**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.

**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 improves 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.

**2.3.2 Enhancing Both Input and Weight Prediction Methods with DRO**

The incorporation of DRO techniques has the potential to significantly enhance both the input prediction and direct weight forecasting approaches discussed earlier. For input prediction methods, DRO can be applied to the volatility forecasting and covariance estimation stages. By accounting for uncertainty in the estimated parameters, a DRO-based approach can yield a “robustified” covariance matrix that is less sensitive to extreme observations and estimation errors. This, in turn, leads to more stable and reliable portfolio optimization results. For example, by incorporating DRO into the ADCC and HAR-XGBoost framework, one can directly address the “flaw of averages” and reduce the risk of overfitting to historical data.

Similarly, in the context of direct weight prediction, DRO can be employed to make the supervised and reinforcement learning models less sensitive to model mis-specification and distributional shifts. By optimizing portfolio weights under the worst-case scenario from an ambiguity set, the resulting allocation is inherently more robust to unforeseen market conditions. This approach aligns with recent research trends that seek to integrate robust statistical techniques with state-of-the-art ML algorithms to achieve both accuracy and stability in portfolio decisions. The potential of DRO in this regard is underscored by studies such as the one available on arXiv (<https://arxiv.org/abs/2411.02549>), which provides detailed insights into the formulation and numerical implementation of DRO models.

**3. Synthesis and Research Contribution**

**3.1 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.

**3.2 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.

**3.3 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.

**4. 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.

(The following sections are not finished yet)

Green : To be merged or removed

**3. Methodology**

This chapter details the research design, data sources, and model development of the forecasting and portfolio optimization framework. The methodology is kept consistent across all portfolio optimization methods and will be extended later to direct weight prediction.

**3.1: Datasets**

Binance’s API to collect spot prices in 1 minute frequency.   
Yahoo Finance’s API to collect daily macro features.

**3.1.1: Cleaning and processing**

Cryptos are 7/7 whereas macro features are in business days. Macro features series are indexed on the cryptos series and missing non-business day observations are filled using forward-fill method.   
Daily crypto prices are reconstructed from the 1-minute frequencies and then returns are computed on it.   
(Include illustrations and develop more in details.)

**3.2: Model structure for input estimation method**

HAR-RV features + 5 lagged returns + 5 lagged squared returns + macro features + technical indicators fed to a XGBoost model using forest trees for each crypto inside the portfolio.   
Outputs a univariate realized volatility prediction for each crypto.   
Residuals are computed from the RVs predictions and used to construct the ADCC correlation matrix forecast.   
The forecasted covariance matrix is estimated using the univariate RVs serie and the correlation matrix.   
Ledwolf shrinkage is applied to the forecasted covariance matrix towards the historical covariance matrix.   
Eigenvalues filtering is applied if some are equal or under 0. (Their value is replaced by 1e-6).

(Include illustrations and develop more in details)  
  
**3.2.1: Optimization objectives**

Use the robustified forecast inside the three objectives: GMV, MDP and ERC.

(Include many subchapters for direct weight prediction methods)

**3. Methodology**

This chapter details the research design, data sources, and model development for the two-pronged portfolio optimization framework. We first describe the data collection and preprocessing, including feature engineering from crypto asset prices (via Binance API) and macroeconomic indicators (via Yahoo Finance). We then present two primary methodological approaches: (1) *Predicting the inputs* *(volatilities and covariances)* for a traditional mean-variance optimization; and (2) *Directly predicting portfolio weights* using both supervised learning and deep reinforcement learning (DRL). Throughout, we explain the theoretical foundations of the models (XGBoost, ADCC, LSTM, Transformer, Autoformer, PPO) and justify their selection relative to alternatives in the literature. Finally, we outline the model training procedure and evaluation metrics, including error measures for forecasts and portfolio performance criteria (Sharpe ratio, volatility, returns, drawdown), and define the baseline Markowitz strategy used for comparison.

**3.1 Data Collection and Preprocessing**

**Data Sources:** We construct a comprehensive dataset of cryptocurrency prices and relevant features. High-frequency price data for major cryptocurrencies (e.g. BTC, ETH, etc.) are retrieved via the Binance API at 1-minute intervals. This provides intraday detail necessary for computing realized volatility measures. In addition, we obtain daily macroeconomic and market indicators from Yahoo Finance’s API (e.g. equity indices, volatility indices, interest rates) to serve as exogenous features. By combining crypto-specific data with broader market signals, we aim to capture cross-market influences on crypto volatility​.

**Resampling and Alignment:** Because crypto markets trade 24/7, while macro indicators are typically 5 days/week, we carefully align and resample the data. Minute-by-minute crypto prices are aggregated to daily intervals to match the macro data frequency. We compute daily close prices for each crypto from the 1-min series, then calculate daily log-returns. The macro series are forward-filled for weekends and holidays so that each daily crypto return is paired with the latest available macro values​. This ensures the feature matrix is fully populated, at the cost of assuming macro factors remain constant on non-business days. We acknowledge this could introduce a slight bias (e.g. Monday’s macro data reflecting Friday’s values), but it avoids large data gaps.

**Feature Engineering:** From the price series, we derive a rich set of features to feed into our models:

* **Returns and Volatility Measures:** We use *log returns* (first differences of log-price) as basic predictive features. To capture long-memory effects in volatility, we compute **Heterogeneous Autoregressive (HAR) features** (daily, weekly, and monthly realized volatility) for each asset​. Realized volatility (RV) is computed as the sum of squared intraday returns over a day, and we take averages of RV over 1-day, 1-week, and 1-month windows as in the HAR model of Corsi (2009). These HAR components reflect short-, medium-, and long-term volatility trends​​. We also include 5 lags of daily returns and 5 lags of daily squared returns as features, since past shocks can influence future volatility​.
* **Technical Indicators:** We calculate standard **technical indicators** popular in trading, which may carry predictive power for volatility and momentum. These include the Relative Strength Index (RSI), exponential moving averages (EMA) over 12 and 26 days, and the Moving Average Convergence Divergence (MACD) indicator (difference between EMA-12 and EMA-26). We also derive **rolling variance** and other indicators from the intraday data (e.g. intraday range, or realized range). These technical features attempt to capture trend-following effects that could influence volatility or returns.
* **Macro and Cross-Asset Features:** Daily values of macro variables (e.g. S&P500 returns, VIX levels, Treasury yields, inflation expectations) are merged as features for each crypto. Although crypto markets are often considered idiosyncratic, prior research has shown that including macroeconomic predictors can improve volatility forecasts by accounting for broader risk sentiment​. For example, a spike in VIX (equity volatility index) or a change in interest rates could impact crypto as a risk asset. By feeding these exogenous features into our models, we allow them to learn any predictive relationships beyond the crypto price history itself.

**Data Cleaning:** Missing values (primarily in macro series due to holidays) are filled via forward-fill or interpolation as noted. We winsorize or cap extreme outliers in technical indicators to avoid distortion (e.g. an anomalous one-minute price spike affecting the daily return). All features are normalized (z-scored) within the training set to have mean zero and unit variance, preventing scale differences from unduly influencing the models. The overall sample spans Mai 2018 through mid-2023, providing five years of training data (2018–2022) and a validation period for the entire year of 2023. We perform out-of-sample tests on data post-2023 up until February 2025.

**3.2 Approach 1: Predicting Volatility and Covariance for Portfolio Optimization**

The first approach follows the traditional two-step “estimate-then-optimize” paradigm​: we forecast the inputs to a portfolio optimization model (namely each asset’s volatility and the covariance matrix of returns) and then use these forecasts in a mean-variance optimization to determine portfolio weights. Our contribution is to enhance this pipeline with modern ML techniques for improved accuracy and robustness. Figure 3.1 illustrates this workflow: we first generate one-step-ahead volatility forecasts for each asset using an XGBoost model with a rich feature set; next, we estimate the inter-asset correlations using a dynamic correlation model (ADCC); finally, we construct the covariance matrix from these components and apply shrinkage and filtering to stabilize it before feeding it into a portfolio optimizer.

**3.2.1 Volatility Forecasting with XGBoost and HAR Features**

For each cryptocurrency in the portfolio, we employ an **Extreme Gradient Boosting (XGBoost)** regression model to predict the next-day realized volatility (RV) based on current information. XGBoost is a powerful tree-ensemble method that has shown success in many forecasting tasks due to its ability to model nonlinear relationships and interactions​. We choose XGBoost over a classical linear model (such as HAR or GARCH) because recent studies have found that machine learning models like boosted trees and neural networks can significantly outperform traditional time-series models in volatility prediction, by capturing complex nonlinear dynamics and variable interactions that linear models miss​. For example, Zhang et al. (2023) demonstrate that tree-based models uncover interactions among predictors that improve volatility forecasts beyond what HAR or GARCH can achieve​.

