Volatility forecasting with component GARCH enhanced machine learning models

A thesis submitted in partial fulfillment for the

degree of

Master of Science

(MSc) Financial Engineering

in the

Decision Sciences Faculty
HEC Montréal

Proposed by: Yissan Yaro

Supervisor: Christian Dorion

April 2024

Abstract

Inspired by Ramos-Pérez et al. (2021), our goal with this study is to compare the versatility and then the performance of recent machine learning models on volatility forecasting and assess their superiority over econometric benchmarks.
To have 2 axes of comparisons, we performed our experiment on 5 different assets, and for 5 forecasting horizons. For each asset, we studied the models' performance for forecasting the volatility for 5, 10, 15, 20, and 60 days ahead. We also studied the performance gains from combining each of our machine learning models with a Component GARCH model.
According to our evaluation framework, our results suggest that the extreme gradient boosting model is the most versatile model for forecasting volatility, closely followed by the natural gradient boosting model. Both boosting models are particularly the best-performing models on the equity indices at every horizon, with the extreme gradient boosting outperforming the natural gradient boosting on these indices. The extreme gradient-boosting model is also one of the best models for forecasting oil volatility. The Long-Short-Term Memory and the transformer neural network turned out to be the best models for forecasting gold volatility for any horizon. Moreover, we found no significant forecasting gain in combining all our models in an ensemble, or in augmenting the machine learning models with a component GARCH.

Contents

List of Tables

List of Figures

Acronyms

ADF Augmented Dickey-Fueller test for unit-root

ANN Artificial Neural Network

CART Classification And Regression Trees

CGARCH Component Generalized Autoregressive Conditional Heteroskedastic model

FFN Feed Forward neural Network

GARCH Generalized Autoregressive Conditional Heteroskedastic model

HV Historical Volatility

JB Jarque-Berra test for normality

LSTM Long Short Term Memory network

LSTM CGARCH Feed forward ensemble model combining LSTM and CGARCH models

MCS Model Confidence Set

MGW Multivariate Giacomini-White test

ML Machine learning

MLP Muli Layer Perceptron

NGBOOST Natural gradient Boosting model

NG CGARCH Feed forward ensemble model combining Ngboost and CGARCH models

OF Overvaluation Frequency

QLIKE Quasi Likelihood function

RelU Rectified Linear Unit

RNN Recurrent neural Network

RV Realized Volatility

TRANS CGARCH Feed forward ensemble model combining Transformer and CGARCH

XGBOOST Extreme Gradient Boosting model

XG CGARCH Feed forward ensemble model combining Xgboost and CGARCH models

X-N-L-T Feed forward ensemble model combining Xgboost, NgBoost, LSTM, Transformer

X-N-L-T CGARCH Feed forward ensemble model combining Xgboost, NgBoost, LSTM, Transformer, and CGARCH model

1 Introduction

This study evaluates the versatility and performance of recent machine learning models on the problem of volatility forecasting and compares them with classical state-of-the-art econometric models. We study the performance along two axes, which are the forecasting horizon (for which we have 5), and the asset on which the forecast is performed (of which we have 5).

We also study the benefits of enhancing these machine learning models with a component GARCH model (Lee and Engle (1993)), as well as combining all of them in an ensemble approach. The machine learning architectures (referred to as vanilla models) considered here are the transformer network, the Long Short Term Memory Network, the Natural gradient boosting model, and the extreme gradient boosting model. The proposed methodologies integrate the strengths of the component GARCH, which captures conditional heteroskedasticity, and the four machine learning architectures which excel in capturing long-range dependencies. Particularly, the LSTM and transformer architectures were purposely created for sequence modeling and forecasting. The boosting models as far as they are concerned, represent some of the most ingenious and flexible architectures in the machine learning world, in the sense that they are swarms of weak models iteratively trained to reduce the mistakes of their predecessors. These combinations leverage the ability of the models to effectively learn complex patterns, different features and dependencies in the volatility time series data.

To evaluate the performance of the proposed approaches, a comprehensive and rigorous empirical study is conducted using a dataset comprising daily realized volatility and returns from our 5 assets. The realized volatility time series were obtained from intra-day prices of our 5 assets. Since this is an auto-regressive point forecasting study, the inputs to our machine learning models consisted of lags of this realized volatility. For every forecasting problem, the dataset is divided into training, validation, and test sets, with no overlap over them. The final evaluation metrics and tests presented in the result tables were computed on the test set. The developed models are compared against two benchmark models, namely the Component GARCH and the HAR model.

We use an extensive evaluation framework to ensure the rigor of our analysis. First, we compute some classic model evaluation metrics, which are the root mean square error (RMSE), Overvaluation Frequency (OF), and Quasi Likelihood (Q-LIKE). For the RMSE and Overvaluation Frequency, lower values are desirable, while higher values are desirable for the Quasi Likelihood. Then, we use the Multivariate Giacomi-White test (MGW), which is also a statistical test that allows us to assess if two models have the same forecasting abilities on a particular problem. We finish by using the model confidence set methodology, which allows us, with a statistical test on forecasted values, to determine the best subset of models for a forecasting problem with a given confidence level (see Annex A). The two latter are very useful tools to complement our analysis of model versatility.

To be more precise, we define model versatility as the ability of a model to outperform other models on multiple assets and horizons. In our case specifically, leveraging our evaluation

framework, we use the following set of criteria that must be met for a model to be considered versatile:

- The model must outperform all other models according to **at least one scoring metric** (RMSE or Quasi-Likelihood), **on every horizon for at least 3 assets**
- On the assets for which the model is superior according to a scoring metric, the model must pass the statistical difference test against both benchmarks (MGW test).
- According to the Model Confidence Set methodology, the model must belong to the best set of models for forecasting the 3 assets' volatility at every horizon.

After determining the most versatile models, we then focus on specific asset volatility forecasting problems. This means that on the assets for which the versatile models performed poorly, we see if another particular model demonstrates superiority for forecasting this single asset's volatility according to a scoring metric, MGW test, and the model's confidence set. We do the same on the horizon dimension, trying to determine if there is a model that outperforms all others for only one forecasting horizon across assets.

Essentially, we try to find empirical answers to such questions as: Is there a specific model that demonstrates complete superiority along both axes, meaning forecasting superiority across horizons and assets? Are there some models that demonstrate superiority in a particular horizon or a specific asset? Is there a clear benefit across assets and/or horizon to enhancing our machine learning models with CGARCH or to combining all of them as an ensemble model? This thesis addresses and answers these questions through a rigorous empirical study.

2 Litterature review

In a comprehensive literature review of multi-horizon volatility forecasting models enhanced by machine learning techniques, it's essential to trace the evolution from traditional econometric models to the integration of advanced machine learning algorithms. One challenge in studying volatility models' literature is to have a comprehensive enough classification framework. Previous studies by Engle and Patton (2001) have attempted this exercise and classify volatility models in two parent categories: models that formulate conditional volatility as a function of observables, and latent volatility models, often coined stochastic volatility models. All of the models explored in this study fall into the first category with respect to this classification. A subsequent seminal classification framework was proposed in Poon and Clive (2003) and Poon and Granger (2005) in their Review of volatility forecasting models, in which they classify volatility models in the following families: time series models based on past realization of standard-deviation family, the ARCH models family, the stochastic volatility family, the non-parametric model's family, the options-based models family, and the machine learning models family.

Ge et al. (2023a) conducted a systematic review, highlighting the efficacious deployment of neural networks in financial volatility forecasting, emphasizing the nuanced ability of these models to approximate both linear and nonlinear dynamics without prior knowledge of the data-generating process. Their study provides an in-depth analysis and comprehensive examination of the advances in Neural Network (NN) applications for financial volatility forecasting. The study examines 35 publications post-2015, identifying issues like the difficulty in meaningful comparisons between models due to a lack of standardization and the disparity between contemporary machine learning (ML) and financial forecasting models. The review identifies several key issues prevalent in the current landscape of neural network-based volatility forecasting. One of the primary challenges noted is the difficulty in conducting easy and meaningful comparisons between different models. This challenge stems from the diverse range of approaches and methodologies employed across studies, making it hard to directly compare their effectiveness or draw conclusive insights. Another challenge is the mere definition of volatility, which varies vastly from one study to another. The review reveals a preference for using historical volatility (HV) over theoretically more robust alternative proxies like implied volatility (IV) or realized volatility (RV) in the literature, partly due to the ease of access to necessary data and the simplicity of the models.

Initially, volatility forecasting was dominated by econometric models, with the Autoregressive Conditional Heteroskedasticity (ARCH) model introduced by Engle (1982) pioneering the field. The generalized ARCH (GARCH) model by Bollerslev (1986) further refined this approach by allowing past conditional variances to influence current estimates, addressing the persistence often observed in financial volatility. The literature then expanded to include models that account for asymmetries and leverage effects, such as the Exponential GARCH (EGARCH) model proposed by Nelson (1991) and the Glosten-Jagannathan-Runkle (GJR) GARCH model (Glosten et al. (1993)). These models capture the phenomenon where negative shocks induce greater volatility than positive shocks of the same magnitude, a reflection of the leverage effect.

With the advent of high-frequency trading and intraday data, realized volatility measures and proxies using high-frequency returns were proposed, enhancing the accuracy of volatil-

ity estimation. Andersen et al. (2003) were instrumental in this development, leading to a variety of realized measures such as realized variance and realized range. The subsequent development of realized volatility estimation and forecasting can be attributed to a series of papers by Barndorff-Nielsen and Shephard, Bandi and Russel, and Ait-Sahalia in which they extensively establish best practices in terms of estimating, sampling, and forecasting realized volatility.

In the landscape of volatility forecasting, the Heterogeneous Autoregressive (HAR) model stands out due to its effectiveness and simplicity in capturing the dynamics of realized volatility. The HAR model can be viewed as a specific autoregressive application of the Ghysels et al. (2007) MIDAS (Mixed Data Sampling) approach, where daily, weekly, and monthly volatilities are harmoniously integrated into a single forecasting model. It emerged from the need to address the limitations of existing models in capturing the long memory and multi-scale nature of volatility. Traditional models like ARCH and GARCH, while effective in many contexts, often fell short in accurately representing the persistence and heterogeneous market behaviors influencing volatility. First introduced in Corsi (2008), it presented a novel approach to volatility modeling, allowing capturing the strong persistence observed in realized volatility at multiple scales (usually daily, weekly, and monthly) within a single framework. Its versatility is further evidenced by the introduction of logarithmic transformations of realized volatility or exogenous regressors, enhancing its compatibility with standard time series procedures. Empirical evidence from various markets, including equity, bond, and commodity markets, consistently underscores the HAR model's superior forecasting performance compared to traditional ARCH-family models. This versatility in application underscores the model's adaptability and robustness. Furthermore, recent advancements have seen the integration of machine learning techniques with HAR models, opening new avenues for enhancing forecasting accuracy.

As machine learning started to reshape many domains, its applications in volatility forecasting gained traction. The literature reveals a shift towards leveraging machine learning's ability to handle non-linearities and complex interactions within data. Initial efforts to forecast volatility with neural networks were made by Miranda and Burgess (1997) , who used them to predict intraday volatilities for the Spanish stock market or Tino et al. (2001) to show how they outperform ARCH models in predicting the volatility of the Austrian stock market. Additionally, an influential study by Hamid and Iqbal (2004) found that ANN's realized volatility forecasts on the S&P500 outperform implied volatility from the Barone-Adesi and Whaley model in anticipating future realized volatility. More recently, deep learning models, particularly Long Short-Term Memory (LSTM) networks, have been employed for their ability to capture long-term dependencies in time series data. Works by Fischer and Krauss (2018) demonstrated the potential of LSTMs to outperform traditional GARCH models in forecasting volatility. Comparative studies, such as those by Hansen and Lunde (2005), have evaluated the effectiveness of these models against traditional GARCH-type models. While machine learning models often outperform in capturing complex market dynamics, they also pose challenges in interpretability and computational intensity. This prompted finance academics to develop hybrid models that combined the strengths of econometric models and machine learning. For instance, incorporating GARCH-based features into neural networks or SVMs can yield models that are both interpretable and powerful in forecasting. Hybrid models that combine machine learning with traditional GARCH-type models have emerged,

seeking to fuse econometric rigor with machine learning's flexibility.

Moreover, the literature reflects an interest in ensemble methods for volatility forecasting. These methods pool predictions from a collection of different models, capitalizing on the strength of each. The blend of various machine learning models into an ensemble, as proposed by Ramos-Pérez et al. (2021), can often yield more accurate and robust forecasts than individual models alone. One of the first studies to try to combine a GARCH model with artificial neural networks for return volatility forecasting was Donaldson and Kamstra (1997), followed by Hu and Tsoukalas (1999). Another notable study was conducted by Malliaris and Salchenberger (1996) to forecast an index Implied volatility for risk management purposes using an ANN. Dunis and Huang (2002) experimented somewhat similar to what we intend to do: They explored the relevancy of Recurrent neural networks and neural network regressions for forecasting and trading currency volatility (using options straddles), as well as combinations of these models with GARCH models. They find very promising results from the RNN-based volatility trading strategy. A recent study by Christensen et al. (2022) finds evidence of the superior predictive ability of neural networks and regression trees over HAR models, especially over long horizons. They attribute the forecast gains to higher persistence captured by the ML models. Lu et al. (2016) demonstrates the outperformance of a stacked EGARCH-ANN model over traditional GARCH models for forecasting the Chinese energy markets' log returns volatility. Similarly, Kristjanpoller and Minutolo (2015) and Kristjanpoller and Minutolo (2016) respectively show the superiority of GARCH-enhanced ANN in forecasting gold and oil price volatility. A recent attempt to use XgBoost models for volatility forecasting was made by Teller et al. (2022). They find evidence of Xgboost models outperforming HAR models for different forecasting horizons. They also find that using non-linear specifications for base learners performs better than non-linear ones on longer horizon forecasts.

More recently, transformers, first introduced in the seminal paper "Attention is All You Need" by Vaswani et al. (2017), revolutionized the field of deep learning by eschewing the traditional recurrent structures in favor of self-attention mechanisms. In a groundbreaking study, Liu et al. (2023) put forth a transformer model leveraging mixed-frequency data to forecast stock volatility. Their application further illustrates the versatility of Transformer architectures in finance across a spectrum of assets. As reported in research covered by Ge et al. (2023b) , the temporal fusion transformer Lim et al. (2021), a variant of the canonical Transformer, has been applied to forecast the volatility of diverse assets such as S&P500, NASDAQ100, gold, silver, and oil and outperforms RNNs, MLPs, and GARCH models for each of these assets.

Closest to our study, Ramos-Pérez et al. (2021) found empirical evidence for the outperformance of an extended transformer architecture called the multi-transformer in generating more accurate one-day ahead forecasts of the S&P500 log returns volatility over GARCH, LSTM, ANN, and traditional transformer. They also found evidence of the outperformance of the GARCH-enhanced versions of these models. Ge et al. (2023a) conducted a systematic review, highlighting the efficacious deployment of neural networks in financial volatility forecasting, emphasizing the nuanced ability of these models to approximate both linear and nonlinear dynamics without prior knowledge of the data-generating process.

