Regime-Aware Deep Learning for Investment Suggestion and Portfolio Optimization

Vineet Naik

MSc in Data Science
The University of Bath
2025

Regime-Aware Deep Learning for Investment Suggestion and Portfolio Optimization - Project type.

Submitted by: Vineet Naik

Copyright

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Master of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed: _____

Contents

List of Figures

List of Tables

Chapter 1

1 Introduction

1.1 Background

Financial markets often alternate between distinct regimes – for example, bull vs. bear markets or risk-on vs. risk-off conditions – which exhibit persistent behavior patterns (e.g., high volatility in a crash, low volatility in a stable period) (Shu, Yu and Mulvey, 2024a; London Stock Exchange Group, 2024). Detecting these regimes is crucial to adjusting investment strategies and managing risk. However, financial time series are noisy and nonstationary, causing regime detection (Ahmad, Shadaydeh and Denzler, 2024).

Traditional approaches like Hidden Markov Models (HMMs) or Markov-switching ARIMA often impose restrictive assumptions and may not capture nonlinearities (Shu, Yu and Mulvey, 2024b). In contrast, machine learning (ML) and deep learning (DL) techniques can learn complex data-driven representations, making them promising tools for latent regime detection and investment forecasting (Suárez-Cetrulo, Quintana and Cervantes, 2024).

This project proposes a *regime-aware deep learning* framework to improve investment decisions by:

- **Regime Detection:** Implement models that classify market conditions into a few distinct regimes (e.g., bull / bear, volatility states) based on historical market and factor data. Techniques may include unsupervised methods (clustering, Gaussian mixture models) and deep neural architectures that respect the data geometry.
- **Deep Learning Forecasting:** Build sequence models (e.g., LSTM, CNN-LSTM hybrids) to forecast asset returns or regime transitions, potentially using regime indicators as additional features. Previous work shows that incorporating regime information can improve the accuracy of the deep model (Aramyan, Ramchandani and Skevofylakas, 2023).
- **Investment Suggestion & Portfolio Optimization:** Design a strategy that leverages detected regimes and forecasts to suggest asset allocations or stock picks. For example, allocate more to equities in a "risk-on" regime and switch to safer assets in a "risk-off" regime. Portfolio optimization techniques (mean-variance, risk-parity, or others) will be used in each regime to tailor suggestions.
- **Prototype and Backtesting:** Develop a prototype pipeline (in Python) that includes

data ingestion, model training, and backtesting. Using historical data, evaluate performance improvements over a baseline (e.g., buy-and-hold). Emphasis will be on a proof-of-concept rather than live trading, focusing on model performance, backtesting results, and interpretability.

- **Explainability & Accessibility:** Given that the target users include novice investors, explainable AI techniques will be incorporated (e.g., visualizing which features or regimes influenced a suggestion) alongside an intuitive user interface (e.g., a Streamlit app). This approach ensures transparency, allowing users to understand the rationale behind a model's recommendations.

The outcome will be a prototype system in Python, combining data ingestion, regime detection, model training, and backtesting with a user-friendly interface to visualize insights. The focus is on improving model performance by adapting forecasting to different market regimes, enhancing regime interpretability, and providing accessible, actionable insights for novice investors through dynamic portfolio recommendations.

1.1.1 Research Statement

Financial markets often undergo regime changes - shifts in underlying behavior such as volatility, correlation, or return patterns - that static models struggle to detect or adapt to effectively. This research investigates how regime-specific forecasting models, combined with adaptive portfolio rebalancing, can improve the detection of market regimes and enhance portfolio allocation strategies in response to these shifts.

1.1.2 Novelty of the research

The novelty of this research lies in its systematic framework for evaluating how different regime-aware strategies - spanning statistical, machine learning, and hybrid approaches - can be adapted for non-expert investors.

Chapter 2

Literature, Technology and Data Survey

This review will explore the evolving landscape of financial asset management, focusing on three key areas: identifying market regimes, forecasting market behavior, and integrating deep learning techniques. It will examine various approaches to detecting market regimes, from traditional statistical models to modern machine learning methods, and discuss their implications for investment strategies. Additionally, the review will delve into the evolution of asset allocation strategies, highlighting the shift from classical models to data-driven approaches, and assess how deep learning enhances these strategies by enabling more adaptive and informed decision-making. Building on these foundations, this research further investigates how deep learning models, applied to both regime detection and market forecasting, can improve detection of regime shifts and enhance adaptive portfolio recommendations. By synthesizing insights from these areas, the review aims to comprehensively understand how market regime detection and forecasting can benefit novice investors.

2.1 Market Regime Detection

Over time, financial markets have shown changing patterns, like sharp spikes in volatility and correlations during crashes. These shifts are influenced by behavioral, political, and economic factors (Shu, Yu and Mulvey, 2024b). One way to understand these changes is by identifying specific periods in financial regimes where market behavior tends to be more stable and consistent for an extended time.

Definition and Importance

A financial regime is an extended period during which markets exhibit consistent behaviors. These behaviors can manifest as bull, bear, or range-bound markets, characterized by sustained positive or negative returns, periods of low or high volatility, or risk-on or risk-off sentiment reflecting investor confidence or aversion. Regime-switching patterns have been documented across multiple asset classes, such as equities, fixed-income securities, and currencies(Shu, Yu and Mulvey, 2024b).

One of the key applications of financial regime detection is identifying a down market, which helps investors manage risk and adapt their strategies accordingly. A down market, or bear

market, occurs when a market drops over 20% from its peak price. It is characterized by sustained declines in asset prices, heightened volatility, and negative returns and presents significant challenges for investors. Bear markets are usually short-lived, lasting about 9.6 months or 289 days on average. In comparison, bull markets tend to last much longer, with an average duration of 2.6 years, or 965 days(Hartford Funds, 2023). Detecting this regime involves analyzing various market indicators, such as sharp drops in equity prices, increased market volatility, and deteriorating economic conditions.

Identifying market regime shifts enables investors to adjust their strategies proactively, enhancing risk management and enabling more informed, data-driven investment decisions (London Stock Exchange Group, 2024).

2.1.1 Methodology

This study explores a range of methodologies for detecting financial market regimes and forecasting asset behavior. The literature includes traditional statistical models such as Markov-switching GARCH and vector autoregression, unsupervised machine learning techniques like k-means clustering and Gaussian Mixture Models, and deep learning methods such as Long-Short-Term Memory (LSTM) networks and CNN-LSTM hybrids.

Hybrid approaches - where unsupervised regime detection is combined with deep learning-based forecasting - are exciting. For instance, some studies use clustering methods to identify latent market regimes, which are then incorporated as features into time-series forecasting models. Such methods have been shown to improve predictive performance and portfolio outcomes.

This section explores the range of available methods and identifies promising candidates for further experimentation. In the subsequent phase, selected approaches will be implemented and evaluated through empirical testing and backtesting.

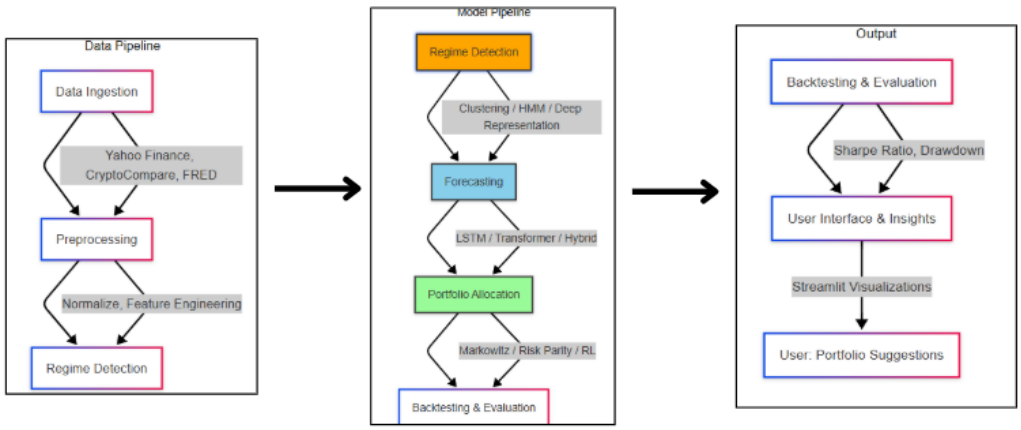

Figure 2.1: Proposed Regime-Aware Portfolio Pipeline

2.1.2 Background

Market regime detection uses statistical or machine learning models to identify persistent market states (e.g., bull vs bear). A variety of methods can be employed for this purpose, ranging from statistical models like vector autoregression and hidden Markov models to machine

learning algorithms such as k-means clustering, agglomerative clustering, and Gaussian mixture models(London Stock Exchange Group, 2024). Many models have been applied, from classical econometric regime-switching models to modern neural architectures. These include Hidden Markov Models (HMMs) and their extensions (Markov-switching AR or GARCH models, semi-Markov or duration-dependent HMMs) (Yuan and Mitra, 2016; Ntantamis, 2009).

Traditional Models: Traditional models, such as Markov-switching models, use latent Markov chains to model transitions between states (Hamilton, 1989).

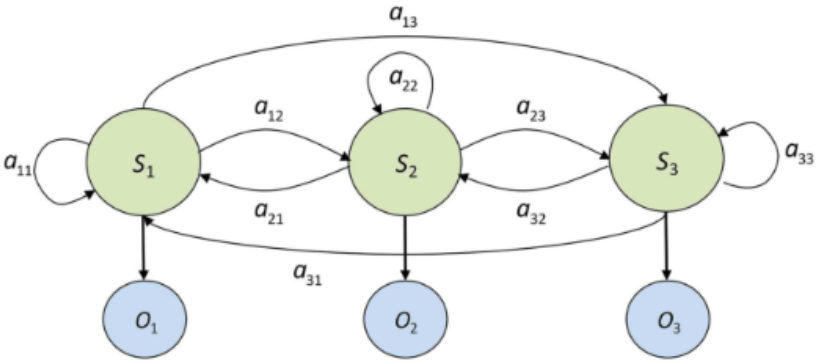

Figure 2.2: Overview of a Hidden Markov Chain

A Hidden Markov Model (HMM) is a statistical tool used to model systems where the states are not directly observable (hidden) but can be inferred based on observable outputs. It consists of hidden states, transition probabilities between them, and emission probabilities that define the likelihood of observing particular outputs from each state. HMMs are widely applied in areas like speech recognition, financial market analysis, and bioinformatics, where they help infer the underlying state sequence based on observable data (QuantConnect, 2025).

In this framework, asset returns or variances switch between several states according to a Markov chain (regime transition probabilities). Extensions include Markov-switching GARCH and VAR models (Shu, Yu and Mulvey, 2024b). These models have been widely applied (e.g., Turner et al., 1989; Ryden et al., 1998, cited in Shu, Yu and Mulvey (2024a))to capture business cycles and market regimes. Statistical hypothesis tests can estimate the number of regimes and transition probabilities. However, such models often assume linear relationships and a limited regime count, which can oversimplify complex markets.

Clustering-Based Methods: More recently, unsupervised machine learning has been applied to regime detection. Since regimes are latent, one can treat regime labeling as a clustering problem on historical data. For example, clustering methods (k-means, Gaussian Mixture Models, hierarchical clustering) can classify periods into clusters with similar return/volatility patterns (London Stock Exchange Group, 2024). A recent study developed a novel clustering method for multivariate time series: they use k-means with Wasserstein-distance metrics on distributions of returns to detect regimes in multidimensional data. They demonstrate that using these regimes for portfolio design yields profitable strategies (Mc Greevy et al., 2024).

Deep Learning: Recent studies have explored applying deep learning techniques for detecting financial market regimes. Orton and Gebbie (2024) employ a deep representation learning approach using specialized neural networks - Seasonal-Periodic Decomposition Network (SPDNet) and its variants - on correlation matrices of asset returns. Their models respect the Riemannian

geometry of covariance matrices and aim to detect market phases within a block-hierarchical market data context. However, they caution that deep models can overfit spatiotemporal correlations and that relying on single summary metrics may be misleading for investment applications. While deep neural networks offer powerful feature extraction capabilities, they require careful tuning to avoid overfitting (Orton and Gebbie, 2024).

Hybrid methods: Hybrid methods that integrate traditional statistical techniques with machine learning (ML) are also prevalent. For instance, Erlwein-Sayer et al. (2021) employ a Hidden Markov Model (HMM) to identify regimes in corporate credit spreads. They then use the identified regime labels as input features for a Long Short-Term Memory (LSTM) time-series model. Their findings indicate that the "HMM-LSTM model is calibrated on credit spreads," in most cases, incorporating regime information enhances the LSTM's forecasting performance. This suggests that integrating regime features can improve the efficacy of deep learning models. The investigation will explore similar hybrid approaches, such as using unsupervised clustering to label regimes and training a neural network that incorporates these labels as inputs(Erlwein-Sayer et al., 2023).

