Project Report: GenAI Advisor

based on the Financial Advisor Bot template

Andreas Mallios
BSc Computer Science (ML and AI)
Student number: 200128357

September 2025

Contents

1 Introduction (970/1000 words)

Artificial Intelligence (AI) continues to reshape global industries, with Generative AI (GenAI) standing out as one of the most influential recent developments. GenAI models, such as large language models (LLMs) and diffusion architectures, have driven widespread adoption and significant investment, particularly in the technology sector. Yet, for retail investors, the tools available to access this growth remain largely generic and inaccessible, both in terms of domain focus and interpretability.

Private equity and venture capital investments in GenAI startups are becoming larger and more targeted as investor strategies evolve with the technology. Funding in GenAI exceeded $56 billion in 2024, nearly double the $29 billion recorded in 2023 (14).

As shown in Figure 1 below, while the number of funding rounds declined in 2024, the overall deal value surged. This suggests a consolidation of capital into larger, infrastructure-oriented bets, aligning with investor confidence in scalable GenAI deployments. These trends highlight the need for tools that help investors evaluate publicly listed companies best positioned to benefit from this structural shift.

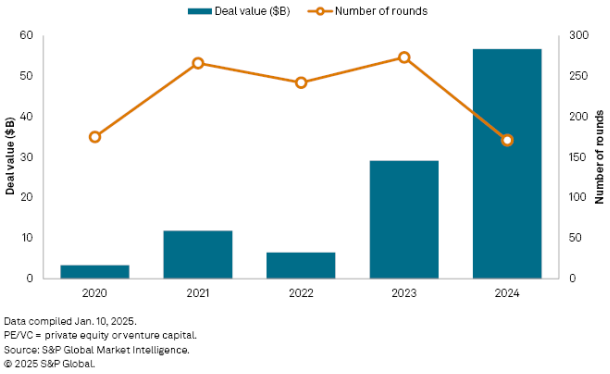


Figure 1: PE/VC investments in GenAI, 2020–2024. Source: S&P Global Market Intelligence (14).

This project responds to that gap by proposing the **GenAI Advisor**, a system that delivers explainable stock recommendations specifically for companies driving or enabling the GenAI revolution. Based on the *Financial Advisor Bot* template, the system is designed for non-technical users, providing accessible, transparent investment signals that go beyond traditional model outputs.

1.1 Concept

The GenAI Advisor is conceived as a personalised financial guidance tool for individual investors interested in the GenAI sector. It provides interpretable recommendations, whether to buy, hold, or sell specific stocks, alongside human-readable explanations for these decisions. Unlike traditional investment tools that obscure decision logic behind opaque algorithms or technical jargon, this system foregrounds explainability and domain relevance.

The intended users of this system are individual retail investors interested in emerging technology themes but lacking the technical background to interpret raw financial signals or operate sophisticated trading tools. These users are typically non-specialists with limited exposure to algorithmic finance or machine learning models. For this group, the GenAI Advisor offers an accessible tool to support their decisions, that reduces complexity while preserving analytical rigour. It is also suitable for students, educators, or early-stage investors exploring explainable AI in financial contexts.

At its core, the GenAI Advisor is built on the belief that financial tools should be not only accurate but also comprehensible. This is particularly important in thematic investing, where

users are drawn not only by quantitative performance but by interest in specific technological trends. In this context, the system acts as a digital intermediary between complex market signals and users' intuitive understanding, offering advice that is grounded and traceable, with a clearly scoped set of companies.

The conceptual novelty lies in its hybrid orientation; it combines evidence-driven financial analysis with a commitment to interpretability. The GenAI Advisor does not aim to outperform the market through high-frequency trading (HFT) or proprietary forecasting. Instead, it prioritises accessibility, domain focus, and trustworthiness, delivering insights users can understand and apply in their decision-making. In doing so, it invites users into the analytical process rather than substituting for it.

Furthermore, the GenAI Advisor embraces thematic integrity by carefully curating its stock universe. It avoids general-purpose technology firms in favour of those with demonstrable involvement in the GenAI ecosystem, whether through infrastructure, tooling, model development, or deployment. The rationale for company inclusion and exclusion is detailed in the project scope (Section 1.3). This conceptual grounding ensures that recommendations are not only technically derived but also strategically aligned with the user's thematic intent.

1.2 Motivation

The motivation for this project is both technical and user-oriented. GenAI represents an emerging thematic investment opportunity that remains underserved by existing tools. Simultaneously, there is increasing demand for transparent and accessible AI systems tailored to the needs of retail investors in the financial domain.

Traditional robo-advisors, such as Nutmeg and Moneyfarm, provide automated portfolio allocation but are domain-agnostic and opaque in how decisions are made. While open-source trading bots like Freqtrade offer flexibility, they require technical expertise and lack natural language interfaces. StockGPT (9) demonstrates the potential of generative models in stock prediction, while recent work on XAI-integrated forecasting (10) highlights the importance of trust and interpretability in financial AI systems.

By combining simple rule-based logic with predictive models and natural language explanations, the GenAI Advisor aims to deliver recommendations that are not only accurate but also intelligible. A local LLM serves as an interpretive layer rather than a decision engine, ensuring that outputs remain grounded in logic and statistical inference, while still being comprehensible to the user.

Beyond usability, the project contributes to responsible AI by ensuring data privacy and reproducibility. No personal or sensitive data is collected, and all sources are public. The LLM is hosted locally to avoid transmitting user inputs externally. The system is explicitly intended for educational and demonstrative use, and includes appropriate disclaimers to distinguish it from regulated financial services.

1.3 Scope

The GenAI Advisor focuses on a curated set of publicly traded companies whose business models are materially driven by Generative AI. The portfolio includes firms across three categories: (1) software developers and strategic investors (e.g., Microsoft, Alphabet, Meta, Amazon); (2) hardware providers (e.g., Nvidia, AMD, Intel, Marvell); and (3) infrastructure suppliers (e.g., TSMC, SK hynix, Broadcom, Arista Networks). These companies were selected for their direct contributions to GenAI model development, deployment, or enablement.

To maintain thematic focus and ensure data availability, the system excludes private companies, indirect holdings, and general technology firms lacking core GenAI offerings (e.g., Apple, Netflix). Chinese firms (e.g., Alibaba, Tencent, Baidu) are also excluded due to market access

limitations (16) and restricted financial transparency due to the Variable Interest Entity (VIE) structure of the BATX (Baidu, Alibaba, Tencent, and Xiaomi) stocks (4).

This scope ensures the system operates within a clearly bounded, analytically defensible domain. It enhances the relevance of recommendations by aligning them with user intent and facilitates reproducible evaluation by focusing on transparent and publicly accessible data.

2 Literature Review (2078/2500 words)

The rapid advancement of Generative AI (GenAI) and its recent commercialisation in late 2022, has sparked significant interest in its application to financial forecasting and investment support systems. Despite notable progress in predictive modelling, current financial advisory tools often prioritise performance metrics at the expense of transparency and interpretability. This trade-off presents a particular challenge for non-technical retail investors, who require not only accurate but also comprehensible recommendations in order to make informed decisions.

This literature review examines recent developments at the intersection of GenAI, explainable AI (XAI), and algorithmic trading. It explores generative models for financial prediction, the integration of XAI techniques in forecasting systems, and the implications of user-centred design in FinTech. The aim is to critically evaluate existing approaches and identify a gap in the current landscape: the lack of accessible, domain-specific, and explainable investment tools. The review concludes by outlining how the *GenAI Advisor* project is designed to address this gap through a hybrid architecture that integrates transparent rule-based logic, supervised machine learning, and natural language explanation layers.

2.1 Generative AI in Financial Forecasting

The application of Generative AI (GenAI) in financial domains has emerged as a frontier area of research, particularly in the context of algorithmic trading and decision support. While traditional predictive systems rely on time-series models and supervised learning techniques, recent work has demonstrated the viability of large language models (LLMs) and generative transformers for tasks such as stock sentiment analysis, earnings prediction, and trade signal generation.

A notable example is *StockGPT* by Mai (9), which applies a fine-tuned generative language model to extract predictive signals from financial text and historical data. The system shows promising results in backtesting but offers limited transparency in terms of feature attribution and decision rationale. This highlights an emerging trade-off in GenAI systems between performance and interpretability, particularly in high-stakes domains like finance.

Generative models have also been used to simulate synthetic market data (15), enabling improved model training under data scarcity. However, these approaches often lack integration with rule-based or statistical baselines, making it difficult to assess the marginal utility of the generative component. Furthermore, the absence of human-centred explanation interfaces in these systems poses a barrier to adoption among non-specialist users.

These limitations underline the need for hybrid architectures that balance the predictive power of generative models with the interpretability of traditional techniques. The *GenAI Advisor* addresses this by separating signal generation from explanation, thereby ensuring that the core decision logic remains auditable while leveraging GenAI to enhance accessibility and user trust.

2.2 Explainable AI in Financial Systems

As AI-driven systems increasingly influence financial decision-making, the demand for explainable artificial intelligence (XAI) has become a critical concern. In domains characterised by uncertainty, regulatory oversight, and end-user scepticism, opaque "black-box" models are often ill-suited for practical deployment (6). This is particularly true for retail investors who may lack technical backgrounds yet require justifiable, transparent investment reasoning.

Marey et al. (10) propose an empirical framework that integrates deep learning with local explanation techniques, such as SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-Agnostic Explanations). Their study demonstrates that user trust and engagement are positively correlated with the presence of intelligible model rationales, especially

when users are allowed to compare multiple explanation formats. However, these explanation mechanisms are typically appended to existing models rather than embedded in the design, leading to potential inconsistencies between what the model decides and what is communicated.

Other work, such as Ribeiro et al. (12), highlights the trade-off between global accuracy and local interpretability. While LIME-style approximations offer useful insight into model behaviour, they can misrepresent model logic under certain conditions, particularly in non-linear financial regimes.

To address these issues, the *GenAI Advisor* integrates XAI at both the structural and presentation layers. Rule-based logic provides a baseline of transparent, auditable recommendations, while machine learning classifiers offer enhanced signal detection. A local language model translates system outputs into natural-language explanations, preserving alignment between system behaviour and user-facing justifications. This layered approach ensures both decision accuracy and explanatory coherence.

2.3 Rule-based vs Machine Learning Investment Strategies

Investment recommendation systems have traditionally relied on rule-based logic grounded in technical indicators such as moving averages, momentum oscillators, and price-volume trends. These heuristics offer a high degree of interpretability, making them attractive to both practitioners and novice investors (11). Strategies based on conditions like moving average crossovers or Relative Strength Index (RSI) thresholds are easily understood and lend themselves to direct explanation.

However, rule-based systems often fail to capture the non-linearities and temporal dependencies inherent in financial time series. This has led to a growing interest in machine learning (ML) models for stock selection and signal generation, such as Random Forests, Gradient Boosting Machines (e.g. XGBoost), and neural networks (5). These models can discover subtle patterns and adapt to changing market regimes, offering superior predictive performance in some contexts.