In the context of multi-horizon forecasting, literature underscores the challenges of predicting volatility over different time frames. The inherent uncertainty of the market and the

impact of external economic events make multi-horizon forecasting particularly complex. Researchers have investigated models that can simultaneously provide short, medium, and long-term volatility forecasts, with mixed success.

3 Data

3.1 Volatility stylized facts

Market volatility, as a measure of risk, uncertainty, or market gauge, is one of the most documented and researched subjects of modern finance. It has been explored both from the forecasting angle and the measuring angle. A couple of stylized facts are universally recognized when it comes to volatility, given the extensive and mathematically sound research that went into proving them.

The stylized facts of financial market volatility are rich and complex, intertwining various phenomena like the leverage effect, the impact of outliers, the influence of news, forecastability issues, non-stationarity, and long memory. These factors are interrelated, each contributing to the intricate behavior of financial markets.
Starting with the leverage effect, this phenomenon highlights a key asymmetry in how markets respond to information. Typically, negative news tends to have a more pronounced impact on volatility compared to positive news of a similar magnitude. According to Black (1976) this asymmetric response can be partly explained by the leverage hypothesis, which suggests that negative market movements increase the leverage of firms (debt-to-equity ratio), thereby raising the risk and volatility. The leverage effect appears to not just be a theoretical construct but a practical consideration in volatility modeling, as it challenges the notion of symmetry in market responses and necessitates the use of models like GJR-GARCH and EGARCH that can account for this asymmetry.
Closely linked to the leverage effect is the impact of news on market volatility. Financial markets are highly sensitive to new information, and the arrival of news can lead to sudden and significant changes in volatility. According to Engle and Ng (1993) and Maheu and McCurdy (2004), the relationship between news and volatility is complex and often depends on the nature of the news, market conditions, and investor sentiment. Positive news might lead to a moderate increase in volatility due to increased trading activity, whereas negative news can cause a sharp spike in volatility, reflecting panic selling or rapid changes in investor sentiment.
The impact of outliers is another critical aspect of financial market volatility. Outliers, which can be caused by extraordinary events such as financial crises, geopolitical tensions, or major economic announcements, can disproportionately affect volatility. These outliers often lead to a heavy-tailed distribution of returns, meaning that extreme changes in prices are more common than would be expected in a normal distribution. This heavy-tailed characteristic of financial returns complicates risk management and forecasting, as standard models based on normal distribution assumptions may underestimate the probability and impact of extreme market movements.

Forecastability of volatility is an ongoing challenge in financial econometrics. Volatility is inherently difficult to predict due to its dynamic non-stationary nature and sensitivity to a wide range of factors. Non-stationarity implies that the statistical properties of volatility, such as the mean and variance, change over time, making it difficult to model and forecast using traditional time series approaches. It may arise from various factors including changing macroeconomic conditions, evolving market structures, or the impact of regulatory changes. Some studies (Perry and R (1982) and Pagan and Schwert (1990)) even find some evidence for a unit root in the volatility time series. While models like GARCH and its variants have been developed to improve volatility forecasting, the accuracy of these predictions can vary significantly depending on the model specification, time period, and market conditions.

The concept of long memory in volatility refers to the persistence of shocks in financial markets over time. Long memory implies that the effects of past market movements, especially large shocks, can influence volatility for an extended period. These characteristics challenge the assumption of short memory in traditional time series models and have led to the development of long-memory models like FIGARCH which captures the decaying influence of past shocks over time but at a slower rate compared to short-memory models, or the component GARCH model. The latter was explicitly designed to allow a separate long-term component in its specification, to accurately capture the persistence shocks in the series. Long memory is a critical concept for understanding the temporal dependencies in financial markets and for developing effective risk management strategies.

Contrasting with long memory (persistence) is the concept of mean reversion, which posits that despite short-term fluctuations, volatility tends to revert to a long-term average over time. This phenomenon indicates that high or low periods of market volatility are temporary and will eventually move back toward a historical average. Mean reversion is a key consideration in risk management and financial modeling, as it implies that extreme market conditions are not permanent and that forecasts should account for this reversion to the mean over the long term.

Volatility clustering, another critical aspect, refers to the tendency of high-volatility periods to be followed by high-volatility periods and low-volatility periods to be followed by low-volatility periods. This phenomenon, observable in financial time series, suggests that volatility exhibits a serial correlation – large changes in prices are often clustered together, followed by periods where prices change minimally. This clustering effect is a crucial factor in volatility forecasting, as it necessitates models that can account for these changing regimes in market conditions. Tackling volatility clustering is what originally motivated the inception and design of GARCH models by Engle and Bollerslev.

These concepts are deeply interrelated. Long memory in volatility suggests the persistence of the effects of market shocks, implying that past market events can influence future volatility for a prolonged period. However, the presence of mean reversion within this framework indicates that while these effects are persistent, they don't last indefinitely, with volatility eventually returning to its long-term average. Meanwhile, volatility clustering reflects how these phenomena can manifest in actual market behavior, with periods of persistent high or low volatility, followed by eventual reversion to mean levels. The interplay of these factors complicates the task of volatility modeling and forecasting. Traditional econometric models have been augmented with features to capture long memory and mean reversion. Yet, accurately capturing these dynamics with a single parametric model remains a challenge due to

the inherent complexity and evolving nature of financial markets.
Overall, the financial market volatility landscape is marked by asymmetry, exogenous factors, and a delicate balance between persistence and mean reversion, with volatility clustering providing observable evidence of these dynamics, making it a challenging yet fascinating subject for study in financial economics.

3.2 Data exploration

This study focuses on forecasting the volatility of 5 assets that have been selected for specific reasons because they have different degrees of sensitivity to each of the aforementioned stylized facts. We attempt to forecast the volatility of the S&P500, the NASDAQ100, the RUSSEL 2000, gold prices, and oil prices.
As one of the most mainstream and followed gauges for the health of the american stock market, the S&P500 has historically been impacted by market sentiment, geopolitical events, and macroeconomic data. It has historically demonstrated high cyclicality, due to its inclusion of the biggest companies of the most cyclical industries.
As per the NASDAQ, it has historically proven much more volatile and fat-tailed than the S&P500 according to Inc (2018) and our descriptive statistics table below, due to the highly uncertain nature of technology businesses and their characteristics: the highly innovation-driven industry, the constant evolution of business conditions, and the relatively low barriers to entry. Some of these companies, although very successful, are often in the early stages of their growth and have not yet established a track record of profitability. Additionally, technology companies are often more reliant on a small number of products or services, which can make them more susceptible to market fluctuations. Finally, the technology industry is subject to rapid changes, which can lead to increased uncertainty and volatility in stock prices.
Similarly, in our analysis period, as per the daily volatility series descriptive statistics table below, the RUSSEL 2000 has historically exhibited more volatility due to the leverage effect and its longer memory, mainly due to the high sensitivity of smaller companies to market and economic conditions, especially to interest rates. It can be seen by remarking how smaller companies perform worse than average in a recession or high interest rate environment, and above average in an expanding economy. They have more debt on their balance sheets, and can therefore better leverage a favorable market environment, but are less resilient to economic shocks.
Concerning oil prices, it is a well-documented fact that they exhibit extreme volatility clustering or severe fat tails around major supply disruption (Kang et al. (2009)), as well as slow mean reversion and very quick reaction to random shocks. On top of supply and demand shifts, oil prices are very sensitive to geopolitical events, instability, weather, and natural disasters, as well as any minor disruption in the highly fragmented chain of production.
Gold, as an asset, has a long history of being a reserve of value, and has historically been the prime vehicle of wealth protection for being inflation-proof. Additionally, just like oil prices, gold prices are driven by supply and demand dynamics, as well as geopolitical shocks, interest rates, and most importantly, inflation and investor behavior. Gold prices are very prone to behavioral drivers like herding or panic, and also display a strong asymmetry because negative shocks tend to increase volatility more than positive shocks.

What makes the attempted exercise of forecasting the volatility of these assets interesting is the different degrees in which we find the documented stylized facts in each of them. Additionally, despite acknowledging the influence of the mentioned exogenous factors impacting the volatility of our assets, our endeavor here is an auto-regressive forecasting study: We focus on harnessing the most predictive power and capturing the stylized facts exclusively from volatility lags.

We can see a display of some descriptive statistics of our returns and volatility series in Table 1, illustrated on Figure A.1 and Figure A.2. The descriptive statistics table also shows us the p-values from the ADF and Jarque-Berra tests. On the considered sample window for all 5 assets, both tests return very small p-values, which is in line with what has been documented so far in most of the literature. This means that for each asset studied here, at a threshold of 1%, the realized volatility series are non-gaussian and non-stationary (and might contain a unit root), as can be visualized on Figure A.2. Except for gold, as per the left panel of Figure A.3, the volatility series exhibit strong auto-correlations at shorter lags (1 to 5), as well as moderate to strong auto-correlation until approximately lags 25 to 30. All the volatility series also exhibit strong partial auto-correlation for short lags, but mostly nonexistent partial auto-correlation after lag 10, as per the right panel of Figure A.3.

Another relevant metric that can be observed in the table below is the Hurst exponent. This metric is used as a measure of long-memory of time-series. First introduced in Hurst (1951), it relates to the auto-correlation of time-series and is an additional tool that helps us quantify the degree of randomness present in a time series. As shown in Mandelbrot and Ness (1968) and explained in Gatheral et al. (2018), it informs us on the memory of a time series by quantifying its tendency to either revert to the mean or to persist in a direction. It is therefore an ideal tool to assess the degree of mean-reversion or persistence present in a series. A value of the exponent between 0.5 and 1 indicates a long-term positive auto-correlation, suggesting that our series is persistent. A value between 0 and 0.5 is indicative of anti-persistence (mean reversion), with a lower value meaning stronger mean reversion. A value of exactly 0.5 characterizes a series with pure randomness (brownian motion). Famously introduced by Hurst (1951) in the domain of hydrology, it was popularized in finance by Mandelbrot and Ness (1968) to introduce their notion of fractional brownian motion which later led to the concept of rough volatility from Comte and Renault (1998), Gatheral et al. (2018) and Calvet and Fisher (2002). In this study, we use the hurst exponent only to assess the long-term properties of our series. The presented results suggest that all of our assets' volatility exhibit some degree of mean-reversion, with the NASDAQ being the "most" mean-reverting and oil being the closest to being a random series, with a hurst exponent of 0.47.

3.3 Realized Volatility

As demonstrated by Barndorff-Nielsen and Shephard (2002), realized volatility is a universally accepted and used proxy for the quadratic variation of a semi-martingale (in this case our return process). Assuming that our price process is described by the simple jump-diffusion process:

$$dS_t = \mu_t dt + \sigma_t dW_t + dJ_t$$

where:

- S_t is the stock price at time t,
- μ_t is the drift coefficient (representing the rate of return),
- σ_t is the diffusion (volatility) coefficient,
- W_t is a Wiener process (or Brownian motion),
- J_t is a Jump process assumed to be defined by $J_t = \sum_{i=1}^{N_t} Y_i$, with $\{Y_i\}_{i\in\mathbb{N}}$ the jump sizes and N a counting process (Cox or Poisson)

The quadratic variation of this stochastic process is a measure of the cumulative variance of the process over a time interval, consisting of the variance from jumps and the integrated variance. For a stochastic process S_t represented by the stochastic differential equation above, the quadratic variation over an interval $[0, T]$ is given by:

$$[S]_T = \underbrace{\int_0^T \sigma_t^2 dt}_{\text{Integrated variance}} + \underbrace{\sum_{i=1}^{N_T} Y_i^2}_{\text{Sum of the squared jumps}} = \lim_{n\to\infty} \sum_{i=1}^{n} (S_{t_i} - S_{t_{i-1}})^2$$

if the limit exists. This represents the cumulative variance of the stock price process over the time interval $[0, T]$. In a series of papers, Barndorff-Nielsen and Shephard propose using the realized volatility as a proxy for the quadratic variation of a semi-martingale. Let us denote the intraday return as:

$$R_{t+i\frac{\tau}{n}} = \log(S_{t+i\frac{\tau}{n}}) - \log(S_{t+(i-1)\frac{\tau}{n}})$$

The realized variance $RV_{t,t+\tau}$ is defined as:

$$RV_{t,t+\tau} = \sum_{i=1}^{n} \left(R_{t+i\frac{\tau}{n}}\right)^2$$

where where $\tau = \frac{1}{252}$ is the length (in years) of a business day and n is the number of periods per day. There are 6.5 hours of trading activities per day, meaning that n = 390 if we sample every minute, and n = 78 if we sample every 5 minutes. The daily quadratic variation satisfies:

$$QV_{t,t+\tau} = \lim_{n\to\infty} \sum_{i=1}^{n} \left(\log(S_{t+i\frac{\tau}{n}}) - \log(S_{t+(i-1)\frac{\tau}{n}})\right)^2 = \lim_{n\to\infty} \sum_{i=1}^{n} \left(R_{t+i\frac{\tau}{n}}\right)^2$$

Hence, as n tends to infinity:

$$QV_{t,t+\tau} \approx RV_{t,t+\tau}$$

For a given $n \in \mathbb{N}$, the realized volatility is a proxy for the daily quadratic variation.
Using realized volatility to model variance grants us the ability to bypass reliance on theoretical models by directly utilizing a variable that can be observed. This approach presents a clear benefit; however, challenges arise when it comes to the actual application of this method. Research indicates that micro-structure noise, which stems from various factors like bid-ask spreads and the discrete nature of price formation, exists within financial markets. This noise, as discussed in works by Amihud and Mendelson (1987), Harris and Raviv (1991), and Madhavan (2000), has the potential to skew the accurate estimation of realized volatility.

3.4 Collection, construction and preprossessing

The data used in this study is extracted from the Trades and Quotes (TAQ) database. Since our assets of interest were not directly observable on the market, we used the following ETFs as proxies:

Asset	Proxy's ticker	Proxy's name
S&P500	SPY	SPDR S&P 500 ETF
NASDAQ	IYW	iShares US Technology ETF
RUSSEL 2000	IWM	iShares Russell 2000 ETF
GOLD	IAU	iShares Gold Trust
OIL	USO	United States Oil ETF

For each ETF, we extracted a series of millisecond intraday trade prices between December 2005 and December 2021. The raw data consisted, for each ETF, of nationwide aggregated trade data at each millisecond of the day. Following the findings of Aït-Sahalia et al. (2011), Bandi et al. (2008) and Bandi and Russell (2008) proposing that it is reasonably safe to treat five-minutes sampled returns as noise-free on most liquid assets in the recent years, we chose to extract prices at five-minute intervals, from which intraday returns are computed, which are then summed as per the formula above to obtain our daily realized volatility series.
Data cleaning and standardization are crucial steps in preparing time-series data for forecasting with machine learning. Machine learning algorithms often require data to be on a similar scale, typically achieved through normalization or standardization. To effectively improve the training of our models, we chose to re-scale each volatility time series using a z-score normalization often referred to as standardization. This method transforms the data to have a zero mean and a standard deviation of one. For a data point x_t, the standardized value z_t is:

$$z_t = \frac{x_t - \mu}{\sigma}$$

where μ is the mean and σ is the standard deviation both estimated only on the training set of each series. In other words, for each time-series we estimate the mean and standard-deviation of the training set, which are used to re-scale the whole dataset. The purpose of proceeding this way is to prevent data leakage, meaning to prevent information from unseen data (test or validation set) to impact or training set.