Rule-based definition (20% drawdown). An alternative to model-based definitions is a simple rule-of-thumb used in practitioner contexts to demarcate broad market states: the **20% peak-to-trough drawdown rule** (London Stock Exchange Group, 2024). Let P_t denote the price, define the running peak

$$\text{peak}_t = \max_{s \le t} P_s,$$

and the drawdown

$$\text{DD}_t = \frac{P_t - \text{peak}_t}{\text{peak}_t}.$$

Operationally:

- **Bear** if $\text{DD}_t \le -0.20$ for at least m consecutive trading days (a short persistence window to suppress transient noise).
- **Bull** once price sets a new high ($P_t > \text{peak}_{t-1}$) or remains above the threshold for k days.
- Otherwise, **sideways/recovery**.

Variants adjust the 20% threshold and/or the persistence parameters (m, k); when used as a benchmark, sensitivity to these choices is often reported alongside statistical or machine-learning regime detectors.

Summary. Studies show that incorporating regime information boosts performance (Shu, Yu and Mulvey, 2024b). Hybrid techniques and deep representation learning promise to detect and act on regime signals.

2.2 Time-Series Forecasting with Deep Learning

Statistical vs. Machine Learning: Traditional time-series methods (ARIMA, ARIMAX, GARCH) are well-understood and perform reasonably on linear patterns. However, they have limitations: they assume linearity and often lack long-term memory. Bao et al. (2024) note that while ARIMA-type models are efficient, they struggle with non-linear dynamics and long-term forecasts (Vuong et al., 2024).

Recurrent Neural Networks (RNNs) and LSTM: Recurrent models, especially Long Short-Term Memory (LSTM) networks, are widely used for sequential financial data. LSTMs can capture long-range dependencies in price series and learn complex patterns from raw inputs. For instance, numerous studies apply LSTM to predict stock prices, indices, or volatility and report better accuracy than simple baselines (Vuong et al., 2024). In the context of regimes, the HMM-LSTM suggests LSTM benefits from regime features (Erlwein-Sayer et al., 2023). However, RNNs can overfit and require substantial data and tuning.

Convolutional Neural Networks (CNNs) and Hybrids: CNNs have also been adapted to time-series analysis. A one-dimensional CNN can learn local patterns in sequences, effectively acting as an automated feature extractor. Many finance papers use 1D CNNs or hybrid CNN-LSTM architectures: for example, a recent study trains a custom CNN-LSTM on stock market data and reports high predictive accuracy (A et al., 2023). Let CNN layers detect short-term local patterns (or time-invariant features) and then let LSTM layers integrate those features over time. In general, hybrid models often outperform single models by leveraging the strengths of each component (Vuong et al., 2024).

Transformers (attention-based): Transformers replace recurrence with self-attention, enabling direct modeling of long-range temporal dependencies and flexible conditioning on covariates. Recent adaptations to forecasting include encoder–decoder and encoder-only designs, as well as specialized architectures such as the Temporal Fusion Transformer (TFT) for multi-horizon forecasting with variable selection and interpretable attention, Informer for efficient long-sequence forecasting via probabilistic sparse attention, and Autoformer with decomposition and auto-correlation mechanisms (Vaswani et al., 2017). Empirically, Transformers can excel when horizons are long and exogenous inputs matter, but they are data-hungry and benefit from careful regularization and early stopping.

Gradient Boosting Machines (GBM): Tree-based boosting methods (e.g., XGBoost, LightGBM, CatBoost) are strong baselines for tabular time-series formulated as supervised learning on lagged features, rolling statistics, and calendar effects. They capture non-linearities and higher-order interactions, handle heterogeneous feature scales, and train quickly; with walk-forward validation and leakage controls, GBMs often rival deep nets on medium-sized datasets (Chen and Guestrin, 2016). Their main limitation is that temporal dependence is represented only through engineered features, not learned sequence dynamics.

Forecasting Targets: The objective is to predict future asset returns or directly forecast regime probabilities using classification or regression approaches. For instance, an LSTM could be trained to predict the next-day regime label or forecast future returns under different regimes. The model's output could adjust asset allocations (e.g., shifting towards bonds if a bear regime is predicted). The literature indicates that even simple use of regime indicators can enhance performance (Erlwein-Sayer et al., 2023). Therefore, this study will assess the most effective methods for integrating these predictions into the allocation process.

2.3 Portfolio Optimization and Strategy Generation

2.3.1 Markowitz mean-variance portfolio theory

Traditional Portfolio Optimization: The classical approach (**Markowitz mean-variance**) constructs efficient frontiers, assuming asset returns are jointly normal and using estimated means and covariances. The theory focuses on constructing a portfolio of assets that maximizes

expected return for a given level of risk or equivalently minimizes risk for a given level of expected return. The efficient frontier is a key result of Markowitz's theory. It represents a set of portfolios that offer the highest expected return for a given level of risk or the lowest risk for a given level of expected return (AnalystPrep, 2024). Any portfolio that lies below the efficient frontier is suboptimal because it does not offer the best return for the level of risk(Investopedia, 2024).

Markowitz's theory has been fundamental in shaping financial portfolio management, though it assumes constant returns and normal distribution of asset returns, which limits its practical application. Later studies have extended the model to incorporate more realistic market conditions, such as non-normal return distributions and transaction costs (Shu, Yu and Mulvey, 2024b).

Limitations of Markowitz Mean-Variance Portfolio Theory: A major limitation of the Markowitz model is its high sensitivity to input estimates - especially expected returns. Even small errors in return estimates can result in significantly different and potentially suboptimal portfolio allocations(AnalystPrep, 2024).

2.3.2 Dynamic Allocation Models:

Beyond statically optimizing within a regime, one can dynamically shift weights when regimes change. In practice, simple rule-based adjustments may be employed: e.g. if a "bear" regime is detected, reduce equity weight by X%.A more systematic approach would involve using Reinforcement Learning (RL) to learn an optimal shifting policy. Recent studies apply deep RL to portfolio allocation (e.g. combining Modern Portfolio Theory with deep RL (Huang, Zhou and Song, 2025).

2.3.3 Water-filling Algorithm

The water-filling algorithm is an optimization technique that allocates a constrained resource (e.g., power, capital) across multiple channels (e.g., frequency bands, assets) to maximize the total utility (He et al., 2013)

As the primary aim of this research is not centered on portfolio optimization, the researcher experiments with applying the water-filling algorithm for capital allocation. Although this algorithm is theoretically optimal in communications engineering for maximizing information capacity under power constraints (Qi, Minturn and Yang, 2012), it remains non-standard in finance and lacks a proven track record within portfolio theory. Its use here therefore represents a methodological novelty rather than a conventional practice. Importantly, the approach may not align with typical financial objectives such as maximizing risk-adjusted returns (e.g., Sharpe Ratio) or minimizing drawdowns. The capital allocation strategy was designed to mirror the theoretical optimum for allocating a limited resource (capital) across channels (assets) with varying signal quality (predictive advantage). The water-filling algorithm was selected for this role.

2.4 Data Sources and Collection

The research utilizes publicly available financial datasets to ensure transparency. Key sources include:

- **Market Data:** Historical equity and cryptocurrency prices from `Yahoo Finance` providing OHLCV data.
- **Macroeconomic indicators (considered, not used as exogenous features):** Although the initial survey proposed using macroeconomic indicators (e.g., policy rates, GDP, CPI) as exogenous features, these were excluded from the final models for following reasons:

  (i) *release lags and revision bias* - most macro series are published monthly/quarterly and revised ex post, which risks look-ahead leakage unless real-time vintages are used.

  (ii) *frequency mismatch* - aligning low-frequency macro with daily targets introduces ragged-edge nowcasting and interpolation choices that add modelling assumptions without clear out-of-sample benefits.

  (iii) *empirical yield* - preliminary ablations showed limited and inconsistent incremental improvement beyond regime-aware, price-derived features on the chosen horizons.

2.4.1 Characteristics of Financial Time Series Data

Financial time series are typically non-stationary, noisy, and exhibit heteroskedasticity, posing challenges for regime detection and forecasting(Guo, 2025). Key challenges include:

- **Temporal Dependencies:** Long-term trends and short-term volatility require models capable of capturing multi-scale patterns, such as Long Short-Term Memory (LSTM) networks and Transformers.
- **Regime Shifts:** Structural breaks (e.g., the 2020 market crash and rebound) highlight the need for adaptive models that can dynamically adjust to changing market conditions.

Chapter 3

Requirements Analysis and Specification

3.1 Introduction

This chapter outlines the analysis of 'Retail investing for novice investors' and presents the formal requirements specification for the development of BullVision, a decision-support system tailored for novice investors. Its primary aim is to bridge the considerable gap between the complexity of financial markets and the limited technical expertise of its intended users. The requirements are shaped through a systematic examination of the challenges faced in retail investing and are expressed as user goals and system functionalities.

3.2 Stakeholder input

To ground the system in real user needs, informal interviews were conducted with family and friends who represent novice investors. These discussions revealed common difficulties such as confusion with financial terminology, uncertainty in interpreting price movements, and reliance on hearsay or social media for decisions. The insights confirmed the need for a simple, transparent tool that explains market conditions in plain language and supports structured, disciplined investing.

3.3 Requirements Analysis

The contemporary investment environment, with its growing accessibility to equities, exchange-traded funds (ETFs), and cryptocurrencies, presents unique challenges for beginner investors. While access to markets has never been easier, this accessibility is not matched by a clear understanding of how these markets function. As a result, novice investors often struggle to interpret basic market movements, trading practices, and investment principles. This disconnect creates several key obstacles:

- **Analysis Paralysis:** The sheer volume of data, charts, and financial news is intimidating, causing beginners to delay investing or make impulsive decisions.
- **Increased Risk of Loss:** Without a firm grasp of how markets operate, investment

decisions often rely on emotion rather than logic. Common pitfalls include buying assets during hype-driven peaks (Fear of Missing Out) and selling during downturns driven by fear, which leads to repeated financial losses.

- **Lack of a Basic Framework:** Many beginners are unfamiliar with essential concepts such as diversification (spreading investments to reduce risk) and position sizing (how much to invest in a single asset). This lack of structure makes their portfolios highly vulnerable to volatility.

3.4 The Affected Audience

This problem primarily affects novice and beginner retail investors who are interested in growing their wealth through stocks and cryptocurrencies but do not have a background in finance or trading.

3.5 Requirements Specification

The following constitutes a precise specification of the objectives the prototype is required to fulfil, structured hierarchically to define overarching user goals and the subsequent functional and non-functional requirements.

3.5.1 User Goals

- **UG1:** The user must be able to obtain a quantitative forecast for a specified financial instrument without necessitating comprehension of the underlying analytical models or technical indicators.
- **UG2:** The user must be able to generate a personalised, diversified investment portfolio predicated on a defined capital budget and a subjective risk tolerance level.
- **UG3:** The user must be afforded insight into the rationale underpinning the portfolio construction, including the forecasted performance and an appraisal of associated risks.
- **UG4:** The user must be able to analyse the projected performance of a custom, user-defined portfolio of assets and quantities.

3.5.2 Functional Requirements

To meet the user goals, the system shall support the following core functions:

- **FR1: Data Acquisition and Management** The system allows users to select instruments from a list or input tickers, retrieves historical OHLCV data from a financial API, supports time range selection, and performs automated currency conversion.
- **FR2: Market Regime Analysis and Indicators** The system computes technical indicators, determines bull/bear regimes using rule-based logic, and selects the most relevant features while avoiding data leakage.
- **FR3: Forecasting Module** The system generates price forecasts for chosen horizons using multiple models (SARIMAX, GBM, LSTM, Transformers), evaluates performance, and stores results.

- **FR4: Model Evaluation and Selection** The system compares and ranks models by accuracy, designates or allows manual selection of the best model, and maintains a record of optimal models for each instrument.
- **FR5: Portfolio Construction** The system enables users to select instruments, declare capital, and set risk tolerance. It allocates funds based on forecasts, volatility, and regimes, enforces diversification limits, and outputs an executable investment plan with projected portfolio value.
- **FR6: Custom Portfolio Analysis** The system shall allow users to define a portfolio by specifying a list of tickers and their respective quantities. It shall then project the future value of this custom portfolio over multiple time horizons using its best available forecasts for each constituent asset.

3.5.3 Non-Functional Requirements

- **NFR1: Usability** — The system shall provide a simple, guided web interface that hides technical complexity and remains accessible to novice users.
- **NFR2: Reliability** — The system shall handle errors gracefully (e.g., invalid tickers, API downtime) and provide clear user feedback without crashing.
- **NFR3: Performance** — Core operations such as data retrieval, forecasting, and portfolio generation shall complete quickly (ideally under one minute).
- **NFR4: Explainability** — The system shall justify its outputs in plain language (e.g., why a model or asset was selected) to build trust and understanding.
- **NFR5:Data Persistence** - The system shall maintain a persistent registry of model selections and engineered datasets to ensure consistency across different analysis modules.

