

# Bibliography

**Université de Lausanne**  
**Hautes Etudes Commerciales**

Study programme: Master in Finance



**Marko Blanus**

Comparative Analysis of Portfolio Optimization Techniques: Classical Markowitz Method vs. Forecasting-Based Input Optimization and Direct Weight Optimization

Master's thesis

Supervisor: **Marc-Aurèle Divernois**

Lausanne, 2025

# Chapter 1

## Introduction

In today’s financial markets, the rapid evolution of asset dynamics and the inherent uncertainty of market behavior have spurred the development of increasingly sophisticated techniques for forecasting volatility and optimizing portfolio allocations. The traditional paradigm in portfolio management—embodied by Markowitz’s mean–variance framework—relies on a two-step process of estimating input parameters (means and covariances) and then optimizing portfolio weights accordingly. However, this “estimate-then-optimize” approach suffers from several limitations, including estimation errors, extreme sensitivity to input parameters, and poor out-of-sample performance. Consequently, recent research has focused on leveraging advanced statistical and machine learning (ML) methods to improve both the forecasting of key market inputs and the direct prediction of portfolio weights. This thesis addresses these challenges by developing a twofold approach. First, it explores an enhanced framework for volatility forecasting and covariance estimation that combines well-established econometric models, such as the Heterogeneous AutoRegressive (HAR) model, with modern machine learning algorithms such as XGBoost and Long Short-Term Memory (LSTM) networks. Second, the thesis investigates an alternative strategy whereby portfolio weights are predicted directly using supervised learning techniques and reinforcement learning approaches (e.g., Proximal Policy Optimization (PPO)). In addition, recent advances in distributionally robust optimization (DRO) are considered as a means to further enhance both methodologies by mitigating the impact of input uncertainty and model misspecification. In the remainder of this chapter, we provide an in-depth review of the existing literature in three main areas: (1) volatility forecasting and input prediction methods; (2) direct portfolio weight prediction approaches; and (3) the application of DRO in portfolio optimization. By critically examining these streams of research, we identify gaps and opportunities that motivate the present work.

# Chapter 2

## Literature Review

### 2.1 Volatility Forecasting and Input Prediction Methods

#### 2.1.1 The HAR Framework and Its Extensions

The seminal work of Corsi (2009) introduced the Heterogeneous AutoRegressive (HAR) model as a parsimonious framework for capturing the long memory and multi-scale nature of volatility. The HAR model decomposes realized volatility into components corresponding to different time horizons—daily, weekly, and monthly—thus accommodating the heterogeneous behavior of market participants. Despite its simplicity, the HAR model has been widely adopted as a benchmark in the literature for forecasting volatility, particularly in contexts where high-frequency data are available. Subsequent studies have sought to enhance the forecasting power of the HAR model by incorporating additional explanatory variables. For instance, the integration of macroeconomic indicators, lagged returns, and lagged squared returns has been shown to improve forecast accuracy by capturing further nonlinear dynamics in volatility evolution. Moreover, the combination of HAR with advanced machine learning techniques, such as XGBoost, has attracted considerable attention. In “Volatility Forecasting with Machine Learning and Intraday Commonality” (Zhang et al., 2023), the authors demonstrate that neural networks and tree-based models outperform classical linear regressions by uncovering complex interactions among predictors. This line of research suggests that ML techniques are particularly well suited to address the nonlinearity inherent in financial time series data.

### 2.1.2 Machine Learning Approaches: XGBoost, LSTM, and Beyond

A growing body of literature supports the integration of ML methods into volatility forecasting. Christensen et al. (2021) in their study “A Machine Learning Approach to Volatility Forecasting” illustrate that off-the-shelf implementations of regularized regression, tree-based algorithms, and neural networks can yield significant improvements in one-day-ahead volatility forecasts over the traditional HAR model. Their empirical findings, based on data from major indices such as the Dow Jones Industrial Average, indicate that ML models can extract incremental information from additional predictors, even when these predictors exhibit strong collinearity or nonlinear relationships. Another stream of research has focused on forecasting volatility in the cryptocurrency domain, where extreme volatility and market-specific idiosyncrasies challenge standard models. In “Crypto Volatility Forecasting: Mounting a HAR, Sentiment, and Machine Learning Horserace” (Brauneis and Sahiner, 2024), the authors extend the HAR framework by incorporating investor sentiment data derived from news and social media, and then compare its performance to several ML models. Their results indicate that although sentiment data do not significantly enhance the HAR model’s performance when used in isolation, the integration of sentiment indicators within ML frameworks such as LightGBM, XGBoost, and LSTM models yields superior predictive accuracy in a nonlinear setting. In addition to these studies, research available on SSRN (abstract id=3766999) further reinforces the view that ML methods automatically decipher the intricate nonlinear relationships between volatility predictors—revealing both consensus on the dominant predictors and disagreement on their ranking. Such findings underscore the importance of robust variable selection and model interpretability techniques (e.g., accumulated local effect plots) to fully exploit the predictive power of ML models.

### 2.1.3 Covariance Matrix Estimation and Robustification Techniques

Beyond volatility forecasting, accurately estimating the covariance matrix of asset returns is critical for portfolio optimization. Traditional approaches often suffer from high estimation risk, leading to unstable portfolios. To mitigate these issues, techniques such as shrinkage estimators and eigenvalue filtering have been proposed. In practice, the covariance matrix estimated via models such as ADCC (Asymmetric Dynamic Conditional Correlation) – combined with univariate volatility forecasts from HAR-XGBoost models – is often “robustified” by shrinking it toward a historical covariance matrix and filtering out spurious eigenvalues. This process reduces the sensitivity of the optimization to estimation errors and should improve out-of-sample performance. These methods

are detailed in various studies (e.g., Corsi’s HAR-XGBoost paper and the ML approach studies) and have become a cornerstone in advanced portfolio management.

## **2.2 Direct Portfolio Weight Prediction Methods**

### **2.2.1 The Rationale for Direct Weight Forecasting**

While much of the existing literature on portfolio optimization focuses on the “estimate-then-optimize” paradigm, recent research has begun to explore methods that predict portfolio weights directly from historical data. This direct approach seeks to bypass the intermediate step of forecasting parameters (such as expected returns or covariances), which are notoriously difficult to estimate accurately. By predicting the optimal asset allocation directly, these methods aim to reduce the propagation of estimation errors and achieve superior out-of-sample performance. Recent studies have demonstrated that both supervised learning techniques and reinforcement learning algorithms can be employed for direct weight prediction. For instance, the work available at ScienceDirect (<https://www.sciencedirect.com/science/article/pii/S0169207024000918>) and SSRN (abstract id=4988124) present methodologies that utilize automated machine learning frameworks to directly predict optimal portfolio weights. These studies frame portfolio selection as a penalized regression or classification problem, where the loss function is designed to capture both return and risk characteristics of the portfolio. Their empirical results suggest that direct weight prediction methods can outperform traditional two-stage models, especially when the underlying data are noisy and exhibit structural changes.

### **2.2.2 Supervised Learning and Reinforcement Learning Approaches**

In the supervised learning paradigm, models such as XGBoost and LSTM are trained to map historical asset returns, market conditions, and macroeconomic indicators directly to portfolio weights. The idea is to learn an optimal weighting strategy from past data that generalizes well to new, unseen market conditions. For example, studies like “Single-Stage Portfolio Optimization with Automated Machine Learning for M6” illustrate how automated model selection and hyperparameter tuning can be integrated into the optimization process. These methods directly target the investment decision, thereby sidestepping the issues related to parameter estimation errors inherent in the two-step approach. On the other hand, unsupervised and reinforcement learning methods offer an alternative avenue for direct weight prediction. Deep Reinforcement Learning (DRL) techniques, such as Proximal Policy Optimization (PPO), have been applied to the portfolio allocation problem by formulating it as a sequential decision-making task. In these

models, an agent learns a policy that maps the current state of the market to an optimal portfolio allocation, while taking into account realistic trading constraints such as transaction costs, slippage, and liquidation rules. A working paper titled “Machine Learning in Portfolio Decisions” explores the application of DRL to directly predict portfolio weights and demonstrates that such methods are capable of capturing complex, nonlinear dependencies that traditional optimization methods often miss. The integration of both supervised and reinforcement learning approaches in the direct weight prediction framework not only improves forecasting accuracy but also provides robustness against market regime shifts and structural changes. By directly optimizing for portfolio performance metrics (such as the Sharpe ratio or downside risk measures), these models are inherently better equipped to handle the uncertainties and non-stationarities that characterize financial markets.

## **2.3 Distributionally Robust Optimization in Portfolio Management**

### **2.3.1 The Need for Robust Optimization Under Uncertainty**

A common challenge in both input prediction and direct weight forecasting is the sensitivity to model misspecification and the inherent uncertainty in financial data. Traditional optimization methods often assume that the underlying probability distributions of asset returns are known or can be estimated accurately from historical data. However, this assumption rarely holds in practice, leading to portfolios that perform well in-sample but poorly out-of-sample. Distributionally Robust Optimization (DRO) offers a powerful framework to address these challenges by seeking solutions that perform well across a range of plausible probability distributions. Instead of relying on a single estimated distribution, DRO models define an ambiguity set—a family of distributions that are consistent with available information (e.g., moments, support, or statistical distances such as the Wasserstein distance). The optimization problem is then reformulated to find the best decision under the worst-case distribution from this ambiguity set. This approach not only enhances the robustness of the solution but also provides theoretical guarantees regarding the performance of the optimized portfolio. The foundational concepts of DRO are thoroughly discussed in the survey “Distributionally Robust Optimization” by Kuhn et al. , which provides a unified treatment of ambiguity sets, duality theory, and numerical solution methods. Recent advancements in this area, including discrepancy-based ambiguity sets and connections with adversarial training in machine learning, have further enriched the DRO literature. Notably, the application of DRO in portfolio optimization has shown promising results in mitigating estimation risk and enhancing the

stability of portfolio performance in volatile markets.

## **2.4 Synthesis and Research Contribution**

### **2.4.1 Integration of Hybrid Forecasting and Direct Weight Prediction**

The literature reviewed above demonstrates that both forecasting the inputs for portfolio optimization and predicting portfolio weights directly have their unique advantages and limitations. Traditional econometric models like HAR remain valuable for their interpretability and parsimony, yet they fall short in capturing nonlinearities inherent in high-frequency data. On the other hand, ML techniques—such as XGBoost, LSTM, and DRL—excel in extracting complex patterns but often suffer from interpretability issues and sensitivity to overfitting when used in isolation. This thesis aims to bridge these two paradigms by proposing a hybrid approach. In the first phase, a robust volatility forecasting framework is developed by combining HAR-based models with machine learning enhancements (e.g., XGBoost) and incorporating techniques for covariance matrix robustification. The empirical findings from studies such as “Volatility Forecasting with Machine Learning and Intraday Commonality” and “Crypto Volatility Forecasting: Mounting a HAR, Sentiment, and Machine Learning Horserace” provide a strong empirical basis for this approach. In the second phase, the focus shifts to the direct prediction of portfolio weights. This involves the development of both supervised learning models—where algorithms are trained to map historical predictors directly to optimal weights—and reinforcement learning models that frame the problem as a sequential decision-making task. Early results from “Single-Stage Portfolio Optimization with Automated Machine Learning for M6” and “Machine Learning in Portfolio Decisions” suggest that these approaches are not only feasible but also capable of outperforming traditional two-stage methods, particularly in noisy and non-stationary environments.

## **2.5 The Role of Distributionally Robust Optimization**

A significant innovation proposed in this thesis is the integration of DRO techniques into both stages of the investment decision process. By constructing ambiguity sets around the estimated volatility, correlations, and even the directly predicted weights, the DRO framework provides an additional layer of protection against model uncertainty. The duality theory and numerical methods developed in the DRO literature (as reviewed in Kuhn et al. ) offer a rigorous foundation for this integration. The resulting framework



is expected to yield portfolios that are not only optimized for historical performance but are also robust to adverse future market scenarios.

## 2.6 Expected Contributions and Implications

The research presented in this thesis is expected to make several contributions to the fields of financial econometrics and portfolio management: **Enhanced Volatility Forecasting:** By integrating HAR-based methods with advanced ML algorithms and robust covariance estimation techniques, the thesis aims to provide a forecasting model that more accurately captures the dynamic behavior of volatility, particularly in volatile asset classes such as cryptocurrencies. **Direct Weight Prediction:** The exploration of both supervised and reinforcement learning approaches for direct weight prediction seeks to challenge the conventional “estimate-then-optimize” paradigm. The research will provide empirical evidence on the performance improvements offered by these direct methods. **Robust Optimization via DRO:** Incorporating DRO techniques addresses a critical shortcoming of existing methods—their sensitivity to input estimation errors and distributional shifts. The robust optimization framework developed herein is anticipated to lead to more stable portfolio performance, especially under market stress. **Practical Implementation:** The proposed methodologies are not only theoretically motivated but also designed with practical constraints in mind, such as transaction costs, market frictions, and realistic trading rules. This practical orientation is essential for the adoption of these methods by practitioners in asset management and quantitative finance.

## 2.7 Conclusion

In summary, this literature review highlights the evolution of volatility forecasting and portfolio optimization from traditional econometric models to modern machine learning and robust optimization frameworks. The reviewed studies underscore the limitations of conventional methods—particularly the challenges associated with parameter estimation and sensitivity to market uncertainty—and point toward promising alternatives such as direct weight prediction and DRO-enhanced optimization. The hybrid approach developed in this thesis, which combines enhanced input forecasting with direct portfolio weight prediction and is further fortified by DRO techniques, represents a significant step forward in the quest for robust, efficient, and practical portfolio management strategies. By addressing both the estimation and optimization stages of the portfolio construction process, the proposed methodology is expected to yield superior risk-adjusted returns and greater resilience in the face of market uncertainty. This introductory chapter sets the stage for the empirical and methodological contributions that follow. The remainder of the thesis will detail the model specifications, data sources, and experimental designs used

to test the proposed framework, as well as a comprehensive evaluation of its performance relative to traditional approaches.

# Chapter 3

## Methodology

### 3.1 Introduction and Objectives

This chapter introduces a comprehensive methodological framework designed to systematically address the cryptocurrency portfolio optimization problem. We propose and rigorously examine two distinct yet complementary approaches, each motivated by unique strengths in capturing the complexities inherent in financial time series and portfolio allocation.