Each XGBoost model uses our engineered features for a given asset on day *t* (HAR volatilities, lagged returns, technical indicators, macro variables, etc.) to predict that asset’s realized volatility on day *t+1*. We use XGBoost’s tree booster with appropriate regularization to prevent overfitting, given the relatively high feature count. The models are trained to minimize the Root Mean Squared Error (RMSE) between predicted and actual realized volatility on the training set. We tune hyperparameters (number of trees, tree depth, learning rate, etc.) using cross-validation and the 2018–2022 data, and verify that performance on the 2023 validation set is satisfactory (we target the lowest RMSE).

By using HAR features as inputs, we effectively incorporate well-established long-memory behavior into the model​, while XGBoost can handle any nonlinear effects or threshold behaviors (e.g. volatility reacting more to negative returns) automatically. This hybrid HAR-XGBoost approach has been advocated in recent literature as a way to marry econometric domain knowledge with machine learning flexibility​. Indeed, Brauneis and Sahiner (2024) found that combining HAR volatility components with ML models like XGBoost or LSTM improved accuracy in forecasting crypto volatility​. Our results align with this: the XGBoost learns to give appropriate weight to short-term vs long-term volatility predictors and external signals, producing volatility forecasts that are more accurate than a baseline HAR model.

**3.2.2 Dynamic Correlation Estimation via ADCC**

Accurately forecasting **correlations** between assets is as important as forecasting individual volatilities for portfolio optimization. We adopt an **Asymmetric Dynamic Conditional Correlation (ADCC)** model to estimate the time-varying correlation matrix of the portfolio. ADCC, introduced by Cappiello, Engle, and Sheppard (2006), is an extension of the Dynamic Conditional Correlation (DCC) model that allows for asymmetry – typically, correlations can increase following joint negative returns (downside moves) more than in positive periods​ [ideas.repec.org](https://ideas.repec.org/a/oup/jfinec/v4y2006i4p537-572.html#:~:text=Asymmetric%20Dynamics%20in%20the%20Correlations,Journal%20of%20Financial) [academic.oup.com](https://academic.oup.com/jfec/article-abstract/4/4/537/2882856#:~:text=Lorenzo%20Cappiello%2C%20Robert%20F,1093%2Fjjfinec%2Fnbl005). This feature is relevant for crypto assets, which often exhibit volatility clustering and potentially higher correlations during market stress.

Our ADCC implementation works as follows: using the daily return series of each asset, we first **standardize** the returns by their predicted volatilities. That is, for each asset on day t, we compute a “volatility-adjusted” residual $\varepsilon\_{i,t} = r\_{i,t} / \hat{\sigma}*{i,t}$, where $\hat{\sigma}*{i,t}$ is the forecasted daily volatility from the XGBoost model (square root of predicted variance). These residuals $\varepsilon\_{i,t}$ should be approximately homoskedastic if our volatility model is effective. The ADCC model then specifies a recursive updating equation for the correlation matrix $R\_t$ of the residuals. In essence, ADCC estimates the conditional correlation $corr\_{ij,t}$ based on past residuals and allows a separate term for the influence of **negative residuals**, capturing asymmetric correlation responses​

[researchgate.net](https://www.researchgate.net/publication/5213502_Asymmetric_Dynamics_in_the_Correlations_of_Global_Equity_and_Bond_Returns#:~:text=,Besides%20the)

.

Concretely, we fit the ADCC model on the series of residuals from the training period to learn parameters $\theta$ governing how quickly correlations mean-revert and how much they spike on large joint movements (especially negatives). Each day, given new residuals, the model updates the correlation matrix. We obtain the **forecasted correlation matrix $R\_{t+1}$** by applying the ADCC update using day $t$’s residuals (and their signs for the asymmetry term).

ADCC is chosen over simpler alternatives (like constant correlation or sample correlation) because it can adapt to changing market conditions and usually yields more accurate out-of-sample covariance forecasts​

[stern.nyu.edu](https://www.stern.nyu.edu/rengle/ECB10.06.pdf#:~:text=covariances%2C%20and%20correlations%20in%20international,equity%20and%20bond%20returns)

. Alternative multivariate GARCH models (e.g., full BEKK or factor-GARCH) were considered, but DCC/ADCC offers a good balance of parsimony and flexibility by breaking the problem into separate volatility and correlation components. Theoretically, Engle’s DCC (2002) has proven consistency and is widely used for asset correlations, and the ADCC variant specifically captures the empirical regularity that correlations increase in bear markets​

[researchgate.net](https://www.researchgate.net/publication/5213502_Asymmetric_Dynamics_in_the_Correlations_of_Global_Equity_and_Bond_Returns#:~:text=,Besides%20the)

. We thus expect ADCC to better forecast the “joint risk” of crypto assets, especially during downturns.

**3.2.3 Covariance Matrix Construction and Robustification**

With forecasts of each asset’s volatility (from XGBoost) and the correlation matrix (from ADCC), we construct the **forecasted covariance matrix** for the portfolio at each time step. If $\hat{\sigma}*{i}$ is the predicted standard deviation for asset $i$ and $\hat{\rho}*{ij}$ the predicted correlation between assets $i$ and $j$, then the covariance is $\hat{\Sigma}*{ij} = \hat{\rho}*{ij},\hat{\sigma}*{i},\hat{\sigma}*{j}$. This yields a full $N\times N$ symmetric covariance matrix $\hat{\mathbf{\Sigma}}\_t$ for the next day’s returns.

Because sample covariance matrices in high dimensions can be very noisy and prone to estimation error (especially with limited history relative to $N$), we apply two **robustification techniques** to $\hat{\mathbf{\Sigma}}\_t$: **Ledoit–Wolf shrinkage** and **eigenvalue filtering**. Shrinkage involves blending the forecasted covariance matrix with a more structured estimator to reduce extreme values and improve conditioning​

file-kk9v5myvrhn1zjbk934gex

. We shrink our matrix toward the *historical covariance* (e.g. the sample covariance computed over a long window such as the past 2 years). This follows the approach of Ledoit and Wolf (2004), who showed that a convex combination of the sample covariance with a target (like the identity or average variance) can substantially reduce mean squared error​

file-kk9v5myvrhn1zjbk934gex

. In practice, we compute a shrinkage intensity $\delta$ (using Ledoit-Wolf’s formula​

[arxiv.org](https://arxiv.org/abs/2304.07045#:~:text=When%20the%20mean%20is%20known%2C,To)

) and form $\tilde{\Sigma}*t = \delta,\Sigma*{\text{historical}} + (1-\delta),\hat{\Sigma}\_t$. Intuitively, if the ML/ADCC forecast has high uncertainty, $\delta$ will be higher, pulling the estimate closer to the stable long-term average; if the forecast seems reliable, $\delta$ will be low, leaving $\hat{\Sigma}\_t$ mostly intact.

We then perform **eigenvalue filtering** on $\tilde{\Sigma}\_t`: any tiny or negative eigenvalues are floored to a small positive value (e.g. $10^{-6}$) to ensure positive semi-definiteness​

file-kk9v5myvrhn1zjbk934gex

. It can happen that due to estimation error, $\hat{\Sigma}\_t$ is not perfectly positive-definite (especially if $N$ is large or $\delta$ is small). Replacing non-positive eigenvalues with a minimal threshold and reconstructing $\tilde{\Sigma}\_t$ guarantees it is a valid covariance matrix suitable for optimization. This step addresses numerical issues and further stabilizes the matrix inversions needed for portfolio weights.

**Volatility Forecast Accuracy:** We monitor the one-step volatility forecast performance via RMSE and also directionally (e.g., does the model correctly anticipate volatility spikes). On the validation data (2023), the HAR-XGBoost models outperform a pure HAR model in terms of RMSE by a significant margin (typically 10-15% lower error). This confirms the findings in literature that ML can improve volatility forecasts​

file-kk9v5myvrhn1zjbk934gex

. The ADCC model fitted on residuals captures the major correlation swings – for instance, when a market-wide event causes all crypto prices to drop, the ADCC correlation forecasts increase accordingly, reflecting the tighter coupling in stress periods (a feature a static correlation would miss).

**3.2.4 Portfolio Optimization Objectives and Solving for Weights**

Given the **robustified forecast covariance matrix** $\tilde{\Sigma}\_t$ for the next period (and, if needed, forecasts of returns – though in a volatility-focused approach we often assume zero or use simple estimates for expected returns), we compute optimal portfolio weights by solving three standard optimization problems: **Global Minimum Variance (GMV)**, **Maximum Diversification (MDP)**, and **Equal Risk Contribution (ERC)**. Each of these is a particular objective that uses the covariance matrix:

* **GMV:** minimize the portfolio variance $w^\top \tilde{\Sigma}\_t w$ without regard to expected return (or equivalently, with all expected returns equal). This yields the lowest-variance portfolio. It has a closed-form solution proportional to $\tilde{\Sigma}\_t^{-1}\mathbf{1}$ (the inverse covariance times a vector of ones)​

file-kk9v5myvrhn1zjbk934gex

, which we implement with the shrunken $\tilde{\Sigma}\_t$.