3.5.4 Constraints

- **C1:** The system depends on third-party financial APIs, subject to availability, rate limits, and licensing.
- **C2:** Forecasting accuracy is limited by the uncertainty of financial markets; the system is advisory, not a guarantee of returns.
- **C3:** The system requires pre-trained models (e.g., LSTM, Transformer) to ensure feasible performance on standard hardware.

Chapter 4

Overall Design

4.1 Analysis of the Problem and Solution Approach

The problem, as defined in the requirements specification, is the asymmetry between the complexity of financial markets and the analytical capability of the novice investor. The solution must therefore perform sophisticated data analysis and modelling while presenting an interface of extreme simplicity.

This dictates a two-tiered design:

- **Backend Engine:** Responsible for complex computations including data acquisition, feature engineering, feature selection, machine learning forecasting, best model selection, and portfolio optimisation.
- **Frontend Interface:** Comprising the Home Page, Investment Planner, and Customised Portfolio Forecast. This acts as a guided wizard that presents the user with minimal, sequential choices and delivers clear, interpretable results.

The most appropriate architectural pattern for this system is a *modular, data-flow pipeline*. Data is processed through a sequence of distinct stages, with each module transforming the output of the previous one. This design ensures high cohesion (each module has a single, well-defined purpose) and low coupling (modules interact through well-defined data interfaces), thereby improving maintainability, testing, and development efficiency.

Finally, the choice of a web application framework is optimal for accessibility, enabling users to access the tool without requiring complex installation procedures.

" Although the dissertation centres on regime-aware deep learning, competent ***machine-learning baselines (SARIMAX, GBM) are included to provide rigorous comparative evidence****. These benchmarks are interpretable, reflect strong practitioner standards, and expose complementary inductive biases that can outperform in short or low-signal windows. Side-by-side evaluation isolates the incremental value of the regime signal and of deep sequence modelling, while a predeclared selection rule (RMSE/Naive → Test RMSE → Accuracy) controls model risk and demonstrates that observed gains are genuine rather than artefacts of tuning. "*

4.2 High-Level System Architecture

The overall system is structured as a pipeline where data and user intent flow sequentially through a series of processing stages, supporting two distinct user workflows: automated portfolio construction and custom portfolio analysis. The high-level architecture and data flow are illustrated in the figure below and described in the subsequent sections.

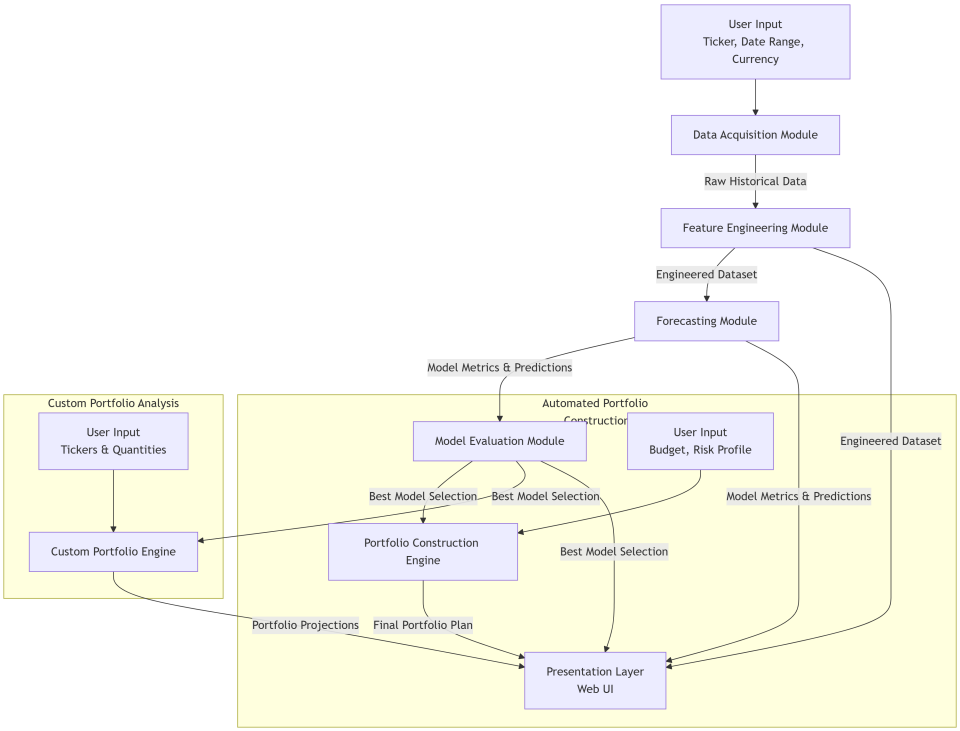

Figure 4.1: High Level System Architecture

The design is implemented as a multi-page Streamlit application. Streamlit provides a powerful framework for rapidly building data-driven web applications in Python, perfectly aligning with the project's need to seamlessly integrate backend Python logic (e.g., Pandas, Scikit-learn) with a presentable frontend. Each page in the application corresponds to a major stage in the pipeline.

4.3 Component specification and modularisation

The application is decomposed into eight modules (Fig. 4.2): acquisition, feature engineering, feature selection, forecasting, model evaluation, guided portfolio construction, custom portfolio projection, and presentation.

System architecture: $01 \rightarrow 02 \rightarrow 03 \rightarrow 04 \rightarrow 05 \rightarrow 07/08$, with artefacts in `metrics/`, `portfolio_datasets/`, and `forecasts/`.

Figure 4.2: High-level architecture and artefact flow.

4.3.1 Data Acquisition Module

The purpose of this module is to retrieve and normalise historical market data so that it can be used consistently in downstream modelling tasks.

Purpose	Retrieve and normalise historical market data suitable for downstream modelling.
Public Interface	`fetch_prices(ticker, start, end)` → DataFrame[OHLCV] `prep_df(raw_df)` → DataFrame[OHLCV] `convert_currency(df, src, dst)` → DataFrame[OHLCV]
Inputs/Outputs	Returns a Pandas DataFrame with columns {Date, Open, High, Low, Close, Adj Close, Volume}; persists a copy in Streamlit session state and optionally to CSV.
Side Effects	Network I/O to the market data API; file writes to `portfolio_datasets/`.
Rationale	Decouples acquisition from modelling and guarantees a stable schema.
Failure Modes	Transient API errors, missing fields; mitigated via schema validation and retries.

Table 4.1: Specification of the Data Acquisition Module

4.3.2 Feature Engineering and Regime Tagging

This module computes technical indicators from historical prices and attaches a drawdown-based market regime label. It produces both a full-history feature set for exploratory analysis and a recomputed 2020+ slice restricted to post-2020 data, avoiding any lookback leakage from earlier periods. The regime classification follows a 20% rolling peak drawdown rule, with thresholds adjusted for highly volatile assets such as cryptocurrencies. In addition to price-based indicators (returns, SMAs, EMAs, MACD, volatility, Bollinger Bands, RSI), the module also adds calendar covariates (day of week and month). The engineered datasets are saved to disk and made available for download, while also being persisted in session state for downstream feature selection and forecasting. A complete list of the variables generated by this module, together with their definitions, is provided in Appendix 7.

Purpose	Compute technical indicators and derive a drawdown-based regime label.
Public Interface	`compute_indicators(df)` $\rightarrow$ DataFrame[features]
Features Produced	Returns (simple, log), SMA/EMA, MACD (line, signal, histogram), volatility (20, 60), Bollinger Bands (20), RSI(14), day of week, month.
Data Contract	Produces (i) full-history dataset for plotting and (ii) recomputed 2020+ slice for training/validation. Regime is defined by a 20% rolling peak drawdown rule; for crypto, thresholds/lookbacks are optionally adjusted. Outputs include `Regime`, `Regime_Int`, and lagged `Regime_L1`.
Persistence	Saves full engineered dataset and recomputed 2020+ dataset as CSVs in `preprocessed_data/`. Both versions are downloadable via the UI and stored in session state.
Rationale	Provides transparent, inspectable transformations while avoiding look-ahead leakage.
Failure Modes	Short histories reduce available features; handled via minimum-period guards. Missing price columns trigger validation errors.

Table 4.2: Specification of the Feature Engineering and Regime Tagging Module

4.3.3 Feature Selection

This module identifies predictive features from the engineered dataset while preventing target leakage. It ensures that the regime-aware design is respected by force-including `Regime_L1` (known at time t) and excluding `Regime_Int` (unknown at t). The process involves pruning collinear features, ranking candidates using multiple metrics (Random Forest importance, permutation importance, mutual information, and correlation), and selecting the top-K features. Deterministic Fourier time features (weekly, monthly, quarterly, yearly) are added to capture seasonal effects. The selected feature set is cached, stored in session state, and automatically saved to the portfolio datasets directory, ensuring consistent availability for downstream forecasting.

Purpose	Identify predictive features while preventing leakage.
Public Interface	`_prune_collinear(X, cols, thr)` → List[str] `_full_rank(y, X, cols)` → Ranked feature list
Design Rules	Force-include `Regime_L1` (safe, known at t) and hard-exclude `Regime_Int` (unsafe, unknown at t).
Additional Features	Deterministic Fourier seasonal features (weekly, monthly, quarterly, yearly).
Outputs	Selected feature set stored in session state; engineered dataset and metadata cached and auto-saved to `portfolio_datasets/`.
Rationale	Ensures transparency, prevents look-ahead leakage, and improves model stability through feature ranking and pruning.
Failure Modes	Insufficient data history or high collinearity may reduce usable features; handled via guards, thresholds, and fallback modes (cache → pretrained → full ranking).

Table 4.3: Specification of the Feature Selection Module

4.3.4 Forecasting

This module generates price forecasts on the engineered 2020+ slice (i.e, all data after 2020) using multiple model backends. It dispatches to available pretrained or fast models and returns training, testing, and forecast frames alongside a consistent metrics payload for later best–model selection. Diagnostics (baseline comparisons, leakage tripwires, and time-shuffled checks) are computed to make results auditable. Metrics are persisted both in session state and as JSON files for reproducibility and downstream evaluation.

Backtesting protocol. Evaluation uses the most recent **4 years** of data ($\approx$ 1,008 trading days). A chronological split is applied: the **test set** is the last **252** trading days ($\approx$ 1 year); the **training set** is the preceding $\approx$ 756 trading days ($\approx$ 3 years). One-step walk-forward prediction is used on the test window, benchmarked against a naïve baseline $\hat{P}_t^{\text{naive}} = P_{t-1}$. Reported metrics include *Train*: RMSE, sMAPE; *Test*: RMSE, MAPE, sMAPE, Accuracy (100 − MAPE), RMSE/Naive, returns-RMSE (model and naïve), time-shuffled RMSE (and ratio), and a shift-back tripwire. Sequence backends consume only information available up to $t - 1$. **SARIMAX is not pretrained**: ARIMA (p, d, q) parameters and coefficients are estimated per ticker. GBM, Transformer, LSTM, and Hybrid backends use pretrained weights for inference. Rolling/blocked multi-window backtests are not implemented. For exact pretraining commands and hyperparameters used by the pretrained backends, see Appendix 9.

Purpose	Generate price forecasts on the engineered 2020+ dataset using interchangeable backends and record comparable metrics.
Public Interface	Dispatcher to available backends: SARIMAX (Regime_L1 fast), GBM (exog), Transformer (exog), LSTM (exog, pretrained), and Hybrid (SARIMAX + residual LSTM).
Inputs	2020+ frame (`fe_df`); selected exogenous columns (`exog_cols`); target `Close`; forecast horizon (UI slider); optional pretrained weights/meta for GBM/Transformer/LSTM/Hybrid.
Outputs	Result object with `train_df`, `test_df`, `fcst_df` (with CIs when provided), and `metrics`. UI shows training fit, test vs naive baseline, and last ∼63 days + forecast with CI.
Metrics & Diagnostics	RMSE, MAPE, sMAPE, Accuracy = 100 − MAPE; RMSE/Naive ratio; returns RMSE vs naive; time-shuffled RMSE and ratio; "shift-back better?" leakage tripwire.
Persistence	Session: `model_metrics` (canonical), `forecast_runs_v1` (rich runs), files map; disk: JSON to `metrics/`; CSV metrics download from UI.
Rationale	Isolates model-specific code behind a common dispatcher to enable like-for-like comparison and reproducible evaluation.
Failure Modes	Missing engineered dataset; backend unavailable or missing pretrained files; malformed backend outputs. Mitigation: guarded imports, disabled buttons, and a safe wrapper that surfaces tracebacks in the UI.

Table 4.4: Specification of the Forecasting Module

4.3.5 Model Evaluation and Selection

This module *reads only saved metrics and cached frames* produced by the Forecasting page (no re-fit, no re-run) to compare candidate backends and select a best model per ticker. A deterministic scoring rule ranks models by (i) lower RMSE/Naive, (ii) lower Test RMSE, and (iii) higher Accuracy = 100−MAPE. The precise tie-break algorithm is given in Appendix 8.2.