Yet, the adoption of ML in retail investment contexts is hindered by concerns over transparency and robustness. Ryll and Seidens (13) highlight that black-box models frequently suffer from overfitting, poor generalisation, and limited interpretability when applied to financial data. Chakraborty and Joseph (3) similarly observe that financial institutions remain cautious about deploying ML in high-stakes environments due to its opacity and the risk of unpredictable behaviour under changing market conditions. These issues are especially problematic for non-expert users, who rely on trustworthy and intelligible decision support systems.

The *GenAI Advisor* bridges this gap by employing a hybrid strategy engine. Transparent rules serve as a reliable baseline, while supervised learning models (Random Forest and XGBoost) are used to enhance signal strength by capturing complex interactions. This dual approach enables comparative evaluation and supports user trust, as each recommendation is either traceable to a logical rule or supported by model-driven evidence.

2.4 Designing for Non-Technical Users in FinTech

The usability and accessibility of financial technologies remain a significant challenge, particularly for non-technical users who may lack domain expertise in data science or quantitative trading. FinTech interfaces that prioritise raw performance often neglect explainability and user-centred design, which are essential for effective decision support and adoption among retail investors. Ben David et al. (2) demonstrate through experimental evidence that the inclusion of explainable AI elements significantly improves trust and adoption rates among users of financial algorithmic advisors. Their findings underscore the critical role of intelligibility and transparency in interface design for enhancing user engagement.

Studies have shown that users without a financial background exhibit higher levels of anxiety and lower confidence when interacting with complex or opaque recommendation systems. Without clear explanations, users may disregard otherwise effective recommendations, especially in high-stakes settings involving personal investments. This issue is compounded by the fact that many retail-focused platforms do not disclose the reasoning behind their advice, instead presenting deterministic outcomes with minimal context (17).

Explainable user interfaces (XUIs) have been proposed as a solution, incorporating elements such as visual signal annotation, confidence metrics, and natural-language summaries (2). These approaches have demonstrated success in increasing perceived trust, interpretability, and willingness to act upon AI-generated recommendations. However, implementation of such systems is often inconsistent, and few are tailored to specific thematic portfolios like those in GenAI-focused sectors.

The *GenAI Advisor* responds to this gap by embedding a natural language explanation layer that interprets rule-based and ML-driven signals into plain-English rationales. By adopting a language-first, model-agnostic approach, the system enables users to engage meaningfully with its outputs, fostering transparency and informed decision-making without requiring technical fluency.

2.5 Backtesting and Evaluation in FinTech

Robust evaluation is a critical component of financial algorithm design, ensuring that investment strategies are not only theoretically sound but also empirically validated under realistic market conditions. Backtesting a strategy using historical data—remains the standard approach for evaluating financial models. However, its reliability depends on careful control for biases such as lookahead error, data snooping, and overfitting (1).

Evaluation metrics in the FinTech domain extend beyond accuracy to include financial risk/return measures. The Sharpe Ratio, for instance, assesses risk-adjusted return by penalising volatility, while maximum drawdown quantifies peak-to-trough losses, both essential for comparing the robustness of competing strategies (8). Other common metrics include total return, alpha and beta coefficients, and classification-based indicators such as precision, recall, and F1-score when using ML classifiers for signal prediction.

Tools such as `backtrader` and `pandas`-based frameworks facilitate integration of financial indicators, portfolio simulation, and metric reporting, making them essential in contemporary algorithmic finance development. Nevertheless, evaluation practices remain inconsistent, with many academic and commercial models reporting high in-sample performance without adequate out-of-sample or cross-validation testing (7).

The *GenAI Advisor* explicitly incorporates backtesting into its pipeline using reproducible Python-based workflows. It compares rule-based and machine learning strategies across multiple indicators and time frames, reporting both financial and statistical performance metrics. This dual-layered evaluation, provides transparency and supports empirical justification for investment recommendations.

2.6 Synthesis and Research Gap

The review of existing literature reveals a rich landscape of innovation at the intersection of AI, finance, and user-centred design. However, it also exposes clear limitations in current approaches when evaluated through the lens of accessibility, explainability, and domain specificity, three attributes that are critical for supporting non-technical investors seeking exposure to GenAI-focused equities.

Generative models such as *StockGPT* demonstrate that large language models (LLMs) can be fine-tuned to extract financial signals from unstructured data, yet their outputs often lack interpretability and alignment with human-understandable trading logic (9). While this presents

a promising avenue for performance gains, it simultaneously creates a transparency deficit. Similarly, Takahashi and Mizuno (15) propose using diffusion models to generate synthetic financial time series that replicate complex market behaviours. Although valuable for addressing data scarcity and irregularities, these techniques rarely incorporate explanatory mechanisms or support integration with domain-informed baselines, limiting their applicability in systems designed for human-in-the-loop decision making.

Research on XAI has provided a broad array of model-agnostic tools such as SHAP and LIME (12), which can help reveal internal model dynamics. Yet, as highlighted by Marey et al. (10), these methods are frequently bolted onto systems post hoc and suffer from a mismatch between model behaviour and user comprehension. Additionally, the form and delivery of explanations have not been standardised across FinTech applications, leaving a usability gap that affects trust and adoption.

This concern is amplified when considering ML-only investment systems. Despite their performance advantages, such models are often developed without sufficient consideration for interpretability or operational auditability. Ryll and Seidens (13) highlight that machine learning models in financial forecasting frequently suffer from overfitting and poor generalisation, particularly under changing market regimes. Similarly, Chakraborty and Joseph (3) caution that opaque model architectures pose a challenge for real-world deployment in financial institutions, where accountability and stability are essential. Conversely, rule-based systems offer clarity and transparency but often lack the adaptability required to navigate complex, non-linear market dynamics.

From a usability perspective, research indicates that FinTech platforms frequently underestimate the cognitive demands placed on users, often failing to present decision rationales in a digestible and actionable format. Ben David et al. (2) demonstrate that the inclusion of explainable elements—such as natural-language justifications and transparency cues, significantly improves user trust and the likelihood of adopting AI-generated financial advice. While some interfaces include features like confidence bands or risk scores, few extend to personalised, intelligible explanations grounded in model logic and user context. Moreover, no existing platform explicitly addresses the needs of investors seeking targeted exposure to the GenAI sector, a growing interest area underserved by conventional robo-advisors.

In terms of evaluation, there is a consistent pattern of insufficient out-of-sample validation and limited comparative benchmarking across strategy types (1; 7). Financial metrics such as Sharpe Ratio or drawdown are inconsistently applied, and many studies prioritise classification accuracy without accounting for investor risk tolerance or decision thresholds.

Taken together, these findings point to a clear research and development gap: there is currently no publicly available tool that offers GenAI-specific stock recommendations through a hybrid system combining auditable rules, machine learning models, and explainable outputs tailored to non-expert users. Moreover, there is a lack of reproducible, modular pipelines that support comparative evaluation of rule-based and ML-based strategies in the context of GenAI sector investment.

The *GenAI Advisor* aims to fill this gap by providing an integrated, open-source prototype that embodies the insights drawn from this literature. It does so through: (1) a hybrid strategy engine that combines clarity and complexity; (2) a local LLM explanation layer to generate user-friendly rationales; (3) focused stock screening based on GenAI sector relevance; and (4) a rigorous evaluation framework using both financial and statistical metrics. By unifying these components, the system advances the current state of practice in explainable FinTech design for thematic investing.

In summary, while research has advanced in AI-driven forecasting and XAI, there remains no system that unites explainability, thematic focus, and evaluative rigor for retail investors in the GenAI sector.

The next section outlines how these insights informed the system design and architecture of

the GenAI Advisor.

3 System Design (1912/2000 words)

This chapter presents a detailed account of the design of the GenAI Advisor system, designed to deliver explainable, user-friendly financial insights for retail investors with an interest in Generative AI equities.

3.1 System Overview & Design Rationale

The architecture of the GenAI Advisor system is composed of five distinct yet interdependent layers:

1. **Data Pipeline**, responsible for sourcing, cleaning, and transforming financial data into formats suitable for analysis.
2. **Strategy Engine**, the analytical core that integrates retrieved knowledge to generate structured investment insights.
3. **Explanation Generator**, producing justifications and rationales for each recommendation.
4. **User Interface**, delivers an intuitive interaction point for retail users, visualising results in a digestible and accessible format.
5. **Backtesting & Evaluation**, supports validation of generated recommendations through historical simulations, user feedback loops and performance scoring.

Each layer encapsulates a single responsibility while remaining interoperable with adjacent layers, reflecting a separation-of-concerns design philosophy. This modular approach enables independent development and testing of each component, reducing coupling and simplifying future iterations or extensions.

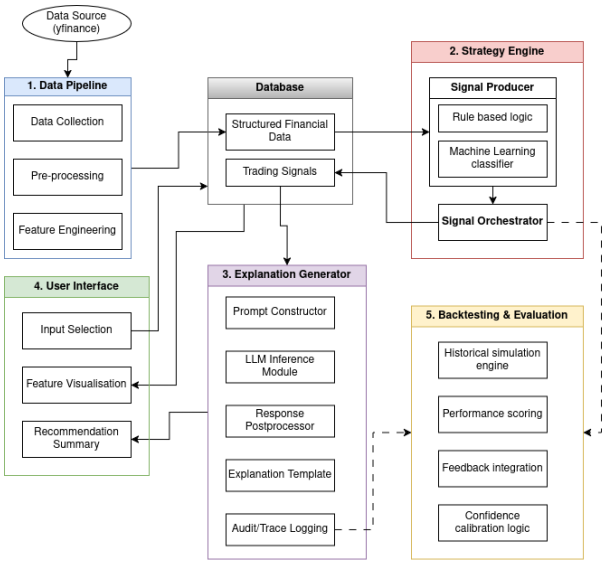


Figure 2: Five-layer Architecture of GenAI Advisor

The design rationale also reflects broader system requirements, particularly the need for explainability, adaptability, and integration with real-time financial data streams. Embedding traceability into the architecture ensures that every recommendation is grounded in verifiable data sources, supporting responsible AI usage. Furthermore, by situating the explanation generator as a distinct layer, the architecture highlights the importance of user trust and transparency as core design principles.

Scalability is not a goal of the current system, and no scaling mechanisms will be implemented in either the prototype or the final version. However, scalability remains an important consideration in the broader context of deploying AI systems in production environments. Should the system be developed further, future work could explore stateless service design, containerised deployment, and asynchronous processing to support larger data volumes and concurrent users. Flexibility across infrastructure backends and compatibility with different classes of language models could also be considered to enable wider applicability.

3.2 Data Pipeline

This foundational layer is responsible for acquiring, preparing, and transforming financial data into structured formats that support both retrieval and modelling. It ensures that the inputs to the system are consistent and semantically rich.