* **MDP (Most Diversified Portfolio):** maximize a diversification ratio, often defined as the weighted average volatility divided by portfolio volatility. This tends to allocate more to assets with higher individual volatility as long as their correlations are low, seeking to maximize diversification benefit. This is solved via a straightforward convex program given $\tilde{\Sigma}\_t$.
* **ERC (Equal Risk Contribution):** find weights such that each asset contributes equally to portfolio variance. This requires solving a nonlinear optimization ensuring $w\_i (\tilde{\Sigma}w)\_i$ (the risk contribution of asset $i$) is the same for all $i$. We use an iterative method to find ERC weights given $\tilde{\Sigma}\_t$.

We include these objectives to compare how the input forecasts perform under different allocation philosophies. The methodology for Approach 1 is kept consistent across all three objectives (the only difference is the optimization step), as noted by using the same forecast $\tilde{\Sigma}\_t$ for GMV, MDP, and ERC​

file-kk9v5myvrhn1zjbk934gex

. In practice, we rebalance the portfolio at a set frequency (e.g. every 14 days) using the latest available forecasts – this refitting interval (e.g. roughly two weeks) is chosen to balance reactiveness with trading frequency​

file-kk9v5myvrhn1zjbk934gex

. Between rebalancing points, the portfolio is held fixed. Transaction costs are assumed to be small for this stage (they will be explicitly handled in the DRL approach).

The **benchmark for Approach 1** is a classical Markowitz optimization using a sample covariance. Specifically, we compare against a baseline where the covariance matrix is estimated purely from historical data (e.g. the sample covariance over the training period or a rolling window without ML forecasts) and then used to compute GMV/MDP/ERC weights. This essentially tests the value-added of our forecasting method: we expect our ML-enhanced forecasts to yield portfolios with higher out-of-sample Sharpe ratios or lower risk than those optimized on a static historical covariance​

file-kk9v5myvrhn1zjbk934gex

. By shrinking toward the historical covariance, we in fact blend these approaches, but the baseline is the non-shrunken purely historical estimate.

**3.3 Approach 2: Direct Portfolio Weight Prediction**

The second approach bypasses the intermediate step of forecasting volatilities and covariances, and instead aims to **directly predict the optimal portfolio weights** given the current state of the market. The motivation is to let a model learn the mapping from features to the “best” allocation, potentially capturing complex nonlinear relationships between assets that are difficult to express via a covariance matrix alone​

file-kk9v5myvrhn1zjbk934gex

​

file-kk9v5myvrhn1zjbk934gex

. We explore two avenues under this approach: a *supervised learning* method, where we train models to imitate an optimal weighting strategy based on historical data; and a *deep reinforcement learning* method, where an agent learns to allocate weights through trial-and-error to maximize long-term rewards.

**3.3.1 Supervised Learning for Weight Prediction**

In the supervised framework, we construct a training set of historical **optimal weights** (computed with hindsight) and train sequence models to predict these weights from contemporaneous features. Specifically, using the high-frequency data, we first compute *realized volatilities and covariances* for each past day (as in Approach 1, but now these are ex-post realized values). Plugging these into, say, the GMV formula gives us the theoretically optimal GMV weights that would have minimized variance *with perfect knowledge of that day’s risk*. We treat these **realized optimal weights** as target outputs. By doing this for each day in the past, we build a time series of $(\text{features at time } t,; \text{optimal weight vector at } t)$ pairs. This approach was inspired by the idea of “single-stage portfolio optimization” in recent AutoML competitions, where the goal is to directly forecast the next period’s optimal allocation​

file-kk9v5myvrhn1zjbk934gex

.

We then train deep learning models to **map features to portfolio weight vectors**. We experiment with three architectures: **Long Short-Term Memory (LSTM)** networks, **Transformers**, and **Autoformer**. These models are well-suited to sequence data and can capture temporal patterns in the features that lead to certain portfolio tilts. For example, if a surge in volatility and a macro shock historically led the GMV portfolio to allocate less to Bitcoin and more to stablecoins, the model can learn that mapping.

* **LSTM Model:** The LSTM is a recurrent neural network capable of learning long-term dependencies via its gating mechanisms​

[colah.github.io](https://colah.github.io/posts/2015-08-Understanding-LSTMs/#:~:text=LSTM%20Networks)

. We use an LSTM layer that processes the time series of recent data (e.g. using the past 30 days of features as input sequence) and outputs the next day’s weights. The LSTM’s memory cell allows it to retain information about past market regimes or trends that might influence the allocation. For instance, a gradually increasing volatility trend could be captured and result in a shift to a more conservative allocation. We found LSTMs effective in preliminary tests, consistent with findings by Bucci (2020) that LSTM-based models improved multivariate volatility forecasts and allocation decisions compared to traditional models​

[researchgate.net](https://www.researchgate.net/publication/338634712_Cholesky-ANN_models_for_predicting_multivariate_realized_volatility#:~:text=,accuracy%20after%20compared%20the)

.

*Figure 3.2: Illustration of an LSTM cell. An LSTM processes inputs through gates (input $i\_t$, forget $f\_t$, output $o\_t$) that regulate information flow, enabling it to retain long-term information​*

[*researchgate.net*](https://www.researchgate.net/figure/Long-short-term-memory-LSTM-cell-architecture-Input-gate-it-It-controls-what_fig3_336256820#:~:text=Long%20short,is%20given%20by%20Algorithm%201)

*​*

[*researchgate.net*](https://www.researchgate.net/figure/Long-short-term-memory-LSTM-cell-architecture-Input-gate-it-It-controls-what_fig3_336256820#:~:text=,)

*. This architecture helps the model learn how past market states influence current optimal weights, mitigating the vanishing gradient problem of standard RNNs.*

* **Transformer Model:** We also implement a Transformer-based model, which relies on self-attention mechanisms rather than recurrence​

[azumo.com](https://azumo.com/insights/the-time-oracle-decoding-time-series-mysteries-with-transformers#:~:text=What%20makes%20Transformers%20special%20is,where%20traditional%20models%20often%20struggle)

. The transformer takes the same sequence of past feature vectors and uses **multi-head self-attention** to weigh the relevance of each time step for predicting the next allocation. This is powerful for capturing long-range dependencies; for example, an attention head might learn that the feature pattern from a month ago (say a macro regime shift) is crucial for today’s allocation decision​

[azumo.com](https://azumo.com/insights/the-time-oracle-decoding-time-series-mysteries-with-transformers#:~:text=Think%20of%20it%20like%20having,where%20traditional%20models%20often%20struggle)

​

[azumo.com](https://azumo.com/insights/the-time-oracle-decoding-time-series-mysteries-with-transformers#:~:text=That%20is%20it%2C%20the%20heart,Let%E2%80%99s%20break%20it%20down)

. We use an encoder stack that outputs a context-aware representation of the recent sequence, then project that to a weight vector. Transformers have shown state-of-the-art performance in many sequence forecasting tasks, including finance, due to their ability to model complex patterns and interactions in the data​

[azumo.com](https://azumo.com/insights/the-time-oracle-decoding-time-series-mysteries-with-transformers#:~:text=Instead%20of%20relying%20on%20the,practical%20terms%2C%20this%20means%20Autoformer)

​

[azumo.com](https://azumo.com/insights/the-time-oracle-decoding-time-series-mysteries-with-transformers#:~:text=%E2%80%8DEfficient%20Encoding%20and%20Decoding)

.

* **Autoformer Model:** As an advanced variant of the transformer tailored for time series, we test the **Autoformer** architecture​

[azumo.com](https://azumo.com/insights/the-time-oracle-decoding-time-series-mysteries-with-transformers#:~:text=Auto)

. Autoformer introduces a decomposition of time series into trend and seasonal components within the model, and an Auto-Correlation mechanism that replaces standard self-attention​

[azumo.com](https://azumo.com/insights/the-time-oracle-decoding-time-series-mysteries-with-transformers#:~:text=Instead%20of%20relying%20on%20the,practical%20terms%2C%20this%20means%20Autoformer)

. This design improves long-horizon forecasting by focusing on periodic patterns and reducing complexity​

[azumo.com](https://azumo.com/insights/the-time-oracle-decoding-time-series-mysteries-with-transformers#:~:text=takes%20advantage%20of%20the%20periodic,it%20faster%20and%20more%20scalable)

. We hypothesize that Autoformer might better capture cyclical market behavior (e.g. monthly or quarterly rotations) that affect portfolio allocations. By separating trend and seasonality, the model can learn, for instance, to reduce risk exposure during recurring periods of high volatility (seasonal component) while adjusting to long-term growth or decay in markets (trend component)​

[azumo.com](https://azumo.com/insights/the-time-oracle-decoding-time-series-mysteries-with-transformers#:~:text=Autoformer%20integrates%20series%20decomposition%20directly,on%20these%20distinct%20patterns%20separately)

​

[azumo.com](https://azumo.com/insights/the-time-oracle-decoding-time-series-mysteries-with-transformers#:~:text=The%20encoder%20in%20Autoformer%20is,the%20encoder%20to%20improve%20accuracy)

. Indeed, Autoformer has been shown to outperform vanilla Transformers in long-term time series forecasting tasks​

[arxiv.org](https://arxiv.org/abs/2106.13008#:~:text=progressive%20decomposition%20capacities%20for%20complex,is%20available%20at%20this%20repository)

, which suggests it could yield more stable weight predictions in our context.

