareness of downside risks through scenario analysis and sensitivity checks. Over time, the system will support benchmarking of its outputs against actual market outcomes, providing a feedback loop on accuracy. For example, for a mid-cap stock, the module should be able to generate valuation estimates reasonably close to market consensus, even within a ±10–20% range, acknowledging market volatility.

6. Expected Challenges

6.1 News Browsing Module

Large-scale copying with minor modifications can cause content inflation and introduce ranking bias.

Company aliases and cross-language naming can lead to errors in linking entities to securities.

Controlling system complexity and operational stability while meeting low latency and high freshness requirements is challenging.

6.2 Stock Trend Module

6.2.1 Expected Challenges for Recommendation System

The development and deployment of the recommendation engine present several anticipated challenges that the project will need to address:

Cold Start Problem: A significant challenge will be providing accurate and engaging recommendations for new users who have no browsing history, making their profile vector non-existent or very sparse.

Defining and Evaluating 'Success': While offline metrics like Recall@K are valuable, the true measure of success is user engagement and investment decisions, which are more difficult to quantify. How to accurately evaluate the recommendation engine's business impact will be a non-trivial challenge.

Dynamic Nature of Financial Data: User interests and market conditions change rapidly. A user's profile vector must be updated frequently to reflect their evolving preferences. How to determine the optimal update frequency and decay rate for the user profile to remain responsive without becoming overly volatile

6.2.2 Expected Challenges for Prediction System

Market Noise and Non-Stationarity: A core challenge will be designing models and retraining strategies that remain robust across regimes without overfitting to recent patterns.

Balancing Short-Term vs. Long-Term Horizons: Determining the optimal forecasting horizon and tuning models to maximize both statistical accuracy and user utility will be a persistent challenge.

Feature Stability and Data Latency: Ensuring stable feature engineering pipelines, timely ingestion, and monitoring for data drift will be critical to maintain reliable predictions.

6.2.3 Expected Challenges for RAG Q&A Module

Data Quality & Preprocessing: Financial reports, news, and conference call transcripts have complex formats and are prone to redundancy, noise, or omissions.Text segmentation strategies directly impact search performance and require continuous optimization.

Retrieval Accuracy: Vector search may recall fragments that are superficially similar to the question but semantically irrelevant. Over-reliance on semantic similarity may overlook timeliness and company relevance. Top-k and rerank strategies require a balance between recall and precision.

Answer Faithfulness & Hallucination：Even with search support, large models may still fabricate data or exaggerate conclusions.

6.3 Combined Investment Recommendation Module

Challenge	Reason / Risk	Possible Mitigation
Model risk / over-fitting	Quant models might do well historically but perform poorly going forward; market regime changes (e.g. macro shocks) can invalidate assumptions.	Use out-of-sample testing; stress tests; include scenario analyses; build in model monitoring; update models periodically.
Interpretability & human trust	Users (analysts, clients) may not trust "black box" outputs, especially from AI/NLP; explanations are needed.	Make every recommendation accompanied by evidence, footnotes; allow drill-downs; include sensitivity tables; keep some human-in-loop.
Bias, noise, and errors from news / sentiment	False information, hype, errors; sentiment models may misinterpret context; overreaction to non-material events.	Use reliable sources; filter; validate events; have thresholds for triggering; allow human review for material items.
Maintaining freshness & up-dates	If the model becomes stale (e.g. macro environment shifts), or data feed breaks, output will degrade.	Set up monitoring; periodic retraining or recalibration; modular design so parts can be updated.

7. Conclusion

This implementation draws directly on concepts from our Intelligent Reasoning Systems course. By combining knowledge representation, reasoning under uncertainty, and data-driven inference, the system goes beyond raw computation to provide structured, explainable, and evidence-backed financial recommendations. The ability to integrate symbolic reasoning (rules, scenarios, constraints) with statistical methods (LLMs, embeddings, sentiment models) demonstrates how intelligent reasoning frameworks can be applied to real-world domains like financial analysis and investment decision-support.

B

Mapped System Functionalities

Table B.1: Mapping of System Functionalities against Knowledge, Techniques, and Skills from MR, RS, and CGS Courses

Course Module	Techniques Covered	Mapped FinSight Functionalities
MR	■ Knowledge representation and reasoning for problem solving ■ Search and constraint satisfaction reasoning ■ Machine reasoning over knowledge graphs ■ Neural and inductive reasoning models	■ Knowledge-based stock analysis within the AI Analyst (RAG Assistant) module ■ Reasoning engine for contextual understanding of user queries ■ Use of inductive reasoning for suggesting stock recommend explanations
RS	■ Market-based and similarity-based recommender systems ■ Model-based hybrid recommendation design ■ Knowledge discovery from large datasets	■ Stock Recommendation module using similarity and hybrid recommender approaches ■ User profiling and vector-based preference optimization (content-based + collaborative signals) ■ News browsing feedback loop employing incremental optimization of user preference vectors ■ Data-driven reasoning for refining recommendations from large-scale market data
CGS	■ Natural language cognition and LLM reasoning ■ Knowledge representation and reasoning in cognitive systems ■ HumanAI interaction and chatbot-based communication	■ RAG-based AI Financial Analyst for conversational reasoning and decision support ■ Cognitive interface enabling natural-language interaction between user and system ■ Contextual retrieval and reasoning using LLMs for personalized financial insights ■ Knowledge-based dialogue that integrates user intent with domain reasoning