**Approach 1** adopts a two-stage prediction-optimization paradigm, explicitly modeling the covariance structure of asset returns before portfolio construction. Within Approach 1, we introduce two differentiated methodological branches. The first branch (*Method A*) employs a combination of econometric and gradient-boosted decision-tree models, namely Extreme Gradient Boosting (XGBoost) paired with the Asymmetric Dynamic Conditional Correlation (ADCC) model. This combination leverages classical financial econometrics and powerful machine learning algorithms, explicitly capturing volatility clustering, asymmetric volatility effects, and dynamic correlation structures extensively documented in financial time series literature. The second branch (*Method B*) advances this concept further, employing state-of-the-art supervised deep learning models—Long Short-Term Memory (LSTM) networks, Transformers, and Autoformers—to predict directly the Cholesky decomposition of daily realized covariance matrices. By targeting Cholesky decompositions, we guarantee the positive definiteness of predicted covariance matrices, thus ensuring the numerical stability and theoretical validity essential for subsequent portfolio optimization. These sophisticated sequence models offer a powerful alternative to classical econometric approaches, effectively capturing intricate temporal dependencies, cross-sectional interactions, and nonlinear relationships present in high-dimensional cryptocurrency markets.

In contrast, **Approach 2** takes a radically different perspective, directly predicting optimal portfolio weights through a single-stage, end-to-end approach. Within Approach

2, we employ supervised deep learning methods similar to those presented in Approach 1—LSTM, Transformer, and Autoformer models—but here, instead of forecasting covariance matrices, these architectures directly predict asset allocations that optimize future portfolio performance. Furthermore, Approach 2 introduces deep reinforcement learning methods (specifically Proximal Policy Optimization, PPO) to learn optimal allocation policies through interaction with the simulated market environment. These reinforcement learning models directly optimize cumulative reward functions, dynamically adapting to evolving market conditions, thereby bypassing explicit covariance or volatility forecasting altogether.

The primary objective of explicitly delineating these methodological approaches and their subcomponents is to thoroughly assess their comparative strengths and weaknesses in terms of predictive accuracy, robustness to market conditions, computational efficiency, and ultimately, out-of-sample portfolio performance. By combining traditional econometric models with cutting-edge machine learning methodologies, we aim not only to quantify the incremental benefits of advanced prediction techniques but also to explore how these methods complement each other, potentially yielding superior risk-adjusted returns, improved stability, and greater interpretability of portfolio decisions.

The selection of cryptocurrencies as the exclusive focus of this study is driven by both practical and theoretical considerations. Practically, cryptocurrencies constitute an asset class characterized by substantial liquidity, continuous trading availability, and high-quality granular data across multiple frequencies. The chosen portfolio includes prominent cryptocurrencies such as Bitcoin (BTC), Ethereum (ETH), Cardano (ADA), and Binance Coin (BNB), selected primarily for their significant market capitalization, widespread investor recognition, and central roles within the cryptocurrency ecosystem. Furthermore, we include assets such as Litecoin (LTC), EOS, NEO, Tron (TRX), Stellar (XLM), and Ripple (XRP), whose selection was primarily motivated by comprehensive historical data availability. Although some of these assets have experienced a decline in market prominence, their extended historical datasets offer robust opportunities for model training, validation, and out-of-sample evaluation across diverse market cycles.

Theoretically, focusing exclusively on cryptocurrencies provides several distinct methodological advantages. Cryptocurrency markets are notably less efficient and more sentiment-driven than traditional equity or fixed-income markets, characterized by extreme volatility, speculative trading behavior, and significant susceptibility to investor psychology and market sentiment. Such market conditions inherently present enhanced opportunities for arbitrage, alpha generation, and exploitation of market inefficiencies—precisely the types of opportunities sophisticated machine learning models are designed to identify and leverage. Moreover, limiting the analysis strictly to cryptocurrencies substantially simplifies interpretability, allowing clear attribution of portfolio performance variations specifically to methodological differences and modeling choices rather than confounding

factors introduced by heterogeneous asset classes.

By comprehensively exploring these contrasting yet complementary methodological pathways—traditional econometric modeling versus advanced supervised and reinforcement learning techniques—we not only aim to evaluate their relative performance but also to establish a clear understanding of their applicability, strengths, and limitations within the emerging and rapidly evolving cryptocurrency investment landscape.

In the subsequent sections of this methodology chapter, we will rigorously detail each method, presenting clear theoretical justifications, detailed mathematical foundations, comprehensive workflow descriptions, and illustrative diagrams that facilitate deeper insight into the models’ inner workings. Additionally, relevant and up-to-date academic references are systematically included to position our contributions within the broader existing literature on statistical finance, econometrics, and machine learning applications to portfolio optimization.

## 3.2 Data Sources and Preprocessing

The empirical analyses presented in this thesis utilize comprehensive financial time series data sourced from a strategically curated portfolio of cryptocurrencies. Specifically, our study focuses on ten prominent digital assets: Bitcoin (BTC), Ethereum (ETH), Cardano (ADA), Binance Coin (BNB), EOS, Litecoin (LTC), NEO, Tron (TRX), Stellar (XLM), and Ripple (XRP). These assets were deliberately selected based on two primary criteria. First, cryptocurrencies such as Bitcoin, Ethereum, Cardano, and Binance Coin were chosen due to their dominant market positions, robust liquidity profiles, substantial market capitalizations, and widespread investor recognition. These assets are commonly perceived as representative benchmarks within cryptocurrency markets, facilitating broad generalizability of results. Conversely, the inclusion of Litecoin, EOS, NEO, Tron, Stellar, and Ripple was primarily guided by the extensive availability of detailed historical data, even though some of these assets have experienced declines in their market prominence in recent years. Their rich historical data series remain highly valuable, enabling rigorous backtesting, robust model training, and comprehensive evaluation across diverse market conditions.

Data was primarily retrieved via the Binance API, a leading cryptocurrency exchange known for its depth of market coverage, liquidity, and data accessibility. Prices were originally obtained at one-minute intervals for each cryptocurrency, spanning from May 18, 2018, through February 2025. This highly granular intraday data served multiple purposes across our methodological framework. First, daily log returns were computed for each asset by resampling the original minute-level data to daily frequency. Formally,

daily log returns were calculated as:

$$r_{i,t} = \ln \left( \frac{P_{i,t}}{P_{i,t-1}} \right), \quad (3.1)$$

where  $P_{i,t}$  denotes the closing price of cryptocurrency  $i$  on day  $t$ . These daily returns constituted primary input features across both approaches—essential for supervised learning models and econometric forecasting.

For volatility modeling within Approach 1 (specifically Method A involving the XGBoost and ADCC models), the high-frequency intraday data served an additional critical role. Daily realized variance ( $RV_{i,t}$ ) for each cryptocurrency was computed as the summation of squared intraday log returns over each trading day:

$$RV_{i,t} = \sum_{j=1}^M (r_{i,t,j})^2, \quad (3.2)$$

where  $r_{i,t,j}$  is the intraday log return for cryptocurrency  $i$  during minute  $j$  of day  $t$ , and  $M$  is the total number of intraday intervals per day. In instances where minute-level data was partially missing or incomplete, volatility was approximated using high-low range proxies, specifically the Parkinson estimator:

$$RV_{i,t}^{\text{Parkinson}} = \frac{(\ln H_{i,t} - \ln L_{i,t})^2}{4 \ln(2)}, \quad (3.3)$$

with  $H_{i,t}$  and  $L_{i,t}$  being the highest and lowest observed prices of cryptocurrency  $i$  on day  $t$ . To capture volatility clustering and long-memory properties in volatilities—crucial for the HAR model features used within XGBoost—these daily realized volatilities were further aggregated into weekly and monthly realized variance measures, enhancing the predictive capacity of volatility forecasts.

The supervised learning methodology (Approach 1 Method B) necessitated a more elaborate and precise treatment of realized covariance matrices. Daily realized covariance matrices ( $\Sigma_t^{\text{realized}}$ ) were systematically computed from minute-frequency returns across all cryptocurrencies as follows:

$$\Sigma_t^{\text{realized}} = \sum_{j=1}^M r_{t,j} r_{t,j}^\top, \quad (3.4)$$

where  $r_{t,j}$  represents the vector of intraday returns for all assets at interval  $j$ . To ensure numerical stability and mathematical validity in subsequent supervised prediction tasks, these covariance matrices were transformed using the Cholesky decomposition, which

guarantees positive definiteness:

$$\Sigma_t^{\text{realized}} = L_t L_t^\top, \quad (3.5)$$

with  $L_t$  being a lower-triangular matrix targeted directly by LSTM, Transformer, and Autoformer models during training.

Unlike traditional financial markets, cryptocurrencies operate continuously without conventional market closures such as weekends, holidays, or overnight gaps. This characteristic significantly simplifies certain aspects of data preprocessing, particularly eliminating the necessity for market-specific adjustments such as dividends, stock splits, or trading suspensions. However, cryptocurrency markets occasionally exhibit isolated instances of missing observations due to transient technical disruptions or API limitations. These sparse gaps were effectively managed through linear interpolation or forward-filling techniques, ensuring data integrity while minimizing potential biases in subsequent analyses.

Additionally, we integrated macroeconomic and broader market indicators sourced from Yahoo Finance’s API as supplemental exogenous predictors, recognizing the documented impact of macroeconomic factors and global sentiment on cryptocurrency prices. Because these macro indicators are typically reported only during weekdays, forward-filling methods were employed to align and populate weekend and holiday observations. This alignment assumes macroeconomic conditions remain unchanged during non-reporting periods, an assumption justified by the typically low volatility of macro indicators at short-term horizons.

A deliberate methodological choice was made concerning feature normalization: we opted not to perform any feature scaling or normalization across the input datasets, preserving the interpretability and original scale of the economic variables. This decision was motivated by the desire to maintain transparency in economic interpretation and to allow the machine learning models—particularly the advanced sequence-based architectures—to autonomously learn optimal internal transformations from raw features, potentially enhancing their predictive power and flexibility.

Regarding feature engineering, several widely recognized technical indicators and derived metrics were systematically calculated from daily price series to enrich predictive modeling. These included rolling moving averages, exponential moving averages (EMAs), volatility proxies, the Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), and other sentiment-driven technical metrics extensively validated in prior cryptocurrency and sentiment-focused financial market studies. These indicators are instrumental in capturing latent investor behaviors and market momentum signals inherent in highly speculative cryptocurrency markets.

Prior to modeling, the full dataset was meticulously divided into chronologically con-

tiguous subsets to ensure robust out-of-sample evaluation and hyperparameter tuning devoid of look-ahead bias. Explicitly, the dataset was partitioned as follows:

- **Training set:** May 18, 2018 – December 31, 2022. This extensive training window maximizes the richness of captured market regimes, ensuring comprehensive learning across bull markets, bear markets, volatility spikes, and regulatory developments.
- **Validation set:** January 1, 2023 – December 31, 2023. Utilized explicitly for hyperparameter tuning, early stopping in neural network training, and model selection, enabling identification of optimal model architectures without contaminating out-of-sample predictive evaluation.
- **Test set:** January 1, 2024 – February 28, 2025. Exclusively reserved for rigorous, unbiased evaluation of model performance, predictive robustness, and portfolio strategies developed within the two methodological frameworks. The test set explicitly simulates real-world forecasting scenarios, critically assessing generalizability beyond observed training data.

This carefully structured preprocessing framework not only ensures methodological rigor and data integrity but also maximizes the potential for predictive accuracy, robustness, interpretability, and ultimately, practical applicability of derived cryptocurrency portfolio strategies across diverse market conditions and future time periods.

### 3.3 Approach 1: Forecasting Covariance for Portfolio Optimization

Approach 1 encompasses two distinct methodologies for forecasting asset covariance matrices, which form the basis of subsequent portfolio optimization:

- **Method A: Two-Step Forecasting using XGBoost and ADCC.** First, individual asset volatilities are forecasted using XGBoost with HAR-based predictors. Second, dynamic correlations between assets are modeled via ADCC. These volatility and correlation forecasts are combined into a robustified covariance matrix.
- **Method B: Direct Covariance Forecasting via Supervised Deep Learning.** Supervised learning models directly forecast the Cholesky decomposition of realized covariance matrices. Predictions are guaranteed to yield positive-definite covariance matrices.

Both methodologies provide alternative perspectives and modeling advantages, allowing an empirical comparison of traditional econometric versus advanced deep learning approaches.



### 3.3.1 Method A: Two-Step Forecasting using XGBoost and ADCC

Method A of Approach 1 is a structured, two-step procedure designed to robustly forecast the covariance structure of asset returns, crucial for precise portfolio optimization. Initially, we forecast asset-specific volatilities employing an extended Heterogeneous Autoregressive (HAR) model (?), augmented by Extreme Gradient Boosting (XGBoost) (?) to capture nonlinear dynamics. Subsequently, dynamic correlations among asset returns are modeled using the Asymmetric Dynamic Conditional Correlation (ADCC) model (?), explicitly accounting for asymmetric market conditions. The resulting covariance matrix is robustified through shrinkage and eigenvalue filtering before feeding into various portfolio optimization frameworks. Figure 3.1 illustrates the comprehensive workflow of Method A.

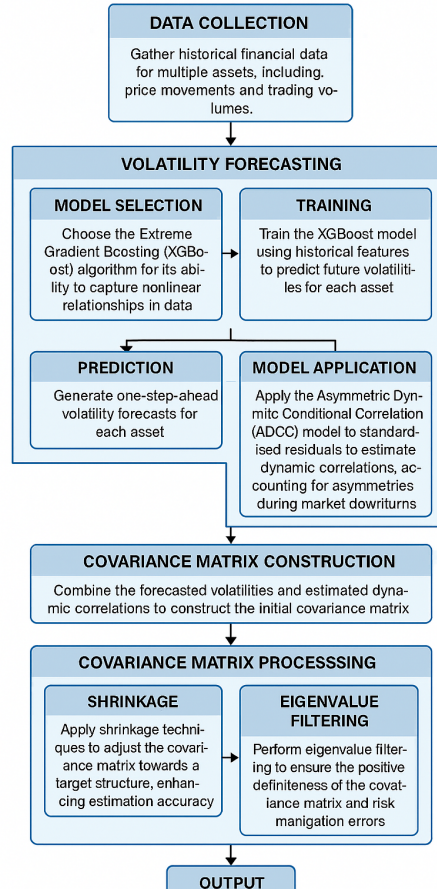


Figure 3.1: Flowchart depicting the process of forecasting a robust covariance matrix using historical data, XGBoost for volatility prediction, and ADCC for dynamic correlation estimation. The covariance matrix is refined with shrinkage and eigenvalue filtering, producing reliable inputs for portfolio optimization and risk management in financial markets.