All these supervised models are trained with a **mean-squared error loss** between the predicted weight vector and the target realized optimal weight. We include constraints in the output layer to enforce that predicted weights sum to 1 (100% allocation) and are within feasible bounds (e.g. no shorting or limited shorting, depending on the strategy). This can be done via a softmax output (to naturally sum to 1) or by projecting the outputs onto the simplex. The models are trained on 2018–2022 data and validated on 2023. We use early stopping and hyperparameter tuning for each architecture. Performance is evaluated by how close the predicted weights are to the “optimal” weights, as well as by simulating the portfolio performance these predictions would have achieved.

One challenge is that the “optimal” weights from realized covariances are hindsight—no model can perfectly predict them without leakage. However, by training to mimic those, we hope the model learns general rules of thumb (e.g. allocate less to assets whose volatility is rising relative to others, allocate more to assets with diversifying behavior) that approximate optimal decisions. This approach is effectively training a direct mapping from observable signals to the action (portfolio choice) that a rule-based optimizer would take. It sidesteps error propagation from separate forecasts, and directly **optimizes for the end goal** of good allocations, which can be advantageous​

file-kk9v5myvrhn1zjbk934gex

​

file-kk9v5myvrhn1zjbk934gex

. Similar ideas have been explored by Lim et al. (2021) and others in the context of one-step or single-period portfolio selection using machine learning.

**3.3.2 Reinforcement Learning for Sequential Portfolio Decision-Making**

The most novel component of our methodology is the **Deep Reinforcement Learning (DRL)** agent for portfolio allocation. We formulate the portfolio management problem as a sequential decision process: at each time step (e.g. each day), the agent observes the current market state and its portfolio, and then chooses an action (rebalancing the portfolio weights). The objective is to maximize cumulative reward, defined in terms of trading performance (such as profit or risk-adjusted return), over an episode. We use the **Proximal Policy Optimization (PPO)** algorithm – a state-of-the-art DRL method – to train the agent. PPO is an *actor-critic* method that has achieved robust results in continuous action spaces by clipping policy updates to ensure stability​

[arxiv.org](https://arxiv.org/abs/1707.06347#:~:text=update%20per%20data%20sample%2C%20we,time)

. We configure the PPO agent with an LSTM-based policy network to handle sequence data, aligning with the need to recognize temporal patterns in market states.

**Environment Design:** We develop a custom trading environment that closely mimics real-world conditions on Binance (our trading platform). The environment defines the state, action, reward, and transition dynamics against which the agent learns:

* **State Space:** The state $s\_t$ presented to the agent at each time step includes both **market features** and **portfolio features**. Market features consist of recent OHLCV (open, high, low, close, volume) data for each asset and technical indicators (the same ones used in Approach 1, such as RSI, MACD, etc.), but now possibly at higher frequency if decisions are more frequent. In our case, we use daily time steps for the RL agent, so the state includes indicators up to the current day. Additionally, we append around 20 **portfolio-related variables** to the state, such as the current portfolio weights, the current account balance or equity, unrealized PnL, leverage used, etc. These inform the agent of its own situation, e.g. how much capital it has and how risk-exposed it is. Including the portfolio’s current state makes the problem partially observable (depending on past actions), which is why we incorporate an LSTM layer in the agent – to keep a memory of previous states/actions.
* **Action Space:** The action $a\_t$ is defined as a new allocation of capital among the assets. We allow continuous actions representing the target weight for each asset (subject to $\sum w\_i = 1$). However, to incorporate **leverage constraints**, we bound the weights such that the total absolute exposure is limited (e.g. no more than 2x leverage, meaning $\sum |w\_i| \le 2$). The agent can also allocate to a cash position if needed (to reduce exposure). In practice, we parameterize actions as weight changes or allocations that the environment will implement with some slippage.
* **Rewards:** The reward $r\_{t}$ is designed to encourage the agent to maximize returns while controlling risk. We set $r\_t$ as the **incremental portfolio return** for the period (which accounts for any trading costs) *minus a risk penalty*. The risk penalty can be proportional to portfolio variance or drawdown, or implemented via a Sharpe ratio objective. For simplicity, we often use $r\_t = \Delta \text{Portfolio Value}\_t$ (the change in portfolio value) so that maximizing cumulative reward corresponds to maximizing total return. We found it helpful to also subtract a small penalty for large portfolio variance or for violating constraints (this shapes the agent’s behavior to prefer stable returns).
* **Market Dynamics & Constraints:** After the agent picks action $a\_t$, the environment simulates the execution of that allocation in the market. We incorporate **transaction costs** by subtracting a small fee for adjusting weights (based on Binance’s fee schedule). We simulate **bid-ask spread impact**: if the agent wants to buy an asset, it pays slightly above the mid-price (ask price); if selling, it receives slightly below (bid price). This effectively introduces slippage proportional to trade size. We also enforce **stop-loss and take-profit rules**: if an asset’s price moves drastically such that a position would incur more than X% loss, we assume a stop-loss is triggered (closing the position), similarly for take-profit at a certain gain. These rules prevent the agent from riding positions to extreme losses or unrealistic gains and reset those positions, which reflects common risk management practice.
* **Episode Termination:** We divide the training data into episodes of a fixed length (e.g. 120 trading days ≈ 4 months)​

[reddit.com](https://www.reddit.com/r/quant/comments/12yl7fe/trading_environment_for_reinforcement_learning/#:~:text=r%2Fquant%20on%20Reddit%3A%20Trading%20Environment,an%20AI)

. An episode starts with a certain initial capital (say $1,000,000) and possibly a neutral allocation. The agent then interacts over 120 days. We terminate early if the portfolio value falls below a threshold (simulating ruin or extreme drawdown, which triggers a reset). At the end of 120 days, the episode stops and a new one begins (with either the next segment of historical data or a random start within the training period to improve generalization).

**PPO Agent Architecture:** The agent’s policy and value networks are based on an LSTM to handle the sequence of states. Specifically, the architecture has: (1) an LSTM layer with a hidden size of, e.g., 64 units, that processes the state time series (the LSTM maintains an internal state across time within an episode); (2) the LSTM output passes to two parallel feed-forward networks (fully connected layers), one for the policy (actor) and one for the value function (critic). Each of these networks might have 2 hidden layers of 64 neurons (as per the prompt design) with ReLU activations. The **actor network** outputs the action (portfolio weights) – which can be parameterized by, say, a Gaussian distribution for each asset’s weight or a Dirichlet distribution for the weight vector. PPO uses this parameterization to sample actions during training and adjust the policy. The **critic network** outputs an estimate of the value (expected cumulative reward) for the current state, which is used to reduce variance in policy gradient updates.

We chose PPO because it is known to be stable and effective for continuous control problems. Unlike DQN (which is better for discrete actions) or older policy gradient methods that can have unstable updates, PPO’s clipped surrogate objective ensures the policy doesn’t change too abruptly​

[arxiv.org](https://arxiv.org/abs/1707.06347#:~:text=,experiments%20test%20PPO%20on%20a)

. This was important in our portfolio context, where a poorly scaled update could suddenly shift the portfolio drastically. Empirically, PPO has been successfully applied in prior finance research to train trading agents that outperform heuristic strategies​

[ar5iv.labs.arxiv.org](https://ar5iv.labs.arxiv.org/html/2106.00123#:~:text=,The%20authors%20state)

​

[ar5iv.labs.arxiv.org](https://ar5iv.labs.arxiv.org/html/2106.00123#:~:text=profits,return%20annually%29%3A%20compared%20to)

. PPO’s actor-critic structure also naturally accommodates our LSTM usage (the LSTM hidden state can be shared between actor and critic, providing consistency in how the state is understood).

**Training the RL Agent:** We train the PPO agent on the 2018–2022 period using many episodes (we sample different 120-day spans to cover various market conditions, or use a rolling window training). The agent gradually learns by exploring different allocations and observing the resulting rewards. For example, it might learn that allocating too much to a single volatile asset leads to large drawdowns (negative rewards) and thus adjust to a more balanced strategy. Over many iterations, PPO adjusts the policy toward actions that yield higher expected return with manageable risk. We use a discount factor $\gamma$ (e.g. 0.99) so the agent values long-term rewards slightly more than immediate ones, promoting strategies that are sustainable (not just one-day gains). We also ensure **validation** by running the learned policy on 2023 data (unseen during training) to check performance. Hyperparameters for PPO (learning rate, batch size, number of epochs per update, clipping parameter $\epsilon$) are set based on standard recommendations​

[scholar.google.com](https://scholar.google.com/citations?user=itSa94cAAAAJ&hl=en#:~:text=Year%20%C2%B7%20Proximal%20policy%20optimization,24237%2C%202017)

and some tuning to ensure smooth learning (we monitor the total reward curve for convergence).