Data Sources: Financial data is sourced via the `yfinance` Python library, which extracts structured data from Yahoo Finance. This includes:

- Daily price data
- Adjusted close prices accounting for dividends and splits
- Market fundamentals including P/E ratio, EPS, market capitalisation, and beta
- Dividend history, earnings results, and analyst forecasts
- Corporate actions such as stock splits and ex-dividend dates

Pre-processing: All raw data is immediately pre-processed in memory using `pandas` and associated libraries before being persisted. This includes:

- Handling of missing values and filtering of anomalous entries
- Normalisation and transformation of numerical columns
- Resampling and forward-filling to align time indices across features
- Annotating rows with relevant metadata

This pre-processing ensures consistency across all instruments and time periods and reduces computational load in downstream stages.

Data Storage: Once cleaned and transformed, the processed dataset is stored in a relational MariaDB database. This allows for fast, structured access by subsequent system components and avoids repeated API calls to Yahoo Finance. Tables are designed to support efficient filtering by date, ticker, and feature type. Indexes are maintained on primary keys and time columns to optimise query latency.

Feature Engineering: This stage derives higher-order signals and representations from the cleaned data:

- **Technical indicators:** including moving averages, MACD, RSI, and volatility bands
- **Fundamental deltas:** e.g. growth rates in earnings, changes in valuation multiples
- **Risk signals:** calculated from rolling variance, beta, and return dispersion
- **Embeddings:** tabular records are converted into dense vector representations using Sentence-BERT for semantic similarity and retrieval

Design Justification: The choice of MariaDB for structured data storage reflects a balance between transparency, performance, and system complexity. Relational databases offer well-defined schema enforcement, support for indexed queries, and seamless integration with Python data analysis libraries. This aligns with the project's emphasis on explainability, traceability, and reproducibility. Alternative solutions were considered, including DuckDB for embedded analytics or Parquet-backed storage for columnar efficiency. However, these options introduce either added complexity or are optimised for larger-scale, concurrent systems. For the scope of this project, focusing on iterative prototyping, backtesting, and clarity, MariaDB provides a robust and pedagogically sound foundation.

3.3 Strategy Engine

The strategy engine forms the analytical core of the GenAI Advisor system, responsible for evaluating investment opportunities using structured signals and domain-specific rules. It interprets features derived from historical data and generates intermediate outputs such as buy/sell/hold flags, risk scores, and confidence levels. This layer operates independently of any generative model.

Signal Retrieval: At runtime, the strategy engine queries the MariaDB backend for time-aligned features such as technical indicators, valuation metrics, volatility profiles, and momentum scores. Data is filtered by ticker, date, and sector using predefined criteria. Missing values are handled via forward-filling or exclusion depending on the indicator's sensitivity.

Rule-based Scoring: Initial decision logic is implemented using deterministic rules. For example:

- *Momentum signal:* Generate a buy signal if the closing price exceeds the 50-day simple moving average and RSI is in the range 50–70.
- *Valuation screen:* Flag as overvalued if the P/E ratio is greater than the sector median by more than one standard deviation.
- *Risk classification:* Assign a "high risk" label if beta exceeds 1.5 or if recent earnings variance is above a defined threshold.

These rule sets are configurable and grounded in established financial heuristics.

Machine Learning Models: Where applicable, supervised models such as logistic regression or gradient-boosted decision trees may be used to estimate directional signals. Input features include moving averages, volatility, return profiles, and sector-adjusted valuation ratios. Outputs are binary or probabilistic classifications of investment suitability.

Strategy Output: The final output is a structured intermediate recommendation, for example:

- *Ticker:* MSFT
- *Action:* HOLD
- *Confidence:* 72%
- *Tags:* "Moderate Risk", "Overvalued"

This output is passed to the explanation layer, where it is contextualised and communicated to the end-user through a natural language interface.

The strict separation of decision logic from explanation ensures that recommendations are reproducible and auditable across time.

3.4 Explanation Generator

The explanation generator is responsible for converting structured strategy outputs into human-readable narratives. This layer interfaces with a large language model (LLM) to articulate the rationale behind each recommendation, ensuring the system remains explainable, and transparent.

Prompt Construction: Each recommendation generated by the strategy engine is converted into a structured prompt. This prompt contains:

- The recommended action (e.g. BUY, HOLD, SELL)
- Associated features (e.g. P/E ratio, RSI, volatility)
- Descriptive tags (e.g. "Overvalued", "Moderate Risk")
- Ticker metadata (e.g. sector, market cap, recent price movement)

The prompt template enforces a consistent narrative structure and ensures that only relevant, pre-computed data is surfaced to the LLM. No raw retrieval or document inference is performed at this stage.

LLM Invocation: The prompt is submitted to a locally hosted language model, with inference restricted to models that can be executed on a GPU with 8GB of VRAM. This design constraint promotes offline operability.

Post-processing and Attribution: The LLM output is parsed and linked to its underlying features. Numerical drivers are tagged and cross-referenced with the original strategy inputs to enable traceability. Where applicable, confidence indicators and relevant feature values are surfaced alongside the textual response.

Model Selection Rationale: For the purpose of this project, fine-tuning large language models is deliberately excluded due to its high resource demands and limited added value in the context of structured explanation generation. Instead, the system leverages pre-trained models served locally via the Ollama framework, which supports quantised variants of open-source architectures such as Mistral, Phi-2, and LLaMA2. These models strike a practical balance between performance and hardware feasibility, running comfortably on an 8GB VRAM GPU. The focus is placed on prompt design and response evaluation rather than model adaptation. This strategy aligns with the project's emphasis on transparency and reproducibility, while avoiding the complexity and risks associated with custom model training.

By isolating the explanation function from decision-making logic and constraining it to local, auditable models, this design ensures the advisory system remains intelligible, trustworthy, and compliant with practical deployment constraints.

3.5 User Interface

The user interface (UI) serves as the primary point of interaction between the end-user and the advisory system. It is designed to surface system outputs in a clear and context-aware format while abstracting away the technical complexity of the underlying architecture.

Interface Design Principles: The UI is guided by the principles of usability, transparency, and minimalism. It aims to provide information density without cognitive overload, and supports both exploratory queries and structured evaluations.

Interaction Model: The primary mode of interaction is a ticker selector and a time window. Queries are routed through the backend pipeline and responses are returned in astructured, explanation-rich format.

Presentation Layer: Output is displayed in modular response cards, with each card corresponding to a company. Each card includes:

- The recommended action (BUY, HOLD, SELL)

- A summarised rationale generated by the explanation layer
- Key features (e.g. P/E ratio, RSI, beta) highlighted inline
- Option to expand for full explanation and numerical context

Cards are sorted based on confidence or relevance, and users can collapse or pin cards for comparison.

Technical Implementation: The frontend is built using a lightweight web framework such as Streamlit. This ensures rapid prototyping, cross-platform compatibility, and support for secure API calls to backend services. The UI is deployed as a local web application accessible via browser.

Responsiveness and Accessibility: The interface is optimised for both desktop and mobile devices. Layouts use responsive containers and typography is chosen for readability. Accessibility guidelines such as contrast ratios and semantic labelling are followed to ensure inclusivity.

This interface completes the user-facing layer of the system, enabling seamless communication of insights derived from structured analysis and local language model generation.

3.6 Backtesting & Evaluation

This layer is responsible for assessing the performance, consistency, and interpretability of the GenAI Advisor system. Evaluation is conducted along two axes: (i) financial effectiveness of the recommendations and (ii) fidelity and stability of the generated explanations.

Backtesting of Strategy Output: The system supports retrospective evaluation of strategy outputs using historical financial data stored in MariaDB. Given a time window and a set of tickers, the engine replays strategy decisions based on data available at the time. The simulated trades are benchmarked using standard performance metrics such as:

- Cumulative return
- Sharpe ratio
- Maximum drawdown
- Hit rate (proportion of correct directional calls)

These simulations validate whether the rule-based and ML-driven components produce economically viable signals under realistic conditions.

Evaluation of Explanation Consistency: Because the system includes a generative component, it is essential to assess the consistency and fidelity of explanations. For a given strategy output, repeated LLM invocations are evaluated for semantic agreement and factual stability. Techniques include:

- *Stability under identical prompts:* The same input is submitted multiple times to check for drift.
- *Paraphrase testing:* Slightly altered prompts are compared for explanation robustness.
- *Key fact alignment:* Extracted metrics in the explanation are compared against known strategy outputs.

These tests ensure that the narrative remains faithful to the underlying numerical reasoning.

User Feedback Loop: The UI logs user reactions to recommendations through binary feedback (e.g. thumbs up/down). Feedback is stored with associated prompt and strategy metadata, enabling post-hoc review and potential retraining of classification thresholds or strategy weights.

Qualitative Auditing: In addition to automated tests, a sample of responses is manually audited to assess:

- Clarity of the natural language explanation
- Justification coverage (i.e., do the reasons align with the action?)
- Absence of hallucinated content or misleading terminology

This step supports ethical AI principles by ensuring outputs remain comprehensible and appropriate for retail consumption.

This evaluation layer closes the loop between system logic, narrative explanation, and end-user experience. It provides the foundations for continuous improvement while maintaining interpretability and trust.

4 Implementation (2399/2500 words)

4.1 Overview

The GenAI Advisor was developed using a Minimum Viable Product (MVP) approach, structured around four modular components: data ingestion, strategy engine, explanation generation, and a Streamlit-based user interface. The system was designed to be fully offline, prioritising data privacy, transparency, and reproducibility. Initial efforts focused on building a thin end-to-end pipeline to demonstrate feasibility and guide future sprints.

4.2 System Architecture

The overall system architecture comprises:

- **Frontend:** Streamlit application for user interaction
- **Backend:** Modular Python packages handling data ingestion, strategy evaluation, and explanation generation
- **LLM Inference:** Mistral 8B model run locally via Ollama
- **Testing and Evaluation:** Pytest-driven unit tests and backtesting utilities

A modular layout allowed for parallel development and easier testing of individual system components. All source code and evaluation assets are available at `https://github.com/andreasmallios/genaiadvisor`, ensuring transparency, reproducibility, and compliance with open-source principles.

4.3 Development Process

An MVP-first approach guided the early stages. Agile-inspired sprints were used to iteratively add functionality and refine components. Git was used for version control, with feature branches for modular additions and frequent integration testing.

4.4 Key Modules and Features

4.4.1 Data Ingestion

Equity price data is retrieved from Yahoo Finance via the `yfinance` API, using a custom function designed to streamline data access and ensure reproducibility. The function `fetch_ticker_data` encapsulates logic for both online retrieval and local caching. When data for a given ticker symbol already exists in the local cache (CSV format), it is loaded directly using `pandas.read_csv`, with the `Date` column parsed into a datetime index to facilitate time series analysis. This not only minimises reliance on external API calls, but also enhances the determinism of backtests.

In the absence of cached data, the function instantiates a `yfinance.Ticker` object and invokes `Ticker.history()` with a default window of one year and daily frequency. Metadata fields unrelated to price analysis (e.g., `Dividends`, `Stock Splits`) are explicitly excluded to avoid downstream noise. The retrieved data frame is then persisted locally to disk, ensuring future runs use identical input data unless manually refreshed.

All data is indexed by date, with explicit naming of the index to `Date` for consistency across modules. Additionally, the local directory used for caching is dynamically resolved using Python's `os.path` utilities, enabling portability across environments. Although not shown directly in the function, timezone consistency is enforced at the point of usage through `tz_localize`, ensuring temporal coherence across the ingestion and backtesting pipeline.