Overfitting check. A generalisation gap is computed as $\Delta_{\text{gen}} = \text{Test MAPE} - \text{Train MAPE}$ (percentage points). Rows with $\Delta_{\text{gen}} > \tau$ are flagged as *Overfit?* in the comparison table; the threshold τ is user-adjustable in the UI (default $\tau = 0.50$ pp). This uses only saved frames/metrics (no re-fit). For formal definitions of all evaluation metrics and formulas, see Appendix 10; variable names and symbols are listed in Appendix 7.

Purpose	Evaluate saved runs for a ticker and select the best model using a deterministic, reproducible rule (no recomputation).
Public Interface	`_get_metrics_sources_for_ticker(ticker)` → `Dict` (collect session/disk JSON metrics); `_score_for_best(payload)` → `Tuple` (lexicographic: RMSE/Naive ↓, Test RMSE ↓, Accuracy ↑).
Inputs	Saved metrics JSONs under `metrics/` and in-session maps; optional cached frames (`train_df`, `test_df`) to backfill Train/Test MAPE if missing.
Scoring Rule	For each model payload, compute $\mathbf{s} = (\text{RMSE/Naive},\ \text{Test RMSE},\ -\text{Accuracy\%}\,(100-\text{MAPE}))$ and select the model with the minimal **s** (lexicographic). See Appendix 8.2.
Diagnostics	Displays Train/Test windows; Train RMSE, sMAPE, (backfilled) MAPE; Test RMSE, MAPE, sMAPE, Accuracy, RMSE/Naive; returns-RMSE (model/naïve); time-shuffled RMSE and ratio; *shift-back* leakage tripwire; and a user-set overfit flag via Gen gap = Test MAPE − Train MAPE (pp).
Outputs	Comparison table (sortable); chosen label written to the latest instrument `*.meta.json` as `best_model` and to `portfolio_datasets/best_models.json`. CSV export of the comparison table.
Persistence	Session keys: `model_metrics`, `model_metrics_files`, `forecast_runs_v1`. Disk: per-model JSONs in `metrics/`; best-model index in `portfolio_datasets/best_models.json`; in-place update of the latest `*.meta.json`.
Rationale	Selection is based strictly on saved, comparable metrics to avoid recomputation skew and ensure reproducibility.
Failure Modes	Missing/invalid metrics JSON; absent cached frames for MAPE backfill; I/O errors when writing `best_models.json` or updating `*.meta.json`. Mitigations: defensive parsing, informative UI messages, and non-fatal fallbacks.

Table 4.5: Specification of the Model Evaluation and Selection Module

4.3.6 Guided Portfolio Construction

This module combines the user's budget and risk preference with per–ticker short-horizon forecasts to produce an *executable* buy list. It is regime-aware, enforces a hard per-asset cap, and buys whole shares. The allocator distributes capital via a water-filling procedure and then greedily tops up the highest-score affordable names, never breaching the cap.

Purpose	Form an executable plan that maps budget and risk settings to integer share quantities using regime-aware model signals.
Selection policy	Prefer the saved *Best Model* for each ticker (from the Model Evaluation page). If absent, auto-select by the deterministic tie-break: lowest **RMSE/Naive**, then lowest **Test RMSE**.
Key operations	`_regime_by_rolling_peak(prices, lookback)` for regime tagging and scoring nudge; `allocate_with_cap(scores, cap)` (water-filling allocator with a hard per-asset cap); `run_model_for_ticker(...)` to fetch the preferred/auto model result.
Inputs	User budget (USD), risk preset (*Conservative/Balanced/Aggressive*) which sets volatility penalty and a small upside floor, per-asset cap, and per-ticker forecasts (with last price, projected horizon price, recent 30d volatility, and regime flag).
Scoring rule	For each ticker, score = $\max(\mu_H - \mu_{\text{floor}}, 0) \,/\, \sigma_{30}^{\gamma} \times$ RegimeMultiplier, where μ_H is the model view for the next H days, σ_{30} is recent volatility, and γ depends on the risk preset.
Allocation policy	Water-fill proportional to scores with a *hard* per-asset cap (no renormalization after capping). Convert target weights to whole shares; then greedily deploy leftover cash to the highest-score affordable names, still respecting the cap.
Outputs	Plan DataFrame: {`Ticker`, `Model`, `Qty` (integer), `Cost_$`, `ForecastPrice`, `ForecastValue`}. Also surfaces regime/lookback, recent volatility, target weight, and leftover cash.
Failure modes	Missing dataset or unavailable preferred model; no positive model view for the chosen horizon; budget too small for a single share under the cap. These are surfaced as UI warnings and the plan is suppressed.

Table 4.6: Specification of the Guided Portfolio Construction module

See Appendix 8.3 for the water-filling allocator pseudo-code and Appendix 8.1 for the rolling-peak regime rule. Variable names and symbols appear in Appendix 7.

4.3.7 Custom portfolio projection

This module projects the value of a user-defined basket {Ticker $\mapsto$ Quantity} over short horizons using the best-available forecast per ticker and reports results in *currency units*. It runs the saved Best Model where present; otherwise it auto-selects by the deterministic tie-break (lowest **RMSE/Naive**, then lowest **Test RMSE**). Baseline (0d) portfolio value is included for reference. Variable and metric definitions appear in Appendix 7 and Table 7.1; allocator and regime rules are given in Appendices 8.3 and 8.1.

Purpose	Project absolute portfolio values for a user-specified basket across multiple horizons.
Public interface	`_run_model_for_ticker()` runs the preferred/best-available backend up to the maximum horizon and returns `fcst_df`. `_proj_price(fcst_df, h)` extracts the price forecast at horizon h.
Inputs	Basket map {Ticker $\mapsto$ Qty}; latest engineered dataset per ticker; horizons $\{0, 5, 10, 15, 30\}$ days; saved Best Model label if available.
Data contract	Per-ticker table with **absolute** values: `Position_$ = Last * Qty`, and projected position values `F5d, F10d, F15d, F30d = P(h) * Qty`. Also produces a portfolio totals series $\{(h, \sum_i P_i(h) \cdot \text{Qty}_i)\}$ for plotting and a % return column relative to h=0.
Outputs	(1) Per-ticker summary: {`Ticker, Qty, LastClose, Total, F5d, F10d, F15d, F30d, Model`}; (2) Portfolio projection table: {`Horizon, Portfolio (USD), % Return`}; (3) Line chart of projected portfolio value.
Rationale	Reporting in currency units (rather than only percentages) aligns with practical decision-making for sizing and cash planning.
Failure modes	Missing dataset/model for a ticker, or empty `fcst_df`; handled by inserting `NaN` projections and surfacing warnings while still computing portfolio totals from available series.

Table 4.7: Specification of the Custom Portfolio Projection module

4.3.8 Presentation layer (Streamlit)

The Streamlit presentation layer provides interactive pages for data acquisition, analysis, modelling, evaluation, and planning, organised in a "one step per page" flow (selection $\rightarrow$ analysis $\rightarrow$ configuration $\rightarrow$ results). User interfaces rely on `st.dataframe`, `st.metric`, and `st.altair_chart`, with application state maintained in `st.session_state`. Controlled file writes support reproducibility: per-run metrics are saved to `metrics/`, engineered datasets and metadata to `portfolio_datasets/`, and optional forecasts to `forecasts/` (see Table 4.8). Variable and metric definitions referenced by the UI are consolidated in Appendix 7 and Table 7.1.

Artefact summary

Module $\rightarrow$ artefacts	Location
Saved metrics (per run)	`metrics/{TICKER}_{model}_*.json`
Engineered datasets + meta	`portfolio_datasets/*.csv`, `portfolio_datasets/*.meta.json`
Best model index	`portfolio_datasets/best_models.json`
Optional forecasts	`forecasts/*.csv`, `forecasts/*.json`

Table 4.8: Files written by the Streamlit presentation layer.

4.3.9 Reproducibility and environment

The application fixes random seeds where applicable, caches deterministic transformations, and records timestamps and file paths in per-run metadata. Experiments were executed on Python 3.8.10 with Pandas, NumPy, Altair, and Streamlit; model backends additionally use scikit-learn, statsmodels, and PyTorch. All artefact paths are relative to the project root. Session-derived choices (e.g., selected features and forecast horizons) are persisted alongside metrics to ensure run-to-run traceability. A full dependency listing is provided in Appendix 12.

4.4 Considered and rejected alternatives

Monolithic script. Rejected due to poor maintainability and testability; modularisation enables clear interfaces, unit tests, and independent evolution.

Database backend. A DBMS (e.g. SQLite/PostgreSQL) is appropriate for multi-user deployment, but for this single-user research prototype the filesystem plus session state reduce operational complexity without compromising validity.

Full web stack (React/Django). A custom frontend would offer greater UI flexibility, but Streamlit maximises research velocity: it provides rich widgets and plotting without maintaining a separate API layer.

Real-time streaming. Streaming architectures are orthogonal to the research question (batch, forecast-driven planning) and add considerable complexity; windowed forecasts on end-of-day data suffice for the evaluation.

Alternative regime definitions. Thresholds other than a 20% rolling-peak drawdown (or higher for crypto) were considered (e.g. trend filters), but the adopted rule offers transparency, ease of auditing, and low risk of leakage.

4.4.1 Detailed design highlights

The system's core functionality is driven by several key algorithms. For brevity, only the non-trivial components are detailed here; the connective logic is omitted.

Regime Detection

A critical step in adapting the forecasting strategy is to identify the current market regime (e.g., "Bull" or "Bear" market). This is achieved using a rolling-window peak drawdown method. For each time point, the algorithm looks back over a defined window (L), identifies the peak price within that window, and calculates the current drawdown from that peak. If the drawdown exceeds a set threshold, the market is classified as a "Bear" regime; otherwise, it is considered "Bull." The formal pseudo-code for this algorithm is presented in Appendix 8.1.

Model Selection

To choose the best forecasting model from a candidate set, a deterministic scoring and tie-breaking system is employed. Each model is scored based on three criteria: its RMSE improvement over a naive baseline, its absolute test RMSE, and its inverse MAPE. The models are then ranked, with the top model selected based on a primary sort on the first criterion, a secondary sort on the second, and a final sort on the third. The precise scoring and sorting logic is provided in Appendix 8.2.

Portfolio Allocation

The capital allocation strategy is implemented via a water-filling algorithm, constrained by a per-asset capital cap. This algorithm distributes the investment budget (B) across assets in proportion to their predictive scores (w_i), ensuring no single asset receives more than a predefined fraction (c) of the total portfolio. The iterative procedure for this allocation is detailed in Appendix 8.3.

4.4.2 Data representations

- **Raw/engineered:** `Date (datetime64)`, `Close (float64)`, OHLCV; engineered indicators as extra float columns.
- **Backtests:** `train_df`, `test_df` with `Date`, `Actual`, `Pred`.
- **Forecasts:** `fcst_df` with `Date`, `Pred`, optional `Lower`, `Upper`.
- **Portfolio tables:** `Ticker (str)`, `Qty (int)`, `Last (float)`, `Position_$ = Last * Qty`, and absolute projections `Val5d_$, Val10d_$, Val15d_$, Val20d_$, Val30d_$ = Pred(h) * Qty`.

4.4.3 Metrics and tie-break rule

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2}, \quad \text{MAPE} = \frac{100}{n}\sum_{t=1}^{n}\left|\frac{y_t - \hat{y}_t}{y_t}\right|, \quad \text{sMAPE} = \frac{200}{n}\sum_{t=1}^{n}\frac{|y_t - \hat{y}_t|}{|y_t| + |\hat{y}_t|}.$$

Variable Definitions:

- n: The number of observations (data points) in the test set. This is the held-out portion of the data that the model has not seen during training and is used for final evaluation.
- t: An index representing a specific time period within the test set. It runs from 1 to n.
- y_t: The actual, real-world observed value of the target variable at time t. *In this case: the true, recorded `Close` price on day t of the test set.*
- $\hat{y}_t$: The predicted value or forecast made by the model for time t.

Model selection sorts by (i) RMSE/Naive (ascending), (ii) Test RMSE (ascending), (iii) $100 - \text{MAPE}$ (descending).

4.4.4 Implementation issues and mitigations

- **Altair axis collapse:** categorical horizons formed via `astype(str)` (not `str(Series)`).
- **Projection semantics:** show absolute currency values; avoid percent deltas in the custom portfolio.
- **Leakage guards:** exclude `Regime_Int`; include `Regime_L1`; add shift-back tripwire and time-shuffled RMSE.
- **Crypto regimes:** higher threshold / shorter lookback for high-volatility assets; rule displayed alongside last close.

4.4.5 Evaluation protocol (summary)

Backtesting: rolling 1-step tests on data 2020 onwards (2020+ slice); naive baseline = previous close.
KPIs: cumulative return, Sharpe, max drawdown; forecasting KPIs as above.
Benchmarks: buy-and-hold, equal-weight, static risk parity.

Reporting: selection by RMSE/Naive → Test RMSE → 100–MAPE; portfolio results shown in absolute USD.