4 Forecasting models

We present every [**A-M-H**] model final architecture in the results section.
We use the following nomenclature throughout the study to refer to our models:

- [**A-M-H**] : Model M for H steps ahead forecasts on asset A
- [**M-H**] : Group of models M for H steps ahead forecasts
- [**A-M**] : Group of models M applied on Asset A ,
- [**A-H**] : H steps ahead foresting problem on asset A ,

With

- **A** ∈ [S&P500, NASDAQ 100, RUSSEL 2000, OIL, GOLD]
- **M** ∈ [CGARCH, HAR, XgBoost, NgBoost, LSTM, Transformer, XG CGARCH, NG CGARCH, LSTM CGARCH, TRANS CGARCH, X-N-L-T, X-N-L-T CGARCH]
- **H** ∈ $[5, 10, 15, 20, 60]$

4.1 Benchmarks

4.1.1 HAR

The Heterogeneous Autoregressive (HAR) model, introduced by Corsi (2008) primarily for modeling and forecasting financial volatility, stands out for its ability to encapsulate different components of volatility operating at varying time scales. This approach is particularly adept at capturing the multi-scale nature of financial market volatility, making it highly suitable for multi day ahead forecasting. The HAR model is built upon the concept that volatility can be influenced by factors operating at different time horizons. It typically includes daily, weekly, and monthly volatility components, allowing it to capture the varying effects of information arriving at different frequencies. The purpose of the HAR model is to account for the long memory and persistent nature of volatility. It integrates different time scales (daily, weekly, monthly) into the volatility forecasting process, reflecting the varying investment horizons and information-processing speeds of market participants. This multi-scale approach allows the HAR model to capture both short-term fluctuations and long-term trends in volatility, making it particularly useful for forecasting over multiple horizons. The mathematical representation of the HAR model for daily data is typically as follows:

$$RV_{t+1|t} = \alpha + \beta_d RV_t + \beta_w RV_t^{(w)} + \beta_m RV_t^{(m)} + \epsilon_{t+1}$$

Here, $RV_{t+1|t}$ is the forecasted realized volatility at time $t+1$, given information up to time t. The model includes three key terms:

- RV_t : The daily realized volatility.
- $RV_t^{(w)}$: The weekly realized volatility, often calculated as the average of daily volatilities over the past week.
- $RV_t^{(m)}$: The monthly realized volatility, typically the average of daily volatilities over the past month.

The parameters α, β_d, β_w, and β_m are estimated using a simple ordinary least-squares technique, and ϵ_{t+1} is the error term.
The HAR model assumes that variance is a continuous, integrable function over the chosen period (daily, weekly, monthly), and that the aggregated volatilities are representative of different market participant horizons. For the HAR model to provide reliable forecasts, the time-series data must satisfy certain statistical properties. Ideally, the data should be ergodic, ensuring that time averages converge to ensemble averages, which is vital for the consistency of the estimated parameters. The model assumes a linear relationship between past realized volatility and future volatility. The error term ϵ_{t+1} is generally assumed to be normally distributed with mean zero and constant variance. The coefficients β_d, β_w, and β_m are estimated using ordinary least squares regression.
For multi-day ahead forecasting, the HAR model leverages its ability to incorporate information from various time horizons. The forecasting process can be iteratively applied to predict future volatilities. For instance, a k-day ahead forecast is iteratively computed following the algorithmic expression:

$$RV_{t+i|t+i-1} = \alpha + \beta_d RV_{t+i-1} + \beta_w RV_{t+i-1}^{(w)} + \beta_m RV_{t+i-1}^{(m)} + \epsilon_{t+i} \text{ for } i = 1, 2, \ldots, k$$

4.1.2 Component GARCH

The Component GARCH model represents a significant evolution in volatility modeling, offering a nuanced understanding of the dynamics of financial market volatility, particularly suited to capture the nuanced behavior of volatility. As detailed by Lee and Engle (1993), the model decomposes the conditional variance of asset returns into two distinct elements: a permanent component q_t that captures the long-term variance, and a transitory component that captures short-term fluctuations. The permanent component is envisioned as a highly persistent process, potentially following a random walk, which is indicative of its stability and influence over extended periods. This is mathematically characterized by an autoregressive parameter ρ close to one, as empirically demonstrated in their paper for the S&P 500 and NIKKEI indices, suggesting the presence of a unit root and hence a non-reverting process that embeds the impact of new information over time.

The original GARCH(1,1) process as per Bollerslev (1986) is expressed as:

$$r_t = m_t + \epsilon_t$$

$$h_t = \omega + \alpha\epsilon_{t-1}^2 + \beta h_{t-1}$$

Mathematically, the Component GARCH model modifies the GARCH(1,1) representation of h_t and can be expressed with the following equations from the Lee and Engle (1993) paper: The conditional variance equation:

$$h_t = q_t + \alpha(\epsilon_{t-1}^2 - q_{t-1}) + \beta(h_{t-1} - q_{t-1})$$

The equation for the permanent component:

$$q_t = \omega + \rho q_{t-1} + \phi(\epsilon_{t-1}^2 - h_{t-1})$$

where:

- h_t is the conditional variance at time t,
- q_t is the permanent component of the variance at time t,
- ϵ_{t-1} is the innovation or shock from the previous time period,
- α and β are parameters capturing the responsiveness of the conditional variance to the innovation and the lagged variance, respectively,
- ω, ϕ, and ρ are parameters specific to the permanent component, with ρ close to one indicating the non-reverting characteristic of q_t.

The transitory component, represented by $\alpha(\epsilon_{t-1}^2 - q_{t-1}) + \beta(h_{t-1} - q_{t-1})$, captures the short-term fluctuations, typically reverting to the long-term trend.
For the Component GARCH model to be meaningful and for its forecasts to be reliable, certain stationarity and convergence conditions must be met. The stationarity condition for the Component GARCH model can be formulated as:

$$(\alpha + \beta)(1 - \rho) + \rho < 1$$

This condition implies that both the permanent component (captured by ρ) and the transitory component (captured by $\alpha + \beta$) must be covariance-stationary
A key requirement of any volatility model is that the volatility estimate should remain non-negative over time. This is crucial for the model's practical applicability and interpretability. The Component GARCH model maintains this property under certain parameter constraints. Specifically, the parameters must satisfy certain inequality constraints to guarantee non-negative conditional variances.
Engle and Lee's empirical analysis using daily stock indices demonstrated the model's efficacy in capturing both the short-term and long-term dynamics of market volatility. The model's parameters were found to be significant, indicating the presence and importance of both permanent and transitory components in the conditional variance. This empirical

validation underscores the model's utility in practical applications, particularly in financial risk management and strategic investment planning.
For forecasting volatility multiple days ahead, we focus on predicting h_{t+k} for $k > 0$, where k is the number of days ahead of the forecast. The forecasting involves the following steps:

- For the permanent component q_t, the forecast q_{t+k} is given by:

$$q_{t+k} = [(1 - \rho^k)/(1 - \rho)]\omega + \rho^k q_t$$

  where ρ is the autoregressive root introduced in the trend equation.
  If $\rho = 1$, which indicates a unit root, the equation simplifies to a linear trend:

$$q_{t+k} = k\omega + q_t$$

- The forecast of h_{t+k} involves predicting the difference between the conditional variance and its trend:

$$h_{t+k} - q_{t+k} = (\alpha + \beta)^k (h_t - q_t)$$

  As $k \to \infty$, this difference converges to zero, indicating that in the long run, the conditional variance aligns with its trend.

- The final forecast for the conditional variance h_{t+k} is obtained by combining the forecasts of q_{t+k} and $h_{t+k} - q_{t+k}$:

$$h_{t+k} = q_{t+k} + (\alpha + \beta)^k (h_t - q_t)$$

  For large k, as $(\alpha + \beta)^k$ approaches zero, the forecast converges to the permanent component:

$$h_{t+k} \approx q_{t+k}$$

4.2 Single models

4.2.1 Transformer

The transformer model, introduced by Vaswani et al. (2017), marked a departure from recurrent neural network architectures. Its key innovation, the self-attention mechanism, allows it to process input sequences in parallel, leading to significant improvements in training efficiency and model performance. Unlike RNNs, which process sequences iteratively, Transformers use self-attention mechanisms to weigh the importance of different parts of the input data, processing the entire sequence simultaneously and allowing for parallel computation. The original architecture comprises an encoder and a decoder, each consisting of multiple layers. The encoder maps an input sequence of data into a continuous representation, and the decoder generates an output sequence. Each data point in the input sequence is referred to as a token. For our use case, each past observation of realized volatility RV_{t-k} is a token. In this study, as per Ramos-Pérez et al. (2021), we only use the encoder part of the transformer's architecture, represented on Figure 3. A more mathematical intuition of the encoder's architecture borrowed from Thickstun (2019) is demonstrated in Annex B.
Key components of the encoder include:

- **Embedding Layers**: Converts input data points into vectors of fixed dimensionality.
- **Positional Encoding**: Adds information about the position of each token in the sequence.
- **Self-Attention**: Most important part innovation of the transformer architecture. It allows the model to consider other tokens in the input sequence when encoding a token.
- **Multi-Head Attention** : Extends self-attention by running it through multiple attention "heads", capturing different aspects of token relationships.
- **Layer Normalization and Feed-Forward Networks** : Each sub-layer (self-attention, feed-forward) in the Transformer has a normalization step and is followed by a feed-forward network. The Transformer's ability to handle sequences in parallel provides a significant performance advantage over sequential models, leading to its adoption in various state-of-the-art domains.
- **Input embedding and Positional encoding:** As mentioned earlier, Transformer layers lack any recurrence mechanism. Consequently, it becomes essential to incorporate information regarding the relative positioning of observations in the time series into the model. This is achieved by augmenting the input data with positional encoding. In our study, we build on the work of Ramos-Pérez et al. (2021) by using a wave function as a positional encoder.

First, the transformer converts input tokens into vectors of fixed dimensionality,

$$\mathbf{E}_i = \text{Embed}(x_i)$$

Where:

- x_i is the i-th element of the input sequence.
- $\mathbf{E}_i$ is its embedding representation.

Then, the Positional Encoding is computed:

$$\mathbf{P}_i = \cos\left(\pi \frac{pos}{N_{\text{pos}} - 1}\right) = \sin\left(\frac{\pi}{2} + \pi \frac{pos}{N_{\text{pos}} - 1}\right)$$

where pos = (0, 1, . . . , Npos 1) is the position of the observation within the time series and Npos maximum lag
The final input representation for each item in the sequence is:

$$\mathbf{Z}_0 = \mathbf{E} + \mathbf{P}$$

Where $\mathbf{Z}_0$ represents the input to the first encoder layer.

Self-attention and Multi-head attention: The self-attention mechanism (See Figure 1) in transformers is a pivotal component and the main innovation driving the success of this architecture across domains. It allows the model to weigh the significance of different parts of the input differently. Also known as intra-attention, it is a mechanism that equips a model to weigh the importance of different parts of the input data. It is a form of attention mechanism that computes the representation of a sequence by relating all positions of the sequence to each other. Unlike previous attention mechanisms that focused on aligning two different sequences (e.g., in sequence-to-sequence models), self-attention focuses on deriving relationships within a single sequence. The fundamental idea is to compute a representation of a sequence by relating different positions of the same sequence. For example, in a sentence, the meaning of a particular word might depend on the context provided by other words in the sentence. For a volatility series, a particular daily value of RV might indicate a completely different signal depending on whether we're in a tumultuous market, a directional trend, or a stable market.
Self-attention provides a way for each RV lag input to consider the entire input sequence when establishing its context. It involves three key components: Queries (Q), Keys (K), and Values (V). These components are learnable vectors derived from the input data but represent different aspects of that data:

- **Queries (Q)**: These are representations of the input used to score how much focus each element of the input sequence should have with respect to other elements. It can be thought of as a request issued by an element to decide which other elements to focus on

- **Keys (K)**: Keys are used in tandem with queries to compute attention scores. Each key is paired with an input element and is used to determine the amount of attention an element of the sequence should get.

- **Values (V)**: Values are the actual representations of the input elements. Once the attention scores are computed using queries and keys, these scores are used to weigh the values, resulting in a weighted sum that represents the output of the self-attention layer for each element

These vectors are not handcrafted; they are learned during the training process. Each query, key, and value is generated by multiplying the input vector by their respective weight matrices, which are parameters learned by the model. Let's consider an input sequence represented by a matrix X, where each row of X corresponds to an element in the sequence (like a past realization of realized volatility):

First, we perform linear transformations on the input matrix X to obtain the query, key, and value matrices. This is done using trainable weight matrices W^Q, W^K, and W^V:

$$Q = XW^Q$$
$$K = XW^K$$

$$V = XW^V$$

Here, Q, K, and V are the resulting query, key, and value matrices, respectively.
Next, we compute attention scores. These scores determine how much focus should be put on other parts of the input sequence when encoding a particular element. The attention score between a query and a key is usually computed using the dot product, although other methods like cosine similarity can also be used.
The raw attention scores indicate the compatibility between queries and keys and are computed as follows:

$$\text{Score} = QK^T$$

Since the magnitude of the dot product grows with the dimensionality of the input, which could lead to very large values and, in turn, push the softmax function into regions where it has extremely small gradients, it is common practice to scale the scores by the square root of the dimensionality of the keys (d_k) :

$$\text{Score} = \frac{QK^T}{\sqrt{d_k}}$$

To turn the scores into probabilities, the softmax function is applied to each row of the scaled attention scores. This step ensures that the weights sum up to 1, thus forming a proper probability distribution.

$$\text{Attention Weights} = \text{softmax}(\text{Score})$$

Finally, attention weights are used to create a weighted sum of the values. The result is a matrix where each row represents the output of the self-attention layer for each element of the sequence:

$$\text{Output} = \text{Attention Weights} \times V$$

The final output of a self-attention unit is summarized as :

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

The key-value-query concepts are borrowed from information retrieval systems. An intuitive understanding can come by drawing a comparison with how a search engine works: When researching a particular subject and trying to retrieve research papers related to it in a scientific repository, we input a couple of keywords (**query**), which the search engine maps against a set of **keys** (paper title, keywords in the abstract, etc) by computing a similarity score. The engine will then present you with the best-matching articles (**values**), meaning the ones having the highest similarity score. The innovations of the self-attention over a retrieval system can be summarized by the following:

- The query, key, and value are obtained by weighting the initial input sequence by learned parameters

- In the attention scores or query-key product described above, every re-weighted step (**query**) in the sequence is compared to all the steps in the sequence (**keys**) via dot product, and attributed a score telling how relevant this step is to the information contained in the whole sequence. The scores obtained for each step are then used as weights for the values vectors that go the the final attention.