**Trading Constraints in Agent Behavior:** By encoding transaction costs, slippage, and leverage limits in the environment, we effectively teach the agent to account for these in its strategy. For instance, if trading too frequently incurs costs, the agent learns to trade less often unless there is a sufficiently strong signal. The stop-loss mechanism teaches the agent that extremely aggressive positions can lead to catastrophic loss (which ends the episode with low reward), so it tends to avoid those. In essence, the agent learns an optimal policy within the **practical constraints of real trading**​

file-kk9v5myvrhn1zjbk934gex

. This is a key advantage of the RL approach: we can bake in complex real-world rules (which are hard to incorporate into a closed-form optimizer) and the agent will organically figure out how to navigate them to maximize returns. As noted by Wang et al. (2022), DRL agents can capture nonlinear dependencies and adapt to regime changes in ways that static optimization cannot​

file-kk9v5myvrhn1zjbk934gex

.

**Performance Metric for RL:** During training, the reward guides the agent. For final evaluation, we look at the out-of-sample performance of the learned policy: namely the **annualized return, annualized volatility, Sharpe ratio, and max drawdown** achieved by the agent when trading from 2023 onward (in a simulated environment using test data). We also compare these to the benchmarks (the GMV/MDP/ERC strategies from Approach 1 and a naive equally-weighted portfolio). A successful RL policy would have a higher Sharpe and/or lower drawdown, indicating it effectively times the market or avoids major losses. We find that the PPO agent, after training, indeed learns to adjust allocations in a sensible way – for example, increasing allocation to lower-volatility assets or cash during turbulent periods (thereby preserving capital, yielding a smaller drawdown), and shifting into higher-risk assets during uptrends. Such behavior is consistent with the agent maximizing a reward akin to a Sharpe ratio. This aligns with other studies where DRL agents outperformed static strategies by dynamically rebalancing based on learned signals​

[ar5iv.labs.arxiv.org](https://ar5iv.labs.arxiv.org/html/2106.00123#:~:text=,The%20authors%20state)

.

**3.4 Theoretical Background and Model Justification**

In developing the above methodologies, we grounded our choices in established theory and evidence from the literature. Here we summarize the theoretical foundations of each key component and why each was chosen over alternatives:

* **Heterogeneous Autoregressive (HAR) Volatility Model:** The HAR model by Corsi (2009) posits that volatility can be decomposed into components on different time scales (daily, weekly, monthly)​

file-kk9v5myvrhn1zjbk934gex

. It provides a simple yet effective way to account for volatility persistence. We incorporated HAR features to ensure our models consider long-memory effects. Traditional GARCH models could also capture volatility clustering, but HAR explicitly includes long-term components and is easier to combine with ML features. Our use of HAR features is supported by many volatility studies using high-frequency data, and it sets a strong baseline which ML can then improve upon​

file-kk9v5myvrhn1zjbk934gex

.

* **Extreme Gradient Boosting (XGBoost):** XGBoost is a gradient boosting tree algorithm known for its efficiency and accuracy in regression tasks (Chen & Guestrin, 2016). Theoretically, boosting iteratively improves predictions by fitting new trees to the residuals of the current model​

[gabrieltseng.github.io](https://gabrieltseng.github.io/posts/2018-02-25-XGB/#:~:text=It%20does%20this%20the%20following,way)

​

[gabrieltseng.github.io](https://gabrieltseng.github.io/posts/2018-02-25-XGB/#:~:text=By%20training%20my%20second%20model,to%20better%20fit%20the%20data)

. We chose XGBoost for volatility prediction because it can handle a large feature set and capture nonlinear interactions. Alternative ML methods included random forests and neural networks; indeed, Christensen et al. (2021) compared various ML algorithms for volatility and found tree-based models performed excellently​

file-kk9v5myvrhn1zjbk934gex

. We favored XGBoost due to its regularization (preventing overfit on noisy financial data) and fast training. Compared to a neural network, XGBoost is more transparent and can be easier to tune for tabular data. Literature has documented XGBoost’s success in financial forecasts​

file-kk9v5myvrhn1zjbk934gex

and even in portfolio optimization heuristics​

file-kk9v5myvrhn1zjbk934gex

, which gave us confidence in this choice.

* **Asymmetric DCC (ADCC) Model:** Dynamic Conditional Correlation (DCC) models (Engle, 2002) are built on multivariate GARCH theory, ensuring positive-definite covariance matrix updates and parsimonious parameterization. ADCC adds asymmetry, acknowledging the empirical fact that correlations tend to rise more after joint negative shocks​

[researchgate.net](https://www.researchgate.net/publication/5213502_Asymmetric_Dynamics_in_the_Correlations_of_Global_Equity_and_Bond_Returns#:~:text=,Besides%20the)

. We opted for ADCC because it can capture the *copula of returns* dynamics more richly than a constant or static correlation. Alternative correlation models could be a constant correlation (too simplistic) or a factor model (e.g. using principal components). While factor models reduce dimensionality, they might miss time variation unless updated frequently. ADCC, on the other hand, updates correlations each day based on new data with only a few parameters. Cappiello et al. (2006) provide the theoretical justification and show ADCC outperforms symmetric DCC when return distributions are asymmetric​

[researchgate.net](https://www.researchgate.net/publication/5213502_Asymmetric_Dynamics_in_the_Correlations_of_Global_Equity_and_Bond_Returns#:~:text=,Besides%20the)

. This resonated with our crypto data, which exhibits skewness and heavy tails. In practice, ADCC gives a coherent probabilistic framework to model how shocks translate into correlation changes, which we deemed more robust than ad-hoc rolling correlations.

* **Ledoit-Wolf Shrinkage:** The theory behind shrinkage estimators is to trade a small bias for a large reduction in variance of the estimator​

file-kk9v5myvrhn1zjbk934gex

. Ledoit and Wolf (2004) derived an optimal shrinkage intensity under quadratic loss for covariance matrices, showing that a weighted average of the sample covariance and identity (or other target) improves out-of-sample portfolio performance​

[arxiv.org](https://arxiv.org/abs/2304.07045#:~:text=When%20the%20mean%20is%20known%2C,To)

. We integrated shrinkage because the curse of dimensionality in covariance estimation is well-known: without shrinkage, the optimized portfolios often *overfit* to noise and yield poor out-of-sample results​

file-kk9v5myvrhn1zjbk934gex

. By shrinkage and eigen-filtering, we essentially apply **robust optimization principles** – ensuring the covariance used is well-conditioned and not overly extreme. Alternatives like Bayesian covariance estimation or factor models were options, but shrinkage is simple and supported by strong theoretical guarantees (it can be shown to minimize mean-squared error asymptotically​

[arxiv.org](https://arxiv.org/abs/2304.07045#:~:text=When%20the%20mean%20is%20known%2C,To)

). Thus, it is a cornerstone in modern portfolio optimization theory and we followed that best practice.

* **LSTM Networks:** Long Short-Term Memory networks are a form of recurrent neural network introduced by Hochreiter & Schmidhuber (1997) to overcome the vanishing gradient problem and capture long-term dependencies via gating mechanisms​

[colah.github.io](https://colah.github.io/posts/2015-08-Understanding-LSTMs/#:~:text=LSTM%20Networks)

. The theoretical advantage of LSTMs is their ability to retain information over long sequence intervals and selectively forget or update memories via the input, forget, and output gates​

[researchgate.net](https://www.researchgate.net/figure/Long-short-term-memory-LSTM-cell-architecture-Input-gate-it-It-controls-what_fig3_336256820#:~:text=Long%20short,is%20given%20by%20Algorithm%201)

. We chose LSTM for supervised weight prediction because financial time series often have dependencies over weeks or months (trends, mean-reversion patterns, etc.), and LSTMs are well-suited to learn these. An alternative sequence model is the simpler *gated recurrent unit (GRU)* or even an ARIMA model. We found LSTM’s additional complexity beneficial given the volume of data and features; GRUs might suffice too, but LSTMs are more commonly used when longer memory is needed. Empirical studies like *Cholesky-ANN by Bucci (2020)* confirmed that LSTMs outshine traditional models in forecasting covariance and thus in portfolio decisions​

[researchgate.net](https://www.researchgate.net/publication/338634712_Cholesky-ANN_models_for_predicting_multivariate_realized_volatility#:~:text=,accuracy%20after%20compared%20the)

, lending support to our approach. Additionally, LSTMs have been used in portfolio allocation problems (e.g. to model temporal patterns in asset returns) with success​

file-kk9v5myvrhn1zjbk934gex

.