Chapter 5

Design of the Experiments

This chapter details the experimental framework designed to evaluate the *BullVision* system. The evaluation is bifocal, assessing (i) the quantitative performance of its predictive models and (ii) the qualitative utility of its portfolio construction and user experience.

5.1 Purpose and Overview of Experimental Strategy

The core hypothesis is that BullVision can provide novice investors with actionable, data-driven investment guidance that is both accurate and accessible. Two research questions (RQs) guide the evaluation:

- **RQ1 (Quantitative):** Does the system's automated model selection consistently choose forecasting techniques that provide better and statistically meaningful predictions over a naive baseline?
- **RQ2 (Qualitative):** Does the system generate portfolio plans that are effective, explainable, and usable for individuals with limited financial expertise?

To answer these, the evaluation leverages the system's built-in capabilities for quantitative analysis and supplements them with a targeted user study.

Table 5.1: Summary of experimental strategy

Research Question	Experiment Type	Method	Primary Metrics
RQ1 (Model Performance)	Quantitative Analysis	Use the Model Evaluation module (`05_Best_Model.py`) to assess performance on a curated basket of assets.	RMSE/Naive, Accuracy (%)
RQ2 (System Utility)	Qualitative User Study	Task-based testing with novice participants using the live application.	Task success rate, thematic feedback

5.2 Experiment 1: Quantitative Model Performance Evaluation

This experiment uses the existing Model Evaluation module to rigorously test the forecasting core of the system.

5.2.1 Methodology and Use of Existing System Components

Objective. Determine whether the model selection and forecasting pipeline provides a measurable advantage over a simple baseline.

Procedure.

1. **Asset selection:** Choose a diverse basket (e.g., {`SPY`, `QQQ`, `GLD`, `BTC-USD`}) spanning equities, commodities, and crypto.
2. **Data preparation:** For each ticker, fetch and prepare historical data using the Data Acquisition module (`01_Market_Data_Scraper.py`).
3. **Pipeline execution:** For each ticker, run Feature Engineering (02), Feature Selection (03), and Forecasting (04) across available backends (SARIMAX, GBM, Transformer, LSTM). This populates `metrics/` with JSON metric files.
4. **Analysis:** Invoke the Model Evaluation module (`05_Best_Model.py`) to gather saved metrics and apply the ranking rule (RMSE/Naive $\rightarrow$ Test RMSE $\rightarrow$ Accuracy). The resulting comparison table forms the primary dataset.
5. **Baseline comparison:** Compare the selected model's metrics (RMSE, Accuracy) against thresholds and the implicit naive baseline (captured via `RMSE_over_Naive`).

5.2.2 Success Criteria and Evaluation Method

The experiment is considered successful if:

- **Practical significance:** For each asset, the selected "best" model achieves RMSE/Naive $<$ 1.0, demonstrating outperformance over the naive forecast.
- **Forecast accuracy:** The selected model achieves Accuracy $= 100 - \text{MAPE} > 50\%$ for the majority of assets in the test basket.

A key step in the quantitative analysis is the aggregation of individual asset metrics to evaluate overall model performance and uncover patterns across different asset classes. This process moves beyond single-asset evaluation to provide a holistic view of the system's forecasting capabilities.

5.3 Experiment 2: Qualitative Usability and Utility Study

5.3.1 Methodology

Objective. Assess whether the system is easy to use, trustworthy, and useful for its target audience of novice investors.

Participants. Recruited 5-10 participants who self-identified as having limited knowledge of stock market analysis.

Tasks. Each participant was advised to use a $10,000 budget in the prototype and asked to complete:

- **Task 1 (Guided):** Use the Guided Portfolio Builder to generate a diversified portfolio plan.
- **Task 2 (Custom):** "You own 15 shares of SPY and 10 shares of DIA. Use the Custom Portfolio tool to project what this might be worth in one month."

Data collection. Direct observation and a short semi-structured interview.

5.3.2 Success Criteria

- **High task efficacy:** 80% of the participants completed both tasks without significant intervention.
- **Positive qualitative feedback:** Thematic analysis indicates the system is understandable and empowering.

5.4 Implementation and Ethical Considerations

Ethics. Participants provided informed consent and were informed that BullVision is a research prototype and not financial advice.

Chapter 6

Analysis of the experiment results

6.1 Individual ticker analysis

Evaluation setup. Experiments operate on the engineered 2020+ slice. For each ticker, the last 252 trading days ($\approx$ 1 year) constitute the test window; all earlier 2020+ observations form the training window. Evaluation is one-step, walk-forward: at day t in the test window, the forecast for t uses only information available up to $t-1$. Regime awareness is enforced uniformly across backends by including the lagged regime indicator `Regime_L1` (safe at t) as a mandatory exogenous feature; the contemporaneous `Regime_Int` is excluded to prevent leakage. A naïve persistence baseline (yesterday's price) is computed in parallel. Reported metrics are RMSE, MAPE, sMAPE, Accuracy (100 − MAPE), the RMSE/Naive ratio, and returns-RMSE (log-return error vs. the naïve). Two diagnostics are logged for auditability: a shift-back tripwire (does $\hat{y}_{t-1}$ beat $\hat{y}_t$?) and a time-shuffled RMSE (predictions compared after shuffling time order).

Aggregate patterns. On many tickers with adequate history, at least one learned model improves upon the persistence baseline ((RMSE/Naive < 1), while for a sizable share the best model's error is very close to the baseline ((RMSE/Naive ≈ 1), indicating limited short-horizon signal. Accuracy and sMAPE naturally degrade as the forecast horizon increases. Where regime persistence and seasonal structure are present, the enforced *Regime_L1* and simple seasonal/technical covariates tend to lower price-RMSE and improve returns-RMSE, with the clearest gains around regime transitions. Results for ticker `QQQ` (Invesco) are given below in Fig. 6.1 to illustrate several of the reported metrics.

Model	Train RMSE	Train sMAPE%	Train MAPE%	Test RMSE	Test sMAPE%	Test MAPE%	RMSE / Naive
Transformer	5.17	1.14	1.15	7.20	0.96	0.96	**0.98**
Hybrid	5.14	1.13	1.13	7.16	0.96	0.96	**0.99**
GBM	5.69	1.27	1.27	7.10	0.95	0.95	**1.00**
LSTM	5.16	1.14	1.14	7.36	0.96	0.96	**1.00**
SARIMAX	14.71	1.38	1.25	7.17	0.96	0.96	**1.01**

Figure 6.1: QQQ : training/test metrics and forecast preview.

6.2 Overall model analysis

6.2.1 Aggregate evaluation methodology

The evaluation compares each forecasting model against a naïve persistence baseline (predicting the last observed value). Key aspects are:

- **Primary Metric.** Models are evaluated using the RMSE ratio relative to the naïve model,
$$\texttt{rmse\_over\_naive} = \frac{\text{RMSE}_{\text{model}}}{\text{RMSE}_{\text{naive}}}.$$
A ratio < 1 indicates the model outperforms the naïve baseline.
- **Win Definition.** A *win* is counted when $\texttt{rmse\_over\_naive} < 1.0$. Results within $\pm\,\texttt{near\_naive\_eps}$ of 1.0 are flagged as "near-naive."
- **Statistical Rigor.**
  - 95% Wilson score confidence intervals are computed for win rates to quantify uncertainty.
  - Small sample sizes ($n <$ `small_n_thresh`) are flagged to highlight potentially unreliable results.
- **Tie-Breaking.** The best model per ticker is selected by (i) lowest RMSE ratio ↓, then (ii) lowest RMSE ↓, then (iii) highest accuracy ↑.
- **Aggregation.** Results are summarized both overall and by predefined asset categories (e.g., ETFs, crypto, bonds, etc.).

See Appendix 11 for concise pseudocode.

6.2.2 Overall Model Performance

The overall results, aggregated across all 34 assets, reveal a clear hierarchy in model performance. Table 6.1 summarizes the key statistics for each model.

Model	Wins	Winrate	Near-naive	Median RMSE ratio	Median RMSE
Transformer	23	67.6%	50.0%	0.984	0.875
Hybrid (SARIMAX + Residual)	26	76.4%	97.05%	0.997	0.877
LSTM	16	47.0%	50.0%	1.001	0.875
GBM	9	27.3%	48.48%	1.028	1.010
SARIMAX	0	0.0%	20.58%	1.033	1.004

Table 6.1: Overall model-wise summary. Lower is better for RMSE/Naive.

Note: Win Rate is the proportion of assets where the model outperformed the naive baseline (RMSE ratio < 1.0). Best Count is the number of assets for which the model was the top performer.

Model	Best count
Transformer	20
Hybrid (SARIMAX + Residual)	10
LSTM	3
GBM	1
SARIMAX	0

Table 6.2: Overall best-model counts (tie-break: RMSE/Naive → Test RMSE → Accuracy).

Key Findings

- **Dominance of Transformer model** The Transformer-based model emerges as the most consistently superior model. It not only has a strong win rate of 67.6% but is also the single best model (best_count) on a majority of assets (20 out of 34), significantly more than any other model. Its median RMSE ratio of 0.984 indicates a median 1.6% improvement over the naive baseline.
- **The Hybrid model paradox.** The Hybrid Residual model presents an interesting case. It boasts the highest overall win rate (76.5%) but a lower best count than the Transformer. Furthermore, its median RMSE ratio (0.997) is very close to 1.0, and it is flagged as "near naive" on 97% of assets. This suggests that while the hybrid model very frequently matches or slightly beats the naive baseline (hence the high win rate), it rarely provides the substantial improvement needed to be the absolute best model (hence the lower best count). It is a model of consistency rather than peak performance.
- **Competitive but less effective models.** The LSTM model is competitive on median RMSE but has a win rate below 50%, indicating inconsistent performance. The gradient boosting model and the SARIMAX model are clearly outperformed by the deep learning approaches, with SARIMAX failing to beat the naive baseline on any asset.

6.2.3 Performance by Asset Category

A more nuanced picture emerges when results are broken down by asset category as cited in Table 6.3. This analysis is crucial as it demonstrates that model efficacy is not universal but is highly dependent on the characteristics of the asset being forecasted.

Category	Top Model	Total Assets in category	No. of Assets won
US Sector ETFs	Transformer	10	7
Commodities	Transformer	5	4
US Index ETFs	Transformer	4	4
Crypto	Transformer	5	3
Bonds	Hybrid	5	3
International ETFs	LSTM	5	2

Table 6.3: Best median RMSE/Naive by category.

Key Findings

- **Transformer for Complex, Volatile Markets** The Transformer model demonstrates remarkable dominance in categories known for complex, non-linear dynamics and higher volatility. It is the top performer in US Sector ETFs, Commodities, US Index ETFs, and Crypto. This strongly suggests that the Transformer's attention mechanism is exceptionally well-suited for capturing intricate patterns and long-range dependencies in these noisy markets.
- **Hybrid Model for Stable, Mean-Reverting Series** The Hybrid model finds its strength in the more stable and potentially mean-reverting Bonds & Rates category. This aligns with the expectation that a model combining linear (ARIMA) and non-linear (LSTM) components can effectively capture the structure in such time series.
- **No Single Best Model** The results for International ETFs show a more fragmented picture, with the LSTM model securing the most wins. This indicates that this category may possess unique characteristics where a simpler sequential model can sometimes outperform the more complex Transformer architecture.

6.2.4 Conclusion of Evaluation

The evaluation unequivocally demonstrates that sophisticated deep learning models, particularly the Transformer architecture with exogenous variables, provide the most significant and consistent forecasting advantage over a naive baseline across a diverse portfolio of financial assets. Its status as the top performer on a majority of assets solidifies its position as the most robust and accurate model tested.

However, the strong performance of the Hybrid Residual model, particularly in specific contexts like Bonds & Rates and its remarkable consistency, highlights that optimal model selection is context-dependent. This suggests a potential trade-off: the hybrid model offers practitioners a reliable, "good-enough" solution with extremely high consistency, while the Transformer architecture provides researchers and quants with superior peak predictive accuracy.

Collectively, this empirical evidence strongly supports the hypothesis that these modern architectures are effectively extracting and utilizing complex patterns—interpretable as regime information—from the sequential data. They thereby deliver a significant and measurable improvement in financial time series forecasting, adding tangible value over simple benchmark models.

6.3 Critical Evaluation of Methodology and Experimental Design

6.3.1 Appropriateness of the Experimental Design

The experimental design was highly appropriate and effectively served the primary goal of the investigation. The results were not only useful but also provided a clear, nuanced hierarchy of model performance.

Strength of the Regime Definition: The choice of a 20% drawdown from peak levels as the criterion for regime detection was a key strength of the methodology. This is not an

implicit or vague definition; it is an explicit, quantitative, and widely accepted rule of thumb in finance for identifying bear markets. This provided a concrete, reproducible filter for the "regime information" central to the hypothesis.