Essentially, what this attention score result will tell us can be seen in our context as: historically, when this consecutive sequence of RVs (or a very similar one) was encountered in the past, how relevant was each step (each lag) of the sequence in predicting future realizations, or what is the informativeness of each step when put in the context of this particular sequence. In terms of intuitive resemblance to econometric models, the self-attention mechanism seems to be closer to a regime-switching HAR model. The essence of self-attention is to be able to contextualize each token in the input sequence by using the same input sequence as a context filter. It uses similarity scoring to find what is the context and then determines which weight to put on each token for this context. A regime-switching HAR would also try to switch the weight attributed to each input conditional on the current regime (context).

Multi-head Self Attention : In practice, self-attention is often implemented through a mechanism known as multi-head attention (Figure 2). This involves running multiple self-attention operations in parallel, each with its own set of linear transformations (i.e., its own W^Q, W^K, W^V). The outputs of these multiple attention heads are then concatenated and linearly transformed again to produce the final output. This allows the model to jointly attend to information from different representation subspaces, capturing various aspects of the input sequence.
If h is the number of heads, the multi-head attention can be represented as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O$$

where each head is defined as:

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

and W^O is the weight matrix for the linear transformation applied after concatenation.
One of the key advantages of self-attention is its ability to handle long-range dependencies with ease. Unlike RNNs, which process sequences step by step, self-attention processes an entire sequence at once, allowing it to consider the full context in one step. This leads to significantly improved training times since parallelization becomes feasible.
Moreover, self-attention provides an interpretability aspect to the neural network's decision-making process, as the attention weights can be analyzed to understand which parts of the input sequence are deemed important when processing a particular element.
In most versions of the transformer architecture, the output of the attention mechanism is passed through a feed-forward network, which is applied to each position separately and identically. This can be represented as:

$$\text{FFN}(x) = \text{ReLU}\left(xW_1 + b_1\right)W_2 + b_2$$

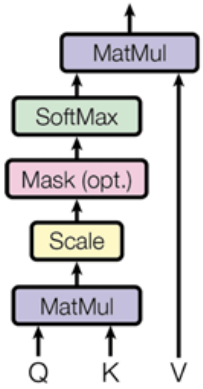


Figure 1: Self-attention mechanism as presented in Vaswani et al. (2017)

The mechanism takes as input three representations of the sequence, which are typically referred to as queries (Q), keys (K), and values (V). These representations are produced through matrix multiplication of the input embeddings with trainable weights. In practice, Q, K, and V might be identical in self-attention, as they come from the same previous layer output. The first step in the self-attention mechanism is to calculate the dot product of the queries with the keys. This operation measures the compatibility or similarity between different positions in the input sequence. The result is a weight matrix where each element represents the attention score from one element to another. The attention scores are then scaled down by dividing them by the square root of the dimension of the keys. This scaling helps in stabilizing the gradients during training, as it prevents the dot products from becoming too large and leading to very small gradients. An optional masking step can be applied, which is typically used in contexts where certain positions should not be attended to, such as padded elements or, in the case of the decoder, to prevent attending to future tokens (for causality) ; The scaled attention scores are passed through a softmax function, which converts them into probabilities. The softmax operation is performed on each row of the matrix, so each row sums up to 1. This step ensures that the attention weights across the sequence add up to 1, forming a probability distribution ; The output of the softmax function is then used to weight the values (V). The attention probabilities are multiplied with the values to produce a weighted sum, which represents the output of the attention mechanism for each position ; Finally, the results of the weighted sum are combined (typically through another matrix multiplication) to produce the final output of the self-attention layer, which then goes to subsequent parts of the Transformer model.

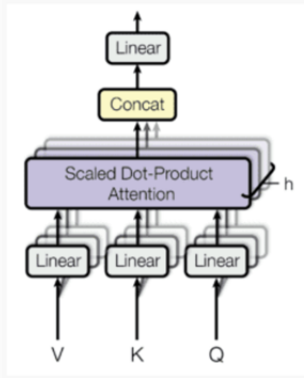


Figure 2: Multi-Head Self-Attention as presented in Vaswani et al. (2017)

This figure depicts us the Multi-Head Self-Attention. Initially, the input vectors are linearly transformed into queries (Q), keys (K), and values (V) for each attention head ; For each head, scaled dot-product attention is computed just as in the single-head case (as illustrated in the previous image). The queries are dotted with the keys, scaled, and then a softmax function is applied to determine the attention weights. These weights are then used to compute a weighted sum of the values ; The output from each head is then concatenated, effectively combining the different learned representations; Finally, this concatenated output is again linearly transformed. This step integrates information from all the heads to produce the final output for the multi-head attention layer. By comparing it to the previous single self-attention mechanism, multi-head attention can be thought of as running several instances of self-attention in parallel (each 'head' being an instance), with the concatenated result capturing a multi-faceted representation of the input sequence. This allows the model to pay attention to different parts of the sequence in varied ways simultaneously, offering a more complex and nuanced understanding than the single-head self-attention

4.2.2 Long-Short Term Memory Network (LSTM)

Long Short-Term Memory (LSTM) network, introduced in 1997 by Hochreiter and Schmidhuber (1997), represents a significant advancement in the field of deep learning, particularly for tasks involving sequential data such as time series analysis and natural language processing. Their unique architecture allows them to remember long-term dependencies, addressing the vanishing gradient problem common in traditional recurrent neural networks (RNNs). The key innovation in LSTM is its ability to maintain a long-term state or memory, which is adjusted through structures called gates.

An LSTM unit consists of a cell (which carries the state across sequence steps, see Figure 4) and three types of gates that regulate the flow of information: the input gate, the forget gate, and the output gate. The ability of the LSTM to maintain memory is provided by the cell state, which is the horizontal line running through the top of the cell. The cell state runs through the entire cell with minor linear transformations performed by the gates. These gates control the extent to which new input should be incorporated into the memory, the degree to which the existing memory should be forgotten, and how much of the memory should contribute to the output, respectively.

The central concept in LSTM is the cell state, denoted as C_t, which runs straight down the entire chain of the network. It can carry relevant information throughout the processing of the sequence. Because of the gated mechanism, information can be added or removed from the cell state.

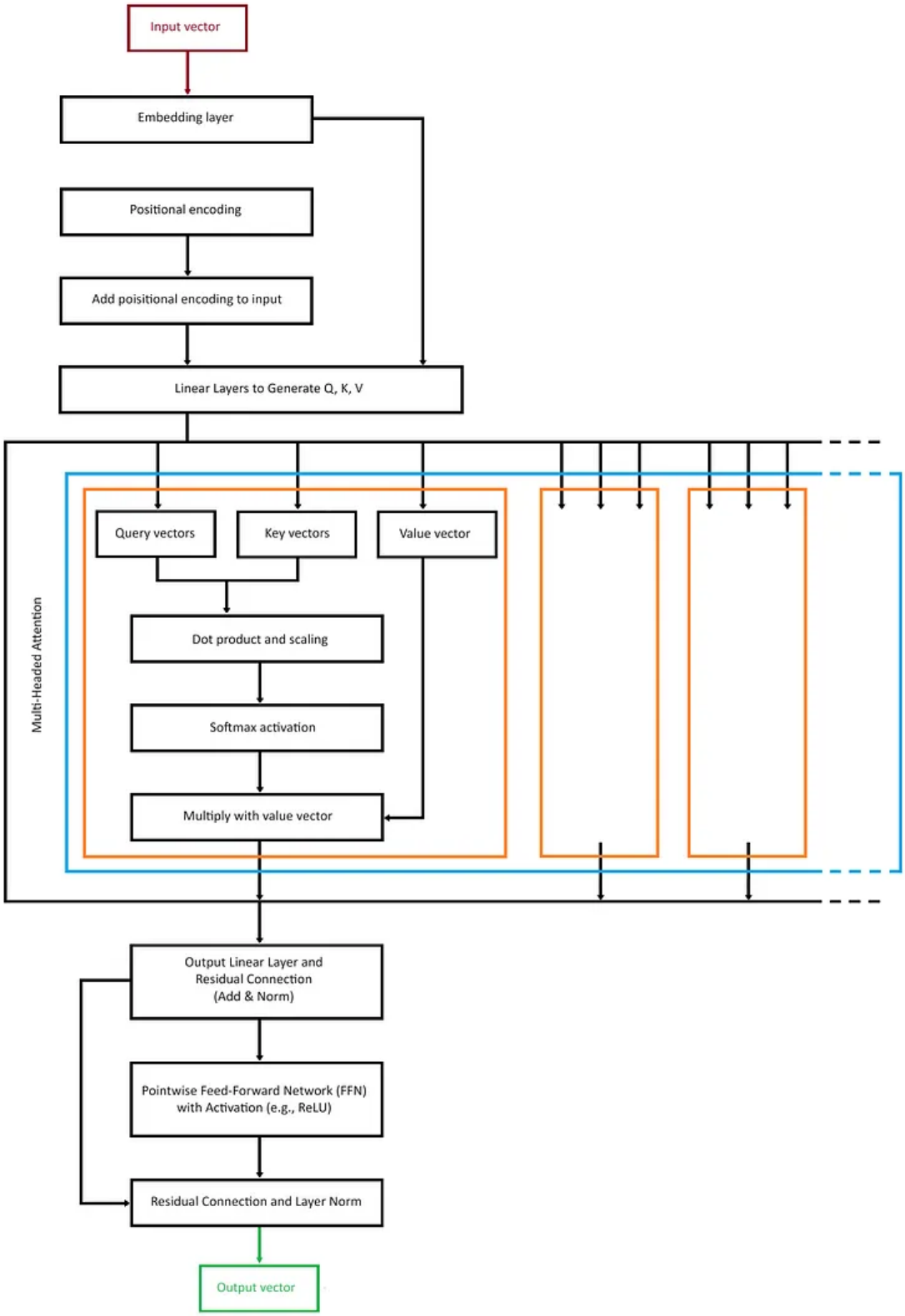


Figure 3: Image from Del-Pra (2023) showing the original encoder architecture as proposed by Vaswani et al. (2017)

The encoder maps an input sequence to a sequence of continuous representations. For each input, embeddings are summed with positional encodings to inject sequence order information. The encoder consists of N identical layers, each with two sub-layers: a multi-head self-attention mechanism and a position-wise feed-forward network, both followed by an add & norm step which applies residual connections and normalization.

- **Forget Gate (f_t)**: As illustrated on Figure 5, this gate decides what information should be discarded from the cell state. It looks at h_{t-1} (the previous hidden state) and x_t (the current input) and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. A 1 represents "completely keep this" while a 0 represents "completely get rid of this".
- **Input Gate (i_t) and Candidate Cell State ($\tilde{C}_t$)**: The input gate decides which values to update, and a candidate layer creates a vector of new candidate values that could be added to the state, as seen on Figure 6 and Figure 7
- **Output Gate (o_t)**: As seen on Figure 8, the output gate decides what the next hidden state h_t should be. The hidden state contains information about previous inputs and is used for predictions.

The operations within an LSTM unit can be formulated as follows:

- **Forget Gate**:
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
- **Input Gate**:
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$
- **Cell State Update**:
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
- **Output Gate and Hidden State**:
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

Where:

- σ represents the sigmoid function.
- W **and** b terms denote weight matrices and bias vectors, respectively.
- $[h_{t-1}, x_t]$ denotes the concatenation of the previous hidden state and the current input.

Training LSTM networks involves backpropagation through time (BPTT), a process similar to back-propagation in other neural networks, but with the unrolled sequence of states. It requires careful handling of gradients, as they can grow or decay exponentially over long sequences (the problem LSTM was designed to address). It is interesting to note that at their core, both classical RNNs and LSTMs draw from the same ideas as GARCH and CGARCH models, which is auto-regression. The two main differences reside in how each model treats

lags of the target variable, and the non-linearities present in RNNs and LSTMs.
Essentially, considering the basic simplified LSTM cell gives the following output:

$$h_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) * \tanh(\sigma(W_f \cdot [h_{t-1}, x_t] + b_f) * C_{t-1} + \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) * \tanh(W_C \cdot [h_{t-1}, x_t] + b_C))$$

with the only three inputs being $x_t, h_{t-1}, and\ C_{t-1}$.

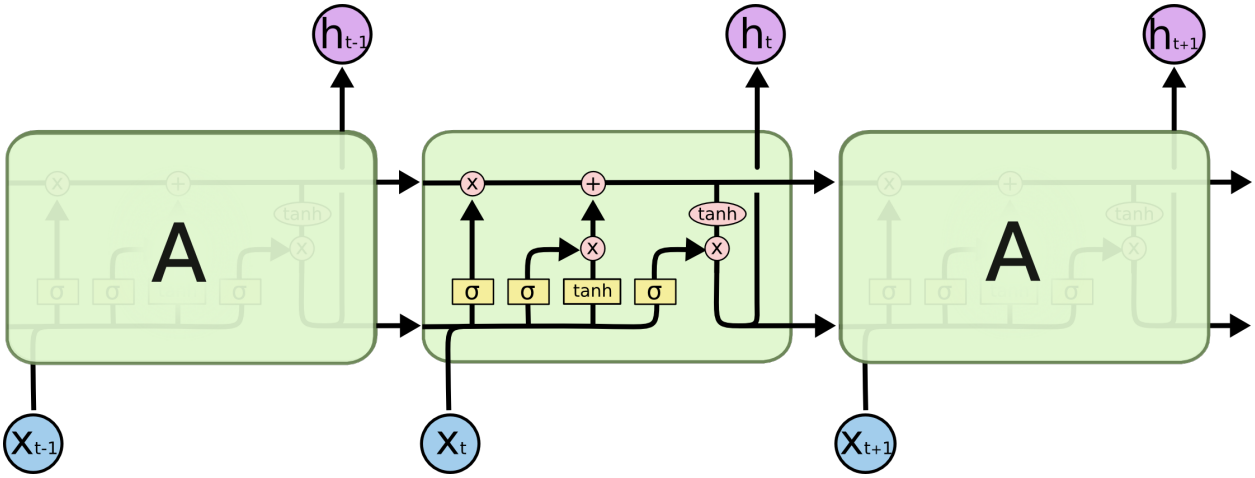
The GARCH model gives the following output:

$$\sigma_t^2 = \omega + \sum_{i=1}^{p} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2$$

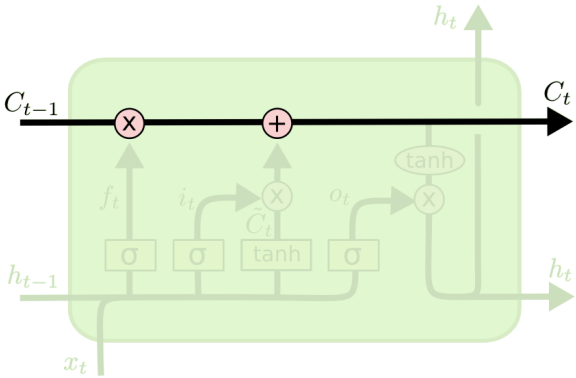
Making abstraction of the non-linearities in the LSTM, we can see that both models output a latent variable at step t, by using as input lags of this latent variable, and new "exogenous" inputs (x for LSTM and ϵ for GARCH). On top of these inputs, the LSTM uses a third input which is the previous cell state C_{t-1}. The previous cell state can be interpreted as a fine-tuned "long memory" or permanent component added to account for longer lags. In this capacity, this third input makes the LSTM architecture very similar to a Component GARCH.
The long memory contained in the cell state is considered fine-tuned since at each step it is selectively modified by the three gates to determine which long-term information is worth propagating to the next steps. The gates used for the cell update also take as only inputs $x\ and\ h$. In this sense, the cell update function plays the role of a more flexible version of the ω, ϕ, and ρ parameters in the permanent component equation in the CGARCH model. Consequently, we can roughly consider that an LSTM is an unconstrained Component GARCH with non-linearities.
Despite the similarity, it's important to point out that these two models are very different in terms of their initial design purpose and the methods used to find their parameters.
An LSTM is a non-parametric model with no distributional assumption which is **trained** by minimizing a loss function, meanwhile a model of the GARCH family is a statistical parametric model built on a distributional assumption, and **estimated** through maximizing a likelihood function.