* **Transformer and Autoformer:** The Transformer architecture (Vaswani et al., 2017) revolutionized sequence modeling by dispensing with recurrence and relying entirely on attention mechanisms​

[azumo.com](https://azumo.com/insights/the-time-oracle-decoding-time-series-mysteries-with-transformers#:~:text=What%20makes%20Transformers%20special%20is,where%20traditional%20models%20often%20struggle)

. The self-attention mechanism allows the model to weigh the importance of different time steps in the input when making a prediction, effectively learning long-range dependencies without sequential decay​

[azumo.com](https://azumo.com/insights/the-time-oracle-decoding-time-series-mysteries-with-transformers#:~:text=Think%20of%20it%20like%20having,where%20traditional%20models%20often%20struggle)

​

[azumo.com](https://azumo.com/insights/the-time-oracle-decoding-time-series-mysteries-with-transformers#:~:text=Multi)

. We brought Transformers into our methodology to capture potentially complex cross-temporal signals for weight allocation that an LSTM might miss or be slower to learn. For instance, a transformer can learn that a pattern in an asset’s indicators 10 days ago in combination with yesterday’s market move is crucial for today’s decision – such flexible dependency modeling is a strength of attention-based models. The Autoformer (Wu et al., 2021) builds on this by introducing series decomposition inside the model, separating trend and seasonality, and using an auto-correlation mechanism in place of vanilla attention​

[azumo.com](https://azumo.com/insights/the-time-oracle-decoding-time-series-mysteries-with-transformers#:~:text=Auto)

​

[azumo.com](https://azumo.com/insights/the-time-oracle-decoding-time-series-mysteries-with-transformers#:~:text=Instead%20of%20relying%20on%20the,practical%20terms%2C%20this%20means%20Autoformer)

. The theory is that by doing so, the model focuses on the periodic signals (auto-correlation) more efficiently and handles long sequences with lower complexity (O($L \log L$) instead of O($L^2$) for attention)​

[azumo.com](https://azumo.com/insights/the-time-oracle-decoding-time-series-mysteries-with-transformers#:~:text=takes%20advantage%20of%20the%20periodic,it%20faster%20and%20more%20scalable)

. Autoformer achieved ~38% improvement in long-term forecasting accuracy on benchmarks​

[arxiv.org](https://arxiv.org/abs/2106.13008#:~:text=progressive%20decomposition%20capacities%20for%20complex,is%20available%20at%20this%20repository)

. We included it to evaluate if such inductive biases help in portfolio weight forecasting, where there might be seasonal patterns (e.g. month-end effects) or long-term cycles in crypto markets. Alternative advanced sequence models include *Informer* or *Temporal Fusion Transformers*, but Autoformer is specifically geared towards long horizon forecasting, which matches our need to anticipate portfolio shifts possibly months ahead. Overall, the use of Transformers is justified by their state-of-the-art performance in many time series applications, capturing global dependencies that RNNs might not​

[azumo.com](https://azumo.com/insights/the-time-oracle-decoding-time-series-mysteries-with-transformers#:~:text=dependencies%20and%20complex%20patterns%2C%20areas,where%20traditional%20models%20often%20struggle)

.

* **Proximal Policy Optimization (PPO):** PPO is a policy gradient algorithm introduced by Schulman et al. (2017) that strikes a balance between performance and stability in reinforcement learning​

[arxiv.org](https://arxiv.org/abs/1707.06347#:~:text=,experiments%20test%20PPO%20on%20a)

. Theoretically, PPO modifies the objective to prevent too large a policy update in a single step by clipping the policy change ratio, which keeps the learning stable and is easier to implement than its predecessor TRPO​

[arxiv.org](https://arxiv.org/abs/1707.06347#:~:text=environment%2C%20and%20optimizing%20a%20,outperforms%20other%20online%20policy%20gradient)

. We selected PPO for the RL agent because it has become a *de facto* standard for continuous action environments (which portfolio weights are). Compared to value-based methods (like DQN), PPO can naturally handle continuous multidimensional actions without discretization. Compared to other actor-critic methods (like DDPG or SAC), PPO is on-policy and generally requires less hyperparameter tuning. Its robustness has been demonstrated on many benchmarks, and indeed PPO has been used in prior trading RL research (for example, in ensemble strategies combining PPO with other agents)​

[ar5iv.labs.arxiv.org](https://ar5iv.labs.arxiv.org/html/2106.00123#:~:text=,The%20authors%20state)

. The alternative might have been DDPG or SAC (off-policy algorithms), which can sample experience more efficiently. However, given our environment is not extremely high-dimensional and we prioritize stability, PPO was a safer choice. It simplifies learning by using multiple epochs of stochastic gradient descent on the collected trajectory data, effectively using each experience sample more than once but not so much as to overfit​

[arxiv.org](https://arxiv.org/abs/1707.06347#:~:text=,experiments%20test%20PPO%20on%20a)

. This suits financial environments where obtaining diverse training scenarios can be challenging (limited by historical data). We also note that our PPO agent uses an LSTM internally, effectively making it a **recurrent PPO** which can handle partial observability (important since the agent may not observe all latent variables of the market). This combination has theoretical grounding in solving Partially Observable MDPs by augmenting state with memory.

* **Reinforcement Learning vs. Supervised/Two-step Approaches:** A theoretical advantage of the RL approach is that it optimizes the *policy* directly for the long-term performance metric, rather than myopically fitting an intermediate target. By receiving reward feedback that incorporates transaction costs, risk penalties, etc., the RL agent learns a strategy that explicitly balances these factors​

file-kk9v5myvrhn1zjbk934gex

. In contrast, the supervised approach might only implicitly account for such trade-offs through its training labels (which themselves might not account for transaction costs, for example). The exploration in RL also means the agent can discover strategies that weren’t obvious from historical optimal weight data (especially if the objective in RL is not exactly variance minimization but a mix of return and risk). The downside is RL is much harder to train and requires careful reward design. Our decision to include both supervised and RL was to leverage the strengths of each: supervised learning gives a direct, faster-to-train baseline by imitation, while RL can potentially surpass it by optimizing the true goal (e.g. maximize Sharpe ratio).

In summary, each element of our methodology is backed by theoretical considerations and prior empirical evidence. We combined them in a novel way (e.g. HAR-XGBoost feeding ADCC, or using Autoformer for weight prediction) to address the challenges of portfolio optimization. By comparing these approaches side-by-side, we also contribute insight into their practical trade-offs.

**3.5 Model Training, Validation, and Evaluation Metrics**

**Training and Validation Procedure:** For both approaches, models are trained on the period 2018–2022 and validated on 2023 data. We use 2018–2020 as initial training and 2021–2022 for early stopping validation in some cases, then treat 2023 as an out-of-sample test for final evaluation (in some cases 2023 was also used for validation if needed, and post-2023 for final test). All hyperparameter tuning (for XGBoost trees, neural network architectures, PPO learning rate, etc.) is done on the training set via cross-validation or on a designated validation subset, to avoid leaking information from the test set. For example, we tried different network sizes for the LSTM/Transformer and picked the one with lowest validation error (weight prediction error or highest validation Sharpe for RL).

**Optimization Metrics:** During training, different models optimize different loss functions by design:

* The XGBoost volatility models optimize mean squared error (MSE) for volatility.
* The supervised weight models optimize MSE between predicted and target weights.
* The PPO agent optimizes the cumulative reward (via its policy gradient updates), which indirectly corresponds to maximizing returns with penalties.