Usefulness of Results: The experiments successfully demonstrated that modern neural networks (Transformer, Hybrid), when provided with this explicit regime context, significantly outperformed both simpler models and the naive baseline. This offers strong, direct support for the hypothesis, showing that regime-aware models add tangible value.

6.3.2 Limitations and Potential for Re-design

While the methodology was strong, the analysis of the results could be extended to provide even deeper insights.

1. **Limitation: Analysis of Regime-Specific Performance**
   Critique: The current results brilliantly show that models trained with regime information perform better overall. However, they do not explicitly show how each model performed specifically during the defined bear market regimes versus bull market regimes. This is a missed opportunity for a deeper layer of analysis.
   Re-design for Deeper Insight: The most valuable enhancement would be to segment the forecast evaluation based on the regime.
   How to do it: Using the 20% drawdown rule, each point in the forecast period can be labeled as "in-regime" (bear market) or "out-of-regime" (bull market or recovery).
   textbfNew Analysis: The metrics (RMSE ratio, Accuracy) should be calculated separately for these two subsets. This would answer critical new questions:

   - Did the Transformer model's superiority come from its performance during the stressful regime, the calm regime, or both?
   - Was the Hybrid model more robust during the drawdown period?

   Why it's better: This redesign would move the conclusion from "Model X works well" to the more powerful and actionable "Model X works well because it excels specifically during Y conditions." This directly tests the mechanism behind the hypothesis.

2. **Limitation: Point Forecasts vs. Predictive Distributions**
   textbfCritique: The evaluation focused on point forecasts. Understanding uncertainty is paramount during volatile, regime-shift periods.
   textbfRe-design for Usefulness: Enhance the models to produce probabilistic forecasts (e.g., predicting a 90% confidence interval). Their performance could then be evaluated on calibration during the high-volatility bear regime: does their 90% interval actually contain the true value 90% of the time? A model that remains well-calibrated during stress is immensely more useful.

3. **Limitation: Comparison to a Simpler Regime-Aware Baseline**
   textbfCritique: The benchmark was a naive model. A stronger, more rigorous comparison would be against a simpler yet still regime-aware model.
   textbfRe-design for Usefulness: Introduce a baseline like a GARCH model or a two-stage model (e.g., "if in drawdown, use model A; else, use model B"). This would test if the complexity of the neural networks is necessary, or if a simpler regime-switching approach captures most of the benefits. It would more precisely isolate the value added by the architecture itself.

6.3.3 Appropriateness of the Overall Methodology

The overall methodology was excellent and highly appropriate. It was built on a solid foundation of an explicit regime definition, a diverse asset universe, and a robust multi-metric evaluation framework.

Future research efforts would prioritize the following redesigns:

1. Conduct a regime-specific performance analysis: This is the most important addition. Segmenting the results by your existing 20% drawdown flag would yield the most significant new insights without any additional data collection or model retraining. The data to do this is likely already present in your results.
2. Compare against a regime-aware baseline: This would strengthen the validity of the conclusions by testing against a more sophisticated benchmark.
3. Incorporate probabilistic forecasting: This would expand the scope of the research from pure accuracy to the highly valuable domain of risk and uncertainty quantification.
4. Incorporate Economic Indicators as Exogenous Variables: These indicators would provide the models with forward-looking, fundamental context that pure price data lacks. This could significantly enhance predictive accuracy, especially around regime transition points, by allowing the model to anticipate changes based on economic conditions rather than just react to them in the price series.

Number of words until this point, excluding front matter: 9817.

Bibliography

A, A., R, R., S, V.R. and Bagde, A.M., 2023. Predicting stock market time-series data using cnn-lstm neural network model [Online]. 2305.14378, Available from: https://arxiv.org/abs/2305.14378.

Ahmad, W., Shadaydeh, M. and Denzler, J., 2024. Regime identification for improving causal analysis in non-stationary timeseries [Online]. 2405.02315, Available from: https://arxiv.org/abs/2405.02315.

AnalystPrep, 2024. Mean-variance portfolio theory [Online]. Accessed: 2025-05-18. Available from: https://analystprep.com/study-notes/actuarial-exams/soa/ifm-investment-and-financial-markets/mean-variance-portfolio-theory/.

Aramyan, H., Ramchandani, J. and Skevofylakas, M., 2023. Market regime detection using statistical and ml-based approaches. https://developers.lseg.com/en/article-catalog/article/market-regime-detection. Accessed: 2025-05-04.

Chen, T. and Guestrin, C., 2016. Xgboost: A scalable tree boosting system. *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. pp.785–794.

Erlwein-Sayer, C., Grimm, S., Pieper, A. and Alsaç, R., 2023. Forecasting corporate credit spreads: Regime-switching in lstm. *Econometrics and statistics*.

Guo, Y., 2025. Applications of time series analysis in quantitative finance. *Available at ssrn 5140015*.

Hamilton, J.D., 1989. A new approach to the economic analysis of nonstationary time series and the business cycle. *Econometrica* [Online], 57(2), pp.357–384. Available from: https://doi.org/10.2307/1912559.

Hartford Funds, 2023. 10 things you should know about bear markets. https://www.hartfordfunds.com/practice-management/client-conversations/managing-volatility/bear-markets.html. Accessed: 2025-04-28.

He, P., Zhao, L., Zhou, S. and Niu, Z., 2013. Water-filling: A geometric approach and its application to solve generalized radio resource allocation problems. *leee transactions on wireless communications* [Online], 12(7), pp.3637–3647. Available from: https://doi.org/10.1109/TWC.2013.061713.130278.

Huang, G., Zhou, X. and Song, Q., 2025. A deep reinforcement learning framework for dynamic portfolio optimization: Evidence from china's stock market [Online]. 2412.18563, Available from: https://arxiv.org/abs/2412.18563.

Investopedia, 2024. *Efficient frontier: What it is and how investors use it* [Online]. Accessed: 2025-04-18. Available from: `https://www.investopedia.com/terms/e/efficientfrontier.asp`.

London Stock Exchange Group, 2024. Market regime detection. `https://developers.lseg.com/en/article-catalog/article/market-regime-detection`. Accessed: 2025-05-04.

Mc Greevy, J., Muguruza, A., Issa, Z., Salvi, C., Chan, J. and Zuric, Z., 2024. Detecting multivariate market regimes via clustering algorithms. *Available at ssrn 4758243*.

Ntantamis, C., 2009. A duration hidden markov model for the identification of regimes in stock market returns. *Available at ssrn 1343726*.

Orton, A. and Gebbie, T., 2024. Representation learning for regime detection in block hierarchical financial markets [Online]. `2410.22346`, Available from: `https://arxiv.org/abs/2410.22346`.

Qi, Q., Minturn, A. and Yang, Y., 2012. An efficient water-filling algorithm for power allocation in ofdm-based cognitive radio systems [Online]. *2012 international conference on systems and informatics (icsai2012)*. pp.2069–2073. Available from: `https://doi.org/10.1109/ICSAI.2012.6223460`.

QuantConnect, 2025. Intraday application of hidden markov models [Online]. Accessed: 2025-04-28. Available from: `https://www.quantconnect.com/research/17900/intraday-application-of-hidden-markov-models/p1`.

Shu, Y., Yu, C. and Mulvey, J., 2024a. Regime-aware asset allocation: a statistical jump model approach. *Ssrn electronic journal* [Online]. Available from: `https://doi.org/10.2139/ssrn.4719989`.

Shu, Y., Yu, C. and Mulvey, J.M., 2024b. Downside risk reduction using regime-switching signals: a statistical jump model approach. *Journal of asset management* [Online], 25(5), p.493–507. Available from: `https://doi.org/10.1057/s41260-024-00376-x`.

Suárez-Cetrulo, A.L., Quintana, D. and Cervantes, A., 2024. Machine learning for financial prediction under regime change using technical analysis: A systematic review. *International journal of interactive multimedia and artificial intelligence* [Online], 9(1), pp.137–148. Available from: `https://doi.org/10.9781/ijimai.2023.06.003`.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Vuong, P.H., Phu, L.H., Van Nguyen, T.H., Duy, L.N., Bao, P.T. and Trinh, T.D., 2024. A bibliometric literature review of stock price forecasting: from statistical model to deep learning approach. *Science progress*, 107(1), p.00368504241236557.

Yuan, Y. and Mitra, G., 2016. Market regime identification using hidden markov models. *Available at ssrn 3406068*.

Chapter 7

Appendix A: Variable Definitions

Variable	Meaning
`Date`	Trading date for each observation.
`Open, High, Low, Close, Adj Close, Volume`	Standard OHLCV price and volume data.
`Return`	Daily percentage return: $\frac{Close_t - Close_{t-1}}{Close_{t-1}}$.
`LogReturn`	Daily log return: $\log(Close_t) - \log(Close_{t-1})$.
`SMA_w`	Simple moving average of closing price over w days (e.g., `SMA_50`, `SMA_200`).
`EMA_w`	Exponential moving average of closing price over w days (e.g., `EMA_12`, `EMA_26`).
`MACD`	Difference between `EMA_12` and `EMA_26` (trend momentum).
`MACD_Signal`	9-day EMA of MACD (signal line).
`MACD_Hist`	Histogram: `MACD` minus `MACD_Signal`.
`Volatility_20, Volatility_60`	Rolling standard deviation of returns over 20/60 days.
`BB_Mid_20`	20-day rolling mean of closing price (Bollinger midpoint).
`BB_Upper_20, BB_Lower_20`	Upper/lower Bollinger Bands: midpoint $\pm 2\times$ rolling std.
`RSI_14`	Relative Strength Index over 14 days (0–100).
`DOW`	Day of week (0 = Monday, 6 = Sunday).
`Month`	Calendar month (1–12).
`Peak`	Running maximum of the closing price (rolling peak).
`Drawdown_from_peak`	Percentage drawdown from the rolling peak: $\frac{Close_t}{Peak_t} - 1$.
`Regime`	Market regime label: Bull or Bear (based on drawdown threshold).
`Regime_Int`	Numeric regime indicator: 1 = Bull, 0 = Bear.
`Regime_L1`	Lagged regime value (safe feature known at time t).
`Fourier Features`	Deterministic seasonal features capturing weekly, monthly, quarterly, and yearly cycles.
`AggScore`	Aggregate feature-importance score combining Random Forest, permutation, mutual information, and correlation metrics.
`RF_importance`	Feature-importance score from a Random Forest model.
`Perm_importance`	Permutation-importance score from a validation set.
`MI`	Mutual information score with the target.
`|Corr|`	Absolute correlation with the target variable ($\|\text{Corr}\|$).

Table 7.1: Variable definitions used in feature engineering, regime tagging, and feature selection.

Chapter 8

Appendix B: Key Algorithms

This appendix contains the formal pseudo-code for the non-trivial algorithms referenced in Chapter 4.

8.1 Regime Detection Algorithm (Rolling-Peak Drawdown)

```
Input: close[1..T], lookback L, threshold
for t in 1..T:
    peak_t = max(close[max(1,t-L+1)..t])
    dd_t = close[t] / peak_t - 1
    regime[t] = "Bear" if dd_t <= -threshold else "Bull"
Output: regime[1..T], dd[1..T]
```

8.2 Best-Model Scoring Algorithm (Deterministic Tie-Break)

```
For each model m:
    s1 = RMSE_over_Naive(m)  # lower is better
    s2 = Test_RMSE(m)        # lower is better
    s3 = 100 - MAPE(m)       # higher is better
Sort models by (s1 asc, s2 asc, s3 desc)
Pick first model in sorted list.
```

8.3 Water-Filling Allocation Algorithm (With Per-Asset Cap)

```
Input: scores w_i >= 0, cap c in (0,1], budget B
a_i = 0; R = 1
while R > eps and exists i with a_i < c and w_i > 0:
    S = {i | a_i < c and w_i > 0}
```

```
    p_i = w_i / sum_{j in S} w_j
    give_i = min(c - a_i, R * p_i)
    a_i += give_i; R -= sum(give_i)
qty_i = floor( (a_i * B) / price_i )
```

Chapter 9

Appendix C: Pretraining Log and Hyperparameters

This appendix records the pretraining commands, artifacts, and hyperparameters for the pretrained backends (Transformer, LSTM, GBM, and Hybrid). Keeping this log updated preserves full reproducibility.