## Volatility Forecasting using HAR and XGBoost

Accurate volatility predictions underpin effective risk management and portfolio allocation. The classical linear HAR model by ? captures long memory and multi-scale volatility features using lagged realized volatility at different frequencies (daily, weekly, monthly):

$$RV_{t+1} = \beta_0 + \beta_D RV_t^{(D)} + \beta_W RV_t^{(W)} + \beta_M RV_t^{(M)} + \varepsilon_{t+1}, \quad (3.6)$$

where lagged realized volatility terms are defined as:

$$RV_t^{(D)} = RV_t, \quad RV_t^{(W)} = \frac{1}{7} \sum_{j=0}^4 RV_{t-j}, \quad RV_t^{(M)} = \frac{1}{28} \sum_{j=0}^{21} RV_{t-j}.$$

Although effective, linear HAR lacks the flexibility to represent nonlinear patterns or interaction effects among predictors like volatility-of-volatility and sentiment indicators (?). Therefore, we enhance HAR by employing the powerful machine learning technique XGBoost, which builds an ensemble of regression trees iteratively via gradient boosting. XGBoost optimizes the following regularized objective (?):

$$\mathcal{L}(\phi) = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k), \quad (3.7)$$

where  $l(y_i, \hat{y}_i)$  is the predictive loss function, typically Mean Squared Error (MSE), and  $\Omega(f_k)$  regularizes tree complexity to prevent overfitting, given explicitly by:

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \|w\|^2,$$

with  $T$  representing the number of leaves and  $w$  leaf weights.

Each iteration constructs a tree by minimizing a second-order approximation of the loss function (?):

$$\tilde{\mathcal{L}}^{(t)} = \sum_i [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t), \quad (3.8)$$

where gradients  $g_i$  and Hessians  $h_i$  guide tree structure formation.

The combined HAR-XGBoost model significantly improves forecast accuracy by capturing nonlinearities and interactions between predictors, outperforming classical HAR models and other linear methods (?). We tune hyperparameters (e.g., tree depth, learning rate, regularization strength) via cross-validation and employ early stopping to ensure model generalization.

**Figures to Include:** - \*\*Figure 2 from ? (page 3):\*\* Illustrating tree splitting and structure score computation, highlighting XGBoost's decision-making mechanics. - \*\*Fig-

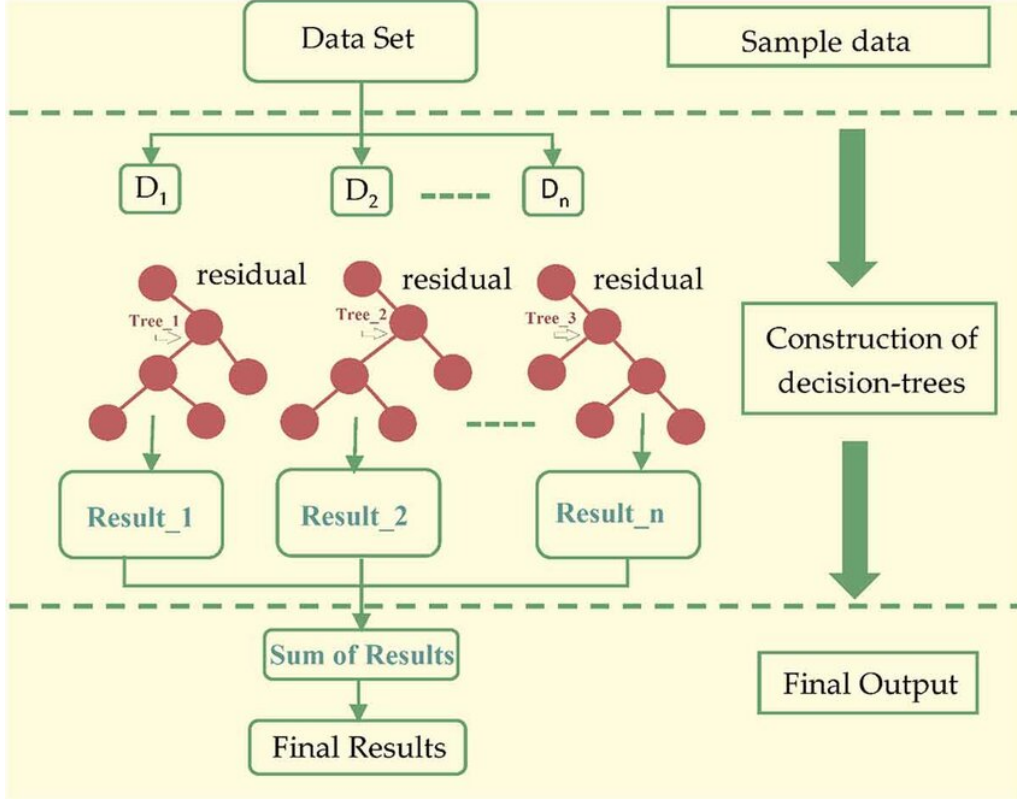


Figure 3.2: Schematic architecture of the volatility forecasting model in Approach 1. HAR features and additional predictors are input into the XGBoost regression algorithm, forecasting one-step-ahead volatility for each asset. These volatility forecasts are combined with ADCC-generated dynamic correlation matrices to produce robust covariance predictions, which guide subsequent portfolio optimization decisions.

ure 1 from ? (page 497):\*\* Performance comparison of HAR and HAR-XGBoost volatility forecasts, demonstrating empirical superiority of the nonlinear model.

### Dynamic Correlation Estimation using ADCC

Given individual volatility forecasts, accurately modeling dynamic asset correlations is essential. We implement the ADCC model introduced by ?, extending Engle's DCC model (?) by incorporating correlation asymmetry during market downturns.

We first define univariate GARCH(1,1) processes for returns  $r_{i,t}$  of each asset  $i$ :

$$r_{i,t} = \mu_{i,t} + \epsilon_{i,t}, \quad \epsilon_{i,t} = \sigma_{i,t} z_{i,t},$$

$$\sigma_{i,t}^2 = \omega_i + \alpha_i \epsilon_{i,t-1}^2 + \beta_i \sigma_{i,t-1}^2,$$

where  $z_{i,t}$  are standardized residuals, i.i.d. with zero mean and unit variance.

The DCC model specifies dynamic correlations among standardized residuals as follows:

$$Q_t = (1 - a - b)\bar{Q} + a(z_{t-1} z_{t-1}^\top) + bQ_{t-1}, \quad (3.9)$$

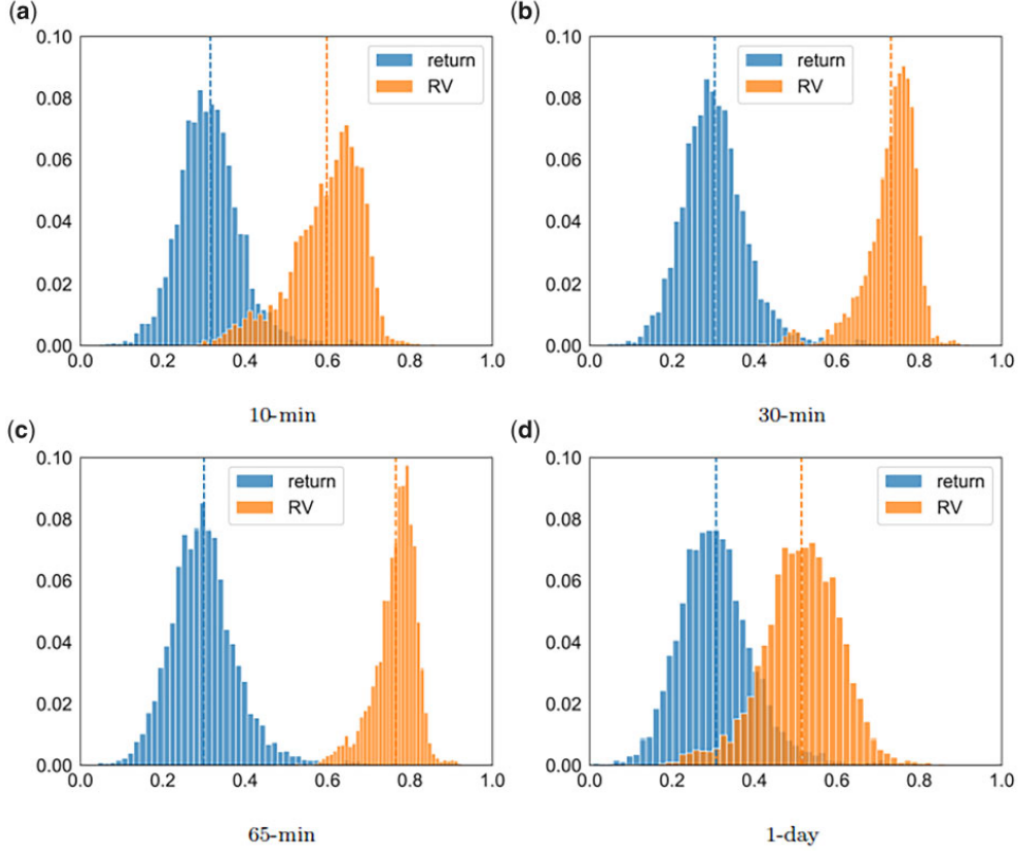


Figure 3.3: Performance comparison of HAR and HAR-XGBoost volatility forecasts, demonstrating empirical superiority of the nonlinear model.

normalizing  $Q_t$  to produce the correlation matrix  $R_t$ :

$$R_t = \text{diag}(Q_t)^{-1/2} Q_t \text{diag}(Q_t)^{-1/2}.$$

The ADCC model augments DCC by adding an asymmetric correlation adjustment term capturing intensified correlations after negative shocks:

$$Q_t = (1 - a - b - g)\bar{Q} + a(z_{t-1}z_{t-1}^\top) + bQ_{t-1} + g(n_{t-1}n_{t-1}^\top), \quad (3.10)$$

where  $n_{t-1}$  denotes indicators for negative returns, and the asymmetry parameter  $g$  captures heightened correlations during downturns. This feature realistically models market behavior, such as contagion effects observed in financial crises. Parameters  $(a, b, g)$  are estimated through a two-step maximum likelihood procedure, enhancing numerical stability and computational efficiency.

**Figure to Include:** - \*\*Figure 1 from ? (page 537):\*\* Depicting correlation dynamics during turbulent periods, visually justifying the importance of the asymmetric ADCC

extension.

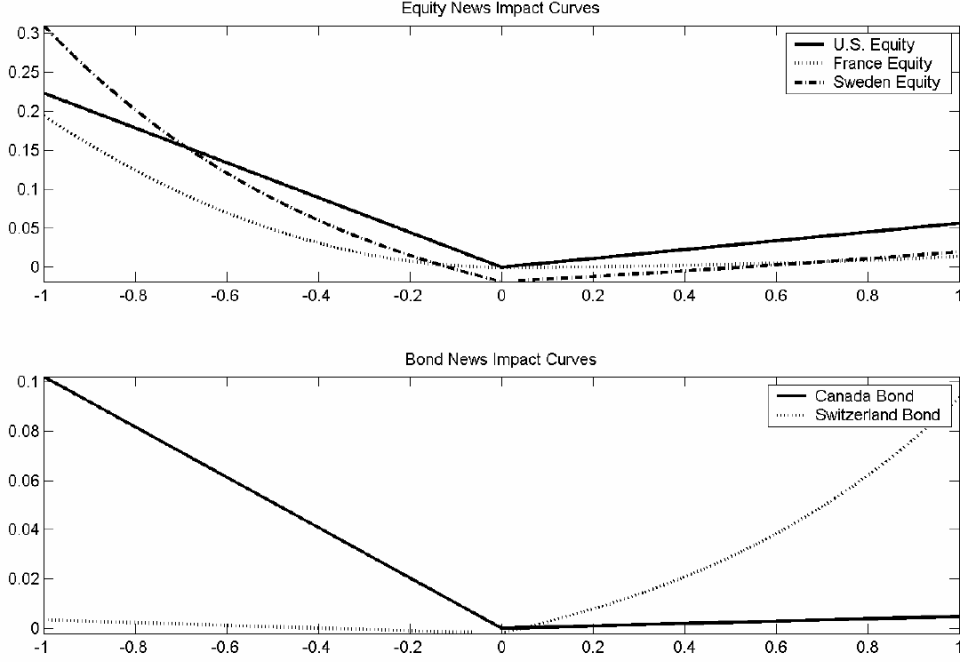


Figure 3.4: Depicting correlation dynamics during turbulent periods, visually justifying the importance of the asymmetric ADCC extension.

**Final Covariance Matrix:** Finally, the predicted covariance matrix  $\hat{\Sigma}_{t+1}$  is constructed from XGBoost volatility forecasts and ADCC-generated correlations:

$$\hat{\Sigma}_{t+1} = D_{t+1} R_{t+1} D_{t+1},$$

where  $D_{t+1}$  is a diagonal matrix of forecasted volatilities from XGBoost. This covariance matrix undergoes further regularization via shrinkage and eigenvalue filtering, ensuring robust and numerically stable inputs for portfolio optimization.

### 3.3.2 Method A: Two-Step Forecasting using XGBoost and ADCC.

Method A of approach 1 consists of a sequence of steps to produce an optimized portfolio. First, we forecast each asset's volatility (or variance) using a machine learning approach that extends the classic HAR (Heterogeneous Autoregressive) model. Next, we estimate time-varying correlations between asset returns using a dynamic conditional correlation model. These volatility and correlation forecasts are then combined into a predicted covariance matrix for the asset returns, which we robustify using shrinkage and eigenvalue filtering techniques. Finally, we input the covariance matrix into various portfolio optimization formulations (such as Global Minimum Variance, Maximum Diversification,

and Equal Risk Contribution portfolios) to determine the asset weights. Figure 3.5 illustrates the workflow of this approach, including the volatility forecasting module and the subsequent portfolio construction.