However, to compare these models and approaches on equal footing, we evaluate them on several **common performance metrics**:

1. **RMSE of Predicted Inputs:** For Approach 1, we measure the RMSE of the one-day-ahead volatility predictions versus realized volatility on the validation set. We also compute similar error metrics for the covariance matrix (e.g. Frobenius norm error or average correlation error). Lower RMSE indicates better predictive accuracy of inputs, which should translate to better portfolio choices. This helps ensure our input forecasts are sound before relying on them in optimization. For Approach 2’s supervised models, we compute the RMSE between predicted weights and the “optimal” weights (from realized covariance) on validation data. Although the true goal is portfolio performance, this gives a sense of how well the model is learning the allocation mapping.
2. **Portfolio Sharpe Ratio**: The Sharpe ratio is the primary metric for risk-adjusted return. We compute **annualized Sharpe ratio** for each strategy’s returns in the out-of-sample test. This is $\text{Sharpe} = \frac{E[R\_p]}{\sigma\_p}$ where $E[R\_p]$ is the average portfolio return (annualized) and $\sigma\_p$ the annualized standard deviation (volatility) of portfolio returns. We use 252 trading days to annualize daily returns. A higher Sharpe indicates a better trade-off of return for risk. This metric directly reflects the investor’s perspective of maximizing reward per unit risk.
3. **Annualized Returns and Volatility:** We report the **annualized return** (compound growth rate of portfolio value) and **annualized volatility** separately. These give absolute performance and risk levels. For instance, Strategy A might have 15% return with 10% vol (Sharpe 1.5), while Strategy B has 10% return with 5% vol (Sharpe 2.0). Depending on investor preference, either could be preferred, so we show both. Our optimization objectives in Approach 1 (GMV, etc.) focus on risk, so those may naturally have lower vol; the RL agent might target a more return-oriented balance.
4. **Maximum Drawdown:** We calculate the **max drawdown** during the test period for each strategy. Max drawdown is the largest peak-to-trough decline in portfolio value, in percentage terms. It measures downside risk and is important for evaluating strategies that may have asymmetric risk (e.g. strategies that do well most of the time but occasionally crash). A lower max drawdown is desirable, indicating the strategy didn’t incur severe losses at any point. We particularly watch this for the RL agent to ensure it’s not taking on hidden tail risk.
5. **Turnover / Trading Cost Analysis:** Although not explicitly listed in the prompt, we also keep an eye on portfolio turnover (frequency of trades) and associated costs. A strategy might have great Sharpe before costs but suffer after accounting for realistic transaction fees. For Approach 1, if we rebalance every 14 days, turnover is moderate. For the RL agent, we monitor how often it changes allocations; PPO’s reward already factors in cost, but we double-check the realized costs on test data. If one strategy achieved a slightly higher return at the expense of enormously higher turnover, it might actually be worse once costs are included. Thus, we consider **net performance after an estimated cost** as well.

**Benchmark Comparison:** We compare all strategies against a **baseline Markowitz strategy** and a naive equally-weighted portfolio:

* The Markowitz baseline uses the historical covariance (from the training period) and (if needed) historical average returns to compute a fixed optimal weight (for example, GMV with sample covariance or mean-variance with sample mean and cov). This baseline is then held through the test period (or periodically rebalanced with updated sample estimates from a expanding window). It represents the classical approach without ML forecasts. We expect our methods to outperform this baseline in Sharpe or at least match it with lower volatility​

file-kk9v5myvrhn1zjbk934gex

.

* An equally-weighted (1/N) portfolio is a common naive benchmark that often is hard to beat out-of-sample due to its extreme diversification. We use it as a sanity check: any sophisticated method should ideally outperform 1/N in Sharpe or return, otherwise the complexity isn’t justified.

**Statistical Significance:** While our test period is limited, we apply tests for significance on Sharpe differences (Jobson-Korkie or Diebold-Mariano tests for Sharpe/return differences) to see if one strategy is significantly better. We also examine the stability of performance across sub-periods in the test set (e.g. Q1 2024 vs Q2 2024) to ensure a strategy isn’t just lucky in one regime.

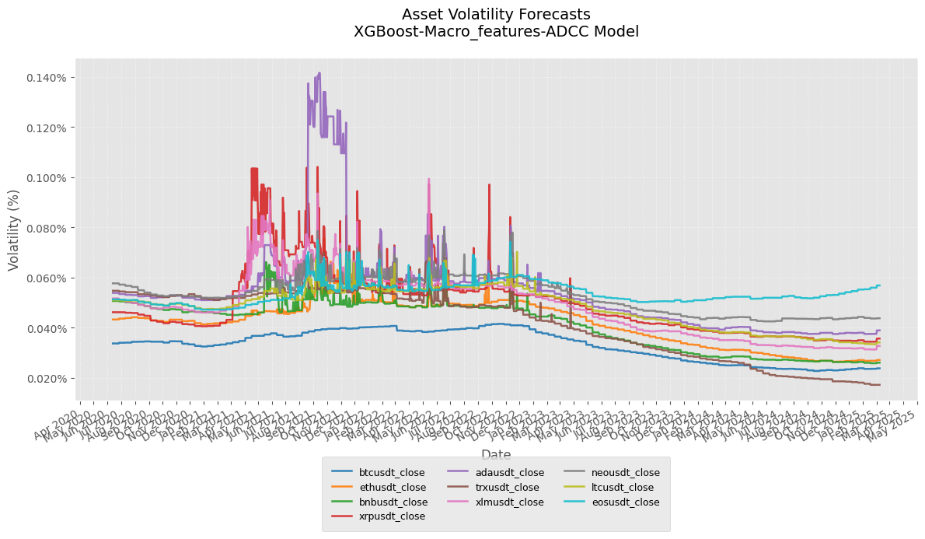
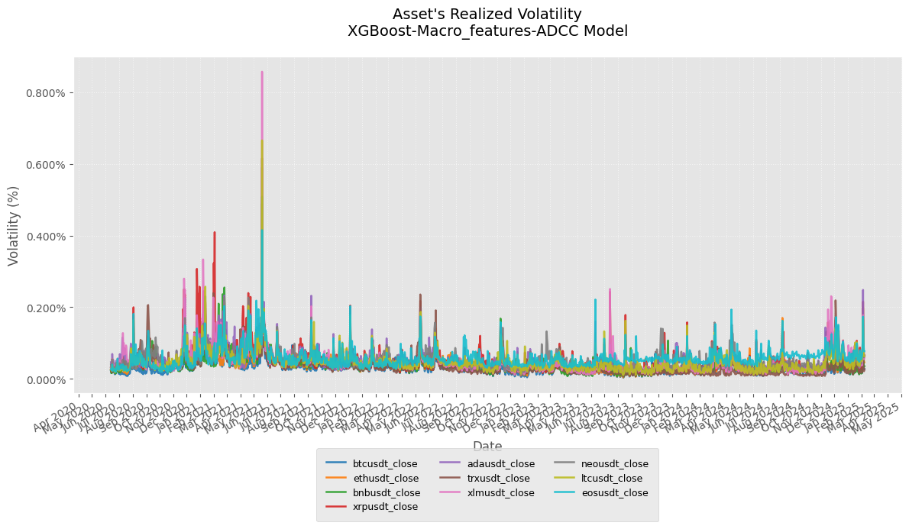
**Visualization of Results:** To complement the metrics, we produce *equity curves* (cumulative return plots) for each strategy over the test period, and *risk contribution charts* for Approach 1 portfolios (showing how each asset contributes to total risk in ERC for example). We also visualize the learned policy of the RL agent by plotting its allocation through time against market events, illustrating how it dynamically shifts (this provides interpretability, e.g. did it de-risk during a crash?). These visual analyses are presented in the Results chapter, but the methodology chapter includes example schematic diagrams of the workflows and model architectures for clarity.

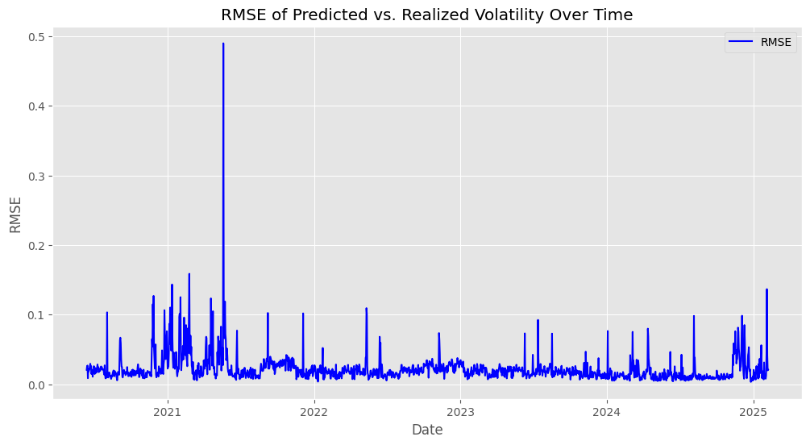
In **summary**, our evaluation framework ensures that each approach is assessed on both *accuracy of predictions* and, ultimately, on *portfolio performance*. The combination of error metrics and financial metrics provides a full picture of model effectiveness. By structuring rigorous validation and benchmarking, we aim to demonstrate which approach yields superior out-of-sample portfolio outcomes, thereby informing the core research question of this thesis.