4.4.2 Strategy Engine

The strategy engine was originally implemented using a simple moving average (SMA) crossover, a well-known momentum indicator. However, it was later modularised and extended into a hybrid ensemble system, integrating both traditional technical indicators and machine learning predictions. Each strategy component was abstracted into its own module with a shared interface returning a `recommendation`, `reason`, and `date`, thereby enabling flexible composition and detailed introspection.

The indicators implemented include SMA crossover, Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), Bollinger Bands, and the Stochastic Oscillator. Each module performs well-defined signal computation:

- **SMA crossover**: Computes short- and long-term SMAs (50-day and 200-day by default) and identifies bullish or bearish crossovers as trading signals.
- **RSI**: Quantifies momentum and detects overbought or oversold regimes, issuing BUY signals when RSI falls below 30 and SELL when above 70.
- **MACD**: Uses exponential moving averages to detect momentum shifts via signal-line crossovers.
- **Bollinger Bands**: Employs a 20-day SMA and standard deviation bands to detect volatility-based breakouts.
- **Stochastic Oscillator**: Evaluates recent closing prices relative to the high-low range over a rolling window to spot potential reversals.

Additionally, a supervised machine learning layer was incorporated using both Random Forest and Logistic Regression classifiers. These were trained on historical technical features, including SMA differentials, RSI, MACD, returns, and volume, with future returns used as labels. Predictions from both models are aggregated and translated into BUY, HOLD, or SELL signals.

All individual signals are passed to a meta-layer defined in `engine.py`, where a weighted voting mechanism combines the recommendations. Each strategy is assigned a normalised weight, and the aggregate score is computed to produce the final advisory signal. A confidence score is also generated, which allows the user to assess the strength of the consensus recommendation.

This modular and explainable architecture enhances the system's adaptability, allowing future indicators or models to be integrated with minimal changes to the existing pipeline.

4.4.3 ML Implementation

The machine learning component of the GenAI Advisor transitioned from baseline `scikit-learn` classifiers to a TensorFlow-based neural network to improve predictive performance while remaining fully auditable. Initial iterations used Random Forest and Logistic Regression to establish a benchmark, leveraging engineered features such as SMA differentials, RSI, MACD, returns, and volumes. Models and scalers were persisted to the `models/` directory to ensure reproducibility.

The final implementation employs a feedforward neural network built with TensorFlow/Keras, trained on combined historical data across all supported tickers. The network comprises two hidden layers (64 and 32 units) with ReLU activations, dropout regularisation, and a softmax output for three classes (BUY, HOLD, SELL). Labels were derived from forward ten-day price movements, with class weighting applied to mitigate label imbalance. Early stopping was used to prevent overfitting, and both the trained model (`tf_model.h5`) and scaler were cached for inference.

This classifier is invoked within the `engine.py` pipeline via the `tensorflow_classifier_signal` function and integrated into the weighted ensemble in `generate_combined_recommendation`. Here, its output complements rule-based signals (e.g., SMA, RSI, MACD) under a defined weighting scheme. This design retains transparency by exposing model-driven rationales alongside deterministic indicators, aligning predictive improvements with the project's explainability objectives.

For inference, TensorFlow was explicitly configured to run on the CPU (`CUDA_VISIBLE_DEVICES=-1`) to avoid contention with the Mistral LLM, which occupies the GPU for explanation generation. Early experiments revealed that concurrent GPU usage by both TensorFlow and Mistral frequently exhausted available VRAM, causing inference failures. Running TensorFlow on the CPU mitigated these conflicts while maintaining acceptable latency for individual predictions. Although training was GPU-accelerated for efficiency, this CPU-only inference design aligns with the offline-first architecture and ensures stable operation on hardware-constrained environments without compromising model availability or reproducibility during backtesting and interactive usage.

4.4.4 Explanation Generator

To enhance transparency and user trust, the system includes a natural language explanation generator powered by a locally hosted instance of the Mistral 8B model, served via Ollama. This design deliberately avoids reliance on proprietary cloud APIs, enabling offline inference and preserving data privacy. Communication with the model is facilitated through Python's `subprocess` module, which programmatically invokes the model via command-line interface.

Prompts are dynamically constructed using the metadata associated with each final recommendation. The resulting prompt embeds a structured format with three components: *Summary*, *Reason*, and *Disclaimer*. These guide the model to produce consistent, human-readable rationales tailored for non-specialist audiences. The prompt instructs the model to contextualise signals such as SMA crossover, RSI, MACD, Bollinger Bands, Stochastic Oscillator, and ML Classifier outputs. It also emphasises educational tone, British English, and a strict 300-word limit to maintain clarity and concision.

By embedding these constraints in the prompt and invoking the model locally, the system achieves explainability without introducing dependencies that compromise reproducibility or user autonomy. Errors in generation are handled gracefully, returning informative fallback messages to aid debugging.

4.4.5 Interface

The user interface is developed using Streamlit, a Python-based rapid application framework tailored for data-centric interfaces. Streamlit was chosen over alternatives such as Flask or Dash due to its simplicity and tight integration with Python's data stack, enabling rapid iteration.This decision facilitated fast prototyping and interactive visualisation, crucial during iterative development and debugging. The application enables users to input stock tickers, retrieve historical market data, view system-generated BUY, HOLD, or SELL recommendations, and inspect the rationale behind each decision.

The main interface consists of two parts: a portfolio overview and a detailed ticker analysis. The portfolio table displays summarised recommendations for all tickers within the watchlist, fetched from a CSV file.

For deeper inspection, users may select a specific ticker, upon which the interface renders price history, a modular signal breakdown, and a natural language explanation. This layered interaction supports transparency and interpretability.

Behind the scenes, the app integrates tightly with the data ingestion pipeline, strategy engine, and explanation generator. Recommendations and signal details are fetched dynamically and

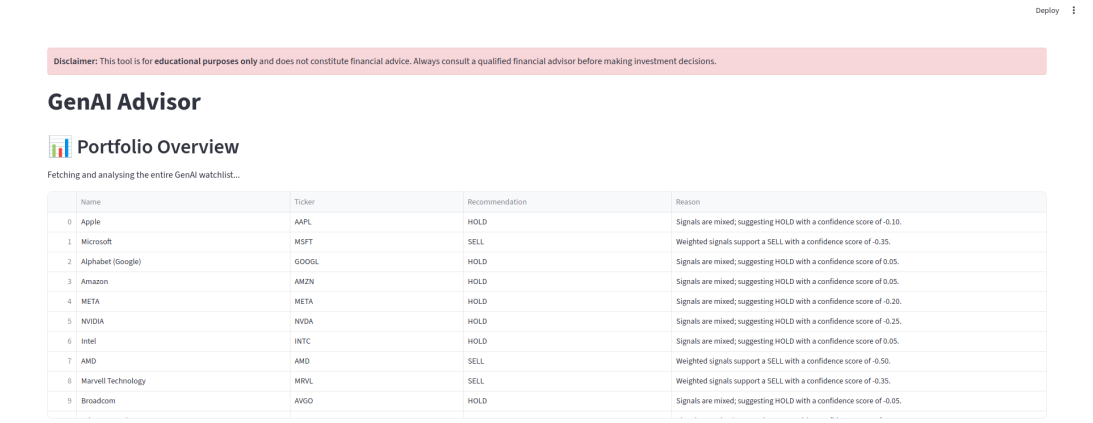


Figure 3: Portfolio overview, showing entire portfolio and recommendations.

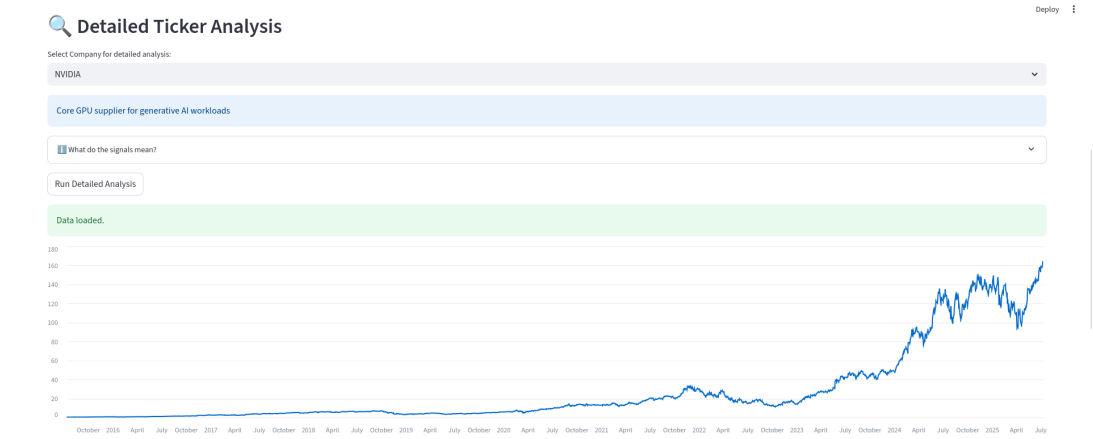


Figure 4: Ticker-level analysis view, showing historical price, technical signals, and explanation.

rendered as structured text. Visual aids, such as historical closing price charts, are plotted using `matplotlib` and embedded into both the interface and downloadable PDF reports. This PDF generation, facilitated by the `fpdf` library, enables offline review and archiving.

Importantly, the interface includes clear disclaimers and educational prompts, ensuring compliance with ethical guidance and reinforcing the tool's non-advisory role. Collectively, the Streamlit-based interface serves not only as a functional front-end, but also as a pedagogical instrument that demystifies algorithmic financial decision-making for non-experts.

4.4.6 Testing Infrastructure and Evaluation Support

Reliability was enforced through a comprehensive `pytest` suite following Test-Driven Development (TDD) principles. Core modules—including data ingestion, technical indicators (SMA, RSI, MACD, Bollinger Bands, Stochastic Oscillator), ensemble logic, and explanation generation—were covered by dedicated tests validating type integrity, output structure, and signal consistency across edge cases.

Integration tests (`test_engine_call.py`) confirmed correct aggregation of weighted strategies into deterministic recommendations. The explanation generator was tested for compliance with its Summary–Reason–Disclaimer format and for robustness against subprocess errors. Pytest was executed with `PYTHONPATH=.` to resolve imports in the nested project structure.

Complementing unit testing, a batch backtesting framework replays strategy outputs against historical data, comparing predictions to forward price movements over configurable horizons

(e.g., 30 days). Metrics such as hit rate, Sharpe ratio, and maximum drawdown were computed, with results logged to timestamped CSVs for notebook-based visualisation and further analysis. This dual approach—functional testing and historical simulation—ensured both software integrity and empirical grounding.

4.4.7 Backtesting Support

Backtesting validated the economic viability of recommendations under realistic market conditions. Using cached historical data, the system generated time-local recommendations which were benchmarked against subsequent price performance. Performance metrics included cumulative return, Sharpe ratio, maximum drawdown, and confusion matrix-derived classification accuracy.