1. News Module -> Knowledge discovery
2. Recommendation Module -> Hybrid recommender optimization Knowledge discovery
3. Forecast Module -> Business resource optimization (evolutionary computing)
4. AI Analyst Module -> Decision automation (knowledge-based reasoning) and Cognitive system design (knowledge graph & natural-language interface)

C

Installation and User Guide

C.1 | Installation

C.1.1 | Backend

C.1.1.1 | Overview

FinSight is an AI-powered financial analytics backend that provides news aggregation, vector-based retrieval, stock forecasting, recommendation, and RAG-based financial research assistance all implemented within a modular `FastAPI` architecture.

This repository hosts the backend micro-services, which include:

- **News Fetching & Aggregation**
- **Stock Trend Forecasting** (ARIMA / Prophet / LSTM / LightGBM / Transformer / Seq2Seq)
- **RAG-based Financial Research Analyst**
- **Content-based & Vector-based Stock Recommendation**
- **User Profiles, Watchlists, and Authentication**
- **MongoDB + Postgres (pgvector) Support**

C.1.1.2 | Project Structure

```
FinSight_BackEnd
 app
     adapters
         db
             database_client.py
             news_repo.py
             pg_profile_repo.py
             rag_conversation_repo.py
             user_repo.py
         embeddings
         fetchers
         llm
         rag
```

```
        vector
    api
        v1
            auth_router.py
            forecast_router.py
            news_router.py
            rag_router.py
            rec_router.py
            stocks_router.py
            user_router.py
    forecasters
        arima_forecaster.py
        prophet_forecaster.py
        lstm_forecaster.py
        transformer_forecaster.py
        stacked_forecaster.py
    services
        news_service.py
        forecast_service.py
        rag_service.py
        rec_service.py
        user_service.py
    main.py
docker-compose.yml
environment.yml
finsight_keypair.pem
pyproject.toml
README.md
requirements.txt
```

All dependencies are fully managed through **`pyproject.toml`**.

C.1.1.3 | Prerequisites

Table C.1: System Prerequisites

Tool	Requirement
Python	3.10+
Docker	Required for Knowledge Base (MongoDB + Postgres)
uv / pip	For dependency installation
Ragflow	For RAG deployment
Ollama	For local LLM provider

Installation of `uv`

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

Database Access (via EC2 Server)

The EC2 server hosts both MongoDB and PostgreSQL databases:

- **MongoDB:** Stores news and optionally vector embeddings.
- **PostgreSQL + pgvector:** Stores user profiles, embeddings, and recommendations.

To connect, simply place `finsight_keypair.pem` in the project root. The program will automatically establish the SSH tunnel and access the databases.

C.1.1.4 | Optional: RAG Component Deployment

Due to limited memory on the EC2 server, the RAG module is deployed locally. This step is optional and does not affect other backend functionalities.

1. **Deploy Ragflow locally:** Follow the official guide at `https://ragflow.io/docs/dev/build_docker_imag`
2. **Run local LLM via Ollama:**

```
    ollama run deepseek-r1:8b

```

3. **Replace your Ragflow key:** Add it to the `.env` file as:

```
    RAGFLOW_API_KEY=<your-key>

```

C.1.1.5 | Installation

```
git clone https://github.com/jiajun-lab/FinSight_BackEnd.git
    FinSight_BackEnd
cd FinSight_BackEnd

# Create virtual environment
uv venv
source .venv/bin/activate

# Install dependencies
uv pip install -e .
# (or pip install -e .)
```

Note: Prophet will compile CmdStan on first run, which may take several minutes.

C.1.1.6 | Running the Backend

Before starting, configure your `.env` file accordingly.

```
uvicorn app.main:app --reload --port 8000
```

API documentation is available at:

```
http://localhost:8000/docs
```

The backend service runs on `http://localhost:8000`, and the frontend retrieves all data from this endpoint.

—

C.1.2 | Frontend

C.1.2.1 | Installation

```
# 1) Clone this repository
git clone https://github.com/jiajun-lab/FinSight_Frontend.git FinSight_FrontEnd
cd FinSight_FrontEnd

# 2) In this folder:
pnpm i

# 3) set backend endpoint (optional; default already localhost):
echo 'VITE_BACKEND_BASE_URL=http://127.0.0.1:8000' > .env

# 4) run
pnpm dev

# 5) quick restart
rm -rf node_modules pnpm-lock.yaml package-lock.json # optional clean reboot
pnpm i
pnpm dev
```

C.2 | News Browsing

C.2.1 | Overview

The FinSight News Browsing Module allows users to explore personalized financial news in an intuitive and interactive interface. The system learns from your actions — such as reading, liking, and bookmarking — to continuously refine future recommendations. This guide explains how to navigate, interact, and get the best experience from the news platform.