9.1 Transformer (Exog)

Command

```
python tools/train_global_transformer_exog.py \
  --data_dir preprocessed_data \
  --out_path pretrained_models/global_transformer_exog.pt \
  --out_meta pretrained_models/global_transformer_exog.meta.json \
  --lookback 30 --epochs 12 --feature_mode union
```

Parameters

lookback	30
epochs	12
feature_mode	union

9.2 LSTM (Exog)

Command

```
python tools/train_global_lstm_exog.py \
  --seq_len 60 --hidden 128 --layers 2 --dropout 0.2 \
  --epochs 30 --batch_size 512 --lr 1e-3
```

Parameters

seq_len	60
hidden	128
layers	2
dropout	0.2
epochs	30
batch_size	512
lr	1×10^{-3}

9.3 GBM (Exog)

Command

```
python tools/train_global_gbm_exog.py \
  --data_dir preprocessed_data \
  --out_path pretrained_models/global_gbm_exog.joblib \
  --trees 600 --depth 3 --lr 0.03
```

Parameters

trees	600
depth	3
lr	0.03

9.4 Hybrid (SARIMAX + Residual LSTM)

Command

```
python tools/train_hybrid_resid_lstm.py \
  --data_dir preprocessed_data \
  --out_model pretrained_models/hybrid_resid_lstm.pt \
  --out_meta pretrained_models/hybrid_resid_lstm.meta.json \
  --seq_len 15 --epochs 30 --lr 5e-4 --hidden 128 --dropout 0.3
```

Parameters

seq_len	15
epochs	30
lr	5×10^{-4}
hidden	128
dropout	0.3

Last updated: 2025-09-06

Chapter 10

Appendix D: Evaluation Metrics and Formulas

All symbols follow the conventions in Appendix 7. Let y_t be the actual price at time t, $\hat{y}_t$ the model prediction, n the number of valid observations in the given window, and $\epsilon > 0$ a small tolerance to avoid division by zero. For returns-based metrics define $r_t = \ln\left(\frac{y_t}{y_{t-1}}\right)$ and $\hat{r}_t = \ln\left(\frac{\hat{y}_t}{y_{t-1}}\right)$. The naïve baseline is $\hat{y}_t^{\text{naive}} = y_{t-1}$.

Metric (label)	**Symbol**	**Formula**	**Meaning / Window**
RMSE (price)	RMSE	$\sqrt{\frac{1}{n}\sum_{t=1}^{n}\left(\hat{y}_t - y_t\right)^2}$	Average error magnitude in price units. Reported for *Train* and *Test*.
MAPE (%)	MAPE	$\frac{100}{m}\sum_{t:\,\lvert y_t\rvert>\epsilon}\left\lvert\frac{y_t-\hat{y}_t}{y_t}\right\rvert$	Scale-free percentage error (undefined near 0). Reported for *Test*.
sMAPE (%)	sMAPE	$\frac{200}{m}\sum_{t=1}^{m}\frac{\lvert y_t-\hat{y}_t\rvert}{\lvert y_t\rvert+\lvert\hat{y}_t\rvert}$	Symmetric percentage error. Reported for *Train* and *Test*.
Accuracy (%)	$100-\text{MAPE}$	$100-\text{MAPE}$	Complement of MAPE; higher is better. *Test* only.
RMSE / Naive (ratio)	RMSE/Naive	$\frac{\text{RMSE}_{\text{model}}}{\text{RMSE}_{\text{naive}}},\quad \text{RMSE}_{\text{naive}}=\sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t-y_{t-1})^2}$	Relative to persistence; <1 means beating naïve. *Test*.
RMSE (returns)	RMSE_r	$\sqrt{\frac{1}{n}\sum_{t=1}^{n}\left(\hat{r}_t-r_t\right)^2},\quad r_t=\ln\frac{y_t}{y_{t-1}},\ \hat{r}_t=\ln\frac{\hat{y}_t}{y_{t-1}}$	Error on *log returns*. *Test*.
RMSE (returns, naïve)	$\text{RMSE}_r^{\text{naive}}$	$\sqrt{\frac{1}{n}\sum_{t=1}^{n}r_t^2}\ \ (\text{since}\ \ \hat{r}_t^{\text{naive}}=0)$	Returns error for persistence baseline. *Test*.
Time-shuffled RMSE	$\text{RMSE}_{\text{shuf}}$	$E_\pi\left[\sqrt{\frac{1}{n}\sum_{t=1}^{n}\left(\hat{y}_{\pi(t)}-y_t\right)^2}\right]$	RMSE after randomly permuting predictions; probes order exploitation. *Test*.
Shuffled/Model RMSE (ratio)	$\text{RMSE}_{\text{shuf}}/\text{RMSE}$	$\frac{\text{RMSE}_{\text{shuf}}}{\text{RMSE}_{\text{model}}}$	>1 suggests the model uses temporal structure. *Test*.
RMSE (shift-back)	$\text{RMSE}_{\text{shift}}$	$\sqrt{\frac{1}{n}\sum_{t=1}^{n}\left(\hat{y}_{t-1}-y_t\right)^2}$	Tripwire for leakage/phase error (uses previous prediction). *Test*.
Leakage tripwire (bool)	1_{leak}	$1\left[\text{RMSE}_{\text{shift}}<\text{RMSE}_{\text{model}}\right]$	Flags if using $\hat{y}_{t-1}$ *beats* $\hat{y}_t$. *Test*.
Generalisation gap (pp)	Δ_{gen}	$\Delta_{\text{gen}}=\text{MAPE}_{\text{Test}}-\text{MAPE}_{\text{Train}}$	Overfitting indicator in percentage points; threshold user-set in UI.

Table 10.1: Metrics computed for model evaluation and diagnostics, with formulas and intended interpretation.

Chapter 11

Appendix E: Metrics summarisation (pseudocode)

```
Inputs: metrics_dir, out_dir, eps=0.02, small_n_thresh=5, TICKER2CAT

Load all JSONs from metrics_dir
Keep newest entry per (ticker, model) by mtime

For each record:
  extract test metrics; map ticker -> category (TICKER2CAT)
  drop if rmse_over_naive is missing
  win  = (rmse_over_naive < 1)
  near = (|rmse_over_naive - 1|  eps)

Overall summary (by model):
  compute n, wins, near, winrate, median_ratio, IQR(ratio),
          median_RMSE, median_Accuracy, median_sMAPE
  add Wilson 95% CI for winrate; flag n < small_n_thresh
  sort by median_ratio ascending

Best-model counts:
  per ticker: pick best by (min ratio → min RMSE → max Accuracy)
  count wins per model

By-category:
  repeat overall and best-model counts grouped by (category, model)

Export:
  overall_model_summary.csv, overall_best_counts.csv,
  summary_by_category.csv, best_counts_by_category.csv
Print brief console summary
```

Chapter 12

Appendix F: Environment and Dependencies

The project targets **Python 3.8.10**. Table 12.1 enumerates the core runtime dependencies and their roles. The pinned `requirements.txt` in Listing 12.1 reproduces the environment on Python 3.8.10.

Package	Version	Role
Python	3.8.10	Interpreter
NumPy	1.24.4	Array & numerical routines (Py 3.8–compatible)
Pandas	2.1.4	DataFrames, IO, time series utilities
SciPy	1.10.1	Numerical helpers used by ML libraries
scikit-learn	1.4.2	Feature selection, metrics, classical ML
statsmodels	0.14.1	SARIMAX and time-series statistics
PyTorch	2.2.2	Neural backends (LSTM/Transformer)
joblib	1.3.2	Model/artefact caching and serialization
Altair	5.1.2	Declarative charts for the UI
Streamlit	1.32.2	Presentation layer and session state
Matplotlib	3.7.5	Base plotting dependency (indirect)
patsy	0.5.6	Formula processing used by statsmodels
typing-extensions	4.7.1	Typing shims required by Torch/Streamlit

Table 12.1: Core dependencies used by the application.

Note. Versions are chosen to be compatible with Python 3.8.10. If the GBM backend uses a third-party library, include one of: `xgboost==1.7.6` or `lightgbm==4.3.0`. These are optional and not required for the feature-selection Random Forest.

```
# requirements.txt (Python 3.8.10)
numpy==1.24.4
pandas==2.1.4
scipy==1.10.1
scikit-learn==1.4.2
statsmodels==0.14.1
torch==2.2.2
joblib==1.3.2
altair==5.1.2
streamlit==1.32.2
matplotlib==3.7.5
patsy==0.5.6
typing-extensions==4.7.1
# Optional (only if your GBM backend uses them):
# xgboost==1.7.6
# lightgbm==4.3.0
```

Figure 12.1: Dependencies required

For archival, export the exact runtime set after installation via: `pip freeze > requirements.lock.txt`. Store this alongside metrics and metadata.

Chapter 13

Appendix G: UI Snapshots

The following figures show the eight Streamlit pages used in the system: Home, Market Data Scraper, Regime Detection, Feature Selection, Forecast, Best Model, Global Investment Planner, and Customized Portfolio.