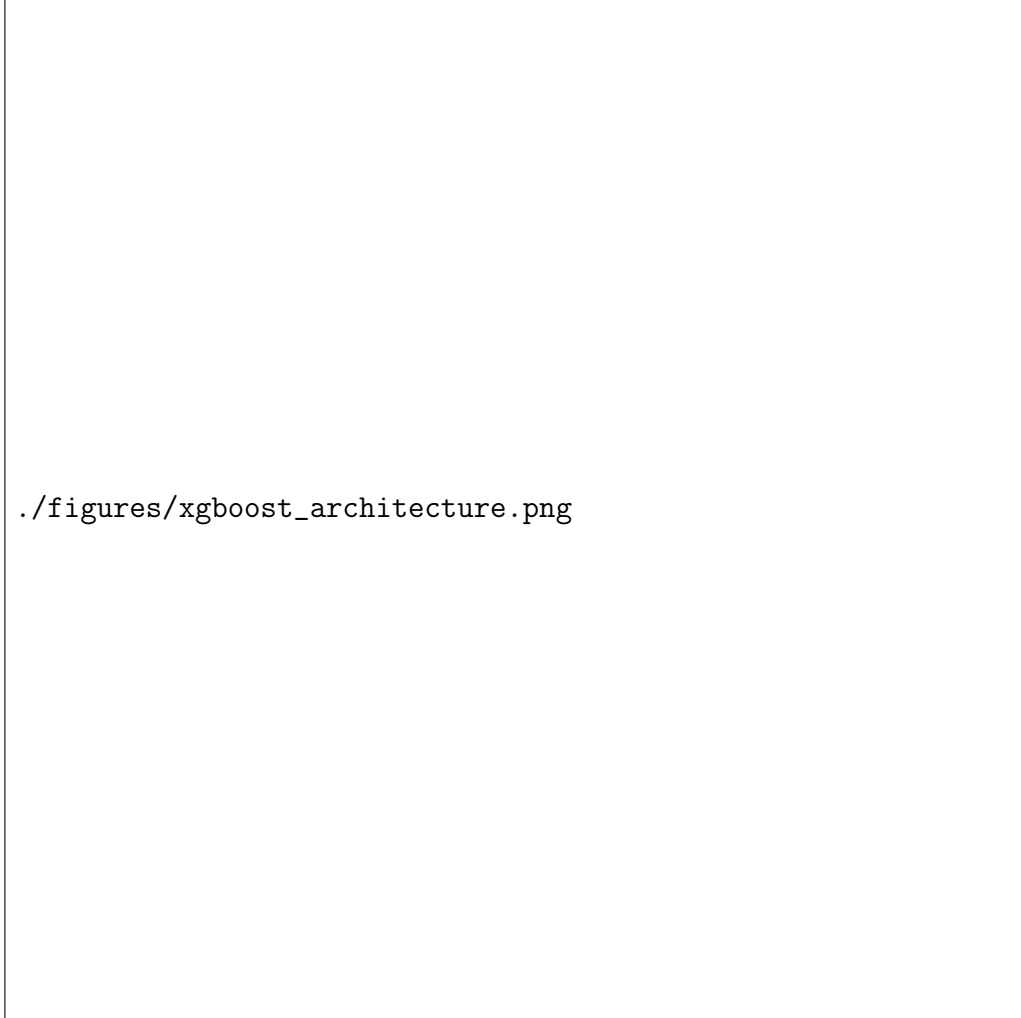


Figure 3.5: Schematic architecture of the volatility forecasting model in Approach 1. The model uses HAR features (daily, weekly, monthly realized volatilities and other predictors) as inputs to an XGBoost regression algorithm, producing a one-step-ahead volatility forecast  $\hat{\sigma}_{i,t+1}$  for each asset  $i$ . These forecasts, combined with a dynamic correlation estimator (ADCC), yield a covariance matrix  $\hat{\Sigma}_{t+1}$ , which feeds into the portfolio optimizer.

### XGBoost with HAR Features for Volatility Forecasting

Accurate volatility forecasts are crucial for risk-aware portfolio optimization. We employ an extended HAR model using Extreme Gradient Boosting (XGBoost) to forecast daily volatility for each asset. The HAR model of ? posits that realized volatility can be predicted by its own lagged values aggregated over different horizons (daily, weekly, monthly), capturing long-memory and multi-scale dynamics in volatility. Formally, a

HAR model for realized variance (RV) can be written as:

$$RV_{t+1} = \beta_0 + \beta_D RV_t^{(D)} + \beta_W RV_t^{(W)} + \beta_M RV_t^{(M)} + \varepsilon_{t+1}, \quad (3.11)$$

where  $RV_t^{(D)} = RV_t$  (daily volatility),  $RV_t^{(W)} = \frac{1}{5} \sum_{j=0}^4 RV_{t-j}$  (last week average), and  $RV_t^{(M)} = \frac{1}{22} \sum_{j=0}^{21} RV_{t-j}$  (last month average) in a typical setting for daily data. This linear model has proven effective for various asset classes (e.g., ?). However, linear HAR may not capture nonlinear patterns or additional predictors (such as macro variables or sentiment).

We therefore use XGBoost, a tree-boosting machine learning algorithm (?), to extend HAR in a nonlinear fashion. XGBoost can handle complex interactions and nonlinearities by fitting an ensemble of decision trees in a stage-wise manner, optimizing a regularized objective. We feed the HAR features (recent daily, weekly, monthly volatilities) into XGBoost, along with other potential features like lagged returns, volatility of volatility, or exogenous variables (e.g., implied volatility indices or sentiment indicators). The model is trained to predict the next-day realized volatility. XGBoost’s gradient boosting framework helps prevent overfitting through shrinkage and tree depth control, and it naturally handles feature importance selection. Empirical studies have shown that machine learning methods like XGBoost can outperform traditional HAR and even deep neural networks in volatility forecasting, especially for short horizons (?). This is attributed to XGBoost’s ability to capture the nonlinear dependence in volatility dynamics and interactions between features (for example, regime-specific responses of volatility to past shocks) that a linear model might miss.

In our setup, for each asset  $i$ , we train an XGBoost regressor  $f_i$  such that  $\hat{\sigma}_{i,t+1}^2 = f_i(RV_{i,t}^{(D)}, RV_{i,t}^{(W)}, RV_{i,t}^{(M)}, \dots)$ , where  $\hat{\sigma}_{i,t+1}$  is the forecasted volatility (standard deviation) for asset  $i$  on day  $t + 1$ . We optimize the hyperparameters of XGBoost (such as the number of trees, maximum tree depth, learning rate, and regularization parameters) using cross-validation on the training set. The loss function is chosen as mean squared error between predicted and realized variance, and we utilize early stopping based on validation error to prevent overfitting. The resulting model provides a daily volatility forecast for each asset, which will be used to construct the covariance matrix.

## ADCC for Dynamic Correlation Estimation

While XGBoost forecasts individual volatilities, we need a model for time-varying correlations between asset returns to fully characterize the joint distribution of returns. We adopt the Asymmetric Dynamic Conditional Correlation (ADCC) model of ?, which extends Engle’s Dynamic Conditional Correlation (DCC) model (?) to allow for asymmetry in correlations during market downturns. The ADCC model is a multivariate GARCH approach: each asset’s return  $r_{i,t}$  is assumed to follow a univariate GARCH process for

its variance, and the correlations between the standardized residuals are modeled as dynamically evolving over time.

In a DCC model, we first specify for each asset  $i$  a univariate GARCH(1,1) process:

$$r_{i,t} = \mu_{i,t} + \epsilon_{i,t}, \quad \epsilon_{i,t} = \sigma_{i,t} z_{i,t},$$

$$\sigma_{i,t}^2 = \omega_i + \alpha_i \epsilon_{i,t-1}^2 + \beta_i \sigma_{i,t-1}^2,$$

where  $z_{i,t}$  are i.i.d. standardized residuals (with zero mean and unit variance). Let  $D_t = \text{diag}(\sigma_{1,t}, \sigma_{2,t}, \dots, \sigma_{N,t})$  be the diagonal matrix of current volatilities. The core of the DCC is to model the correlation matrix  $R_t$  of the residual vector  $z_t = (\epsilon_{1,t}/\sigma_{1,t}, \dots, \epsilon_{N,t}/\sigma_{N,t})$ . The DCC model assumes:

$$Q_t = (1 - a - b) \bar{Q} + a (z_{t-1} z_{t-1}^\top) + b Q_{t-1}, \quad (3.12)$$

where  $Q_t$  is an intermediate “quasi-covariance” matrix,  $\bar{Q}$  is the unconditional covariance of  $z_t$  (estimated from data), and  $a, b$  are scalar parameters (with  $a + b < 1$  for stability). The actual correlation matrix is  $R_t = \text{diag}(Q_t)^{-1/2} Q_t \text{diag}(Q_t)^{-1/2}$ , which standardizes  $Q_t$  to have ones on the diagonal. Equation (3.12) implies that correlations mean-revert to  $\bar{Q}$  and respond to recent shocks  $z_{t-1} z_{t-1}^\top$ . The parameters  $a$  and  $b$  determine how fast correlations react ( $a$ ) and how much they persist ( $b$ ) (?).

The ADCC model adds an asymmetric term to capture the empirical fact that correlations often increase after joint negative returns (e.g., markets becoming more correlated in a crisis). In ADCC, an additional term  $g(n_{t-1} n_{t-1}^\top)$  is added to (3.12), where  $n_{t-1}$  is a vector of indicators for negative returns (e.g.,  $n_{i,t-1} = 1$  if  $\epsilon_{i,t-1} < 0$ , else 0). The modified equation is:

$$Q_t = (1 - a - b - g) \bar{Q} + a (z_{t-1} z_{t-1}^\top) + b Q_{t-1} + g (n_{t-1} n_{t-1}^\top), \quad (3.13)$$

as introduced by ?. The  $g$  term allows recent negative shocks (where one or more  $n_{i,t-1} = 1$ ) to have an extra impact on increasing correlations, reflecting “flight-to-quality” or contagion effects observed in downturns. We estimate the ADCC parameters ( $a, b, g$ ) via maximum likelihood, using a two-step procedure: first estimate each univariate GARCH for volatilities, then estimate the correlation dynamics. The output of the ADCC model is a forecast of the full  $N \times N$  correlation matrix  $R_{t+1}$  for the next day’s returns, which we pair with the volatility forecasts to build the covariance matrix.



### 3.3.3 Method B: Supervised Learning Models for Covariance Prediction (LSTM, Transformer, Autoformer)

In contrast to the two-stage econometric modeling approach of Method A (XGBoost and ADCC), Method B adopts a direct forecasting strategy, employing advanced supervised machine learning techniques to predict the covariance structure of asset returns. This methodology explicitly targets the Cholesky decomposition of daily realized covariance matrices. The principal motivation behind directly predicting the Cholesky decomposition is twofold: it ensures the positive definiteness and numerical stability of forecasted covariance matrices, while simultaneously allowing models to capture complex nonlinear and temporal interdependencies inherent in cryptocurrency return data.

Formally, the daily realized covariance matrix  $\Sigma_t^{\text{realized}}$  is computed from intraday (minute-frequency) returns as:

$$\Sigma_t^{\text{realized}} = \sum_{m=1}^M r_{t,m} r_{t,m}^\top, \quad (3.14)$$

where  $r_{t,m}$  denotes the vector of returns for all assets at minute  $m$  of day  $t$ , and  $M$  represents the number of minute-level observations per day. Subsequently, the covariance matrix undergoes a Cholesky decomposition:

$$\Sigma_t^{\text{realized}} = L_t L_t^\top, \quad (3.15)$$

with  $L_t$  being a unique lower-triangular matrix with strictly positive diagonal elements. This decomposition serves as the prediction target of the supervised models, ensuring predicted covariance matrices are always positive definite.

We explore three advanced sequence modeling architectures extensively documented in modern financial forecasting: Long Short-Term Memory (LSTM) networks, Transformer models, and Autoformer models. Each of these is detailed comprehensively in the following sections.

**Long Short-Term Memory (LSTM) Networks** The LSTM architecture, introduced by ?, is particularly adept at modeling sequential data with long-range temporal dependencies. LSTM addresses the vanishing gradient problem commonly encountered in traditional recurrent neural networks (RNNs), enabling the capture of persistent volatility and correlation dynamics observed in financial data.

*LSTM Architecture and Equations.* Each LSTM unit consists of a cell state ( $c_t$ ) and three gating mechanisms: forget gate ( $f_t$ ), input gate ( $i_t$ ), and output gate ( $o_t$ ), controlling information flow across time steps. Mathematically, the LSTM unit computations are defined as follows:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \quad (\text{Forget gate}) \quad (3.16)$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \quad (\text{Input gate}) \quad (3.17)$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o), \quad (\text{Output gate}) \quad (3.18)$$

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c), \quad (\text{Candidate state}) \quad (3.19)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \quad (\text{Cell state update}) \quad (3.20)$$

$$h_t = o_t \odot \tanh(c_t), \quad (\text{Hidden state update}) \quad (3.21)$$

Here,  $x_t$  denotes the input vector at time  $t$ ,  $h_{t-1}$  is the previous hidden state,  $\sigma$  represents the sigmoid activation function, and  $\odot$  denotes element-wise multiplication. The parameters  $W$ ,  $U$ , and biases  $b$  are learned during model training.

*Application to Covariance Prediction.* In our context, the LSTM network processes rolling sequences of historical Cholesky decomposition matrices  $(L_{t-k}, \dots, L_t)$  as input, predicting the next-day decomposition matrix  $(L_{t+1})$ . The output layer is structured to ensure lower-triangular form, and appropriate activations (e.g., exponential) guarantee positive diagonal elements.

## Long Short-Term Memory (LSTM) Networks

The Long Short-Term Memory (LSTM) architecture, initially proposed by ?, effectively addresses the vanishing and exploding gradient problems inherent in traditional recurrent neural networks (RNNs), allowing for robust modeling of complex temporal dependencies. LSTMs achieve this by introducing specialized gating mechanisms within their recurrent units, enabling selective information flow and long-term memory retention.

*Theoretical Foundations of LSTM.* Standard RNN architectures struggle to propagate gradients through long sequences due to exponential error decay or growth, severely limiting their capacity to capture long-term dependencies. LSTM mitigates this issue by incorporating memory cells and multiplicative gating units, which facilitate controlled storage and retrieval of information over extended time intervals. Central to this mechanism is the *constant error carousel* (CEC), which allows error gradients to flow backward through time without significant alteration.