C.2.2 | Getting Started

1. Open the FinSight web application in your browser.
2. Log in with your registered account.
3. Click on the **News** tab in the top navigation bar to enter the news recommendation page.

Once you enter, the system automatically loads your personalized news feed, showing 9 recommended articles in a clean 3×3 grid layout.

C.2.3 | Browsing the News Feed

- Each news card displays the **title**, **source**, **publication time**, and **related stock or sector**.
- Click the article title to open the full story in a new browser tab.
- Use the **Next** and **Previous** buttons at the bottom to move between pages.
- The system automatically caches previous pages, so navigation is fast and seamless.
- When you reach the last page and click **Next**, new articles will be fetched and displayed automatically.

C.2.4 | Interacting with Articles

You can directly interact with each article to help the system learn your preferences:

- **Like** (♡) Press this button if you find an article relevant or interesting. Future recommendations will prioritize similar topics or sectors.
- **Bookmark** (⋆) Use this to save an article for later reading. It also signals a stronger preference to the system.
- **Read** Simply clicking and reading an article contributes to your profile based on reading duration.

All actions update your profile automatically in the background — there is no need to refresh the page.

C.2.5 | Getting Personalized Recommendations

The news feed automatically adapts based on your reading and interaction history:

- The more you interact, the more accurate your recommendations become.
- You will start seeing more articles from sectors and companies that match your interests.
- Older or irrelevant content will gradually fade out as new interests emerge.

You can always reset your browsing session by reloading the page — the system will load a fresh selection of recommended articles.

C.2.6 | Tips for Best Experience

- Interact regularly (like or bookmark) to personalize your feed faster.
- Use **Next** to load new articles when you reach the end of your feed.
- The system automatically avoids showing repeated or recently viewed items.
- All updates happen instantly without page reloads.

C.3 | Stock Recommendation

The Stock Recommendation System helps investors discover suitable investment opportunities through intelligent algorithms and comprehensive stock analysis tools. This manual explains how to use the six main features of the system.

C.3.1 | Browsing All Stocks

To explore the complete stock universe:

1. Click the **"All Stocks"** tab in the main navigation.
2. Use the search bar at the top to find stocks by symbol or company name.
3. Scroll through the grid layout to view all available stocks.
4. Each stock card displays:
   - Stock symbol and company name
   - Sector with color coding
   - Basic financial metrics when available
5. Click the pagination controls at the bottom to navigate through multiple pages.
6. Hover over any stock card to trigger pre-loading of detailed information.

C.3.2 | Getting Basic Recommendations

To receive personalized stock recommendations, as shown in Fig C.1:

1. Ensure you are logged in to your account
2. Click the **"Recommended"** tab
3. The system automatically displays stocks matching your investment profile
4. Adjust recommendation settings:
   - Use the **"Top K"** input to change the number of recommendations (1-20)
   - Use the **"Diversity"** dropdown to control sector variety
5. Click the **"Refresh"** button to update recommendations
6. Each recommendation shows a similarity score indicating how well it matches your profile

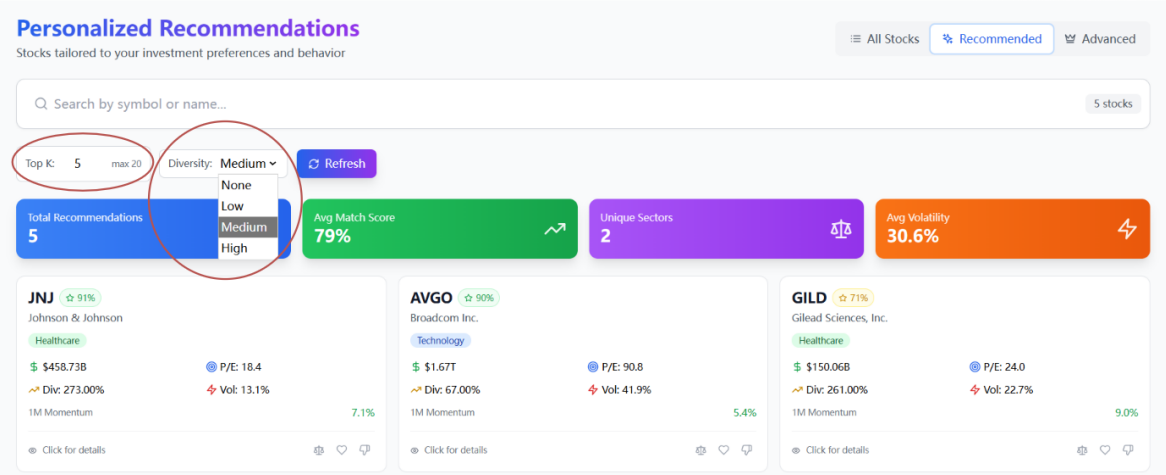


Figure C.1: Basic Recommendation

C.3.3 | Using Advanced Recommendations

For multi-objective optimized recommendations, as Shown in Fig C.2:

1. Click the **"Advanced"** tab
2. Select your risk profile:
   - **Conservative**: Lower risk, stable returns
   - **Balanced**: Moderate risk and return balance
   - **Aggressive**: Higher risk, potential for higher returns
3. View the detailed score breakdown for each recommendation:
   - Final composite score
   - Preference matching score
   - Risk-adjusted return score
   - Diversification benefit score
   - Market timing score
4. Read the explanation for why each stock is recommended
5. Adjust the number of recommendations using the **"Top K"** setting

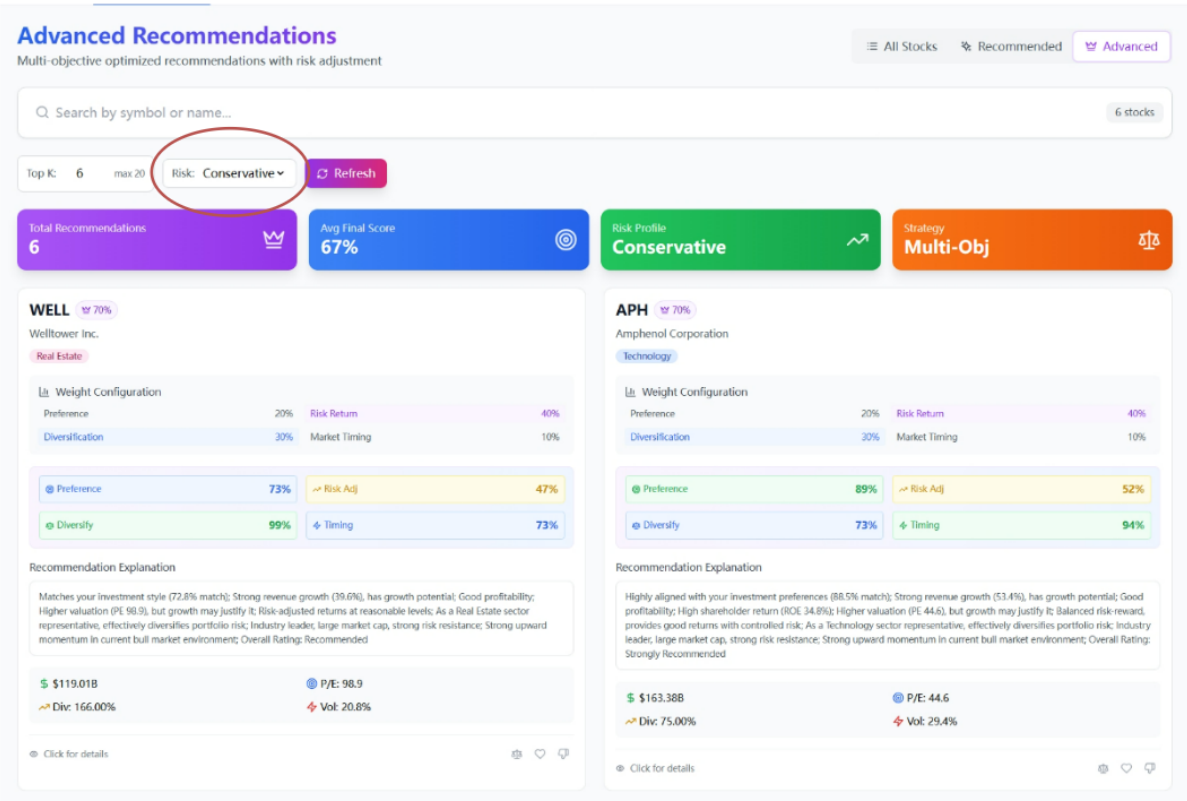


Figure C.2: Advanced Recommendation

C.3.4 | Comparing Stocks

To compare multiple stocks side-by-side, as shown in Fig C.3:

1. Click the scale icon on any stock card to add it to comparison
2. You can add up to 3 stocks for comparison
3. View the comparison panel that appears at the top of the screen
4. Click **"Analyze Comparison"** when you have 2 or more stocks selected
5. In the comparison modal, you can:
   - View side-by-side metrics in a comparison table
   - Analyze valuation differences through bar charts
   - Compare risk levels through volatility visualizations
   - Review growth metric comparisons
   - Read the summary analysis highlighting key differences
6. Remove stocks from comparison by clicking the 'x' button