4.2.3 Extreme Gradient Boosted tree (XgBoost)

Weak Learners
Boosting starts with a base learning algorithm to build weak learners. A weak learner is a simple model (like a decision tree or a linear regressor), but it could be any machine learning algorithm that provides better than random predictions.
Decision trees
A decision tree is a widely used non-parametric machine learning algorithm that can be used for both classification and regression tasks. In the context of regression, a decision tree predicts the value of a target variable by learning simple decision rules inferred from the feature data.



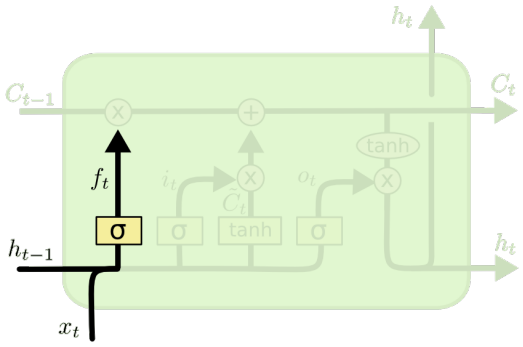
(a) Unrolled Cells



(b) Single Cell

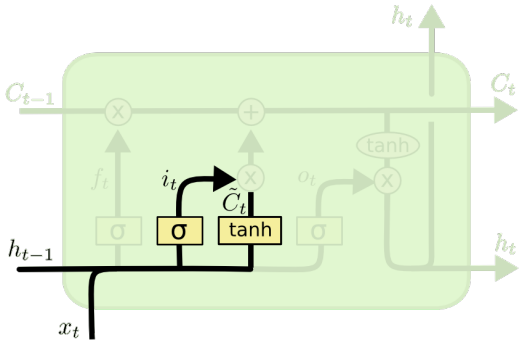
Figure 4: LSTM Cells computational flow from Olah (2015):

Subfigure (a) : The forget gate utilizes a sigmoid (σ) activation to regulate the retention of information from the previous cell state **c**. Simultaneously, the input gate determines potential new information to be added via a sigmoid activation and a tanh-generated vector of candidate values. The cell state is then updated by an element-wise multiplication of the forget gate's output with the old state and an addition of the input gate's scaled new candidate values **x**, allowing the cell to maintain and modify its memory. Lastly, the output gate, through another sigmoid function, filters the updated cell state via a tanh function to produce the next hidden state, which captures the relevant temporal information for the sequence being processed. Both the new hidden state **h** and the updated cell state **c** are forwarded to the next time step in the sequence.
Subfigure (b) : The horizontal line running through the top illustrates how the previous cell's state is modified by the current input and propagated to the next cell.



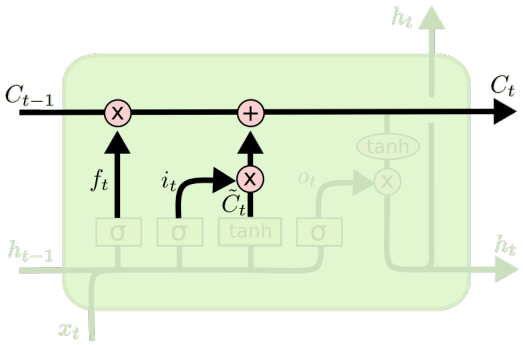
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 5: Forget gate from Olah (2015)



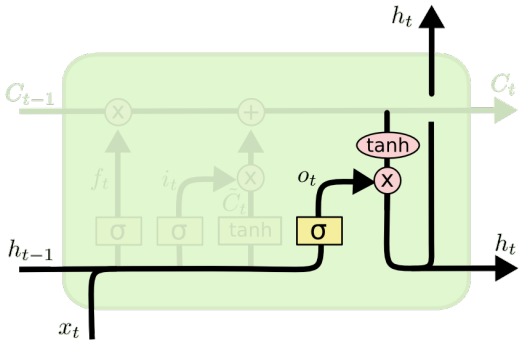
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 6: Input gate from Olah (2015)



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 7: Cell state update from Olah (2015)



$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 8: Output gate from Olah (2015)

The model is structured like an inverted tree with a series of binary splits or decisions. Each internal node of the tree represents a decision on a certain feature, each branch represents the outcome of the decision, and each leaf node represents a final output value (the predicted value for regression tasks). A decision tree splits the data starting from the root node and makes decisions that lead to leaf nodes, where the final continuous output is given.
The entire dataset starts at the root of the tree. The algorithm selects a feature and a split point on that feature to divide the data into two subsets.
The choice of which feature to split on and where to split it is typically based on minimizing a cost function like the Mean Squared Error (MSE) or the Residual Sum of Squares (RSS). The algorithm considers every feature and every possible value of that feature to determine the best split.
This process is recursively applied to each derived subset. The tree continues to grow until a stopping criterion is met (like a maximum depth of the tree, a minimum number of samples required to split a node, etc.). The goal of the process is to be performed multiple times recursively during the training process until only homogeneous nodes are left.
For making a prediction, a new data point is passed down the tree. At each node, a decision is made based on the value of the corresponding feature until a leaf node is reached. The value at the leaf node gives the predicted output.

Regression trees
As a more specialized decision tree, regression trees (Figure 9), a pivotal element in the realm of machine learning, stand out for their efficacy in managing regression problems through a structured, tree-based approach. The origins and evolution of regression trees are intertwined with the development of decision trees. Breiman et al. (1984) CART algorithm forms the conceptual basis for regression trees. It operates by systematically dividing the predictor space into distinct segments, forming a binary tree structure. Just like a general decision tree, this structure is characterized by its internal nodes, which correspond to the input features, and branches, which represent the decisions or splits made based on these features. The culmination of these branches is the terminal nodes or leaves, which hold the predicted values or leaf weights. These leaf weights are essentially the outcomes of the regression tree for the various partitions of the input space. As explained in Teller et al. (2022), for a given input element x_i in a data set S of size n, the regression tree delineates the input space into K distinct regions $R_1, ..., R_K$. The model assigns a predicted value $\hat{y}_i$ to each input element, calculated as the sum of the estimated leaf weights $\hat{w}_k$ for the region to which x_i belongs, expressed through the equation:

$$\hat{y}_i = \sum_{k=1}^{K} \hat{w}_k I(x_i \in R_k)$$

Here, I represents the indicator function, determining whether the input element falls within a specific region R_k. The estimation of leaf weights $\hat{w}_k$ for each region is achieved by minimizing a loss function, typically a squared error loss, across the data points within that region.
The process of determining the best way to split the input space and form these regions is a critical aspect of the CART algorithm. It employs a method known as recursive binary

splitting, combined with least squares optimization, to minimize the residual sum of squares (RSS). This optimization is pivotal in the context of regression trees as it guides the algorithm in evaluating all possible combinations of input variables and their respective threshold split candidates. The RSS is calculated by the equation:

$$\text{RSS} = \sum_{i=1}^{n} \sum_{k=1}^{K} (y_i - \hat{w}_k)^2 I(x_i \in R_k)$$

This equation reflects the greedy nature of regression trees, as the algorithm at each step of tree growth makes locally optimal choices aiming to find the most effective global solution, meaning the split and number of regions that minimize the RSS.

In the broader context of machine learning, ensemble models have built upon and enhanced the basic structure of regression trees. Ensemble methods, recognized for their superior performance over single-algorithm approaches, combine multiple weak learners, often decision trees, to create a more robust model. As stated above, Boosting, a key ensemble technique, iteratively adds new learners to previously fitted ones, focusing on improving predictions, especially for data points that earlier models did not accurately predict.

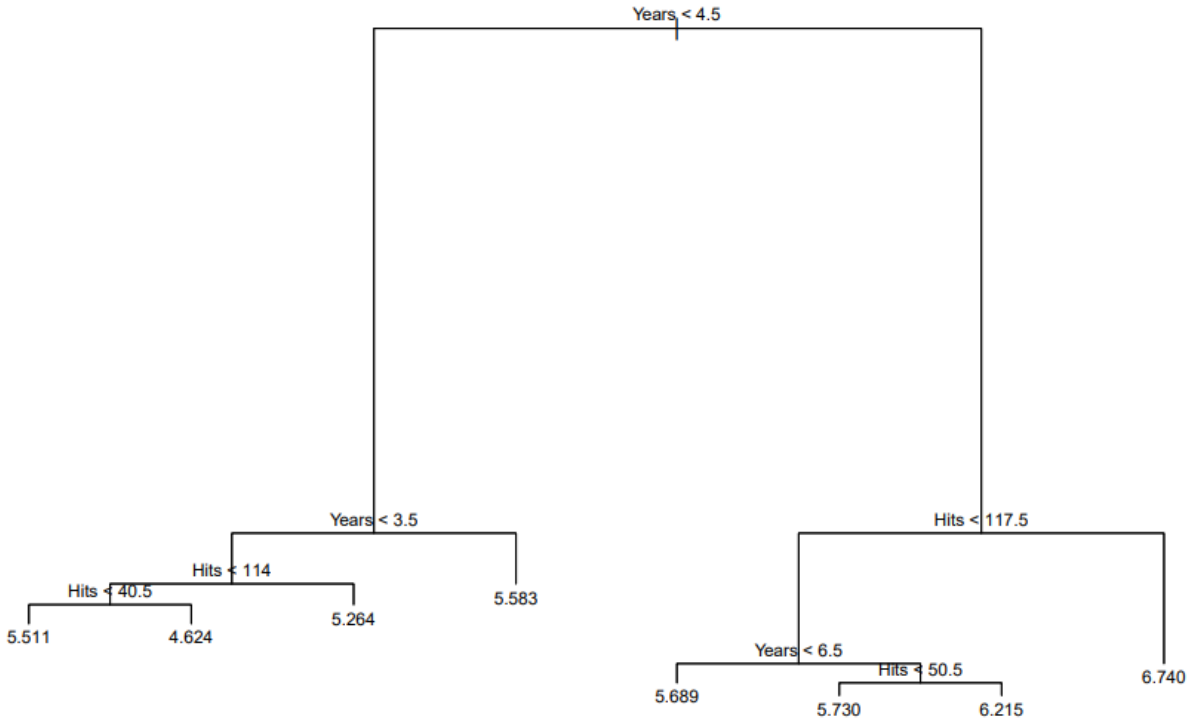


Figure 9: Regression tree for predicting log salary from hits and years played as seen in Steorts (2017)

At each internal node, we ask the associated question, and go to the left child if the answer is "yes", to the right child if the answer is "no".

Boosting and Extreme Gradient Boosting
Boosting is a machine learning ensemble technique that aims to create a strong classifier from several weak classifiers. With boosting, base learners are estimated sequentially, and each base learner aims to reduce the error of its predecessors. The first base learner is trained on the entire dataset and its predictions are used to evaluate its performance using a loss function. In each subsequent iteration, the boosting algorithm adjusts the weight of training instances based on the performance of the previous model. Instances that were misclassified or had higher errors are given more weight, making the algorithm focus more on these in the next iteration. A new model is then trained on the reweighted data. The goal is for this new model to perform better on the instances that the previous model got wrong. The predictions from all models are then combined to make a final prediction. This is typically done through a weighted vote or average, where more accurate models have a bigger influence on the final prediction. As the process continues, the boosting algorithm iteratively improves the accuracy of the overall model. The process is repeated, and new models are added until either a preset number of models are added or no further improvements can be made. The ensemble of weak learners evolves into a single strong learner. To avoid overfitting, recent boosting algorithms usually include regularization techniques, such as limiting the number of weak learners or shrinking the contribution of each through a learning rate.
Extreme Gradient Boosting (XGBoost) , recently intoduced by Chen and Guestrin (2016), is an advanced implementation of gradient boosting algorithm. It has gained popularity due to its efficiency and effectiveness in solving various classification and regression problems. Its main innovation lies in its introduction of row and column sub-sampling to prevent over-fitting and reduce training time, and shrinkage for regularized learning. In the most common implementation of XgBoost as well as for our study, the base learners are regression trees.
In each iteration, Xgboost finds the best split by choosing the features and split points that yield the highest gain. After growing a tree, XgBoost prunes it using the concept of 'depth-first' approach, where splits that provide negative or non-significant gain are removed. Xgboost introduces a shrinkage factor to slow down the learning in each step, adding robustness to the model. It also uses feature sampling to prevent overfitting, similar to random forests. Xgboost effectively combines decision trees, gradient boosting, and regularization, making it a powerful tool for time-series forecasting.
Essentially with XgBoost, at each new iteration, the regularized errors of the previous tree are used as an input to the currently trained tree. Since the model produces a point forecast of our volatility, the closest intuitive parallel to one of our benchmark models would be to view XgBoost as a genetic swarm of HAR models each trained on the unexplained residuals of its predecessor. In this analogy, each new HAR model would use the same conventional RVs inputs, but would also add the last HAR's residual as input.

4.2.4 Natural Gradient Boosted tree (NgBoost)

Natural Gradient Boosting, proposed in 2020 in Duan et al. (2020), is an ensemble model for probabilistic prediction via gradient boosting. The main innovation of this model is to generalize gradient boosting to probabilistic regression by treating the output variable conditional distribution parameters as targets for a boosting algorithm. It therefore comes

inherently with a straightforward way to compute confidence intervals. Its attractiveness relies upon its great flexibility, in the sense that it can be used with any base (weak) learner, any family of parametric probability distributions, and any scoring criteria. Knowing the dynamic, jump-displaying non-gaussian nature of most assets realized volatility, the NgBoost model appeared to be an interesting candidate for modeling this variable. This study is the first in the literature to attempt to use NgBoost for volatility forecasting.
The three key elements of the NGBoost algorithm are The base learner, the parametric probability distribution, the scoring rule.
Base Learners: NGBoost can use any regression algorithm as its base learner. Common choices include decision trees and linear models.