*Detailed LSTM Equations.* An LSTM unit at time  $t$  maintains two distinct state vectors: the hidden state ( $h_t$ ) and the cell state ( $c_t$ ). The cell state acts as long-term memory, regulated by gating units—input gate ( $i_t$ ), forget gate ( $f_t$ ), and output gate ( $o_t$ ). These gates control information flow by dynamically scaling signals entering and leaving the memory cell. The computations within an LSTM unit are formally described by the following equations:



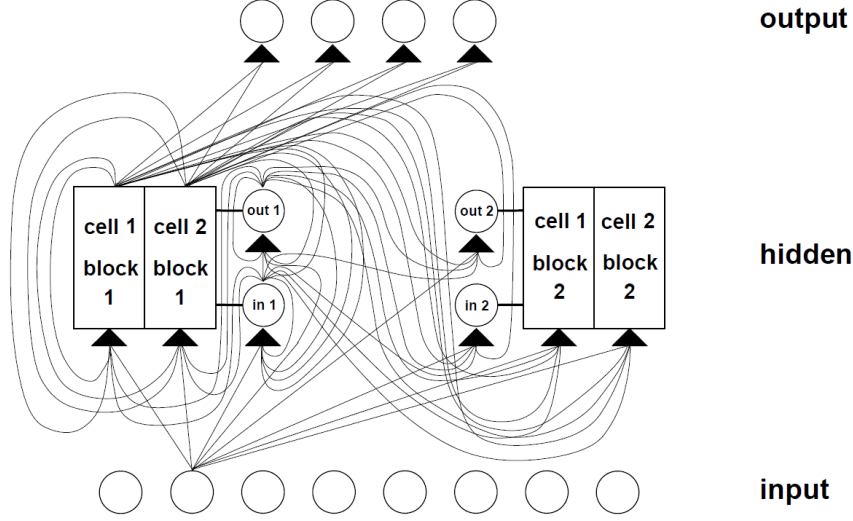


Figure 3.7: Illustration of the LSTM network topology.

enabling the model to predict the next period's Cholesky decomposition ( $L_{t+1}$ ). The model outputs are reshaped into lower-triangular form, ensuring a valid Cholesky matrix with positive diagonal elements through appropriate activation functions (e.g., exponential or softplus).

*Training Procedure and Loss Function.* LSTM training involves minimizing a loss function measuring the discrepancy between predicted and realized Cholesky decompositions, typically using the Frobenius norm:

$$L_{\text{MSE}} = \frac{1}{T} \sum_{t=1}^T \|\hat{L}_{t+1} - L_{t+1}\|_F^2, \quad (3.28)$$

where  $\|\cdot\|_F$  denotes the Frobenius norm. Regularization terms are also introduced to encourage temporal smoothness, penalizing significant deviations between successive predictions to indirectly control portfolio turnover induced by rapid changes in forecasted covariance structures.

*Summary and Benefits.* Overall, the LSTM's gated recurrent structure effectively captures complex, nonlinear temporal relationships in covariance data, crucial for robust financial forecasting. Its inherent stability in gradient propagation positions LSTM as an exceptionally suitable method for long-term covariance prediction tasks, significantly outperforming traditional recurrent neural networks in capturing persistent volatility and correlation structures observed in cryptocurrency markets and other financial assets.

## Transformer Models

Transformer architectures, first proposed by ?, have dramatically transformed sequence modeling tasks due to their innovative self-attention mechanisms. Unlike recurrent neu-

ral networks, Transformers do not process data sequentially, which allows simultaneous modeling of dependencies across all positions within the sequence, significantly increasing computational parallelism and effectively capturing long-range dependencies.

*Transformer Architecture and Theoretical Foundations.* A Transformer comprises encoder and decoder stacks, each containing layers that consist primarily of multi-head self-attention mechanisms and position-wise fully connected feed-forward networks (see Figure 3.8). Both stacks utilize residual connections (?) followed by layer normalization (?), ensuring stable gradient propagation during training.

The cornerstone of the Transformer architecture is its self-attention mechanism, specifically scaled dot-product attention, defined mathematically as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V, \quad (3.29)$$

where the queries ( $Q$ ), keys ( $K$ ), and values ( $V$ ) are matrices representing input embeddings projected into different spaces, and  $d_k$  is the dimensionality of the keys and queries. Scaling by  $\sqrt{d_k}$  helps to stabilize gradients by preventing excessively large dot-product values that can saturate the softmax function (?).

The Transformer further leverages *multi-head attention* to jointly capture diverse patterns and dependencies across multiple representation subspaces. The multi-head mechanism applies parallel attention functions on projected inputs:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O, \quad (3.30)$$

where each individual attention head is computed as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V). \quad (3.31)$$

Here, projection matrices  $W_i^Q, W_i^K, W_i^V$ , and  $W^O$  are learned parameters, projecting inputs into lower-dimensional subspaces. Typically,  $h = 8$  heads are utilized, each with dimension  $d_k = d_v = d_{\text{model}}/h = 64$ , striking an optimal balance between computational efficiency and representational capacity.

Additionally, position-wise feed-forward networks (FFN) complement the attention layers by independently transforming each position's embedding through two linear transformations separated by a ReLU activation:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2. \quad (3.32)$$

Each layer uses identical operations across sequence positions but distinct parameters, thus preserving positional independence within the model layers.

Since the Transformer architecture inherently lacks recurrence or convolution, positional encoding is necessary to provide order information. A sinusoidal positional encod-

ing, added directly to token embeddings, is defined as:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \quad (3.33)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \quad (3.34)$$

where  $pos$  denotes token position and  $i$  represents the embedding dimension index. This encoding enables the model to leverage both relative and absolute positional information effectively.

*Application to Covariance Prediction.* In covariance forecasting tasks, the Transformer processes historical sequences of Cholesky decompositions derived from realized covariance matrices. Its self-attention mechanism enables the model to dynamically weigh past observations based on their relevance, effectively identifying volatility clustering, structural breaks, and cross-asset correlations without explicit temporal constraints. This approach is highly advantageous in capturing subtle patterns inherent in financial markets, particularly during episodes of regime shifts or shocks.

Regularization strategies such as dropout (with rates typically between 0.1 to 0.3), attention dropout, and rigorous hyperparameter tuning (number of layers, heads, dimension sizes, learning rates) are employed to mitigate overfitting risks due to the limited historical data often encountered in financial forecasting applications.

The comprehensive structure and operational flow of the Transformer model for covariance prediction are illustrated in Figure 3.8, clearly demonstrating the interaction between multi-head self-attention layers, positional encodings, and position-wise feed-forward networks.

*Advantages of Transformer for Covariance Forecasting.* The Transformer's attention mechanism naturally addresses the challenge of modeling long-range dependencies prevalent in financial time series, without the gradient vanishing or exploding problems associated with RNN-based models. Its parallelizable computations offer considerable speed and scalability benefits, especially valuable in financial applications involving large asset universes or extensive historical datasets. Moreover, its ability to attend selectively and adaptively to different sequence elements provides interpretability advantages, as attention distributions can explicitly indicate which historical events or periods predominantly influence the covariance forecasts.

Overall, the Transformer architecture, with its theoretical elegance, computational efficiency, and strong empirical performance, represents a potent method for covariance prediction in modern quantitative finance, complementing and sometimes surpassing traditional econometric approaches.

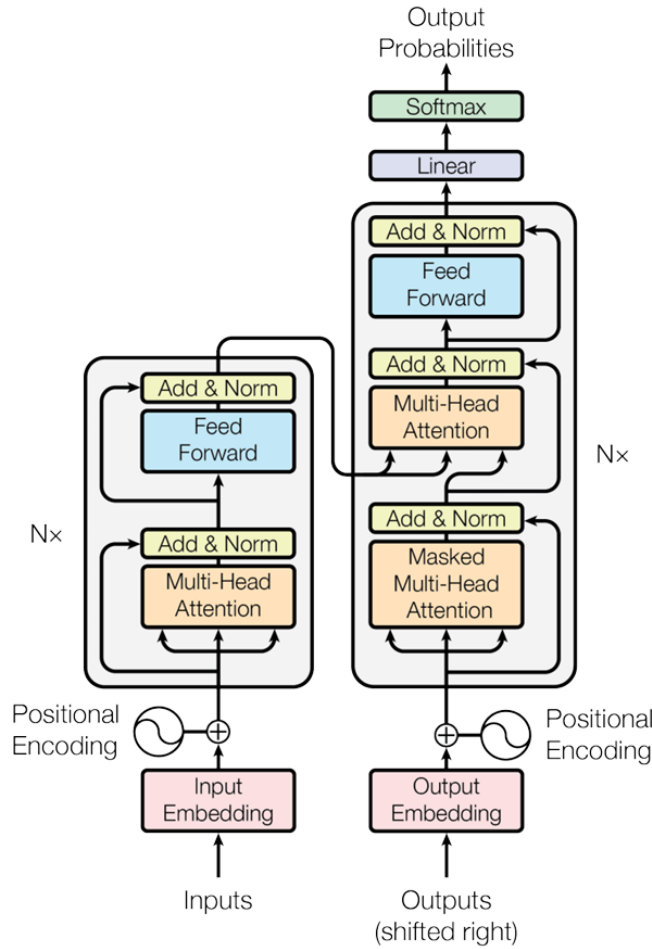


Figure 3.8: Detailed schematic of the Transformer encoder-decoder architecture, illustrating multi-head self-attention mechanisms, position-wise feed-forward layers, positional encodings, and residual connections. (Figure extracted from ?, page 3).

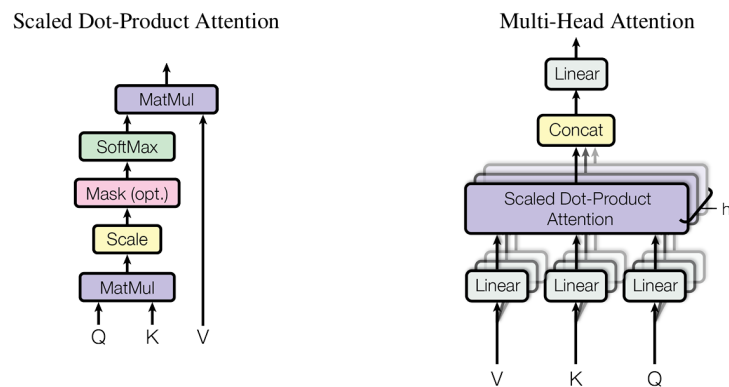


Figure 3.9: Illustration of scaled dot-product attention and multi-head attention mechanisms used in the Transformer, highlighting their parallel structure and information flow. (Adapted from ?, page 4, Figure 2.)

## Autoformer: Auto-Correlation-Based Transformer for Covariance Prediction

The Autoformer, introduced by ?, is a Transformer-based architecture specifically engineered to capture and forecast long-term dependencies within time series data. It introduces two major innovations: an intrinsic **progressive series decomposition** mechanism and an efficient **Auto-Correlation attention mechanism** based on periodicity rather than traditional point-wise self-attention.

### Autoformer Architecture and Series Decomposition

The core idea behind Autoformer is its ability to progressively decompose input series into trend-cyclical and seasonal components. Unlike traditional decomposition methods used purely as preprocessing steps, the Autoformer incorporates decomposition as an inner operation, continuously applied throughout the model’s layers to capture the intricate interactions of future series patterns.

Given a multivariate time series  $X \in \mathbb{R}^{L \times d}$ , the decomposition at each step is performed as:

$$X_t = \text{AvgPool}(\text{Padding}(X)), \quad (3.35)$$

$$X_s = X - X_t, \quad (3.36)$$

where  $X_s$  is the seasonal (short-term fluctuations) component and  $X_t$  the trend-cyclical (long-term progression) component.

The Autoformer adopts an encoder-decoder structure. The encoder extracts seasonal patterns, while the decoder aggregates and refines the trend-cyclical components progressively throughout its layers. Each encoder and decoder layer incorporates multiple series decomposition blocks interleaved with Auto-Correlation attention blocks to gradually refine the prediction (see Figure 1 on page 4 of ?).

**Auto-Correlation Attention Mechanism** Distinct from traditional Transformer architectures, Autoformer replaces the self-attention mechanism with a novel Auto-Correlation attention. This mechanism leverages the inherent periodicity and autocorrelation properties of time series data to identify and aggregate similar sub-series across different periods, greatly enhancing predictive accuracy and efficiency.

Given discrete-time processes  $\{X_t\}$ , the Auto-Correlation function at lag  $\tau$  is calculated by:

$$R_{XX}(\tau) = \lim_{L \rightarrow \infty} \frac{1}{L} \sum_{t=1}^L X_t X_{t-\tau}, \quad (3.37)$$

which captures the similarity between the series and its lagged versions.

The Auto-Correlation mechanism selects the most significant delays (lags)  $\tau_1, \tau_2, \dots, \tau_k$



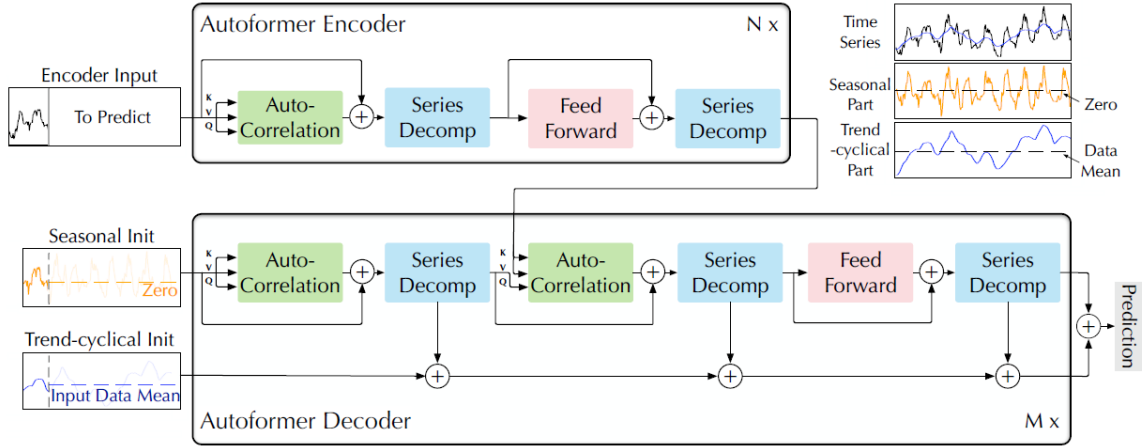


Figure 3.10: Autoformer architecture illustrating the series decomposition blocks (blue) and Auto-Correlation mechanisms (green). The encoder captures seasonal patterns, while the decoder progressively aggregates and refines the trend component (from ?, page 4).

based on the magnitude of  $R_{XX}(\tau)$  and aggregates these time-delayed series via a *time-delay aggregation* operation defined as:

$$\text{Auto-Correlation}(Q, K, V) = \sum_{i=1}^k \text{Roll}(V, \tau_i) \cdot \hat{R}_{Q,K}(\tau_i), \quad (3.38)$$

where  $\text{Roll}(V, \tau)$  shifts the series  $V$  by lag  $\tau$ , aligning sub-series that correspond to similar temporal patterns. The weights  $\hat{R}_{Q,K}(\tau)$  are computed via softmax normalization of the autocorrelations:

$$\hat{R}_{Q,K}(\tau_1), \dots, \hat{R}_{Q,K}(\tau_k) = \text{Softmax}(R_{Q,K}(\tau_1), \dots, R_{Q,K}(\tau_k)). \quad (3.39)$$

This operation significantly differs from standard self-attention, as it focuses explicitly on sub-series alignment rather than isolated point-to-point attention, resulting in higher computational efficiency ( $O(L \log L)$ ) and improved ability to capture complex periodic structures inherent in financial data.