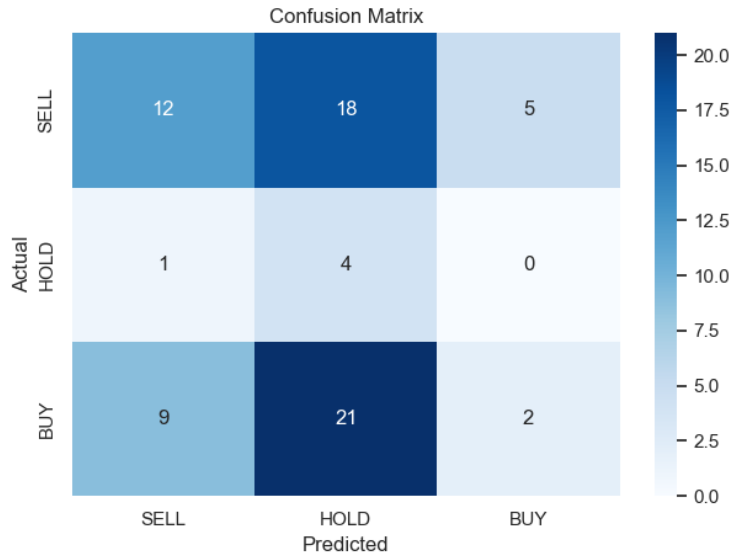


Figure 5: Confusion matrix of classifier predictions versus observed outcomes during evaluation.

This evaluation pipeline provided reproducible, timestamped results consistent with algorithmic trading validation best practices (1).

4.4.8 Logging and Monitoring

Execution logging was implemented using a lightweight CSV-based approach via `logger.py`. Each interaction, including ticker requests, generated recommendations, and explanation invocations, was appended to `logs/usage_log.csv`. This file records timestamps, user inputs, and resulting signals, enabling both auditability and post-hoc debugging. During development, these logs were instrumental in diagnosing runtime errors and tracing data flow across modules. For example, discrepancies between model predictions and ensemble outputs were reconciled by cross-referencing log timestamps with feature generation events. This persistent logging mechanism also facilitated informal monitoring of system behaviour during backtesting and exploratory testing, thereby supporting reproducibility and transparency without requiring external monitoring frameworks.

4.5 Environment and Dependency Management

A Conda-based environment was used to manage dependencies and ensure reproducibility across different development machines. All required packages, including `pandas`, `yfinance`, `tensorflow`, `scikit-learn`, and `streamlit`, were defined within a single `requirements.yml` file stored at

the repository root. This configuration allowed the environment to be recreated deterministically with a single command. Pinning library versions reduced the risk of incompatibilities during integration or evaluation. The approach also simplified transitions between CPU-only and GPU-enabled environments, particularly relevant for TensorFlow training versus inference runs, where GPU acceleration was explicitly disabled for deterministic CPU-bound predictions. This standardised environment setup was crucial for aligning development, testing, and report reproduction in an offline-first context.

4.6 Code Structure and Repository Organisation

The repository is divided into two primary components: the `code/` directory, which contains the full implementation of the GenAI Advisor, and the `report/` directory, which holds the dissertation source files.

- `code/`: Implements the system, organised into:
  - `app/` – Core application logic, including data ingestion, strategy engine, explanation generator, and supporting utilities.
  - `data/` – Cached market data (CSV format) for reproducibility.
  - `models/` – Trained machine learning models and scalers.
  - `evaluation/` – Backtesting outputs and results.
  - `logs/` – Usage logs generated during execution.
  - `tests/` – Pytest suites aligned with application modules.
  - `notebooks/` – Prototyping and exploratory analysis in Jupyter.
  - `streamlit_app/` – User interface built in Streamlit.
- `report/`: Contains the dissertation materials:
  - Chapter-wise `.tex` files and the bibliography.
  - Figures and visualisations in `assets/`.
  - Compiled `main.pdf` and LaTeX build artefacts.

This structure separates implementation from documentation while maintaining cohesion within each domain, enabling systematic development and transparent academic reporting.

4.7 Implementation Challenges

Several technical challenges emerged during implementation, each requiring targeted debugging and design intervention.

- **Timezone mismatch:**
  Historical price data retrieved via Yahoo Finance lacked explicit timezone metadata, which introduced ambiguity when slicing datasets for backtesting. This was particularly problematic when aligning cut-off dates across modules. To mitigate this, all datetime indices were explicitly localised to the `America/New_York` timezone using the `tz_localize` method. This normalisation ensured consistent interpretation of trading days and eliminated subtle off-by-one errors during data slicing.
- **CSV integrity:** Cached datasets were initially stored as CSV files to improve reproducibility and reduce API reliance. However, inconsistencies in how indices were handled during `to_csv()` and `read_csv()` operations led to misaligned time series and malformed DataFrames. This was addressed by explicitly setting the `index_label` parameter during

write operations and enforcing `index_col="Date"` with `parse_dates=True` during reads. These changes ensured referential consistency across modules and safeguarded downstream processing.

- **Subprocess inference:** Local model inference via the `ollama` CLI introduced challenges in handling prompt formatting, encoding, and subprocess errors. The prompt had to be programmatically composed with structured constraints while maintaining compatibility with standard input streams. Errors during model invocation (e.g., non-zero return codes) were caught and reported via Python's `subprocess.CalledProcessError` exception handling, allowing the interface to degrade gracefully with informative fallback messages.

- **Test discovery:** During unit test integration, Pytest failed to resolve relative imports due to the project's nested directory structure. This was resolved by invoking Pytest with the environment variable `PYTHONPATH=.`, thereby ensuring that all modules were discoverable relative to the project root. This solution avoided the need for path rewrites or package restructuring.

- **Backtesting leakage:** Initial backtest prototypes inadvertently included post-cutoff data in model inputs, leading to optimistic and invalid evaluations. This was corrected by enforcing a strict temporal cutoff: recommendations were generated using only data up to the target date, while subsequent price movements were isolated for outcome analysis. This approach ensured methodological rigour and prevented data leakage, a critical consideration in empirical evaluation.

These challenges highlight the complexity of integrating financial data pipelines, local LLM inference, and modular testing in a cohesive system. Their resolution contributed directly to the robustness and reproducibility of the final implementation.

4.8 Summary

This implementation unites modular financial analytics, XAI-driven explanations, and reproducible evaluation workflows in a single offline system. By adhering to robust software engineering practices, leveraging open-source tooling, and prioritising explainability, the GenAI Advisor achieves technical depth, transparency, and educational value suitable for retail investor contexts.

5 Evaluation (2099/2500 words)

5.1 Overview

This section evaluates the performance, reliability, and transparency of the proposed signal generation and recommendation framework. We begin with a quantitative analysis of the model's classification accuracy, consistency across market conditions, and outcome alignment. This is followed by an interpretability study leveraging SHAP values to understand the influence of individual input features. We also examine the quality and usefulness of natural language explanations generated by a local LLM, supported by a manual annotation template. The evaluation concludes with a discussion of current limitations and an overall assessment of the system's strengths and areas for further refinement.

5.2 Evaluation Design

The evaluation process was grounded in the incremental development of the system. As the MVP expanded from a single-indicator engine (SMA crossover) to a modular, ensemble-based architecture, the system was structured to support isolated testing of each signal module. Indicators such as RSI and MACD were added as independent Python modules, allowing for precise attribution and targeted validation.

This modular design also enabled a batch evaluation process, in which the system was tested across multiple tickers and historical timestamps using a strict cut-off methodology. Outputs—including recommendations, explanations, and realised price changes—were logged to structured CSV files. These formed the basis for subsequent analysis.

Supporting Jupyter notebooks were developed to visualise the relationship between recommendation types and price movement outcomes. Visual tools such as histograms and boxplots were used to inspect signal performance, and early empirical trends (e.g., the hit rate of BUY signals) informed further refinement of strategy logic and prompt design.

5.3 Methodology

The evaluation employed a cut-off-based backtesting framework to simulate the system's recommendations in a temporally realistic setting. For each ticker, historical data was truncated at a fixed cut-off date, beyond which no future information was accessible to the strategy engine. A 30-day lookahead horizon was then used to observe forward price movements and assess the quality of the generated recommendation.

Batch evaluations were conducted across a curated list of generative AI-related equities and multiple monthly cut-off dates. This enabled temporal robustness checks and cross-sectional analysis of the model's behaviour. To ensure determinism and reproducibility, all equity data was fetched from local CSV caches generated during prior ingestion runs, thereby avoiding API variability or data drift.

This methodology allowed for an ex-post evaluation of the advisory system while respecting realistic information boundaries and supporting statistical aggregation across time and instruments.

5.4 Implementation of the Backtesting Pipeline

The backtesting pipeline was implemented as a modular Python batch system. Its core function, `backtest_ticker()`, accepts a stock ticker and a historical cut-off date as input. It slices the cached price data to include only records available up to that date, thereby simulating a realistic information boundary. A composite recommendation is generated using the ensemble strategy engine, and an accompanying explanation is produced via the local language model.

To assess recommendation performance, the system fetches post-cutoff price data over a fixed lookahead horizon (30 days), and calculates the forward percentage return. The results are compiled into structured dictionaries containing the ticker, as-of date, recommendation, rationale, explanation, and forward price change. This design ensures strict temporal separation between signal generation and performance evaluation, eliminating data leakage.

A higher-level orchestration script, `batch_backtesting.py`, automates this process over multiple tickers and dates. It reads tickers from a CSV watchlist and iterates through a configurable set of evaluation dates. Outputs from each backtest run are logged to timestamped CSV files in a dedicated evaluation directory, enabling reproducibility and version tracking.

The resulting datasets are subsequently explored using Jupyter notebooks. These notebooks facilitate performance visualisation, signal attribution, and exploratory metrics computation. This architecture supports both batch scalability and transparent post hoc analysis.

5.5 Quantitative Signal Performance

This subsection evaluates the predictive efficacy of the system's generated trading signals—BUY, HOLD, and SELL—using both statistical visualisation and classification metrics.

Distribution Analysis. The violin and box plots (Figures 6 and 7) show the distribution of 30-day forward price changes conditioned on the system's recommendations. The SELL group exhibits the most left-skewed distribution, indicating that negative returns were most prevalent following these signals.