Parametric Probability Distributions: The method assumes a parametric form for the conditional probability distribution $P(y|x)$. It estimates the parameters of this distribution as functions of the input features x.

Proper Scoring Rules: NGBoost uses scoring rules to compare the estimated probability distribution to the observed data. The scoring rule must be proper, meaning it assigns the best score to the true distribution. Common scoring rules include the Logarithmic Score for Maximum Likelihood Estimation (MLE) and the Continuous Ranked Probability Score (CRPS), often considered a more robust alternative to MLE.

The key innovation in NGBoost is the use of the natural gradient rather than the conventional gradient for parameter updates. The natural gradient takes into account the information geometry of the parameter space (the curvature of the function to optimize), leading to more efficient and stable learning.
The natural gradient $\tilde{\nabla}$ is defined as:

$$\tilde{\nabla} S(\theta|y) \propto I_S(\theta)^{-1} \nabla S(\theta|y)$$

where

- $I_S(\theta)$ is the Fisher Information Matrix for the scoring rule S
- $\nabla S(\theta|y)$ is the standard gradient of the scoring rule with respect to the parameters θ.

In essence, NgBoost predicts $y|x$ in the form of a probability distribution P_θ for which the parameters θ represent the outputs obtained from our base learners. For example, if we consider the conditional distribution of $y|x$ to be normal with $\theta = (\mu, \sigma)$, we will train our base learners at each stage to forecast the parameters θ by minimizing a scoring rule. The twist with NgBoost is that instead of using the classical ordinary gradient for finding the direction of steepest descent, we use the natural gradient. The argument invoked by the model's creators Duan et al. (2020) is that the ordinary gradient is not invariant to reparametrization. This essentially means that the "distance" between two values for a parameter does not correspond to an appropriate "distance" between the distributions characterized by these two values. The natural gradient gives us the invariant direction of steepest descent.

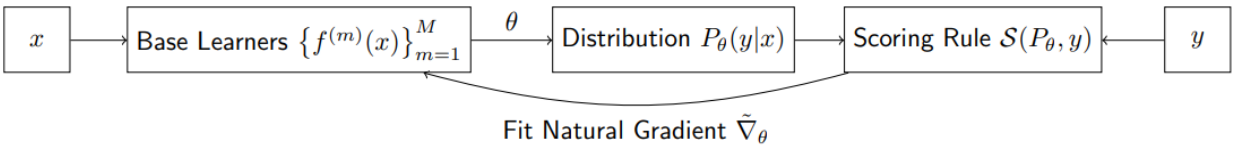


Figure 10: NgBoost learning loop as illustrated by Duan et al. (2020)

In summary, NgBoost learns the parameters sequentially, where each base learner is trained to correct the residuals (errors) of the ensemble so far. The use of the natural gradient makes the updates more efficient and invariant to the choice of parametrization, leading to better training dynamics and performance.
The NgBoost model provides an interesting extension to XgBoost in the sense that it allows for probabilistic forecasting using the same theoretical architecture as XGBoost. The only two differences between them are that NgBoost uses natural gradient descent instead of ordinary gradient descent, and NgBoost tries to maximize a similarity score on the distribution instead of minimizing an error function on point forecasts. Because of the latter, our NgBoost trained with RV lags inputs to forecast RV is intuitively similar to a genetic swarm of AR-GARCH models for which the AR mean process is the volatility.

4.3 Ensemble models

Ensemble learning in machine learning is a robust methodology that involves combining multiple models to improve the overall performance, accuracy, and robustness of predictions. The core idea behind ensemble learning is that a group of already trained learners can come together to form a stronger learner, thereby improving the model's predictions. This approach is particularly beneficial in reducing errors that might arise from a single model misspecification as well as the risk of overfitting, in enhancing the accuracy of predictions, and in dealing with the bias-variance trade-off more effectively.
The development of ensemble models stems from the realization that different models capture different patterns in the data, and thus, combining these models can lead to more comprehensive and accurate predictions. There are primarily three types of ensemble learning methods: Bagging, Boosting, and Stacking, each with its unique mechanism and purpose. As mentioned above, two of the models already covered (XgBoost and NgBoost) are Boosting Ensemble models.
Stacking, or Stacked generalization, is a technique that involves training multiple models on the same data and then using another model, often referred to as a meta-learner, to combine their predictions. The base learners are typically different kinds of algorithms, making stacking a heterogeneous approach. The meta-learner, which could be a different algorithm, is trained on the outputs of the base learners, learning how to best combine their predictions to improve accuracy. Stacking is particularly effective when the base models are significantly diverse, as the meta-learner can effectively capture the different aspects of the data represented by each base model. There exist numerous ways to construct a meta-learner or to combine models' predictions using Stacked generalization, the most common being averaging, weighted averaging, or feeding the Single models' outputs to a subsequent machine learning model.

In comparing these methods, bagging is known for its simplicity and effectiveness in reducing overfitting, especially in the case of complex models like decision trees. Boosting, while more complex, is generally more powerful in terms of performance, capable of improving both bias and variance but at the cost of being more sensitive to noise and outliers. Stacking, being the most sophisticated, can yield even better results by optimally combining diverse models, but it comes with the complexity of training and tuning multiple layers of models. In this study, we chose to make use of stacking by using single-layer feed-forward neural networks as meta-learners, because of the flexibility provided by these architectures. (See Hornik et al. (1989))

4.3.1 X-N-L-T

Our first ensemble model consists of a combination of the four previous ones with a single layer 4 neurons feed-forward network as an ensembling method. We chose to have a Feed-Forward aggregation instead of the averaging aggregation frequently encountered in the literature to allow for more flexibility. We especially try to account for the fact that models' forecast performance is often conditional on market conditions by letting the Feed-Forward Network learn the weight to attribute to models' forecasts depending on market conditions. Essentially, the X-N-L-T model is as follow:

$$\sigma_{t+k} = FFN(X_t) = ReLU(w_1 Transformer(X_t) + w_2 LSTM(X_t) + w_3 XgBoost(X_t) + w_4 NgBoost(X_t))$$

4.3.2 CGARCH enhanced models

This category of models consists of combinations of each of the 5 previous models with our CGARCH model using a 2-neuron single-layer Feed-Forward network, which is essentially a multi-linear regression with a non-linearity (Figure A.5). For each of the 5 ML models, the FFN takes as input the output forecasts from the ML model and the CGARCH model.

CGARCH enhanced Transformer

$$\sigma_{t+k} = FFN(X_t) = ReLU(w_1 Transformer(X_t) + w_2 CGARCH(X_t))$$

CGARCH enhanced Long-Short Term Memory Network

$$\sigma_{t+k} = FFN(X_t) = ReLU(w_1 LSTM(X_t) + w_2 CGARCH(X_t))$$

CGARCH enhanced Extreme Gradient Boosted tree

$$\sigma_{t+k} = FFN(X_t) = ReLU(w_1 XgBoost(X_t) + w_2 CGARCH(X_t))$$

CGARCH enhanced Natural Gradient Boosted tree

$$\sigma_{t+k} = FFN(X_t) = ReLU(w_1 NgBoost(X_t) + w_2 CGARCH(X_t))$$

X-N-L-T - CGARCH (Figure A.6)

$$\sigma_{t+k} = FFN(X_t) =$$

$$ReLU(w_1 Transformer(X_t) + w_2 LSTM(X_t) + w_3 XgBoost(X_t) + w_4 NgBoost(X_t) + w_5 CGARCH(X_t))$$

5 Empirical methodology

As stated above, our study's goal is to compare the versatility and the performance of various learning algorithms on 20 forecasting problems. The problems differ on two axes: Assets and horizon. It essentially comes down to producing a point forecast for an asset's realized daily volatility at t+h. We outline below how we built our datasets for the machine-learning models.

5.1 Rolling window

For this experiment, since our explanatory variables consist of lags of our target variables, we adopt a static sliding rolling window method for building our input and target pairs dataset. The sliding step size is 1. The sliding rolling window methodology, illustrated in (Figure A.7), takes m lags of the realized volatility in order to forecast our target horizon. Let's denote the time series as $\{X_t\}$, where $t = 1, 2, \ldots, T$, and T is the total number of observations. For a rolling window of size m, the training data at time t (for $t > m$) would be $\{X_{t-m}, X_{t-m+1}, \ldots, X_{t-1}\}$. The model uses this data to predict X_{t+h}.
At each time step t, with $t > m$, the model is trained on the window of observations:

$$W_t = \{X_{t-m}, X_{t-m+1}, \ldots, X_{t-1}\}$$

Based on the training data in W_t, the model predicts the next value $\hat{X}_{t+h}$, which is then compared to the actual observed value X_{t+h} to compute the prediction error.
After the prediction at time t, the window is updated by including the next observation X_t and excluding the oldest observation X_{t-m}, and the model moves to the next time step $t+1$. The updated window W_{t+1} becomes:

$$W_{t+1} = \{X_{t-m+1}, X_{t-m+2}, \ldots, X_t\}$$

This process continues until the end of the time series.

At T_0 we use the values between $[T_{0-m}\ ,\ T_0\ ]$ as input to As discussed, the number of lags m was a hyperparameter for each [A-H-M] model. Because of computing resource constraints, we estimate all the hyperparameters only on the 60-day-ahead forecasting version of each asset class for each model. We then use the same architecture for the N-days ahead forecasting problems.

5.2 Dataset partitioning

An important part of our research consists in estimating and selecting among a substantial set of models varying by hyperparameters. We adopt a variation of a popular model selection methodology used by Kelly and Xiu (2023), the fixed training-validation split. Since we evaluate and compare the performance of both single and ensemble models, for each forecasting horizon and each asset, we divide our dataset into four parts : a training set D1 for the single models (50% of dataset), a validation set D2 for the single models(10% of dataset), a training set D3 for the ensemble models(20% of our dataset), and a final test set D4 (20% of dataset), on which we test both the single and ensemble models. Our main concern with proceeding with such a split was to prevent data leakage between the ensemble models and the single models since the ensemble model's inputs consist of outputs from the single models. The splits are temporally ordered, meaning that historically, D1 occurs before D2, which occurs before D3, which occurs before D4. We proceed in such a way to prevent information leakage backward in time (See Kelly and Xiu (2023)). The final test set D4 was only visited once at the end for generating the results tables after all models had been trained on their respective training and validation sets. This means that none of the models (either simple or ensemble) had seen the data in D4 during training or validation, only at the end for generating evaluation and comparison metrics.

5.3 Hyperparameter tuning

In Table 2, we detail all the hyperparameters tested on the [**A-60**] forecasting models. The tuning is performed by grid-search.

6 Results, comparison, and discussion

6.1 Final models architectures

In this section, we see the final values found by our grid-search hyper-parameters tuning process on the [**A-60**] models. The results are displayed in Table 3:

6.2 Results and discussion

In this section, we discuss and compare the final out-of-sample performance of our models. First of all, the out-of-sample performance measured by RMSE, Quasi-likelihood, and Over-valuation Frequency of each model grouped by assets are found in Tables 4 to 8.
Secondly, in order to provide rigorous statistical backing for our discussions, we display the results of models vs benchmarks MGW tests for equivalent predictive abilities for each [**A-H**] problem, which can be found in Table 9. The displayed value in each cell of the table gives us the answer to the MGW test's question: **Does the model in row have a statistically different predictive ability than the Benchmark in Column ?**.
Finally, we display the results of the MCS test at $\alpha = 0.01$ for [**A-H**] groups in Table 10. This essentially tells us what's the smallest best set of superior models for each [**A-H**] problem with a 99% confidence level. Framed differently, it shows us the set of models for which

predictive ability is superior to the rest, but not differentiable among themselves at a 99% confidence level.

To evaluate and compare our model's predictive ability, we use this extensive framework to ensure the robustness of our findings. First, we review the more classical out-of-sample performance metrics displayed in tables 4 to 8. Then we discuss the MGW predictive ability comparative test results against the benchmarks, displayed in table 9. Subsequently we undertake an extensive review of the results displayed in table 10 which allows us to perform a multidimensional comparison of our models: It provides us with the direct answer, for every [**A-H**] problem, to the question: What is the best set of models for this forecasting problem?

Models Out-of-Sample performance metrics (Tables 4 to 8)

Following the results of Patton (2011) on the robustness of the Quasi-Likelihood and the MSE (RMSE) to volatility proxies for model comparison, we chose to use these two metrics as performance metrics to compare our models in this section. The overvaluation frequency is also displayed but only for additional informative purposes, and is not used to rank our models. For all the models, regardless of asset and horizon.

Analyzing the 4 tables, here are the most important conclusions to be drawn:
For the equity indices, on every single forecasting horizon, the boosting models outperform all other models (including benchmarks) in terms of RMSE, including the benchmarks. Also for these indices, the CGARCH-enhanced models have in general a higher RMSE than the single models. On gold volatility at every horizon, the transformer and LSTM models are dominant in terms of RMSE. On oil volatility, XgBoost is the most dominant model at every horizon except at h-20 in terms of RMSE and is tied by the LSTM at h-60. The LSTM CGARCH and TRANS CGARCH models are in general the worst-performing models in terms of RMSE, for all the assets.
An interesting thing to note is that aside from a few occurrences, there are only a few models that beat both benchmarks on the Quasi-Likelihood, and this superiority is not robust to asset and horizon. In general, the CGARCH-enhanced models demonstrate superiority over single models according to the Quasi-Likelihood. This result is in line with what is expected from a machine learning auto-regressive model enhanced with a statistical model designed and trained by maximizing its likelihood to fit the distribution of the data. This outlines the gain in terms of goodness of fit from augmenting machine learning models with statistical or econometric ones. These results are nonetheless contradictory with the ones obtained from RMSE, which point towards a dominance of the single boosting models.

In terms of Over-valuation Frequency, there was no significant pattern worth mentioning across assets and horizons. We however deem it valuable to have the information as an additional decision-making metric when deciding between two models that are tied on other metrics for a specific forecasting problem. Overvaluation of volatility forecasts can turn out to be extremely costly depending on the application. Among the single models in general,

aside from a few exceptions, the LSTM and the Transformer display the lowest overvaluation frequency.
In summary, regarding the RMSE, the most striking result appears to be that XgBoost and NgBoost consistently outperform all other models for every horizon for the equity indices. XgBoost is also the best-performing model on oil volatility, except on the 20-day horizon. When comparing the two boosting models against each other on all assets, it seems like for every horizon, Ngboost is almost always outperformed by XgBoost on RMSE, but consistently outperforms XgBoost in terms of Quasi Likelihood and Overvaluation Frequency. In other terms, NgBoost demonstrates a better fit to the conditional distribution of the realized volatility than XgBoost and overestimates less often than the latter. The transformer and LSTM networks are consistently outperforming other models on gold volatility forecasting at every horizon in terms of RMSE.