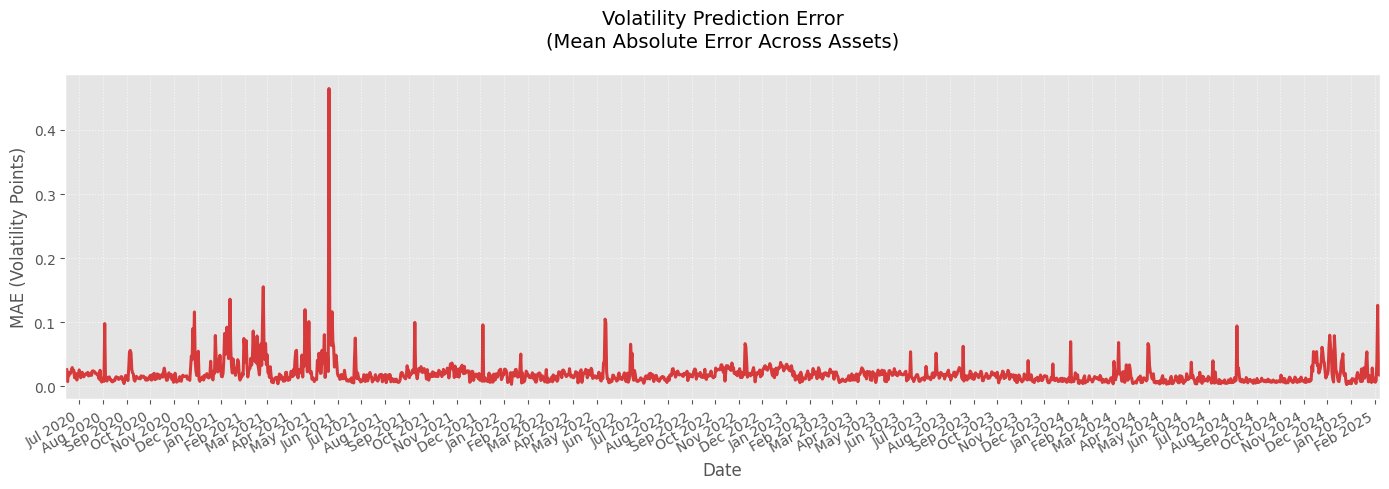
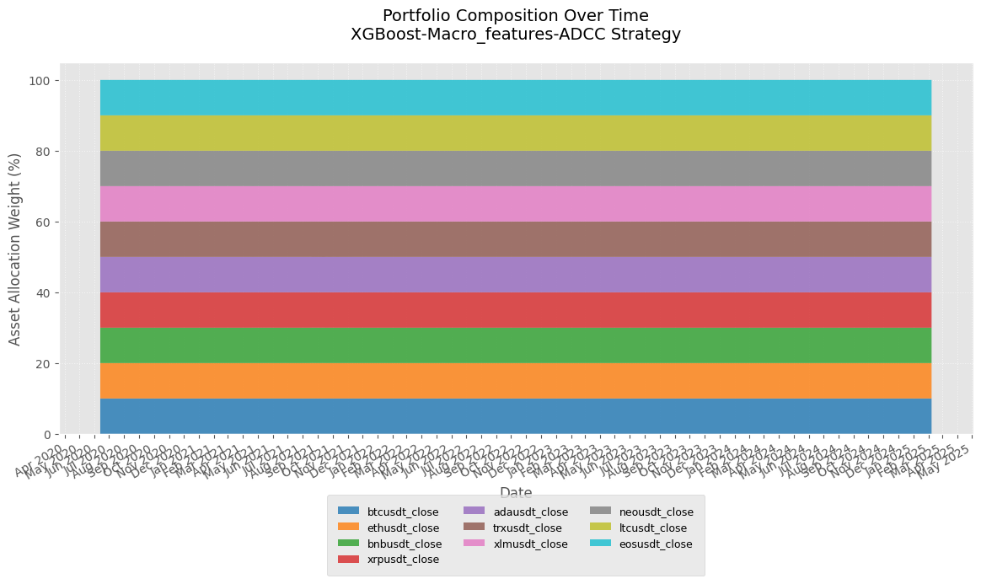
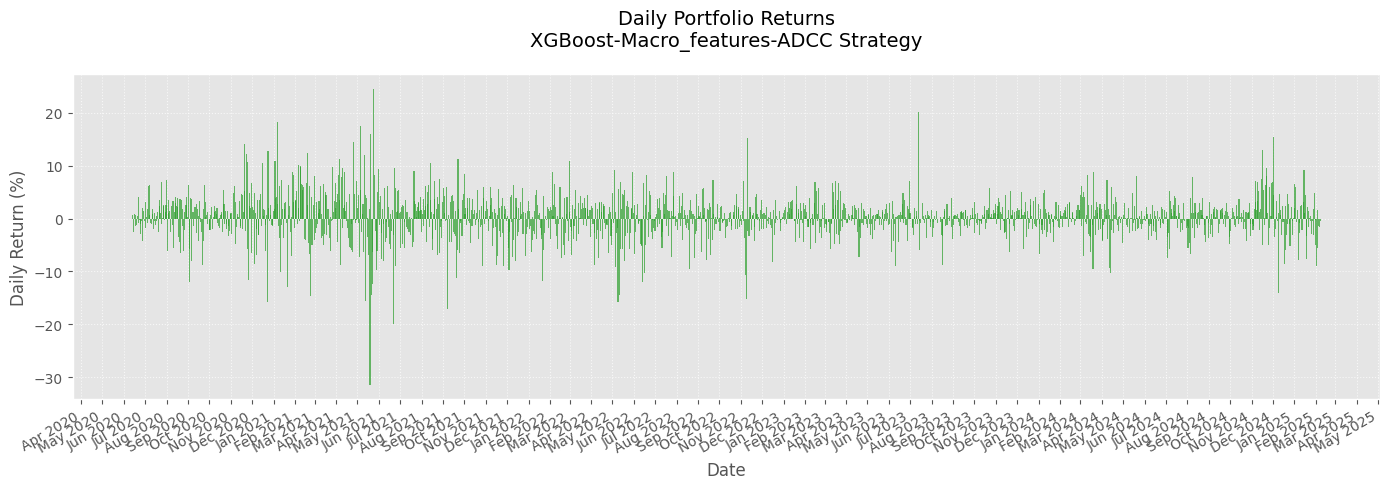
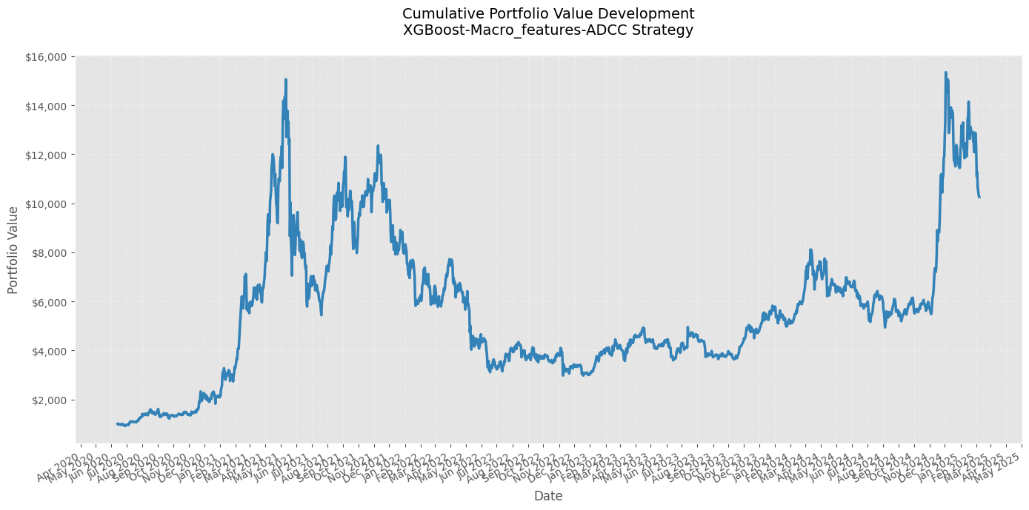
**4. Results**

**4.1: Inputs estimation method**

**4.1.1: GMV objective (24 months window + 14 days refitting)**





5. Analysis

6. Conclusion

*References:*

* Corsi, F. (2009). *Heterogeneous Autoregressive Realized Volatility: A Simple Approach to Long Memory.* Journal of Financial Econometrics. [Link: <https://academic.oup.com/jfec/article/22/2/492/7081291>]
* Brauneis, A. & Sahiner, M. (2024). *Crypto Volatility Forecasting: Mounting a HAR, Sentiment, and Machine Learning Horserace.* Asia-Pacific Financial Markets. [Link: <https://link.springer.com/article/10.1007/s10690-024-09510-6>]
* Christensen, K., Siggaard, M., & Veliyev, B. (2021). *A Machine Learning Approach to Volatility Forecasting.* [Link: <https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3766999>]
* Jain, V., Sahay, R. R., & Nupur (2024). *TODIM with XGBOOST and MVO Metaheuristic Approach for Portfolio Optimization.* [Link: <https://link.springer.com/article/10.1007/s13198-024-02610-6>]
* Huang, X., Newton, D. P., Platanakis, E., & Sutcliffe, C. (2024). *Single-Stage Portfolio Optimization with Automated Machine Learning for M6.* [Link: <https://www.sciencedirect.com/science/article/pii/S0169207024000918>]
* Guidolin, M., Panzeri, G., & Pedio, M. (2024). *Machine Learning in Portfolio Decisions.* [Link: <https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4988124>]
* Kuhn, D., Shafiee, S., & Wiesemann, W. (2024). *Distributionally Robust Optimization.* [Link: <https://arxiv.org/abs/2411.02549>]