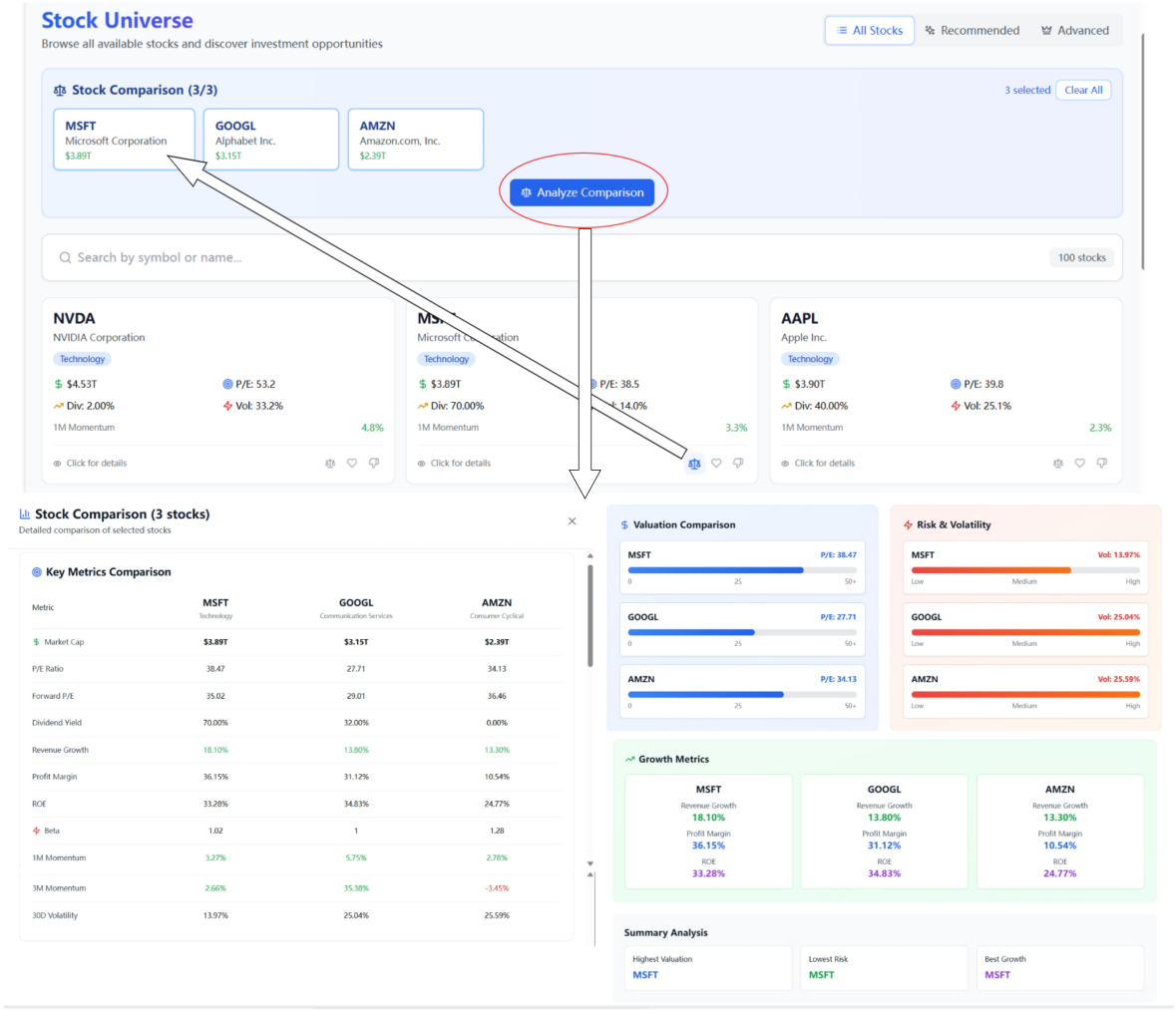


Figure C.3: Stock Comparison

C.3.5 | Viewing Stock Details

To access comprehensive stock information, as shown in Fig C.4:

1. Click on any stock card (except the action buttons)
2. The stock detail modal will open with comprehensive information
3. Navigate through different sections:
   - **Price Charts**: Switch between time ranges (1M, 3M, 6M, 1Y) and price types (Close, Open, High, Low)
   - **Valuation Metrics**: View P/E ratios, market cap, and other valuation data
   - **Financial Ratios**: Analyze profitability, growth, and efficiency metrics
   - **Dividend Information**: Review yield and payout ratios
   - **Performance Metrics**: Check momentum and volatility data
   - **Business Summary**: Read the company description