### Application to Covariance Prediction

In our supervised covariance forecasting framework, Autoformer processes historical sequences of Cholesky decomposition matrices derived from realized covariance matrices. The input sequence is:

$$(L_{t-k}, \dots, L_t) \quad \text{where} \quad \Sigma_t = L_t L_t^\top,$$

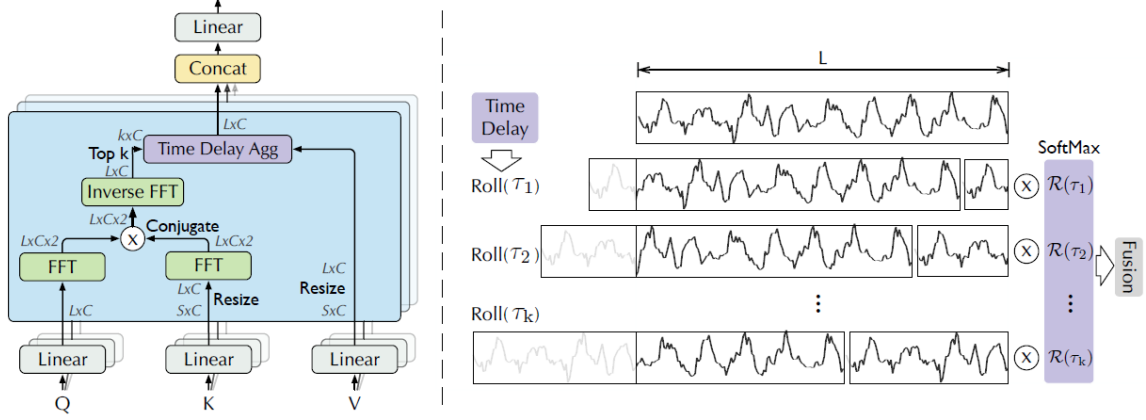


Figure 3.11: Auto-Correlation mechanism and Time Delay Aggregation block. The top- $k$  lags  $(\tau_1, \dots, \tau_k)$  are identified based on autocorrelation, and sub-series at corresponding delays are aggregated (from ?, page 5).

and the model predicts  $L_{t+1}$ . The encoder captures seasonal patterns from past Cholesky decompositions, while the decoder leverages Auto-Correlation and series decomposition blocks to progressively forecast future Cholesky matrices. This ensures the forecasted covariance matrices remain positive definite, stable, and reflective of complex temporal correlations in asset returns.

Model training employs a Frobenius norm-based mean squared error loss:

$$L_{MSE} = \frac{1}{T} \sum_{t=1}^T \|\hat{L}_{t+1} - L_{t+1}\|_F^2, \quad (3.40)$$

with additional regularization terms encouraging smooth transitions between predicted covariance matrices over successive periods, thus indirectly controlling portfolio turnover.

Through its progressive decomposition and advanced autocorrelation-based attention, the Autoformer model adeptly captures intricate temporal and cross-sectional covariance dynamics, presenting a powerful alternative to traditional econometric and standard Transformer-based methods.

### 3.3.4 Processing forecasts of Approach 1

#### Covariance Matrix Construction and Robustification

Given the outputs of the previous two components, we construct the forecasted covariance matrix  $\hat{\Sigma}_{t+1}$  for asset returns on day  $t+1$ . If  $\hat{\sigma}_{i,t+1}$  is the forecast volatility of asset  $i$  (from the XGBoost-HAR model) and  $\hat{R}_{t+1}$  is the forecast correlation matrix (from ADCC), then we form:

$$\hat{\Sigma}_{t+1} = \hat{D}_{t+1} \hat{R}_{t+1} \hat{D}_{t+1}, \quad (3.41)$$

where  $\hat{D}_{t+1} = \text{diag}(\hat{\sigma}_{1,t+1}, \dots, \hat{\sigma}_{N,t+1})$ . In other words,  $\hat{\Sigma}_{ij,t+1} = \hat{\rho}_{ij,t+1} \hat{\sigma}_{i,t+1} \hat{\sigma}_{j,t+1}$  for  $i, j$  assets, which is a natural way to combine volatilities and correlations into covariances.

However, raw covariance matrix estimates, especially those based on limited data or purely statistical models, can be noisy and lead to unstable portfolio weights (?). We implement two forms of regularization to robustify the covariance matrix: *shrinkage* and *eigenvalue filtering*.

**Ledoit-Wolf Shrinkage:** We apply the shrinkage estimator of ? to the sample covariance matrix of recent returns as an additional layer of stability. The idea of shrinkage is to take a weighted average of the empirical covariance matrix  $S_t$  (sampled over a recent window, e.g., past 50-100 days up to  $t$ ) and a structured target matrix  $F$  that is more stable. A common choice for  $F$  is the constant correlation matrix (with all off-diagonal correlations equal to the average correlation) or simply the diagonal of  $S_t$  (which assumes zero correlation). The shrunk covariance is:

$$\tilde{\Sigma}_t = \lambda F + (1 - \lambda) S_t, \quad (3.42)$$

where  $0 \leq \lambda \leq 1$  is the shrinkage intensity. ? provide an analytic formula for the optimal  $\lambda$  that minimizes the mean-squared error between  $\tilde{\Sigma}_t$  and the true covariance, under certain assumptions. We use this approach to adjust  $\hat{\Sigma}_{t+1}$  by shrinking it toward a target. In practice, we might shrink towards the constant correlation structure to avoid extreme correlation estimates. Shrinkage has a solid theoretical foundation in improving covariance estimation, especially when  $N$  is large relative to the observation window (?). By reducing estimation noise, it often improves out-of-sample portfolio performance.

**Eigenvalue Filtering:** Even after shrinkage,  $\hat{\Sigma}_{t+1}$  might have eigenvalues that are very small or very large purely due to noise (particularly if  $N$  is large or the correlation estimator is imperfect). We employ eigenvalue filtering based on Random Matrix Theory (e.g., ?), which suggests that beyond a certain threshold, small eigenvalues of a sample covariance matrix are likely to be dominated by estimation error. We perform a simple eigenvalue adjustment: we eigen-decompose  $\hat{\Sigma}_{t+1} = V \Lambda V^\top$ , where  $\Lambda = \text{diag}(\ell_1, \dots, \ell_N)$  are eigenvalues sorted in ascending order. If any eigenvalue  $\ell_i$  is extremely small or numerically zero (which could make  $\hat{\Sigma}$  nearly singular and cause instability in optimization), we floor it to a minimum level  $\ell_{\min}$  or replace it with an average of neighboring eigenvalues. Likewise, we may cap extremely large eigenvalues if needed. The adjusted covariance is then reconstructed as  $\hat{\Sigma}_{t+1}^{(\text{adj})} = V \Lambda^{(\text{adj})} V^\top$ . This filtering ensures  $\hat{\Sigma}_{t+1}^{(\text{adj})}$  is positive-definite and better-conditioned for optimization. Such eigenvalue cleaning techniques have been shown to improve the estimation of the true underlying covariance by removing noise (?).

At the end of this step, we have a robust forecast covariance matrix for the next

period's asset returns. We denote this matrix as  $\hat{\Sigma}_{t+1}^*$ . This will be the input in the portfolio optimization stage to determine the asset allocation.

### Portfolio Optimization (GMV, MDP, ERC)

With a forecast covariance matrix  $\hat{\Sigma}_{t+1}^*$  in hand, we solve for optimal portfolio weights under various risk-based optimization objectives. In this thesis, we focus on three portfolio constructions: Global Minimum Variance (GMV), Maximum Diversification (MDP), and Equal Risk Contribution (ERC). These portfolios do not require forecasts of expected returns (which are notoriously difficult to estimate), instead relying solely on the covariance matrix to allocate risk efficiently.

**Global Minimum Variance (GMV):** The GMV portfolio is the one that achieves the lowest possible portfolio variance among all portfolios on the efficient frontier (?). Formally, the GMV weights  $w^{GMV}$  solve:

$$\min_w w^\top \hat{\Sigma}_{t+1}^* w, \quad (3.43)$$

$$\text{s.t.} \quad \sum_{i=1}^N w_i = 1, \quad (3.44)$$

$$w_i \geq 0, \quad \forall i, \quad (3.45)$$

assuming no short-selling (we impose  $w_i \geq 0$ ) and full investment. The closed-form solution (without the no-short constraint) is proportional to the inverse of the covariance:  $w^{GMV} \propto (\hat{\Sigma}_{t+1}^*)^{-1} \mathbf{1}$ , where  $\mathbf{1}$  is a vector of ones. The GMV portfolio concentrates more weight in assets with lower volatility and lower correlation to others, achieving minimal total risk. We include the no-short constraint to ensure practical investability and to avoid leveraging extremely low-variance assets. The GMV approach is purely risk-driven and serves as a baseline for minimum-risk allocation.

**Maximum Diversification (MDP):** The Maximum Diversification portfolio, introduced by ?, aims to maximize the “diversification ratio,” which is defined as the ratio of the weighted average volatility of assets to the portfolio volatility. Given asset volatilities (we can use  $\hat{\sigma}_{i,t+1}$ ) and covariance, the diversification ratio of a weight vector  $w$  is:

$$D(w) = \frac{\sum_i w_i \hat{\sigma}_{i,t+1}}{\sqrt{w^\top \hat{\Sigma}_{t+1}^* w}}. \quad (3.46)$$

The MDP weights  $w^{MDP}$  are those that maximize  $D(w)$  subject to  $w$  summing to 1 (and typically  $w_i \geq 0$ ). Equivalently, one can show this is the solution to

$$\max_{w \geq 0} \sum_i w_i \hat{\sigma}_{i,t+1} \quad \text{s.t.} \quad w^\top \hat{\Sigma}_{t+1}^* w = 1, \quad (3.47)$$

which leads to a solution where each asset’s weight is proportional to its volatility-adjusted by its covariance with others. Intuitively, the MDP emphasizes holding assets that contribute to the portfolio volatility in a balanced way—an asset with high standalone volatility can still get a significant weight if it has low correlation with the rest, thus improving diversification. The MDP tends to allocate more to less correlated assets, achieving a portfolio that is maximally diversified across risk sources (?). In practice, we solve this optimization numerically (it is a convex quadratic problem after some algebraic manipulation). The result is a portfolio with typically intermediate risk—higher variance than GMV (because we didn’t strictly minimize variance) but more diversified exposures, which can lead to better risk-adjusted returns if the assumption “diversification is the only free lunch” holds.

**Equal Risk Contribution (ERC):** The Equal Risk Contribution portfolio, also known as the risk parity portfolio, seeks to allocate weights such that each asset contributes equally to the total portfolio risk (?). The risk contribution of asset  $i$  in a portfolio  $w$  is defined as:

$$RC_i(w) = w_i \frac{\partial}{\partial w_i} \sqrt{w^\top \Sigma w} = w_i [\hat{\Sigma}_{t+1}^* w]_i, \quad (3.48)$$

which simplifies to  $RC_i = w_i (\text{row } i \text{ of } \hat{\Sigma}_{t+1}^*) \cdot w = w_i \sum_j \hat{\sigma}_{ij,t+1} w_j$ . In an ERC portfolio,  $RC_i$  is the same for all  $i$ . There is no closed-form solution in general, but it can be found by solving the system of equations  $RC_1 = RC_2 = \dots = RC_N$  with  $\sum w_i = 1, w_i \geq 0$ . Equivalently, one can solve:

$$\min_{w \geq 0} \sum_{i < j} (RC_i(w) - RC_j(w))^2, \quad (3.49)$$

or use specialized algorithms as in ?. We use an iterative method: start with an initial guess (like equal weights) and adjust weights in the direction that equalizes the risk contributions, repeating until convergence. The ERC portfolio tends to put more weight on lower-volatility assets but not as extremely as GMV, because it also accounts for correlation: highly correlated groups of assets together will not dominate the risk. ERC portfolios have gained popularity as a way to balance risk without requiring return forecasts, and they often achieve a compromise between high diversification and low volatility.

All three portfolios (GMV, MDP, ERC) are computed at each rebalancing period using

the forecast covariance  $\hat{\Sigma}_{t+1}^*$ . Notably, if our forecast covariance is accurate, the GMV portfolio will indeed be the ex-post minimum variance; the MDP will capture uncorrelated return streams; and the ERC will avoid concentration in any single asset’s risk. We rebalance the portfolio according to each strategy and record the weights for evaluation. As an additional consideration, we include modest transaction cost assumptions when moving from one period’s weights to the next, to ensure that turnover is kept at realistic levels—however, the core methodology focuses on the weight generation rather than a detailed trading cost model.

### 3.4 Approach 2: Direct Portfolio Weight Prediction

Approach 2 bypasses the explicit estimation of volatilities and covariances by using data-driven models to directly compute the portfolio weights based on historical information. We explore two main directions under this approach: (1) **Supervised Learning models** that treat the next-period optimal weights as a prediction target, and (2) **Deep Reinforcement Learning** that treats portfolio allocation as a sequential decision-making problem and learns a policy to choose weights maximizing cumulative rewards. The motivation here is that a single-stage model might capture nonlinear relationships between asset returns, momentum, and other indicators in a way that directly optimizes the end goal (portfolio performance), potentially outperforming a two-stage approach that might propagate errors from forecasts to allocation.