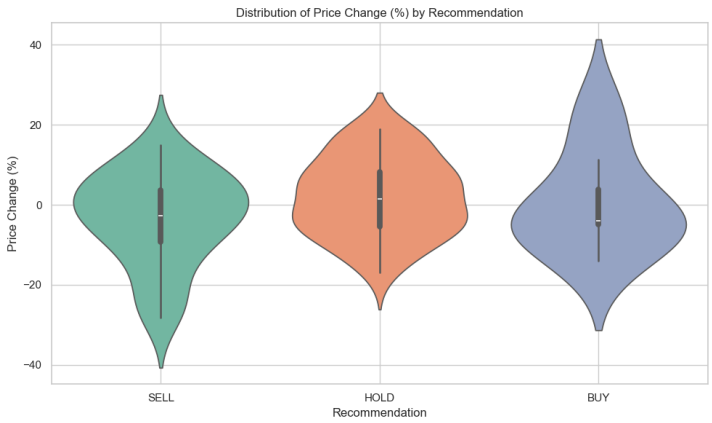


Figure 6: Distribution of Price Change (%) by Recommendation Type

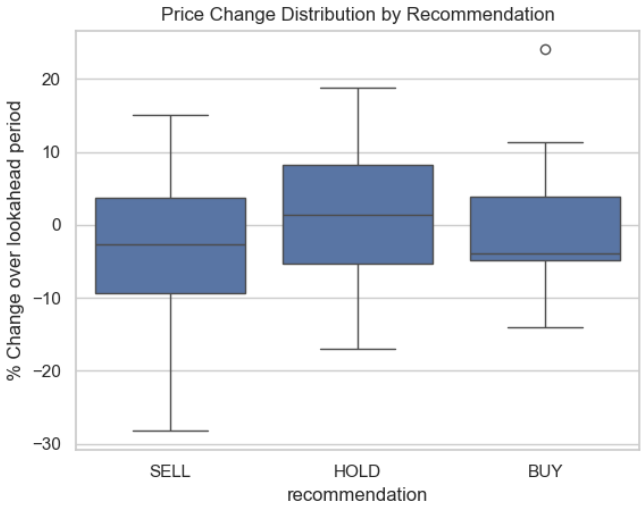


Figure 7: Boxplot of Price Change by Recommendation

Conversely, the BUY group shows a higher tail on the positive side, albeit with greater dispersion. HOLD recommendations, which dominate the output distribution (see Figure 8), tend to centre around zero return, aligning with the system's intent to recommend HOLD in uncertain or balanced cases.

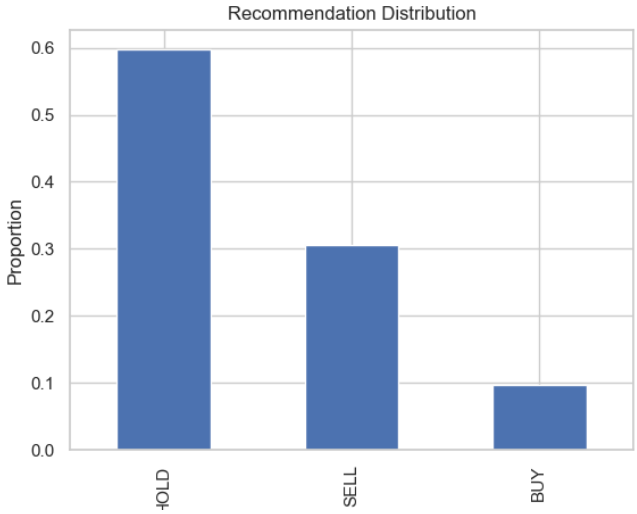


Figure 8: Recommendation Type Distribution

Recommendation Frequency. Figure 8 illustrates the proportion of recommendations. HOLD signals dominate, comprising nearly 60% of total outputs, followed by SELL and then BUY. This asymmetry could stem from conservative aggregation logic in the signal fusion engine or class imbalance during model training.

Prediction Accuracy and Confusion Matrix. Performance metrics derived from backtest outcomes are summarised using a confusion matrix and classification report (Figures 9 and 10). HOLD shows high recall (0.80), indicating the system often correctly predicts when no significant directional movement is expected. However, BUY and SELL classes suffer from lower recall,

suggesting many directional changes are incorrectly classified as HOLD. Precision is highest for SELL (0.55), which implies a relatively high proportion of SELL predictions are correct.

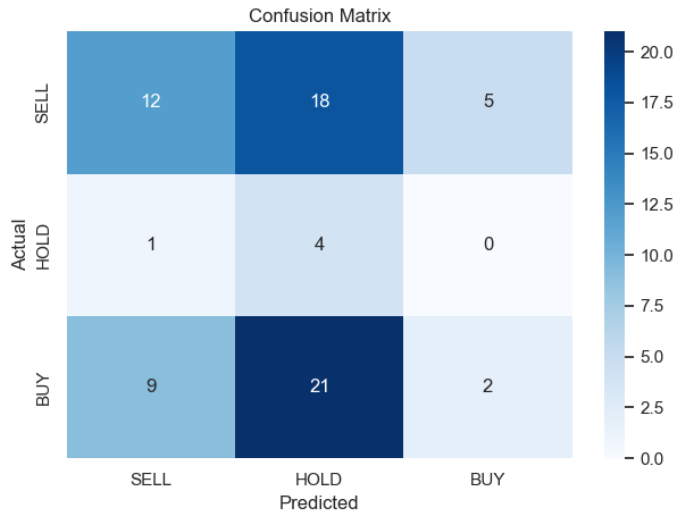


Figure 9: Confusion Matrix by Recommendation Type

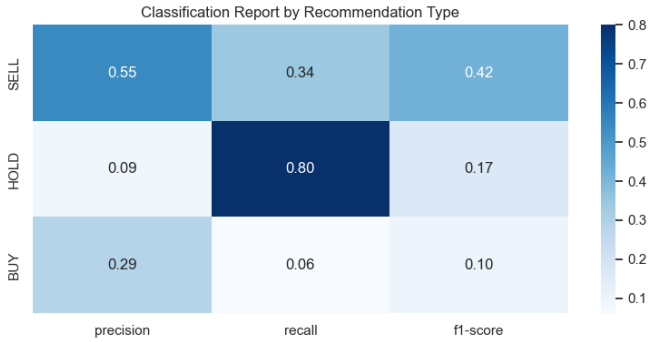


Figure 10: Classification Metrics for Each Recommendation

Outcome Breakdown. Figure 11 details how often each recommendation led to correct, incorrect, or neutral outcomes. Most HOLDs are neutral, while SELLs are more likely to be correct than incorrect. BUY signals remain underrepresented and less accurate, warranting further tuning.

Overall, the model demonstrates competent neutral positioning, but directional recommendations require refinement—particularly in improving recall for BUY signals and balancing class representation in both training and orchestration stages.

5.6 Machine Learning Interpretability

To provide insight into the decision-making process of the TensorFlow-based classifier, we employed SHAP (SHapley Additive exPlanations) values as a post hoc interpretability technique. SHAP values help attribute the contribution of each feature towards the model's output in a consistent and theoretically grounded manner.

Figure 12 illustrates the average absolute SHAP values across the three output classes—SELL (0), HOLD (1), and BUY (2). These values indicate the magnitude of impact each input feature has on the model's predictions, averaged across a sample of recent AAPL data points.

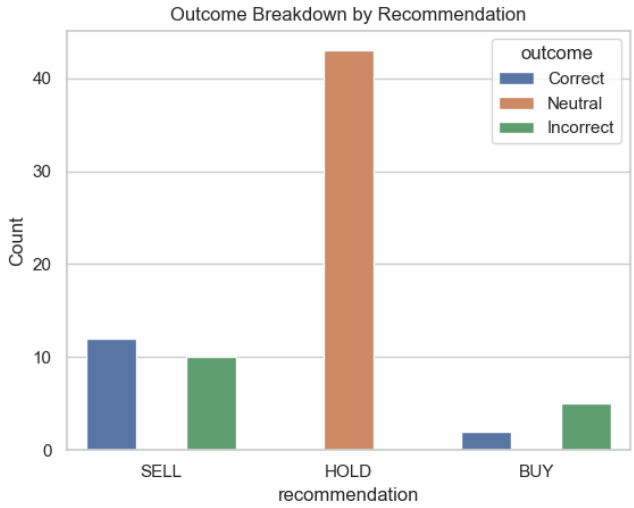


Figure 11: Outcome Breakdown by Recommendation Type

The results highlight the dominant role of short- to medium-term technical indicators. In particular, `SMA_diff` (the difference between the 10-day and 50-day simple moving averages) exerts the greatest influence across all classes. This suggests the model heavily relies on trend momentum for directional prediction. Return-based features (`Return_1d`, `Return_5d`, `Return_10d`) and `MACD` also play substantial roles, especially in distinguishing BUY signals. Conversely, features such as `Volume` and `SMA_10` appear to contribute minimally.

Overall, the SHAP analysis reveals that the model bases its forecasts on interpretable technical signals rather than obscure patterns. This reinforces confidence in the model's behaviour, offering transparency critical for user trust in financial decision support systems.

5.7 Evaluation of Explanations

To evaluate the effectiveness and clarity of the explanations generated by the large language model (LLM) for each recommendation, we conducted a structured manual assessment. The objective was to determine whether these natural language explanations enhance the transparency and interpretability of the machine learning (ML) predictions, particularly for end-users with financial domain expertise.

Each explanation accompanying a predicted signal (BUY, HOLD, or SELL) was reviewed independently across several qualitative dimensions. These dimensions were selected to reflect the dual requirements of technical correctness and human interpretability. For consistency, all evaluations used the following rubric:

Example Evaluation Entry:

- **Ticker:** AAPL
- **Date:** 2025-07-01
- **Recommendation:** BUY
- **LLM Explanation:** "The model predicts a BUY signal due to a strong positive MACD crossover, increasing 5-day returns, and rising RSI, all indicating bullish momentum."
- **Evaluator Comments:** The explanation is clear, accurate, and tailored. It could be improved by including model confidence or a mention of recent volatility.

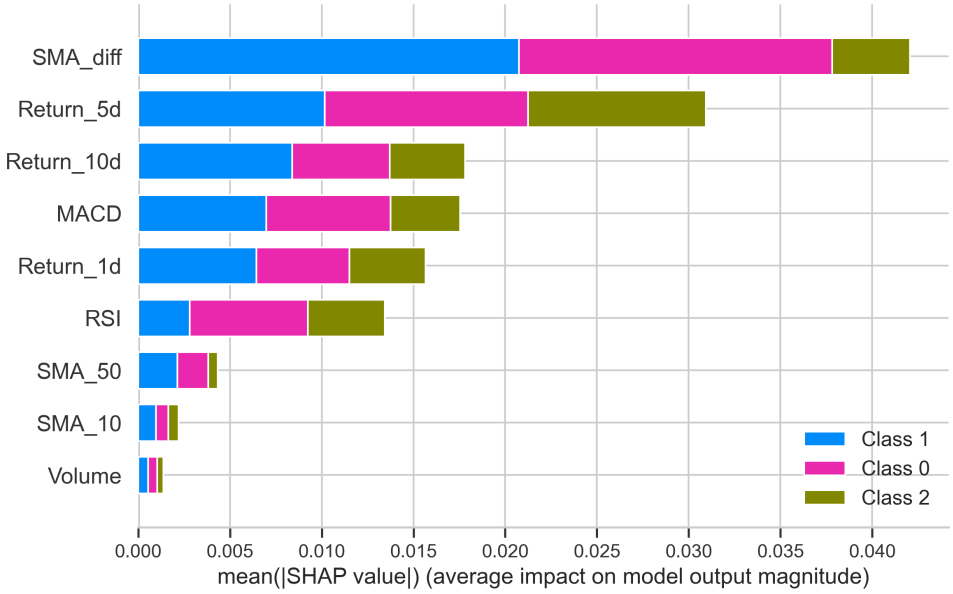


Figure 12: SHAP Summary Plot Showing Average Feature Impact per Class

Table 1: Manual Evaluation Template for LLM Explanations

Criterion	**Score (1–5)**	**Notes**
Relevance		Does the explanation directly relate to the input features or context influencing the recommendation?
Specificity		Is the explanation tailored to the specific case, or does it contain vague or generic language?
Correctness		Are the financial and technical statements accurate based on the model's behaviour?
Interpretability		Is the explanation understandable to a non-technical financial analyst or user?
Justifiability		Does the explanation appear to align with known signals or the model's inferred logic (e.g., SHAP insights)?
Actionability		Would the explanation help an analyst or investor make a better-informed decision?