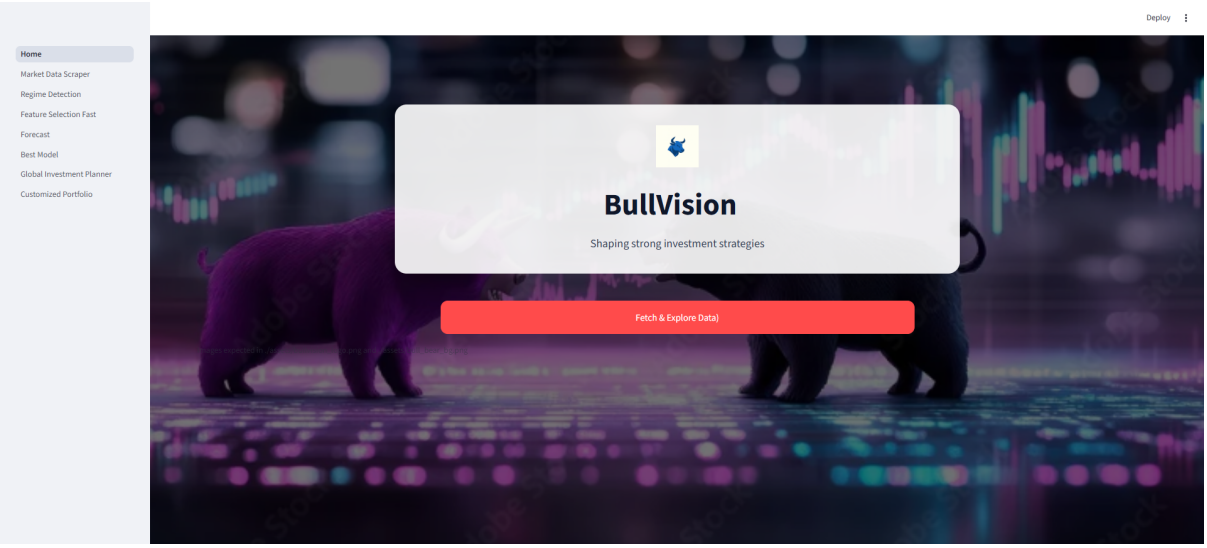

Figure 13.1: Home Page Dashboard

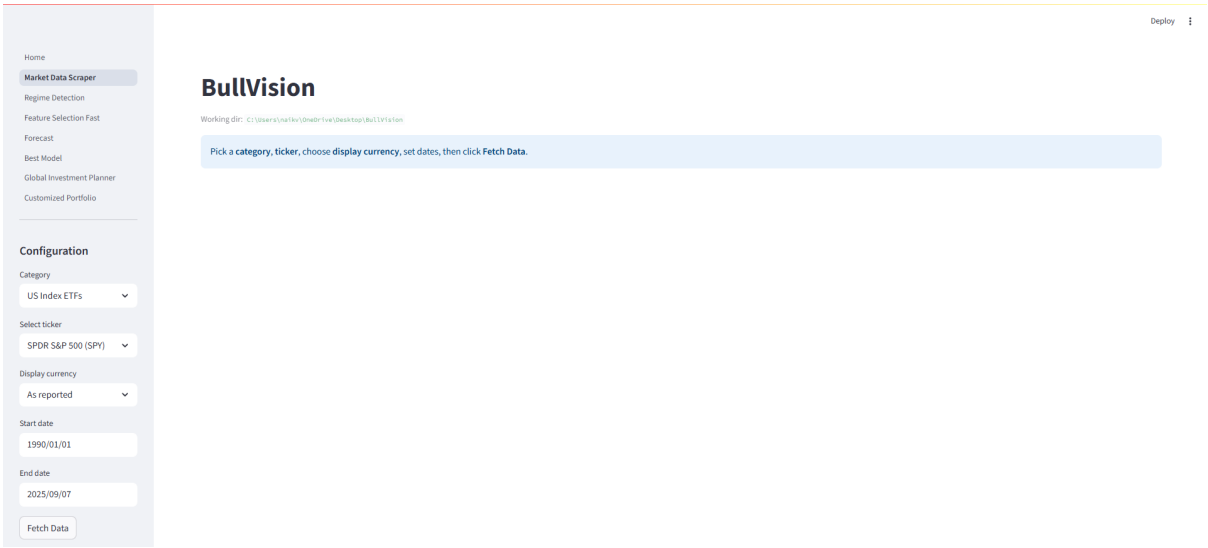

Figure 13.2: Market Data Scraper Dashboard

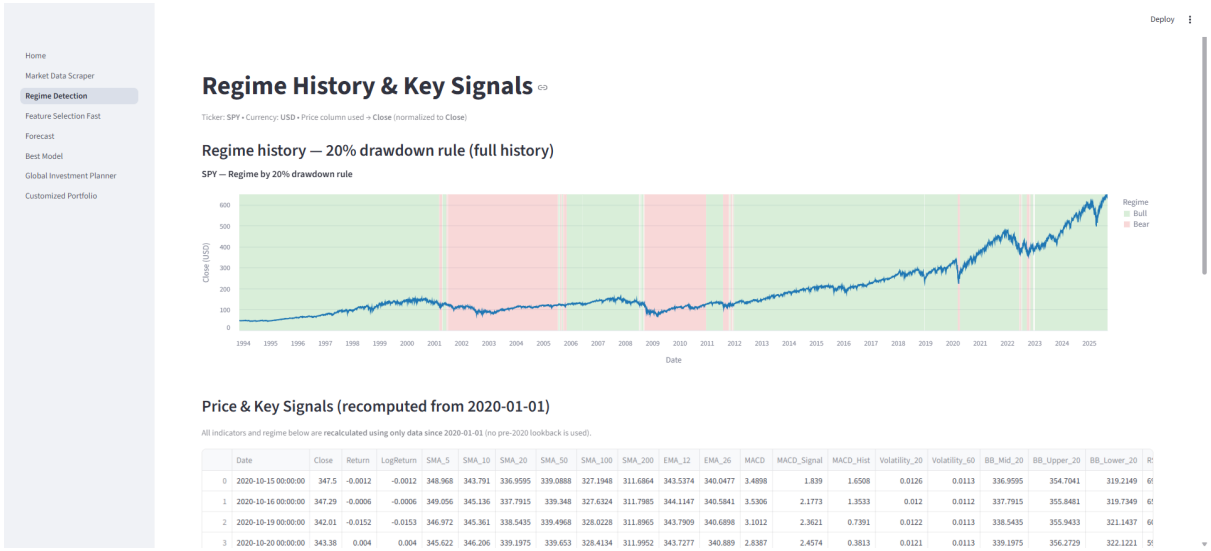

Figure 13.3: Regime Detection Dashboard

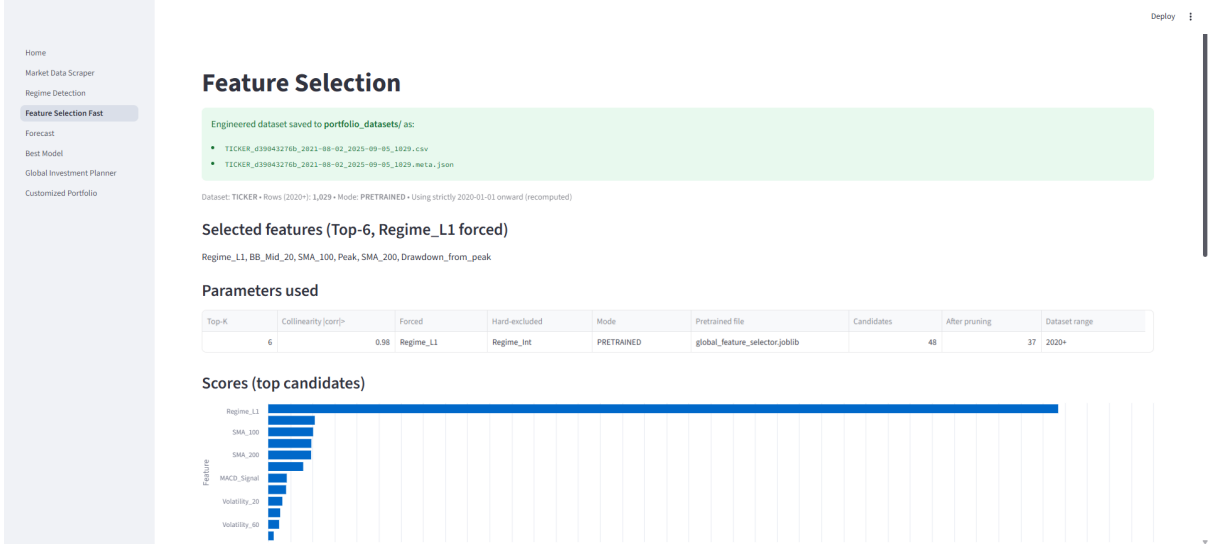

Figure 13.4: Feature Selection Dashboard

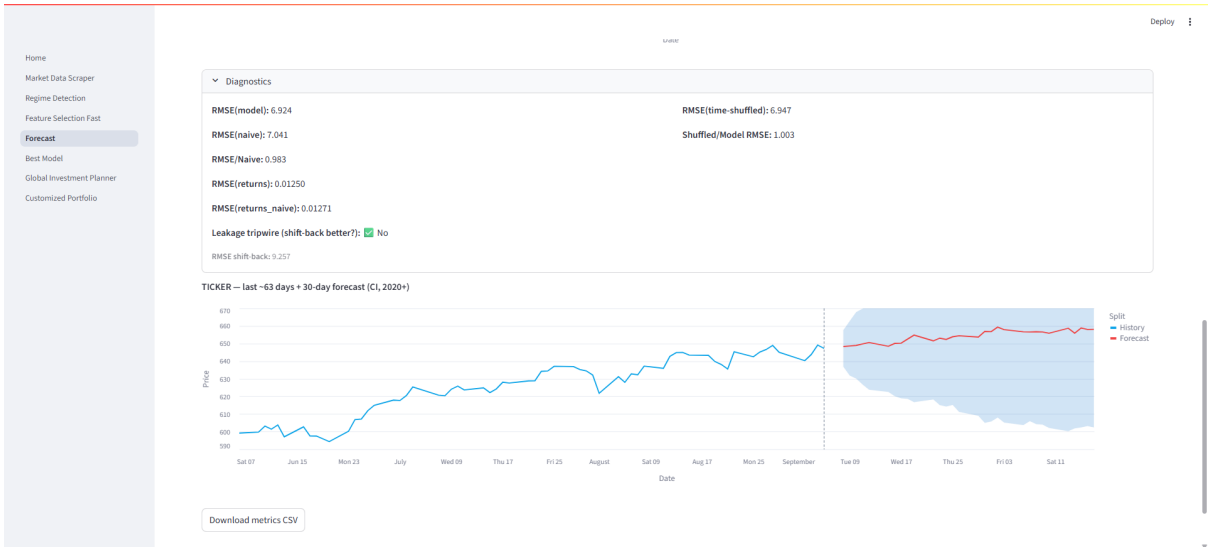

Figure 13.5: Forecasting Dashboard

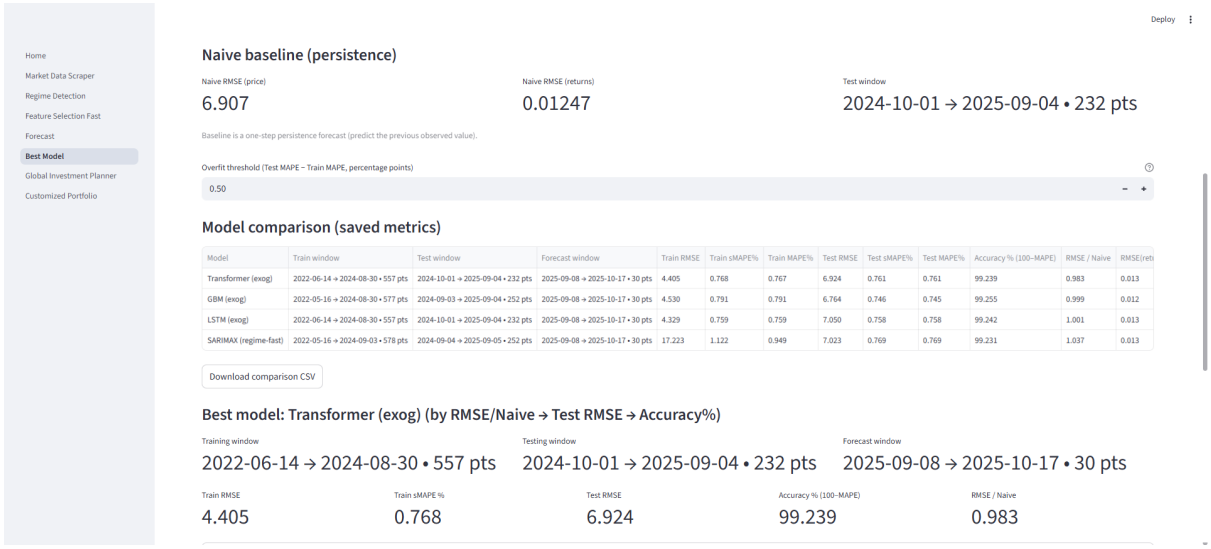

Figure 13.6: Best Model Selection Dashboard

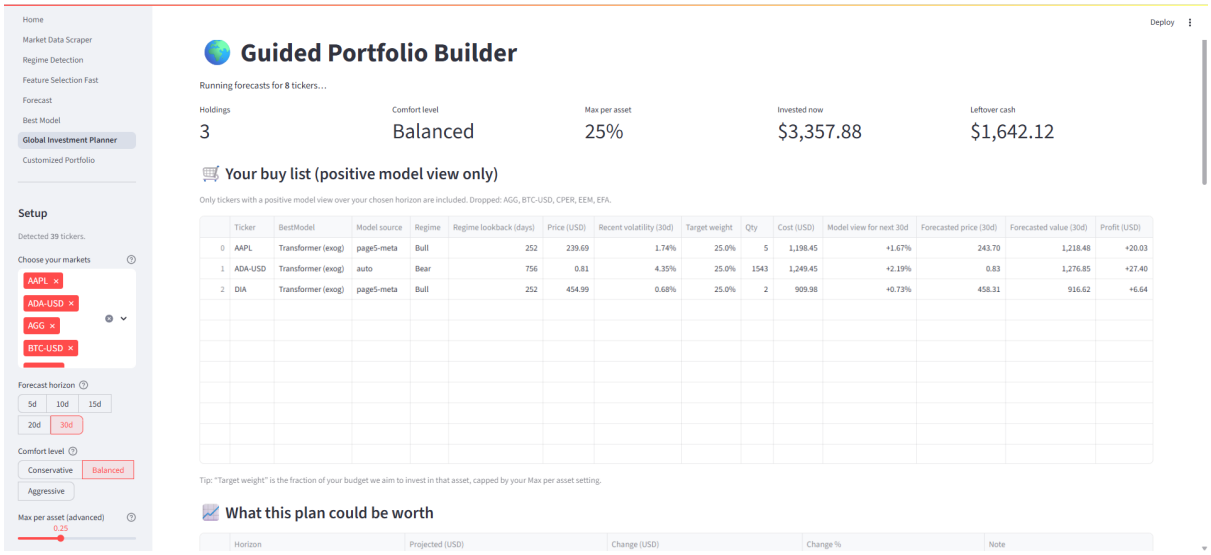

Figure 13.7: Global Investment Planner Dashboard

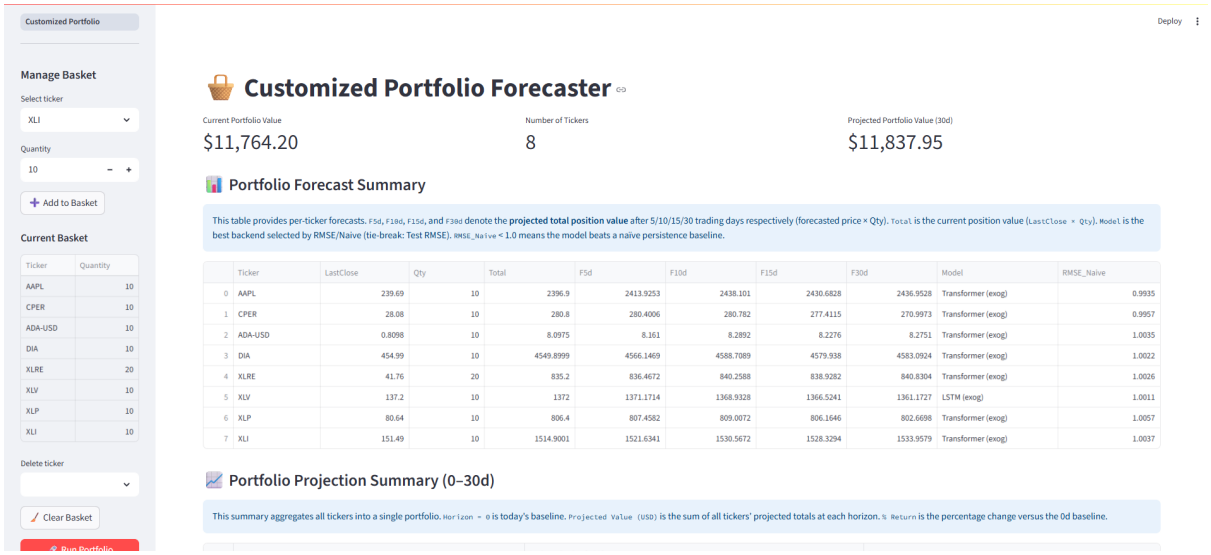

Figure 13.8: Customized Portfolio Dashboard