#### 3.4.1 Supervised Learning Models for Direct Weight Prediction (LSTM, Transformer, Autoformer)

In Approach 2, supervised learning models (LSTM, Transformer, Autoformer), previously detailed in Approach 1 (Method B, section 3.2), are repurposed to directly predict portfolio weights. Unlike covariance forecasting, here the models are trained to map historical asset and market information directly to optimal portfolio weights without explicitly forecasting covariance matrices. The rationale behind directly predicting portfolio allocations is to exploit implicit relationships within market dynamics, potentially capturing nonlinear and complex patterns missed by explicit covariance-based optimizations.

The training procedure, loss functions, and regularization techniques for these supervised models closely parallel those described previously (see section 3.2), adjusted specifically for portfolio weight prediction targets.

### 3.4.2 Supervised Learning Models for Weight Prediction (LSTM, Transformer, Autoformer)

In the supervised learning paradigm, we attempt to learn a mapping from recent market data to the portfolio weights that would be ideal for the next period. This requires constructing a training set of input-output pairs  $\{X_t, w_t^*\}$ , where  $X_t$  is a vector of features summarizing the state of the market at time  $t$  (e.g., recent returns, volatilities, indicators) and  $w_t^*$  is a target weight vector that we consider “optimal” for period  $t + 1$ . One way to obtain  $w_t^*$  for training is to use the retrospective optimal weights (for instance, the weights that would have maximized the portfolio’s reward at  $t + 1$  using information up to  $t$ ). In practice, we might use the weights from Approach 1 or a known strategy as proxy targets. Another approach is to define  $w_t^*$  through some heuristic or rule (e.g., the Markowitz solution using sample covariance from a long window and an assumed expected return vector). Once we have targets, we can fit a supervised model to approximate this mapping.

We experiment with several advanced sequence modeling architectures:

- **Long Short-Term Memory (LSTM) networks:** LSTM is a type of recurrent neural network well-suited for sequence data that can capture long-term dependencies through its gating mechanisms (?). LSTMs have been successfully applied to financial time series forecasting problems, capturing temporal patterns in returns and volatility (e.g., ?). In our context, an LSTM processes a sequence of past observations (such as a rolling window of past  $k$  days of returns or other features) and outputs a prediction for the next period’s portfolio weights. The network consists of an input layer (taking features for each day sequentially), one or more LSTM layers that maintain hidden states, and a final dense layer that produces  $N$  outputs (corresponding to weights for each asset). We constrain the outputs to sum to 1 (e.g., by using a softmax layer or by normalizing the outputs manually) to ensure the network predicts a valid weight vector. The loss function for training can be a mean squared error between predicted and target weights, or even a portfolio performance metric (though the latter makes it a form of reinforcement learning, which we handle in the next subsection). The LSTM’s ability to model nonlinear temporal relationships may allow it to anticipate patterns like momentum or mean reversion that inform the optimal allocation.
- **Transformer-based models:** Transformers (?) have revolutionized sequence modeling by using self-attention mechanisms to capture dependencies between elements in a sequence, without requiring recurrence. We apply a Transformer model to our portfolio data by treating the past  $k$  days as a sequence and the features of all assets on each day as the input at each time step. The self-attention mechanism

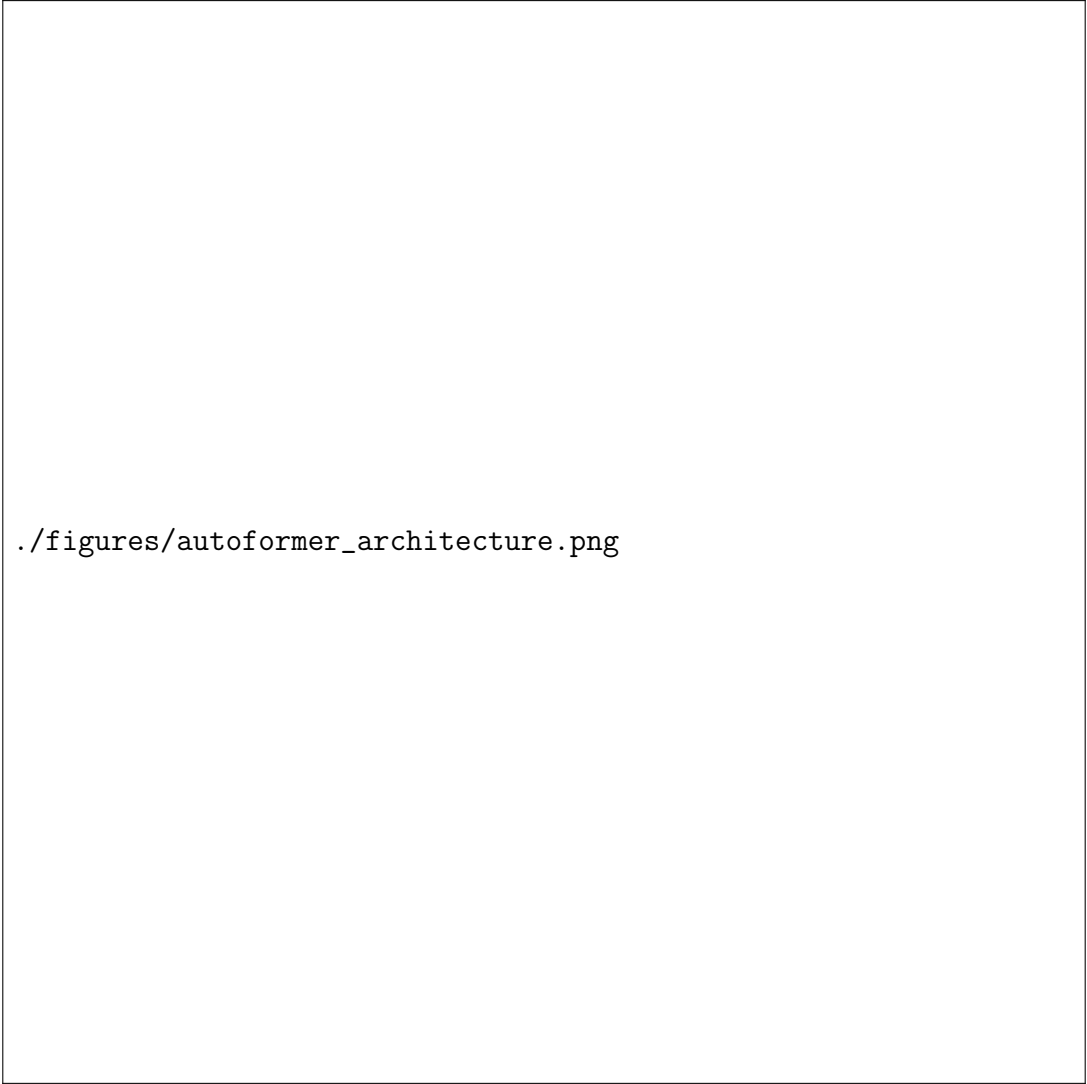
can learn which past days (and which assets on those days) are most relevant to deciding the allocation for the next day. Transformers are advantageous for their ability to handle long sequences and to parallelize computation. However, standard Transformers are data-hungry and may overfit on financial data if not regularized, due to many parameters. Therefore, we also consider specialized time-series Transformers.

- **Autoformer:** We implement the Autoformer model proposed by [\[15\]](#), which is a Transformer-based architecture specifically designed for long-term time series forecasting. Autoformer introduces an “auto-correlation” mechanism and series decomposition blocks to extract seasonality and trend from time series data. It uses an encoder-decoder structure: the encoder learns representations of historical time series segments and the decoder makes predictions for future segments. In our case, we adapt Autoformer to predict future weights by training it on sequences of multi-asset data. The model first decomposes the input series (e.g., price or return sequences of each asset) into trend and seasonal components, then applies self-attention (via auto-correlation) to capture relationships. The output is a forecast of each asset’s contribution or weight. We found Autoformer appealing because it explicitly handles long sequences and can extrapolate patterns beyond the training window by capturing the series’ intrinsic periodicities ([\[15\]](#)). The architecture of Autoformer is shown in Figure 3.12, highlighting its encoder-decoder with multi-head attention and decomposition blocks.

Training these supervised models involves minimizing a loss between the model’s predicted weight vector  $\hat{w}_t$  and the target optimal weight  $w_t^*$ . We typically use an  $L^2$  (MSE) loss:  $L = \frac{1}{T} \sum_t \|\hat{w}_t - w_t^*\|^2$ . To prevent the model from predicting weights that change too erratically, we may augment the loss with a regularization term that penalizes large changes in weights (to implicitly control turnover). We also ensure that the output weight vector is feasible (e.g., no negative weights if not allowed, and sums to unity). This can be handled by the network output layer (using a softmax ensures non-negativity and sum to one). For each model (LSTM, vanilla Transformer, Autoformer), we tune hyperparameters such as the number of layers, hidden units, attention heads, etc., using the validation set. We also employ early stopping to avoid overfitting, given the limited amount of financial time series data relative to typical deep learning tasks.

One challenge in supervised weight prediction is that the “optimal” weights  $w_t^*$  used as targets may be noisy or based on specific model assumptions. However, even if approximate, the supervised models can learn a policy that generalizes those assumptions. For example, if  $w_t^*$  comes from a Markowitz optimization with an assumed return vector, the ML model might learn to infer that return vector from patterns in recent data. In summary, the supervised learning approach leverages powerful sequence models to di-





`./figures/autoformer_architecture.png`

Figure 3.12: Architecture of the Autoformer model (from ?). The model decomposes input time series into trend and seasonal components, applies an auto-correlation mechanism to identify salient periodic patterns, and uses an attention-based encoder-decoder to forecast future values. We adapt this for portfolio allocation by using multi-asset return sequences as inputs and forecasting normalized weight allocations as outputs.

rectly output portfolio decisions, effectively learning an implicit representation of market dynamics and risk-return trade-offs from the data. This approach is data-driven and may capture complex signals that are hard to formalize in a two-step approach.

### 3.4.3 Deep Reinforcement Learning with PPO for Portfolio Allocation

The second branch of Approach 2 uses deep reinforcement learning (DRL) to train an agent that interacts with the financial market environment and learns to make optimal allocation decisions. We frame the portfolio management problem as a Markov Decision

Process (MDP): at each time step  $t$ , the agent observes the state  $s_t$  (which could consist of recent asset returns, technical indicators, economic variables, and even the current portfolio holdings) and takes an action  $a_t$  which is choosing a new set of portfolio weights. After executing this allocation for period  $t$  to  $t + 1$ , the agent receives a reward  $R_{t+1}$ , for example the portfolio’s return or a utility (such as logarithmic return or Sharpe ratio increment), and the state transitions to  $s_{t+1}$ . The goal is to learn a policy  $\pi(a|s)$  that maximizes the expected cumulative reward (e.g., terminal wealth or long-term Sharpe ratio). This setup does not require predefined “optimal” weights for training; instead, the agent learns from its own experience by exploring different strategies and getting feedback via rewards.

We utilize the Proximal Policy Optimization (PPO) algorithm (?) to train our portfolio agent. PPO is a state-of-the-art policy gradient method known for its stability and reliability in training. It iteratively improves the policy by sampling trajectories (sequences of states, actions, rewards) and updating the policy parameters to increase the probability of actions that yield higher advantage (advantage = reward above a baseline). PPO introduces a clipped surrogate objective to prevent large, destabilizing updates to the policy. The objective function for PPO can be expressed as:

$$L^{\text{PPO}}(\theta) = \mathbb{E}_t \left[ \min \left( r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right], \quad (3.50)$$

where  $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$  is the ratio of the new policy probability to the old policy probability for action  $a_t$ ,  $\hat{A}_t$  is the estimated advantage at time  $t$ , and  $\epsilon$  is a small hyperparameter (e.g., 0.2) that defines the clip range. This objective essentially says: improve the policy such that actions with positive advantage are more likely, but do not change the probability ratio by more than  $\pm\epsilon$  to ensure a conservative update (?). By maximizing  $L^{\text{PPO}}$ , PPO finds a good balance between exploring new policies and staying close to the current policy to avoid performance collapse.

In our implementation, the policy  $\pi_\theta$  is represented by a neural network (the “actor”) that takes the state  $s_t$  as input and outputs a distribution over possible actions. We parameterize actions as the portfolio weight vector  $w_t$  itself. One approach is to have the network output  $N$  continuous values which are passed through a softmax function, so the output is a valid weight vector (nonnegative and summing to 1). Alternatively, we can output unconstrained weights and normalize them in a post-processing step. The reward  $R_{t+1}$  can be chosen to guide the agent towards desirable behavior; for example,  $R_{t+1}$  could simply be the portfolio’s logarithmic return from  $t$  to  $t + 1$  (which encourages wealth growth), or an objective that penalizes volatility and drawdowns to directly shape risk-adjusted performance. In our case, we consider  $R_{t+1} = \ln(1 + r_{p,t+1})$  where  $r_{p,t+1} = \sum_i w_{i,t} r_{i,t+1}$  is the portfolio return, which naturally encourages compounding of wealth. We also experimented with  $R_{t+1}$  defined as a trade-off like  $r_{p,t+1} - \alpha \times \text{RiskPenalty}$  (for

some small  $\alpha$ ) to incorporate risk aversion.

The training process involves simulating the agent over historical data. We use a rolling window approach: at each episode, the agent starts with an initial capital and no positions (or a predetermined initial allocation), then for each day in a training period, it observes state  $s_t$ , allocates  $w_t$ , and moves to next state until the end of the period. We accumulate the rewards and use them to update the policy and a value function (the “critic” in actor-critic terminology) which estimates expected returns to go. We repeat this over many episodes (sampling different time periods or adding randomness via exploration noise) so the agent experiences various market conditions. The PPO algorithm’s clipping mechanism and adaptive learning rate (we use a relatively small step size and Adam optimizer) help ensure the training is stable. Over time, the agent learns a policy that, for example, shifts weights towards assets with positive momentum or low volatility, cuts exposure during turbulent periods, etc., in order to maximize the cumulative return.

Our DRL approach draws on a growing body of research applying reinforcement learning to portfolio management. For instance, ? demonstrated a deep RL agent outperforming traditional strategies in cryptocurrency markets by directly learning the allocation policy. Compared to supervised learning, RL has the advantage that it directly optimizes the evaluation objective (portfolio growth) rather than an intermediate error, and it can learn to react to regime changes and sequential dependence of actions (such as the impact of today’s allocation on tomorrow’s opportunity set). Indeed, one can view the RL agent as learning an adaptive strategy that may not correspond to any single static optimization model. However, RL can be data-intensive and sensitive to reward specification and exploration; we mitigate this by using reward structures aligned with common portfolio goals and by baseline comparisons.