MGW equivalent predictive ability test against benchmarks (Table 9)

To rigorously evaluate the predictive power of the various models, we delve into an analysis based on the MGW test for equivalent predictive ability. The p-values obtained from these tests are critical indicators, with lower values suggesting that a model's predictive performance is statistically different from the benchmark model's. We set a significance threshold at **0.01** to delineate models that meaningfully outperform the benchmarks.

The first thing we can point out from analyzing this table is that almost none of our models is simultaneously dominant over both horizons and assets in terms of being statistically different from the benchmarks. The only models to breach this are the boosting models, which demonstrate statistically different predictive ability from the benchmarks on every asset and horizon except at h-60 on oil for XgBoost.

On the S&P500 volatility, the single models, as far as they are concerned, display some mixed performance across the different horizons. Most models demonstrate different predictive abilities against the CGARCH model. The most consistently different single models for forecasting S&P500 volatility on multiple horizons seem to be both boosting models, and the most consistently different of the enhanced models is the Trans CGARCH model. For the NASDAQ, similar to the S&P 500, some various performance is recorded for the single models, with the boosting-based models (and their enhanced versions) demonstrating the absolute statistical difference for all horizons, and most models being statistically different from CGARCH on h-10, h-20, and h-60. On the RUSSEL 2000 volatility nonetheless, aside from the boosting models discussed above, their is no clear pattern emerging in terms of different predictive ability against the benchmarks. The MGW results for GOLD tell us that the boosting models are also unequivocally different from both benchmarks, and all models beat the CGARCH models except for LSTM-CGARCH and NG-CGARCH respectively at h-5 and h-15. Nonetheless, both the results for gold also inform us that both the LSTM and transformer network are statistically different from both benchmarks at every horizon. This strengthens what was observed on the RMSE for gold regarding the LSTM and transformer outperforming other models on this asset. Oil is the only asset for which one of the boosting models (XgBoost) is not statistically different from a benchmark (CGARCH).

Finally, the forecasting problems on which our models have demonstrated the least different predictive ability against the benchmarks are: the RUSSEL at h-5, the RUSSEL at h-15, and oil at h-60.

Additional results

Model Confidence Set Best: models at 99% Confidence level (Table 10)

This section analyzes the model confidence sets constructed for each [**A-H**] problem. Before delving into the comparative analysis, we tabulate the frequency of each model's appearance in the MCS results across assets and horizons to discern the most dominant performers:

- **XgBoost** : Contained in 100% of Model Confidence Sets
- **NgBoost** : Contained in 100% of Model Confidence Sets
- **Transformer** : Contained in 100% of Model Confidence Sets
- **LSTM** : Contained in 92% of Model Confidence Sets

The MCS results, when viewed in totality, reveal the dominance of certain models that are robust across horizons and assets. Surprisingly, the 4 models that happen to appear the most are our 4 single models. The two boosting models model particularly stand out again for their adaptability and resilience, indicating that they encapsulate a balance of responsiveness and structural understanding that applies different market conditions, asset-specific conditions, regimes, and forecasting windows. Both boosting models appear in virtually every set of best models.
Their dominant presence across all assets and horizons is indicative of their robustness. The model's ability to leverage the predictive power of boosted weak learners likely contributes to their wide applicability, specifically to this auto-regressive forecasting endeavor.

The transformer model also appears in all the best model sets. This suggests the effectiveness of the underlying self-attention mechanism in extracting relevant features in diverse market conditions, and its ability to do so without being augmented by an econometric model.
In terms of versatility in both dimensions, we also notice the recurrence of the LSTM model in the best sets of 92% of our [**A-H**] problems. This result is in line with what we would expect from a model that was purposely engineered for sequence modeling. Its absence from a set of best models only happens for forecasting the NASDAQ's volatility at h-60.

We also note that oil is the one with the most models in its model confidence sets across horizons, especially at h-60 where all our models are included in the MCS. This is informative of the fact that the test was unable to distinguish our models from one another in terms of forecasting power. This result is to be expected from a time series that exhibits a unit-root or pure randomness., as there is no trend or mean-reversion to be learned from lagged values. As discussed in the data exploratory section, the oil volatility series on the considered period was the closest to being random, as measured by its 0.47 Hurst exponent.
The MCS results pointed us toward believing that our single models are the most versatile across the two dimensions considered for this task.

Discussion

Overall, the extensive evaluation framework solicited in this study didn't point us toward a unanimously superior model demonstrating absolute versatility across our two analysis dimensions according to every single metric. Nonetheless, there was a distinctive tendency for the boosting models to stand out. These models were able to:

- Score lower RMSE than all other models (including benchmarks) for almost every asset, the only exception being the commodities (mainly gold).
- Demonstrate statistically different predictive ability against the benchmarks on every asset and horizon with a single exception for XgBoost on oil.
- Be included in the set of best forecasting models with a 99% confidence level for every asset and horizon

The only relevant metric on which our boosting models don't dominate unequivocally is the Quasi-Likelihood. Even without scoring high on Q-LIKE, based on their performance demonstrated from the RMSE and MGW test backed by the MCS test, it seems reasonable to consider that our main finding is that in the context of our analysis, XgBoost and NgBoost are the most versatile models for volatility forecasting. This result is not surprising given the fact both models come down to a stacked ensemble of weak learners. The use of multiple learners to extract different features from our models is very similar to what is done by the attention layers in the transformer, or a convolution layer in a convolutional neural network. In this sense, it is not surprising that the transformer would also appear in 100% of model confidence sets.

Additionally, it is worth mentioning that although they both emerge as superior models, a deeper dive into the results shows us the superiority of NgBoost over XgBoost in terms of goodness of fit, which is not surprising given that NgBoost is trained to forecast the parameters of a distribution.

We know that the superiority of XgBoost over HAR and LSTM for 1 day ahead realized volatility forecasting on multiple stocks was already established by Teller et al. (2022); Similarly, the superiority of GARCH enhanced transformers over GARCH, LSTM, ANN, transformer, and GARCH enhanced LSTM and Transformer for 1 day ahead S&P500 historical volatility forecasting was established by Ramos-Pérez et al. (2021).

The results also pointed us towards the superiority of the LSTM and transformer networks for forecasting gold volatility at any horizon. Those two sequence models appear well-suited for capturing the seasonality from supply and demand, geopolitical and macro influences in gold's volatility.

There was also no model that specifically outperformed the others for a particular value of the horizon dimension, meaning that there was no model that was superior to others on one specific horizon regardless of the asset.

Moreover, it turned out that there was no clear edge or advantage from aggregating all our models together using a Feed-Forward Network for ensembling, with or without a component GARCH enhancement. Neither the X-N-L-T nor the X-N-L-T CGARCH demonstrated versatile superiority on any dimension. This result is hardly surprising; The literature suggests that there is rarely an edge to be gained from merely combining outputs from different

specialized architectures without infusing any domain knowledge or intelligently leveraging each architecture's strength on a different part of the problem. Besides, when comparing the single models to their enhanced versions, enhancing any single model with a CGARCH only results in goodness-of-fit gains (Quasi-likelihood), and does not provide generally more accurate forecasts according to RMSE or MGW/MCS tests.
Reflecting on our study, our results for the non-boosting models do not seem conclusive or clear-cut enough to extract any valuable or trustworthy findings regarding the versatility of these models. Nonetheless, the most interesting and trustworthy results we find are: Xgboost and NgBoost shine the best in terms of versatility and forecasting power; Aside from occasional goodness of fit gains, there is no added forecasting value to enhancing our single models with a component GARCH model; The LSTM and transformer networks are the best at forecasting gold volatility.
The superiority of a boosting model is in line with what we've been seeing in the last two biggest time-series forecasting competitions, the Makridakis et al. (2020) M4 and Makridakis et al. (2022) M5 competitions, where boosting models have come out in the top most robust models for forecasting more than 40000 time-series.

7 Conclusion and directions for future work

The undertaking of this thesis was to compare and explore the versatility and performance of different machine learning architectures in forecasting the realized volatility of 5 popular and liquid assets. We used intraday prices sampled at a 5mn frequency from the TAQ database collected from 2005 to 2021 for 5 assets proxied by ETFs. The versatility was tested along two axes, the forecasting horizon, and the asset on which we performed the forecast.
Besides, we also studied the added value from enhancing our plain machine learning models with a component GARCH model in a stacked ensemble manner using a single-layer feed-forward neural network. We adopted rigorous processing and partitioning of our data to make sure that the final evaluation was made on a subset (D4, or test set) that was never seen during training. In order to obtain sound and trustworthy results from our study, we used an extensive evaluation framework comprising the classical performance metrics encountered in academia (RMSE, QLIKE), as well as two statistical tests to assess and differentiate forecasting power among models.
The main result was that the best models regardless of asset or horizon were the boosting models. The LSTM and Transformer networks were found to be the best models for forecasting gold volatility. There was also no material added-value in enhancing our models with a Component GARCH or to combining them all together.
Nonetheless, several avenues for improvement could be undertaken from our results. First of all, a major shortcoming of the study is that it compared only the pure auto-regressive forecasting power of the architectures, in the sense that we didn't include any exogenous regressor than lags of our response variable, not even the returns series. A more complete and thorough study could investigate the robustness of these architectures when enhanced with exogenous regressors such as macroeconomic variables or other market variables related

to the assets. Secondly, the number of assets and horizons could be easily extended in order to have a more comprehensive assessment of robustness.
Furthermore, given that the state-of-the art econometric models for volatility forecasting are most of the time regime-switching hidden markov models, it would be interesting to explore whether or not regime-switching versions of our architectures demonstrate superior robustness and predictive ability against regime-switching benchmarks. It would also be interesting to explore the extent to which the current version of these architectures (non regime switching) are inherently able to capture the different volatility regimes filtered by a state-of-the-art state-space filter (like the Hamilton filter). Also, this study could be extended to look at the multiple assets' volatility forecasting problem as an actual multivariate forecasting problem.
Besides, we only studied point forecasting, which means we're only forecasting the volatility of one point in time into the future; a more useful variation would be to study how well the models perform in terms of range forecasting, which is forecasting the volatility of every day between today and the forecasting horizon.
Another shortcoming of our study is the lack of confidence intervals around our forecasts, or for our parameters. Since most applications using volatility forecasts as input are heavily sensitive to the uncertainty around the forecasts, it is critical to have a statistical assessment of the uncertainty around a model's parameters and forecasts in the form of confidence intervals. To this effect, a relatively new framework for ML forecasts uncertainty quantification known as conformal prediction has emerged. This framework allows us to produce confidence intervals for any forecasting model, even non-parametric and non-statistical ones. Quantifying uncertainty would also allow us to more rigorously benchmark our models against bayesian forecasting models like Gaussian processes.
Since our models are very data-hungry deep learning models that have been proven to get asymptotically better with more training data, another critical improvement to the study that may alter the results is to augment the size of the training data by going further in the past to gather more volatility clusters and long-term behavior. We were limited in this capacity by the hardware constraints that come with treating high-frequency data and the availability of historical intraday series on our source database.
Also, exploring more alternative training frameworks, loss functions, or switching to a quasi-likelihood maximizing framework instead of loss minimization might have given more interesting results. An interesting avenue in this regard is to experiment with other kinds of weak learners in the boosting models instead of only focusing on trees and linear regressors. Moreover, we could improve the ensemble models built from our single models by experimenting with more ensembling schemes that allow to leverage each architecture's strength instead of the naive non-linear weighted averaging we used here.
Lastly one of the most promising avenues is to extend the number of architectures studied to include the state-of-the-art models for time series forecasting. The most promising direction would be to experiment with the following novel architectures which currently constitute the state of the art: state space models like the N-BEATS by Chapados, Bengio, Oreshkin et al. (2020), the N-HITS by Challu et al. (2022), Physics Informed Neural Networks (Cuomo et al. (2022)), the attentional copulas transformer by Ashok et al. (2023) from ServiceNow and MILA, or more recently, the mamba model by Gu and Dao (2023) which is recognized, as of our final submission, as the current frontier in ML time-series forecasting.

References

Yakov Amihud and Haim Mendelson. Trading mechanisms and stock returns: An empirical investigation. *Journal of Finance*, 42:533–553, 07 1987.

Torben G. Andersen, Tim Bollerslev, Francis X. Diebold, and Paul Labys. *Econometrica*, 71(2):579–625, March 2003. ISSN 1468-0262.

Arjun Ashok, Etienne Marcotte, Valentina Zantedeschi, Nicolas Chapados, and Alexandre Drouin. Tactis-2: Better, faster, simpler attentional copulas for multivariate time series. *ArXiv*, abs/2310.01327, 2023. URL https://www.servicenow.com/blogs/2024/tactis-2-time-series-prediction.

Yacine Aït-Sahalia, Per A. Mykland, and Lan Zhang. Ultra high frequency volatility estimation with dependent microstructure noise. *Journal of Econometrics*, 160(1):160–175, January 2011.

Federico M Bandi and Jeffrey R Russell. Microstructure noise, realized variance, and optimal sampling. *Review of Economic Studies*, 75:339–369, 2008.

Federico M Bandi, Jeffrey R Russell, and Chen Yang. Realized volatility forecasting and option pricing. *Journal of Econometrics*, 147:34–46, 11 2008.

Ole E Barndorff-Nielsen and Neil Shephard. Estimating quadratic variation using realized variance. *Journal of Applied Econometrics*, 17:457–477, 2002.

Melike Bildirici and Özgür Ömer Ersin. Improving forecasts of garch family models with the artificial neural networks: An application to the daily returns in istanbul stock exchange. *Expert Systems with Applications*, 36:7355–7362, 05 2009.

Fisher Black. Studies of stock market volatility changes. *Proceedings of the American Statistical Association*, pages 177–181, 1976.

Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31:307–327, 04 1986.

Daniel Borup, Jonas N. Eriksen, Mads M. Kjær, and Martin Thyrsgaard. Predicting bond return predictability. *Management Science*, 70(2):931–951, 2024.

Leo Breiman, Jerome Friedman, Charles J Stone, and Olshen R A. *Classification and regression trees*. Taylor Francis, 01 1984.

Christian T Brownlees, Robert F Engle, and Bryan T Kelly. A practical guide to volatility forecasting through calm and storm. *Journal of Risk*, 14(2):3–22, 2011.

Andrea Bucci. Forecasting realized volatility a review. *Journal of Advanced Studies in Finance*, 8:94–138, 2017.

Laurent Calvet and Adlai Fisher. Multifractality in Asset Returns: Theory and Evidence. *Review of Economics and Statistics*, 84(3):381–406, 08 2002. ISSN 0034-6535. doi: 10.1162/003465302320259420. URL https://doi.org/10.1162/003465302320259420.

Peter Carr, Liuren Wu, and Zhibai Zhang. Using machine learning to predict realized variance. *Journal of Investment Management*, 18(2):307–327, 09 2020.

Cristian Challu, Kin G Olivares, Boris N Oreshkin, Federico Garza, Max Mergenthaler-Canseco, and Artur Dubrawski. N-hits: Neural hierarchical interpolation for time series forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(4): 6989–6997, 2022.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, page 785–794, 2016.