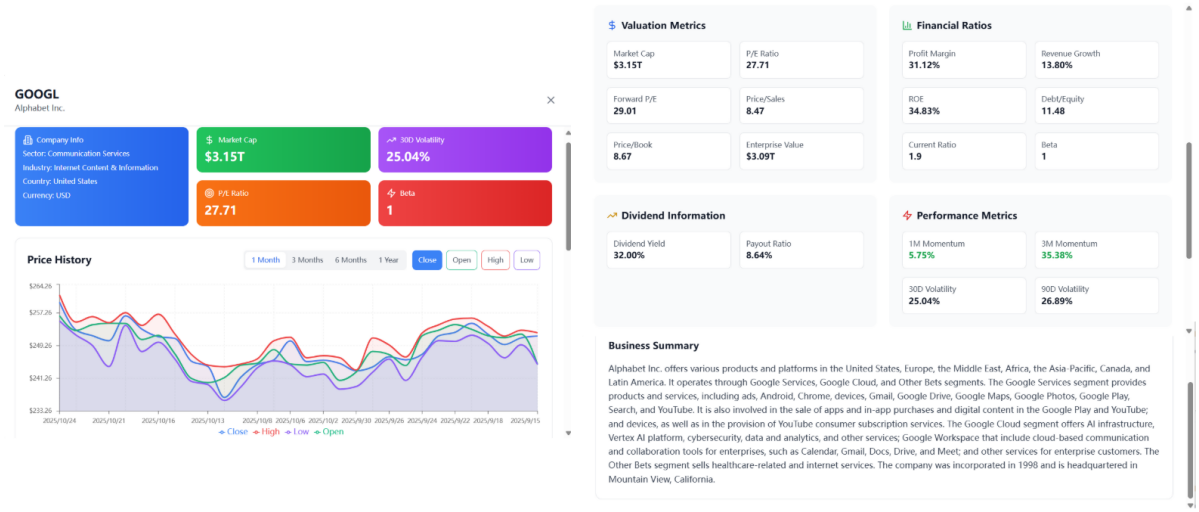


Figure C.4: Stock Detail

C.3.6 | Improving Recommendations Through Behavior

The system learns from your interactions to provide better recommendations, as shown in Fig C.5:

1. **Liking Stocks**: Click the heart icon (♡) on stocks you're interested in
   - This tells the system you prefer similar stocks
   - The heart will turn green and show a count
2. **Disliking Stocks**: Click the thumbs down icon on stocks you want to avoid
   - This helps the system understand what to avoid recommending
   - The icon will turn red and show a count
3. **Viewing Details**: Simply clicking on stocks to view details also influences your profile
4. **Automatic Updates**: The system automatically:
   - Updates your investment profile based on interactions
   - Adjusts future recommendations in real-time
   - Learns your sector preferences and investment style
5. To see the impact of your interactions, refresh the recommendations after several interactions

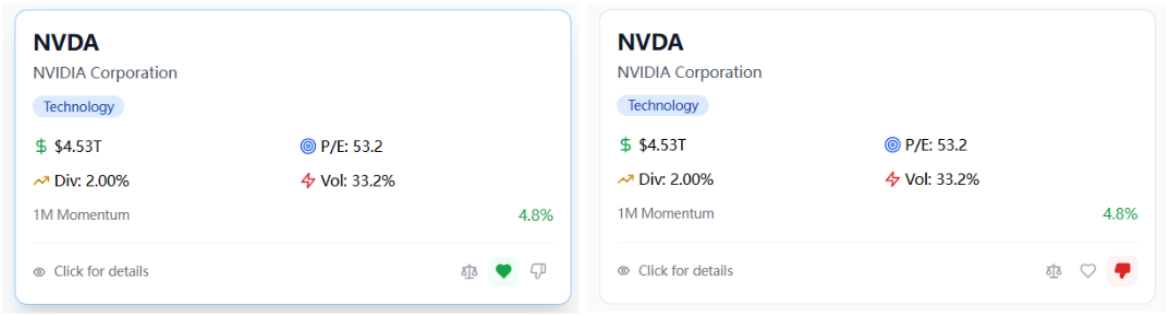


Figure C.5: User Behavior (like, dislike

C.4 | Stock Prediction

C.4.1 | Overview

The FinSight Stock Forecast Module provides users with intelligent, real-time stock trend predictions powered by multiple time-series and machine learning models. It visualizes both recent historical prices and short-term future forecasts, allowing users to intuitively interpret model predictions, confidence levels, and expected changes. This guide explains how to access, interact with, and interpret the stock forecasting interface.