Figure 3.13 illustrates the RL loop for the portfolio problem. The state includes relevant features (such as a history of returns, volatility estimates, etc., similar to the inputs used in supervised models, plus possibly the last action to account for transaction costs or holdings), the action is a new weight vector, and the environment applies that action to generate the next state and reward. Over many iterations, the agent’s policy converges to a strategy that makes locally optimal decisions which generalize well across the time series. In summary, the PPO-based DRL approach provides a model-free way to derive a trading strategy and is a powerful complement to the model-based Approach 1 and the supervised Approach 2.1.

## 3.5 Model Training and Evaluation

Having specified the models and strategies for both approaches, we now describe how they are trained and evaluated. We pay particular attention to preventing overfitting (a

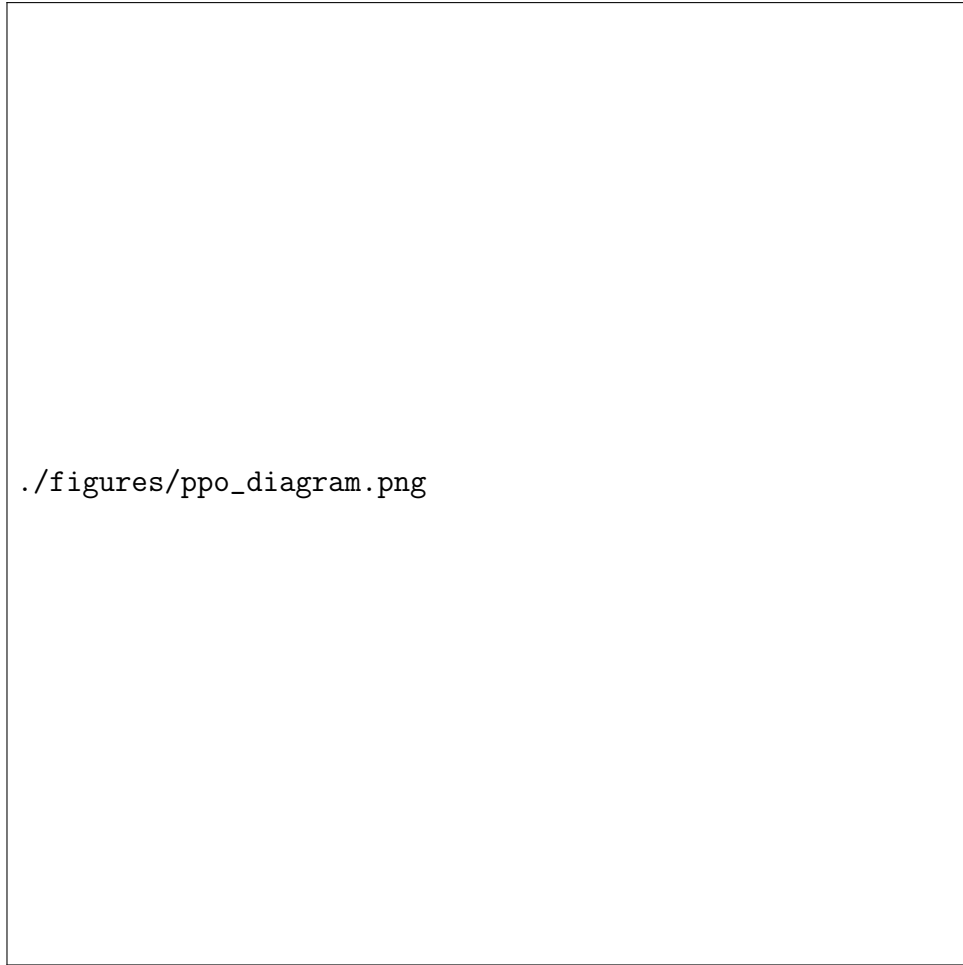


Figure 3.13: Reinforcement learning framework for direct portfolio allocation. The agent (policy network) observes the market state (e.g., recent returns, indicators) and outputs an action (portfolio weights). It then receives a reward equal to the portfolio’s return (or a risk-adjusted utility) for that period. The policy is updated via PPO to maximize long-term rewards. This loop allows the agent to learn an allocation strategy that directly optimizes portfolio performance.

critical issue in financial modeling due to non-stationarity and noise) and to the metrics and statistical tests used to compare performance across methods.

### 3.5.1 Training Process and Data Splits

We divide the historical data into three segments: a training set, a validation set, and a test set. The training period (e.g., Jan 2010–Dec 2018) is used to fit the models (XGBoost, GARCH/ADCC, LSTM, Transformer, etc.) or to train the RL agent. The validation period (e.g., Jan 2019–Dec 2019) is a hold-out sample used to tune hyperparameters and to perform model selection. Finally, the test period (e.g., Jan 2020–Dec 2021) is completely out-of-sample and is used for the final evaluation of performance. This chronological split ensures that our evaluation simulates a real-time forecasting scenario with no look-ahead

bias.

For the machine learning models in Approach 1 (XGBoost volatility and ADCC correlation), training involves maximizing likelihood or minimizing error on the training set. The HAR-XGBoost models are trained on pre-2019 data to predict next-day volatility. We use rolling cross-validation within the training set (e.g., train on 2010-2014, validate on 2015, then slide the window) to ensure the model generalizes to different market regimes and to decide on complexity (like tree depth or number of trees in XGBoost). The ADCC model’s GARCH parameters are estimated using data up to 2018 as well, with validation on 2019 to ensure the correlation dynamics are reasonable (we check, for example, that ADCC doesn’t overfit by looking at log-likelihood on validation data). Approach 1’s portfolio optimization step doesn’t have trainable parameters per se (aside from the shrinkage intensity which we can set via Ledoit-Wolf’s formula or tune slightly on validation). It is mostly an application of the forecasts.

For the supervised learning models in Approach 2 (LSTM, Transformer, Autoformer), training is done with backpropagation on the training set. We use early stopping based on validation loss to determine when to stop training to avoid overfitting, since deep networks can overfit quickly on limited financial data. We also employ techniques like dropout regularization in the LSTM/Transformer layers and limit the number of training epochs. The hyperparameters (learning rate, network depth, etc.) are chosen to minimize the validation error in predicting the target weights  $w_t^*$ . We encountered that beyond a certain model complexity, validation performance degraded, which indicated the need to keep the models reasonably small or heavily regularized for robustness.

The reinforcement learning agent with PPO is trained by interacting with the training set data as the market environment. We implement episodic training: the agent starts at the beginning of 2010 and traverses the data till end of 2018, then resets and repeats, for many epochs. We also randomize the start year of each episode occasionally, or even randomly sample chunks of the training timeline in different orders, to encourage the policy to be robust to different market conditions. The PPO hyperparameters (like clipping  $\epsilon$ , discount factor  $\gamma$ , GAE parameter  $\lambda$ , etc.) were initially set to standard values from ? and then adjusted slightly based on the stability of training and the validation performance (which we measure by the average reward or Sharpe ratio on the validation period when the agent is run without further learning). We limit the number of policy update iterations per episode to prevent overfitting to training sequences. Transaction cost and risk penalty considerations are included in the reward during training so that the agent’s learned policy naturally accounts for them.

All models are retrained or refit at a certain frequency to update them with new data as time progresses. In our study, we use a recursive expanding window scheme on the test set: at the start of the test period (2020), we use the models trained up to 2019 to make predictions in 2020. Then, one could imagine if this were an online process, we might

re-estimate or refit the models at the end of 2020 including that year’s data (especially for GARCH/ADCC which could be updated, or even refit XGBoost with new data) and then make forecasts for 2021, and so on. However, for clarity in evaluation, we mostly keep the models fixed as trained on pre-2020 data, and just simulate their performance on 2020-2021. The RL agent similarly is not updated during the test; we assess how a fixed policy learned on past data performs going forward.

### 3.5.2 Forecast and Allocation Accuracy Measures

To evaluate the models, we employ metrics that assess both the accuracy of intermediate forecasts (for Approach 1) and the quality of final portfolio outcomes (for both approaches).

For volatility forecasting (Approach 1’s first step), we use the Root Mean Squared Error (RMSE) and  $R^2$  (coefficient of determination) between predicted and realized volatility. If  $\hat{\sigma}_{i,t}$  is the predicted volatility for asset  $i$  on day  $t$  and  $\sigma_{i,t}$  the realized volatility (e.g., square root of realized variance) on that day, the RMSE for asset  $i$  over the test period  $T$  is:

$$\text{RMSE}_i = \sqrt{\frac{1}{T} \sum_{t=1}^T (\hat{\sigma}_{i,t} - \sigma_{i,t})^2}. \quad (3.51)$$

We report average RMSE across assets and also compare it to benchmarks (e.g., a simple HAR or GARCH model). We similarly evaluate correlation forecasts by looking at correlation prediction errors, though this is trickier to summarize; we use the Frobenius norm error of the correlation matrix or specific pairwise correlation errors.

For portfolio performance, we focus on standard metrics in asset management:

- **Annualized Sharpe Ratio:** The Sharpe ratio (?) is the average excess return of the portfolio divided by the standard deviation of returns. We compute it on the test set (assuming daily returns) as

$$\text{Sharpe} = \frac{\mu_p}{\sigma_p}, \quad (3.52)$$

where  $\mu_p = \frac{252}{T} \sum_{t=1}^T r_{p,t}$  is the average annualized portfolio return (assuming 252 trading days) and  $\sigma_p = \sqrt{252} \text{std}(r_{p,t})$  is the annualized volatility. If a risk-free rate is considered, we subtract it from  $r_{p,t}$  in  $\mu_p$ , but given the low interest rates in recent years, we often take excess returns as just raw returns. The Sharpe ratio gives a reward-to-risk efficiency measure; higher is better.

- **Max Drawdown (MDD):** Max drawdown is a downside risk measure that captures the largest peak-to-trough decline in the portfolio’s cumulative value during the period. Formally, if  $V_t$  is the cumulative portfolio value at time  $t$  (with  $V_0 = 1$ ),

the drawdown at time  $t$  is  $D_t = 1 - \frac{V_t}{\max_{s \leq t} V_s}$ . The maximum drawdown is  $\max_t D_t$ , the worst percentage loss from a peak. We report MDD as a percentage. Lower MDD indicates better capital preservation. This is especially relevant if two strategies have similar returns but one has a much deeper temporary loss.

- **Calmar Ratio:** We sometimes cite the Calmar ratio, which is the annualized return divided by the max drawdown, as another risk-adjusted metric focusing on drawdowns. It is complementary to Sharpe.
- **Turnover:** Although not listed in the question, we track portfolio turnover, which is the sum of absolute changes in weights quarter-to-quarter or year-to-year. High turnover can indicate overfitting or impracticality due to transaction costs. Our RL agent, for instance, might trade more frequently than a covariance-based strategy; we include this in evaluation to ensure differences in performance are not solely from taking on more trading.

Additionally, we check **forecast-based metrics** for Approach 2 implicitly by looking at how well the supervised model’s predicted weights correlate with some “ideal” weights. But ultimately, the true test of those predictions is the realized portfolio performance they achieve.

We compare all strategies (Approach 1 GMV, Approach 1 MDP, Approach 1 ERC, Approach 2 supervised, Approach 2 RL) on these metrics over the test set. For each metric, we see which approach delivered the best (e.g., highest Sharpe, lowest MDD). We also look at the consistency of performance over time (e.g., rolling 6-month Sharpe) to see if one approach is more stable.

### 3.5.3 Benchmarks and Statistical Comparison

To put the results in context, we include several benchmark strategies:

1. **Equal-Weight (1/N) Portfolio:** A naive benchmark that allocates equally to all assets and rebalances periodically. This strategy requires no forecasting and often is hard to beat out-of-sample (?). It provides a yardstick for whether our complex models add value.
2. **Historical Covariance GMV:** The traditional approach of computing the portfolio using the sample covariance matrix of past returns (e.g., 1-year window) for GMV optimization, without any fancy forecasting or shrinkage. This shows the benefit of our sophisticated Approach 1 pipeline.
3. **Market Index or Reference Fund:** If our asset universe has a natural benchmark (e.g., an S&P500 index if assets are S&P sectors, or a crypto index if assets

are cryptocurrencies), we compare the portfolio performance to that benchmark’s performance.

4. **Oracle (with hindsight) Portfolio:** For reference, we sometimes compute the “oracle” GMV or maximum Sharpe portfolio using full-sample information (or using actual realized covariance on the test set) to see an upper bound. This is not a investable strategy but tells us the maximum Sharpe achievable with perfect information.

To assess whether differences in performance are statistically significant, we conduct several tests: - For forecast accuracy (e.g., volatility RMSE), we use the *Diebold-Mariano test* (?) to check if one model’s forecast errors are significantly lower than another’s. For instance, we test null hypothesis that XGBoost-HAR and a pure HAR have equal predictive accuracy, using the time series of squared error differences. - For Sharpe ratios, we use the Jobson-Korkie test (as modified by ?) or the more robust ? test for difference in Sharpe ratios. These tests account for the non-independence of returns when evaluating if, say, the RL strategy’s Sharpe is higher than the GMV’s Sharpe beyond what could be due to chance. A  $p$ -value less than 0.05 would indicate significance at 95% confidence. - We also compare portfolios’ cumulative returns using the *reality check* of ? or the *SPA* test by ? to ensure that if we declare one strategy best, the performance is not due to data snooping. These tests involve generating a distribution of performance differences under the null of no superior model, often via bootstrap. - For the RL vs supervised approach in particular, we analyze if the differences in their achieved returns each month are significant (using, for example, a paired  $t$ -test or the non-parametric Wilcoxon test on monthly returns of the two strategies).

Finally, we discuss the results in terms of economic significance: beyond  $p$ -values, we consider whether the improvement in Sharpe or reduction in drawdown is meaningful for an investor, given estimation and transaction costs.

In summary, our methodology chapter has presented two complementary approaches to portfolio optimization, grounded in both financial theory and machine learning innovation. We have detailed the models, their theoretical justification with citations to prior literature (e.g., HAR by ?, covariance shrinkage by ?, diversification principles by ?, deep learning models like ?, and reinforcement learning by ?), and we have outlined how we train and evaluate these models on historical data. In the next chapter, we will present the empirical findings, comparing the out-of-sample performance of Approach 1 and Approach 2, and discussing insights such as when a model-driven approach might be preferred over a direct learning approach, and vice versa, in the context of portfolio management.