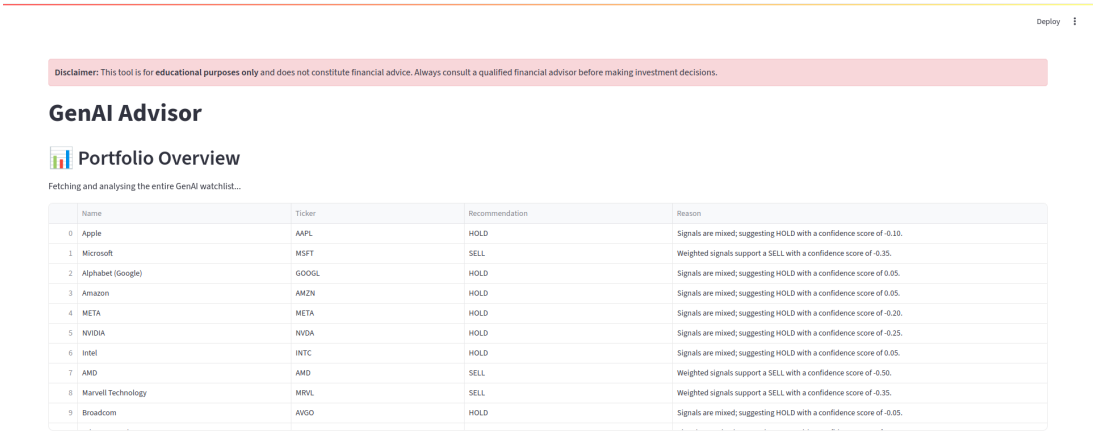
Evaluations using this framework revealed that while most explanations were interpretable and relevant, some lacked depth in justifiability or failed to clearly connect the language to the underlying numerical signals. In future iterations, incorporating SHAP-derived insights into the explanation prompt may improve quality and consistency across recommendations.

5.8 Frontend Evaluation

To ensure accessibility and user engagement, the GenAI Advisor features an interactive web-based frontend built with Streamlit. The application is structured around two primary views: the portfolio-level overview and the detailed ticker analysis, each designed to support both technical users and novices seeking intuitive insights.

Portfolio Overview Figure 13 shows the application's landing view, where all GenAI-related companies are automatically analysed. Each row displays a ticker symbol, its full name, the current recommendation (BUY, HOLD, or SELL), and a succinct reason summarising the signal

consensus. This view is designed for a fast scan of the market sentiment across the watchlist.



	Name	Ticker	Recommendation	Reason
0	Apple	AAPL	HOLD	Signals are mixed; suggesting HOLD with a confidence score of -0.10.
1	Microsoft	MSFT	SELL	Weighted signals support a SELL with a confidence score of -0.35.
2	Alphabet (Google)	GOOGL	HOLD	Signals are mixed; suggesting HOLD with a confidence score of 0.05.
3	Amazon	AMZN	HOLD	Signals are mixed; suggesting HOLD with a confidence score of 0.05.
4	META	META	HOLD	Signals are mixed; suggesting HOLD with a confidence score of -0.20.
5	NVIDIA	NVDA	HOLD	Signals are mixed; suggesting HOLD with a confidence score of -0.25.
6	Intel	INTC	HOLD	Signals are mixed; suggesting HOLD with a confidence score of 0.05.
7	AMD	AMD	SELL	Weighted signals support a SELL with a confidence score of -0.50.
8	Marvell Technology	MRVL	SELL	Weighted signals support a SELL with a confidence score of -0.35.
9	Broadcom	AVGO	HOLD	Signals are mixed; suggesting HOLD with a confidence score of -0.05.

Figure 13: Portfolio-Level Recommendation Summary

Detailed Ticker Analysis Upon selecting a specific company, users are presented with a more granular analysis panel (Figure 14). The system fetches historical stock data (defaulting to 10 years) and visualises the price trend. Subsequently, a hybrid strategy engine computes the combined recommendation, which is further broken down into individual signal components. Each indicator—such as SMA crossover, RSI, MACD, Bollinger Bands, and others—contributes an individual recommendation with a natural language justification.

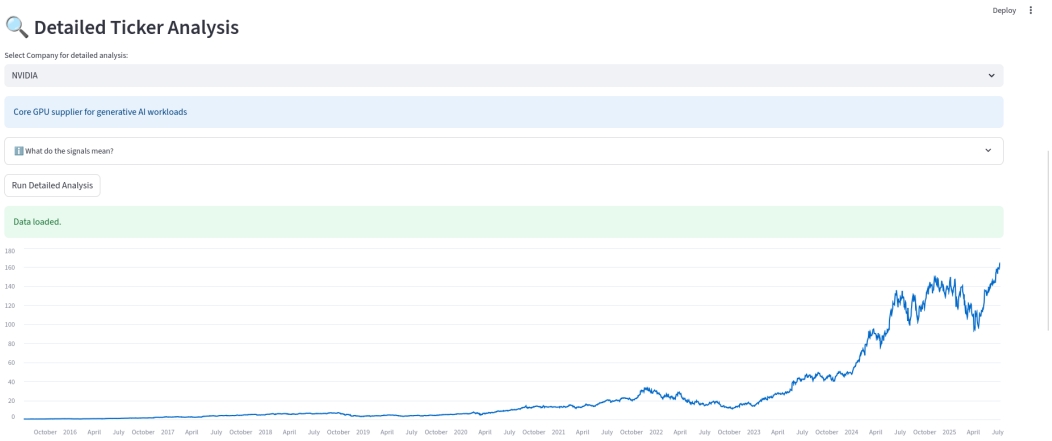


Figure 14: Detailed Ticker Analysis View

Explanation Generation and Export Figure 15 showcases the final segment of the analysis: a large language model (LLM)-generated explanation that contextualises the decision. The system measures and displays the explanation latency, stores the interaction in a log file, and allows users to download a PDF report for offline review or sharing. The inclusion of export functionality and speed metrics enhances both usability and transparency.

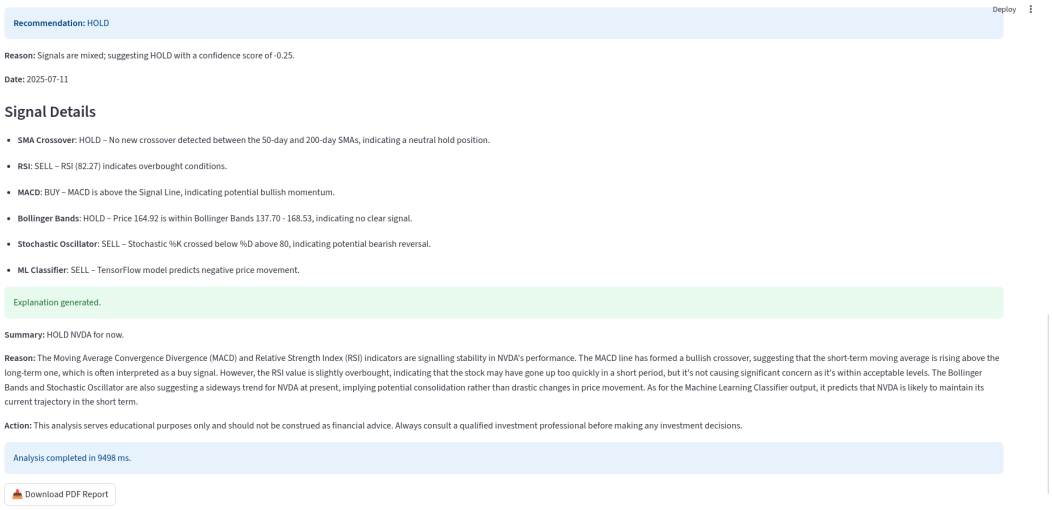


Figure 15: Model Explanation, Signal Breakdown, and Export Features

User Experience Considerations The frontend emphasises clarity and educational value. Tooltips, expandable signal descriptions, and consistent colour schemes help guide the user. Furthermore, a disclaimer is persistently shown to remind users that the tool is intended strictly for educational use and does not replace professional financial advice.

Overall, the frontend complements the backend analytics by delivering insights in an accessible, trustworthy, and aesthetically clean format.

Limitations. Some latency spikes may occur when the local LLM (e.g., Mistral) is under GPU memory contention, or if data retrieval is delayed. Additionally, the current application design assumes end-users are familiar with basic financial concepts, which may limit accessibility for novices.

Future Enhancements Planned improvements include persistent user session logging, mobile responsiveness, SHAP visualisation integration, and model versioning metadata in downloadable reports.

5.9 Limitations

Despite the promising performance and comprehensive evaluation of the machine learning (ML) signal generation and explanation pipeline, several limitations must be acknowledged.

Model Performance and Generalisation. The TensorFlow-based classifier demonstrates moderate performance, especially in correctly identifying BUY signals, which remain underrepresented and less accurate compared to SELL and HOLD. This class imbalance introduces bias into the model and limits its generalisability to unseen or volatile market conditions. While class weights and data augmentation were partially addressed, further refinement is required to balance predictive power across all classes.

Data Scope and Market Conditions. The training data is based on historical daily prices of a limited set of US technology stocks over a 10-year period. This constraint may reduce the model's robustness to macroeconomic shifts, structural breaks, or non-tech sector behaviours. The use of fixed lookback and forecast horizons may also fail to capture dynamics with varying time lags or temporal dependencies.

Explanation Fidelity. Although SHAP values provide local interpretability and the LLM-generated narratives offer human-readable justifications, the explanations may not always faithfully represent the underlying model mechanics. SHAP explanations are based on approximations and are sensitive to input distributions and feature correlation. Meanwhile, the LLM explanations—while fluent—may introduce hallucinations or overstate causal relationships unless tightly grounded in model outputs.

Subjectivity in Manual Evaluation. The assessment of explanation quality relies on human judgement, which introduces subjectivity and potential inconsistencies across reviewers. While the evaluation rubric helps standardise criteria, different levels of domain expertise may influence scoring and interpretation.

Resource Constraints. The interpretability pipeline, especially SHAP computation for deep learning models, remains computationally expensive and was limited to a subset of 100 samples per ticker. This restricts the breadth of explanation analysis and may obscure insights that emerge in longer-term or higher-volume applications.