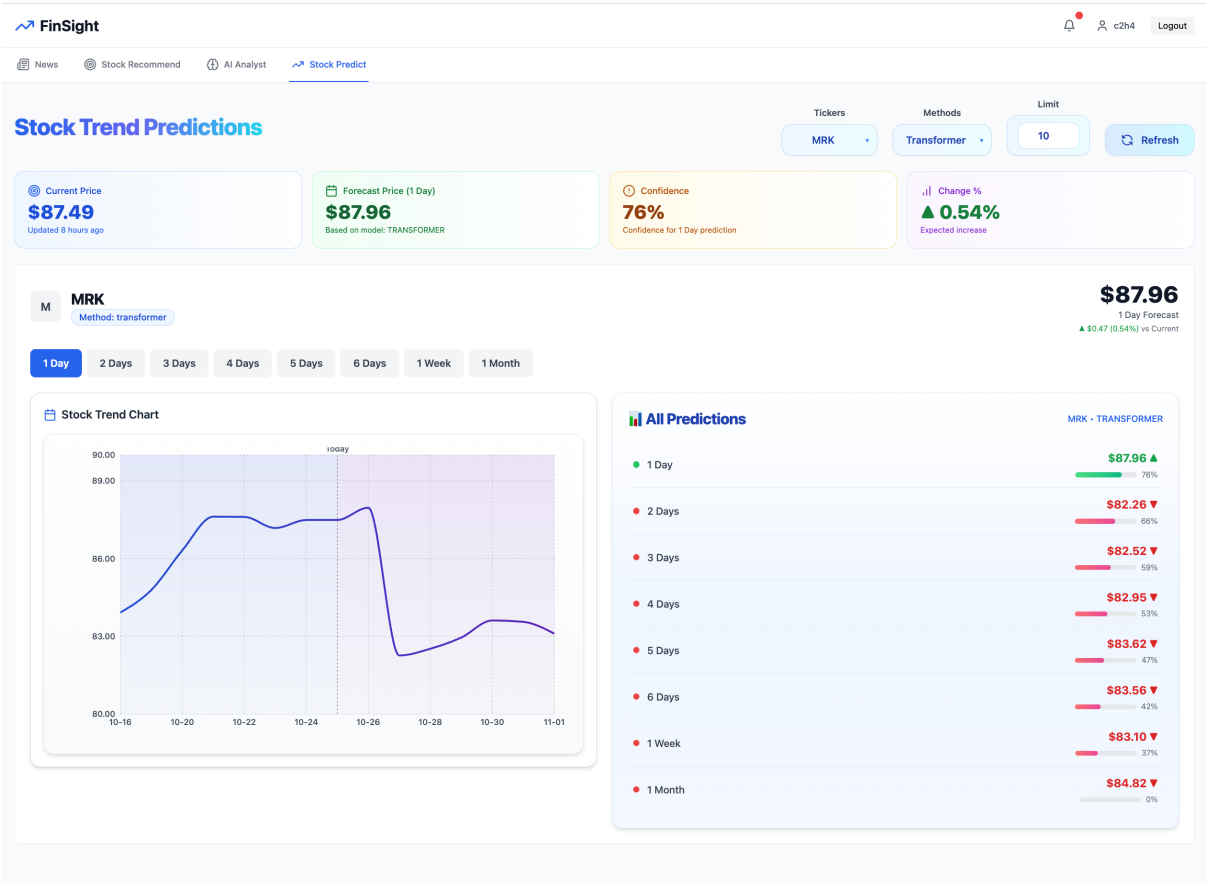


Figure C.6: Stock Prediction User Interface

C.4.2 | Interface Overview

1. Open the FinSight web application in your browser.
2. Log in using your registered account credentials.
3. Navigate to the **Stock Predict** tab from the top navigation bar.
4. Select a stock ticker (e.g., AAPL, TSLA, MRK) and a forecasting model (ARIMA, Prophet, LSTM, Transformer) from the dropdown menus.
5. Click **Refresh** to load the most recent predictions and market data.

Once loaded, the system displays the current price, forecasted price, model confidence, and expected percentage change in a clean and responsive dashboard.

C.4.3 | Understanding the Forecast Dashboard

The main dashboard is divided into three functional regions for clarity and analytical insight:

- **Top Summary Cards:**
  - **Current Price:** Shows the latest closing price retrieved from the market data feed.
  - **Forecast Price:** Displays the models predicted price for the selected horizon (e.g., 1 Day, 1 Week).
  - **Confidence:** Indicates the models reliability, derived from validation metrics such as RMSE or variance.
  - **Change** Shows the expected price movement relative to the current price, color-coded in green (positive) or red (negative).
- **Trend Chart:**
  - Visualizes both recent actual prices (solid blue line) and predicted prices (purple dashed line).
  - A vertical line marks today, separating historical and forecasted data.
  - Shaded areas represent confidence intervals for visualizing prediction uncertainty.
  - Horizon buttons (1 Day, 3 Days, 1 Week, 1 Month) dynamically update the chart without reloading the page.
- **Prediction Summary Panel:**
  - Lists all horizon forecasts with predicted values, confidence scores, and trend arrows.
  - Allows quick comparison between short-term and medium-term predictions.
  - Color-coded bars visualize model confidence for each horizon.

C.4.4 | Interacting with Forecasts

You can explore predictions interactively:

- Change the **ticker symbol** to analyze a different stock instantly.
- Switch between **models** (e.g., LSTM vs. Transformer) to compare forecast behavior.
- Adjust the **limit** or time horizon to view more or fewer prediction intervals.
- Hover over any data point in the chart to view exact date and price details.

All interactions automatically trigger backend API calls, and updated results appear within seconds.

C.4.5 | Interpreting Forecast Results

- A **higher confidence score** (close to 100%) indicates a more stable forecast, while lower scores reflect market volatility or data uncertainty.
- Positive **Change %** suggests an expected upward trend; negative values suggest possible price declines.
- Comparing multiple models provides insight into how traditional and deep learning approaches interpret market patterns differently.
- The charts shaded area width visually represents prediction riskthe wider the band, the greater the uncertainty.

C.4.6 | Tips for Best Experience

- Refresh periodically to fetch the most recent forecast data.
- Compare different models to assess prediction consistency.
- Use shorter horizons (13 days) for near-term signals and longer ones for trend analysis.
- Interpret low-confidence predictions cautiously; they indicate volatile or sparse underlying data.
- Combine forecast insights with other FinSight modules for more informed investment decisions.

C.5 | AI Analyst

The **AI Financial Analyst** module serves as an intelligent assistant powered by a Retrieval-Augmented Generation (RAG) framework. It enables users to ask analytical questions about stocks, sectors, earnings, and market trends, and receive context-aware, knowledge-grounded answers.

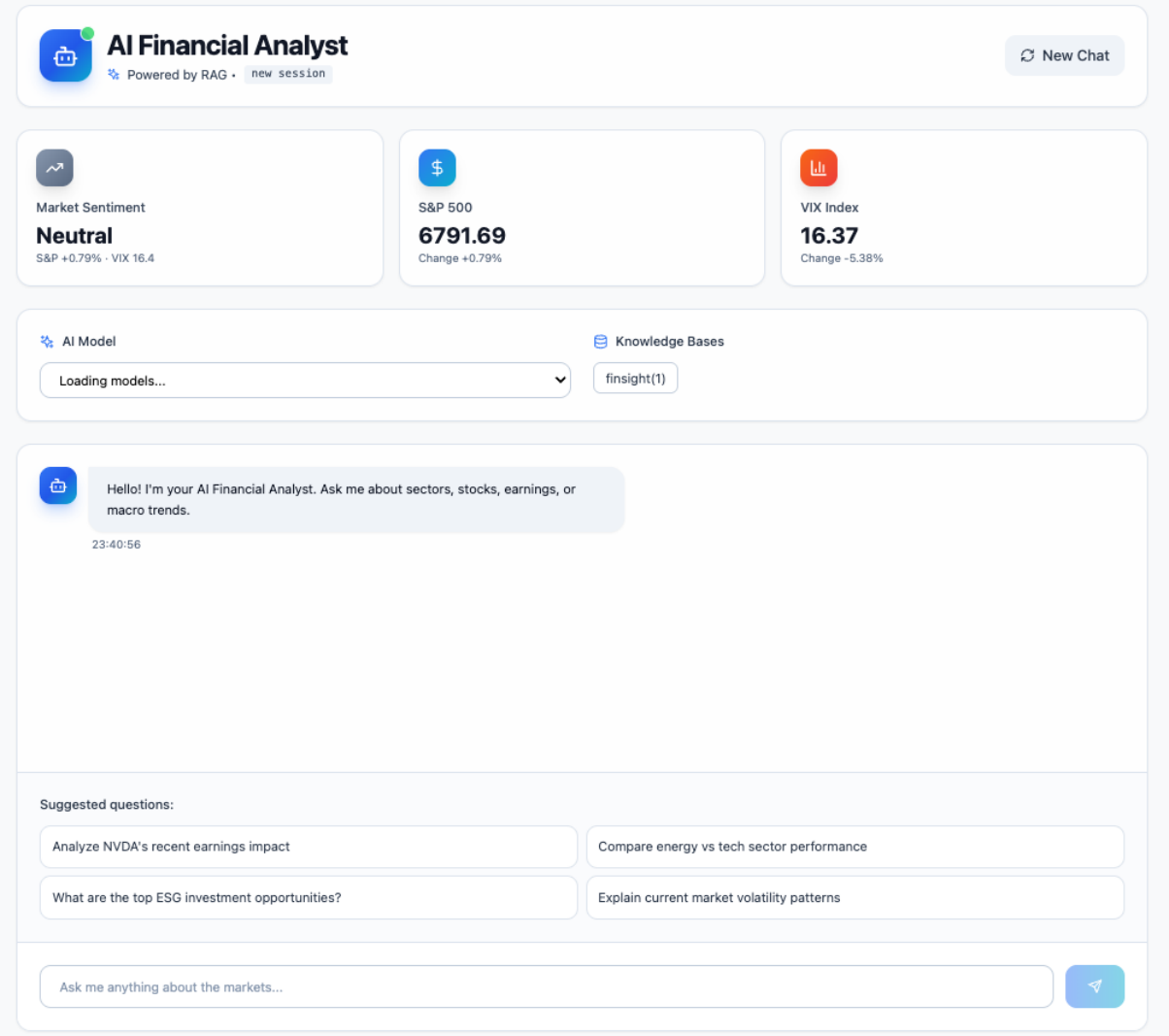


Figure C.7: AI Analyst User Interface

C.5.1 | Interface Overview

- **Market Indicators**
  The dashboard displays key real-time market metrics:
  - **Market Sentiment:** General market mood derived from aggregated financial data.
  - **S&P 500 Index:** Current value and daily percentage change of the U.S. benchmark index.
  - **VIX Index:** Market volatility indicator reflecting overall risk and uncertainty levels.
- **AI Model Selection**
  Users can choose among available AI models (e.g., deepSeek-r1:8b) to power the assistants reasoning and response generation.
- **Knowledge Base Selection**
  The system allows users to select from available financial knowledge bases (e.g., *finsight(1)*). The selected dataset determines the retrieval source used to ground responses with relevant and accurate financial context.

C.5.2 | Functionality

The AI Analyst supports natural-language financial Q&A, allowing users to:

- Ask about market movements, company performance, or sector comparisons.
- Obtain summarized insights, reasoning explanations, and evidence-based references.
- Conduct context-aware, multi-turn analytical conversations supported by the selected model and knowledge base.