Absence of Real-Time Testing. All results reported are based on historical backtesting. The model has not yet been deployed in a live market environment where slippage, transaction costs, and execution latency would impact real-world performance. Without live validation, claims of profitability or decision support remain speculative.

Addressing these limitations in future work will be essential for enhancing the system's reliability, fairness, and operational viability in real-world financial decision-making.

5.10 Summary

This section presented a comprehensive evaluation of the machine learning-based recommendation system, combining both quantitative signal performance and interpretability analysis. Through statistical backtesting, confusion matrices, and classification metrics, we assessed the model's predictive capabilities and identified areas for improvement, particularly in generating BUY signals.

We further examined the system's interpretability using SHAP values to uncover the most influential input features and evaluated the clarity and quality of the LLM-generated natural language explanations using a structured manual rubric.

While the framework shows promise, several limitations persist, including class imbalance, the scope of training data, and the computational cost of interpretability methods. Despite these, the combination of transparent model insights and explainable recommendations provides a solid foundation for iterative refinement and potential deployment in real-world financial advisory contexts.

6 Conclusion (990/1000 words)

6.1 Summary

This project has presented the design, implementation, and evaluation of the *GenAI Advisor*, an offline-first, explainable financial advisory prototype integrating machine learning-driven signal generation, modular technical analysis, and natural language explanations. Through an iterative, MVP-oriented development process, the system evolved from a single-indicator proof of concept into a hybrid ensemble framework combining rule-based heuristics and a TensorFlow-based classifier, coupled with a locally hosted language model for explanation generation.

Key achievements include the successful integration of reproducible data ingestion pipelines, a modular strategy engine extensible to future indicators, and a user-facing interface capable of delivering interpretable recommendations in a pedagogically oriented format. Evaluation demonstrated that the system performed competently in predicting neutral (HOLD) conditions and moderately in directional classification (BUY/SELL), while maintaining a strong emphasis on transparency through feature-level interpretability (via SHAP) and structured language model explanations.

Crucially, the work addresses three interconnected challenges inherent in the deployment of algorithmic decision support for retail investors: *(i)* ensuring reproducibility and determinism in financial data processing, *(ii)* demystifying machine learning outputs through human-centred explanation design, and *(iii)* balancing model-driven predictions with educational transparency to mitigate over-reliance on automated signals. The system's offline-first architecture, grounded in open-source tooling and lightweight dependency management, further supports these aims by reducing infrastructural complexity and enhancing auditability.

Collectively, this research demonstrates that explainable AI (XAI) and local LLM inference can be effectively combined in a single framework for educational financial decision support, laying the groundwork for future applications at the intersection of retail investing and responsible algorithmic transparency.

6.2 Reflection on Contributions

The contributions of this project can be summarised across three dimensions:

1. **Technical integration:** The system combines modular financial indicators, an ML classifier, and an LLM-driven explanation layer in a cohesive, end-to-end workflow. The architecture emphasises reproducibility, portability, and offline operation, distinguishing it from conventional API-reliant fintech prototypes.

2. **Explainability and transparency:** By pairing post hoc interpretability (SHAP) with prompt-engineered natural language explanations, the project situates interpretability as a first-class objective rather than a retrospective add-on. This dual approach - quantitative and narrative - supports both expert and novice understanding of model outputs.

3. **Evaluation methodology:** The development of a cut-off based backtesting framework provided a robust mechanism for temporal validation and reproducible benchmarking of signal performance. The manual rubric for explanation evaluation offers a replicable template for assessing LLM-generated financial narratives.

These elements collectively position the GenAI Advisor as both a functional prototype and a methodological case study in combining XAI, local inference, and financial signal generation under constrained, reproducible conditions.

6.3 Future Work

While the current implementation delivers a functional MVP, several avenues exist for extending its technical depth, empirical robustness, and user-facing capabilities:

1. Migration to SQL-based storage. Data storage currently relies on CSV files for their simplicity and transparency. While sufficient for a single-user, offline MVP, future versions could benefit from migration to a relational database system (e.g., MariaDB or PostgreSQL). This would improve query efficiency, enable concurrent multi-user access, and allow integration with broader portfolio tracking functionality. Structured storage would also support richer metadata—such as per-ticker feature attributions or backtesting statistics—facilitating advanced querying and visualisation.

2. Data diversification and temporal modelling. The model is presently trained on a limited set of technology-focused equities using fixed daily frequencies and a 10-day forecast horizon. Expanding coverage to multi-sector equities, alternative geographies, and higher-frequency intraday data could improve generalisability. Furthermore, temporal models such as LSTMs or Transformers could be explored to capture sequential dependencies beyond fixed-window features, potentially enhancing BUY/SELL recall rates observed during evaluation.

3. Empirical signal weighting and adaptive ensembles. The ensemble strategy currently uses fixed, heuristic weights for signal aggregation. A data-driven weighting mechanism—optimised via historical performance metrics or Bayesian updating—could dynamically reallocate weights across indicators as market conditions evolve. Such adaptivity may reduce the over-representation of HOLD signals and increase responsiveness to trend inflections.

4. Enhanced explanation fidelity. While SHAP values provide feature-level insight and LLM-generated explanations contextualise these signals for end-users, future iterations could explicitly integrate SHAP outputs into the prompt construction pipeline. This would ensure tighter alignment between model mechanics and generated narratives, reducing the risk of over-generalised or ungrounded explanations.

5. Improved frontend interactivity. The Streamlit-based interface could be expanded to include interactive signal overlays, portfolio simulation tools (e.g., scenario analysis sliders), and drill-down visualisations for SHAP feature contributions. Mobile responsiveness and session persistence would further enhance usability, especially for non-technical users.

6. Live deployment and paper trading integration. Backtesting, while necessary for offline validation, cannot fully account for transaction costs, slippage, or real-time data constraints. Future work will involve integrating with paper trading APIs (e.g., Alpaca) to simulate live execution under realistic market latency. Such an extension would bridge the gap between retrospective evaluation and practical deployability.

6.4 Concluding Remarks

This research underscores the viability of uniting machine learning, explainable AI, and generative language models within an offline, reproducible framework for financial education and decision support. The GenAI Advisor not only operationalises these technologies in a technically rigorous manner but also situates them within an ethical and pedagogical lens, emphasising transparency, reproducibility, and user autonomy.

By providing a modular, extensible foundation, this work offers both a functional prototype and a springboard for future research. Subsequent iterations can build upon this groundwork to

incorporate adaptive weighting, richer temporal modelling, and live market testing. Crucially, the dual emphasis on performance and interpretability ensures that such progress does not come at the cost of user trust or responsible usage.

In sum, the project delivers a credible proof of concept for explainable, AI-driven financial signal generation in an offline-first context, demonstrating both immediate educational utility and long-term potential for responsible innovation in algorithmic advisory systems. Future work will focus on scaling this foundation to more diverse markets, more sophisticated architectures, and more user-tailored interfaces, advancing the broader goal of demystifying machine learning in retail finance.

Overall, this project demonstrates that combining modular technical analysis, machine learning-driven prediction, and local language model explanations within an offline-first architecture is both technically feasible and educationally valuable. The GenAI Advisor offers a reproducible, transparent prototype that balances predictive capability with interpretability, addressing common trust deficits in algorithmic financial tools, and laying a foundation for future research into adaptive, responsible, and user-centred financial decision support systems.

References

[1] David H. Bailey, Jonathan M. Borwein, Marcos Lopez de Prado, and Qiji Jim Zhu. 2014. The Probability of Backtest Overfitting. *Journal of Computational Finance* 20, 4 (2014), 39–69.

[2] Daphna Ben David, Yedidya S. Resheff, and Tal Tron. 2021. Explainable AI and Adoption of Financial Algorithmic Advisors: an Experimental Study. *arXiv preprint arXiv:2101.02555* (2021). https://arxiv.org/abs/2101.02555

[3] Chiranjit Chakraborty and Alex Joseph. 2017. *Machine Learning at Central Banks.* Staff Working Paper 674. Bank of England. https://www.bankofengland.co.uk/-/media/boe/files/working-paper/2017/machine-learning-at-central-banks.pdf

[4] Fa Chen. 2021. Variable interest entity structures in China: are legal uncertainties and risks to foreign investors part of China's regulatory policy? *Asia Pacific Law Review* 29, 1 (2021), 1–24. https://doi.org/10.1080/10192557.2021.1995229

[5] Thomas Fischer and Christopher Krauss. 2018. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research* 270, 2 (2018), 654–669. https://doi.org/10.1016/j.ejor.2017.11.054

[6] David Gunning and David Aha. 2019. DARPA's Explainable Artificial Intelligence (XAI) Program. *AI Magazine* 40, 2 (2019), 44–58. https://doi.org/10.1609/aimag.v40i2.2850

[7] I. Kaastra and M. Boyd. 1996. Designing a Neural Network for Forecasting Financial and Economic Time Series. *Neurocomputing* 10, 3 (1996), 215–236. https://doi.org/10.1016/0925-2312(95)00039-9

[8] Andrew W. Lo. 2002. The Statistics of Sharpe Ratios. *Financial Analysts Journal* 58, 4 (2002), 36–52. https://doi.org/10.2469/faj.v58.n4.2453

[9] D. Mai. 2024. StockGPT: A GenAI Model for Stock Prediction and Trading. *SSRN* (2024). https://ssrn.com/abstract=4787199

[10] N. Marey, A. A. Abu-Musa, and M. Ganna. 2024. Integrating Deep Learning and Explainable Artificial Intelligence Techniques for Stock Price Predictions. *International Journal of Accounting and Management Sciences* 3, 4 (2024), 479–504.

[11] John J. Murphy. 1999. *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications.* New York Institute of Finance.

[12] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. Why Should I Trust You? Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 1135–1144.

[13] Lukas Ryll and Sebastian Seidens. 2019. Evaluating the Performance of Machine Learning Algorithms in Financial Market Forecasting: A Comprehensive Survey. *arXiv preprint arXiv:1906.07786* (2019). https://arxiv.org/abs/1906.07786

[14] S&P Global Market Intelligence. 2025. *GenAI funding hits record in 2024, boosted by infrastructure interest.* https://www.spglobal.com/market-intelligence/en/news-insights/articles/2025/1/genai-funding-hits-record-in-2024-boosted-by-infrastructure-interest-87132257 Accessed May 2025.

[15] Tomonori Takahashi and Takayuki Mizuno. 2024. Generation of synthetic financial time series by diffusion models. *arXiv preprint arXiv:2410.18897* (2024).

[16] U.S. Department of Commerce. 2022. *Commerce Implements New Export Controls on Advanced Computing and Semiconductor Manufacturing Items to the People's Republic of China (PRC).* Technical Report 2022-21658. Bureau of Industry and Security. `https://www.govinfo.gov/content/pkg/FR-2022-10-13/pdf/2022-21658.pdf` Federal Register, Vol. 87, No. 197, pp. 62186–62215.

[17] Alex Zarifis and Xusen Cheng. 2024. How to build trust in answers given by Generative AI for specific, and vague, financial questions. *arXiv preprint arXiv:2408.14593* (2024). `https://arxiv.org/abs/2408.14593`