Eunsuk Chong, Chulwoo Han, and Frank C Park. Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies. *Expert Systems with Applications*, 83:187–205, 10 2017.

Kim Christensen, Mathias Siggaard, and Bezirgen Veliyev. A machine learning approach to volatility forecasting. *Journal of Financial Econometrics*, 06 2022.

Fabienne Comte and Eric Renault. Long memory in continuous-time stochastic volatility models. *Mathematical Finance*, 8(4):291–323, October 1998.

F Corsi. A simple approximate long-memory model of realized volatility. *Journal of Financial Econometrics*, 7:174–196, 11 2008.

Fulvio Corsi, Stefan Mittnik, Christian Pigorsch, and Uta Pigorsch. The volatility of realized volatility. *Econometric Reviews*, 27:46–78, 2008.

Salvatore Cuomo, Vincenzo Schiano, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92, 07 2022.

Gordon A D. A review of hierarchical classification. *Journal of the Royal Statistical Society. Series A (General)*, 150:119, 1987.

Zhifeng Dai, Huiting Zhou, Xiaodi Dong, and Jie Kang. Forecasting stock market volatility: A combination approach. *Discrete Dynamics in Nature and Society*, 2020:1–9, 06 2020.

Moin Dalvi. Gradient boosting algorithms from scratch, 2022. URL https://github.com/MoinDalvs/Gradient_Boosting_Algorithms_From_Scratch. [Blog Post Accessed on 2023-07-20].

Stavros Degiannakis. Multiple days ahead realized volatility forecasting: Single, combined and average forecasts. *Global Finance Journal*, 36(C):41–61, 2018.

Marco Del-Pra. Large language models, 2023. URL https://medium.com/@marcodelpra/large-language-models-1a6eec644b30. [Blog Post Accessed on 2023-11-18].

Francis X Diebold and Robert S Mariano. Comparing predictive accuracy. *Journal of Business Economic Statistics*, 20:134–144, 01 2002.

Glen Donaldson and Mark Kamstra. An artificial neural network-garch model for international stock return volatility. *Journal of Empirical Finance*, 4:17–46, 1997.

Christian Dorion and Nicolas Chapados. Volatility forecasting and explanatory variables: A tractable bayesian approach to stochastic volatility. *SSRN Electronic Journal*, 2012. doi: 10.2139/ssrn.1747945.

Tony Duan, Anand Avati, Daisy Yi Ding, Khanh K Thai, Sanjay Basu, Andrew Y Ng, and Alejandro Schuler. Ngboost: Natural gradient boosting for probabilistic prediction. *Proceedings of the 37th International Conference on Machine Learning*, 119, 06 2020.

Christian L Dunis and Xuehuan Huang. Forecasting and trading currency volatility: an application of recurrent neural regression and model combination. *Journal of Forecasting*, 21:317–354, 2002.

Robert F Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50:987–1007, 07 1982.

Robert F. Engle and Victor K. Ng. Measuring and testing the impact of news on volatility. *Journal of Finance*, 48:1749–1778, 1993.

Robert F Engle and A.J. Patton. What good is a volatility model? *Quantitative Finance*, 1(2):237–245, 2001.

Thomas Fischer and Christopher Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270: 654–669, 2018.

Rémi Galarneau-Vincent, Geneviève Gauthier, and Frédéric Godin. Foreseeing the worst: Forecasting electricity dart spikes. *Energy Economics*, 119, 03 2023.

Jim Gatheral, Jaisson Thibault, and Mathieu Rosenbaum. Volatility is rough. *Quantitative Finance*, 18(6):933–949, 2018.

Geneviève Gauthier. Math80631, chapter 8 - realized variance. [Lecture Note], HEC Montréal, 2017.

Wenbo Ge, Pooia Lalbakhsh, Leigh Isai, Artem Lenskiy, and Hanna Suominen. Neural network–based financial volatility forecasting: A systematic review. *ACM Computing Surveys*, 55:1–30, 01 2023a.

Wenbo Ge, Pooia Lalbakhsh, Leigh Isai, Artem Lensky, and Hanna Suominen. Comparing Deep Learning Models for the Task of Volatility Prediction Using Multivariate Data. Papers 2306.12446, arXiv.org, June 2023b. URL https://ideas.repec.org/p/arx/papers/2306.12446.html.

Felix Gers, J Urgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm learning to forget: Continual prediction with lstm. *IEE*, 2:850–855, 1999.

Eric Ghysels, Arthur Sinko, and Rossen Valkanov. Midas regressions: Further results and new directions. *Econometric Reviews*, 26:53–90, 02 2007.

Francesco Giordano, La Rocca, and Cira Perna. Forecasting nonlinear time series with neural network sieve bootstrap. *Computational Statistics Data Analysis*, 51:3871–3884, 05 2007.

Lawrence R Glosten, ravi jagannathan, and david e runkle. On the relation between the expected value and the volatility of the nominal excess return on stocks. *Journal of Finance*, 48:1779–1801, 12 1993.

Open AI.(2023). ChatGPT (GPT-4). [produce an extensive and detailed paragraph to use as a starting point for a litterature review on the subject of volatility forecasting, proceed step by step], a. URL https://chat.openai.com.

Open AI.(2023). ChatGPT (GPT-4). [i need a comprehensive paragraph detailing the theory and mathematics behing each of the following models : Transformer, lstm, xgboost, har. think step by step.], b. URL https://chat.openai.com.

Open AI.(2023). ChatGPT (GPT-4). [i need a comprehensive paragraph to use as a strarting point for presenting the most documented and recognized stylized facts of volatility. proceed step by step and only use reliable sources], c. URL https://chat.openai.com.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. 2023. URL https://openreview.net/forum?id=AL1fq05o7H.

Shihao Gu, Bryan Kelly, and Dacheng Xiu. Empirical Asset Pricing via Machine Learning. *Review of Financial Studies*, 33(5):2223–2273, 02 2020.

Shaikh Hamid. Primer on using neural networks for forecasting market variables. *Southern New Hampshire University*, 2004. URL https://core.ac.uk/download/pdf/71368014.pdf.

Shaikh A Hamid and Zahid Iqbal. Using neural networks for forecasting volatility of sp 500 index futures prices. *Journal of Business Research*, 57:1116–1125, 10 2004.

Peter R Hansen and Asger Lunde. A forecast comparison of volatility models: does anything beat a garch(1,1)? *Journal of Applied Econometrics*, 20:873–889, 2005.

Peter R Hansen, Asger Lunde, and James M Nason. The model confidence set. *Econometrica*, 79:453–497, 2011.

Peter Reinhard Hansen, Asger Lunde, and James M. Nason. Choosing the best volatility models: The model confidence set approach*. *Oxford Bulletin of Economics and Statistics*, 65(s1):839–861, December 2003.

Milton Harris and Artur Raviv. The theory of capital structure. *Journal of Finance*, 46: 297–355, 03 1991.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 11 1997.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 01 1989.

Michael Y Hu and Christos Tsoukalas. Combining conditional volatility forecasts using neural networks: an application to the ems exchange rates. *Journal of International Financial Markets, Institutions and Money*, 9:407–422, 1999.

H. E Hurst. Long term storage capacity of reservoirs. *Transactions of the American Society of Civil Engineers*, page 770-799, 1951.

IBM Inc. What is a decision tree?, 2023. URL https://www.ibm.com/topics/decision-trees. [Blog Post Accessed on 2023-11-20].

Nasdaq Inc. Nasdaq-100 higher volatility than the sp 500: Myths and truths around volatility, 2018. URL https://www.nasdaq.com/articles/nasdaq-100-higher-volatility-sp-500-myths-and-truths-around-volatility-2018-04-03.

Sang Hoon Kang, Sang-Mok Kang, and Seong-Min Yoon. Forecasting volatility of crude oil markets. *Energy Economics*, 31:119–125, 01 2009.

Bryan T Kelly and Dacheng Xiu. Financial machine learning. *Social Science Research Network*, 07 2023.

Ha Young Kim and Chang Hyun Won. Forecasting the volatility of stock price index: A hybrid model integrating lstm with multiple garch-type models. *Expert Systems with Applications*, 103:25–37, 08 2018.

Werner Kristjanpoller and Marcel C Minutolo. Gold price volatility: A forecasting approach using the artificial neural network–garch model. *Expert Systems with Applications*, 42: 7245–7251, 11 2015.

Werner Kristjanpoller and Marcel C Minutolo. Forecasting volatility of oil price using an artificial neural network-garch model. *Expert Systems with Applications*, 65:233–241, 12 2016.

Gary Lee and Robert F Engle. A permanent and transitory component model of stock return volatility. *SSRN Electronic journal*, 10 1993. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5848.

Bryan Lim, Sercan O Arik, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting, 09 2021. ISSN 0169-2070.

Chun Liu and John M Maheu. Forecasting realized volatility: A bayesian model-averaging approach. *Journal of Applied Econometrics*, 24:709–733, 2009.

Qingfeng Liu, Qingsong Yao, and Guoqing Zhao. Model averaging estimation for conditional volatility models with an application to stock market volatility forecast. *Journal of Forecasting*, 01 2020.

Wenting Liu, Zhilun Gui, Guilin Jiang, Li Tang, Lijun Zhou, Wei Leng, Xulong Zhang, and Yujiang Liu. Stock volatility prediction based on transformer model using mixed-frequency data. *arXiv (Cornell University)*, 09 2023.

Xunfa Lu, Danfeng Que, and Guangxi Cao. Volatility forecast based on the hybrid artificial neural network and garch-type models. *Procedia Computer Science*, 91:1044–1049, 2016.

Chuong Luong and Nikolai Dokuchaev. Forecasting of realised volatility with the random forests algorithm. *Journal of Risk and Financial Management*, 11:61, 10 2018.

Štefan Lyócsa and Peter Molnár. Volatility forecasting of strategically linked commodity etfs: gold-silver. *Quantitative Finance*, 16:1809–1822, 2016.

Ananth Madhavan. Market microstructure: A survey. *Journal of Financial Markets*, 3: 205–258, 08 2000.

John M. Maheu and Thomas H. McCurdy. News arrival, jump dynamics, and volatility components for individual stock returns. *Journal of Finance*, 59:755–793, 2004.

Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36: 54–74, 01 2020.

Navaneeth Malingan. What are encoder in transformers, 2023. URL https://www.scaler.com/topics/nlp/transformer-encoder-decoder/. [Blog Post Accessed on 2023-06-06].

Mary Malliaris and Linda Salchenberger. Using neural networks to forecast the sp 100 implied volatility. *Neurocomputing*, 10:183–195, 03 1996.

Benoit B. Mandelbrot and John W. Van Ness. Fractional brownian motions, fractional noises and applications. pages 422 – 437, 1968.

Michael McAleer and Marcelo C Medeiros. Realized volatility: A review. *Econometric Reviews*, 27:10–45, 02 2008.

Gonzalez Miranda and N Burgess. Modelling market volatilities: the neural network perspective. *The European Journal of Finance*, 3:137–157, 06 1997.

Soheil Almasi Monfared and David Enke. Volatility forecasting using a hybrid gjr-garch neural network model. *Procedia Computer Science*, 36:246–253, 2014.

Daniel B Nelson. Conditional heteroskedasticity in asset returns: A new approach. *Econometrica*, 59:347–370, 03 1991.

Nima Nonejad. An overview of dynamic model averaging techniques in time-series econometrics. *Journal of Economic Surveys*, 35:566–614, 01 2021.

Christopher Olah. Understanding lstm, 2015. URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Blog Post Accessed on 2024-03-20].

Boris N Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-beats: Neural basis expansion analysis for interpretable time series forecasting, 02 2020. URL https://arxiv.org/abs/1905.10437.

Adrian R Pagan and G.William Schwert. Alternative models for conditional stock volatility. *Journal of Econometrics*, 45:267–290, 07 1990.

Andrew J Patton. Volatility forecast comparison using imperfect volatility proxies. *Journal of Econometrics*, 160:246–256, 01 2011.

Perry and Philip R. Time-variance relationship of security returns: Implications for the return-generating stochastic process. *Journal of Finance*, 37:857–870, 1982.

Ser-Huang Poon and Clive. Forecasting volatility in financial markets: A review. *Journal of Economic Literature*, 41:478–539, 06 2003.

Ser-Huang Poon and Clive Granger. Practical issues in forecasting volatility. *Financial Analysts Journal*, 61:45–56, 2005.

Quinlan J R. Induction of decision trees. *Machine Learning*, 1:81–106, 03 1986.

Eduardo Ramos-Pérez, Pablo J Alonso-González, and José Javier Núñez-Velázquez. Forecasting volatility with a stacked model based on a hybridized artificial neural network. *Expert Systems with Applications*, 129:1–9, 09 2019.

Eduardo Ramos-Pérez, Pablo J. Alonso-González, and José Javier Núñez-Velázquez. Multi-Transformer: A New Neural Network-Based Architecture for Forecasting S&P Volatility. *Mathematics*, 9(15):1–18, July 2021.

Steven L Salzberg. C4.5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993. *Machine Learning*, 16:235–240, 09 1994.

Hugo Gobato Souto and Amir Moradi. Forecasting realized volatility through financial turbulence and neural networks. *The Poznań University of Economics Review*, 9, 07 2023.

Rebecca Steorts. Sta325, chapter 8 - tree based methods: Regression trees. [Lecture Note], Duke University, 2017. URL https://wwwhttps://2.stat.duke.edu/~rcs46/lectures_2017/08-trees/08-tree-regression.pdf.

Andreas Teller, Uta Pigorsch, and Christian Pigorsch. Short- to long-term realized volatility forecasting using extreme gradient boosting. *SSRN Electronic Journal*, 2022. URL http://dx.doi.org/10.2139/ssrn.4267541.

John Thickstun. Transformer networks. Technical report, Standford University, 2019. URL https://johnthickstun.com/docs/transformers.pdf.

P Tino, C Schittenkopf, and G Dorffner. Financial volatility trading using recurrent neural networks. *IEEE Transactions on Neural Networks*, 12:865–874, 07 2001.

Simon van Norden and Robert Vigfusson. Regime-Switching Models, A guide to the Bank of Canada Gauss Procedures. Staff Working Papers 96-3, Bank of Canada, 1996. URL https://www.bankofcanada.ca/wp-content/uploads/2010/05/wp96-3.pdf.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

Andrés Vidal and Werner Kristjanpoller. Gold volatility prediction using a cnn-lstm approach. *Expert Systems with Applications*, 157:113481, 11 2020.

Ines Wilms, Jeroen Rombouts, and Christophe Croux. Multivariate volatility forecasts for stock market indices. *International Journal of Forecasting*, 09 2020.

Jonathan H Wright. Testing for a unit root in the volatility of asset returns. *Journal of Applied Econometrics*, 14:309–318, 1999.

Jia Zhai, Yi Cao, and Xiaoquan Liu. A neural network enhanced volatility component model. *Quantitative Finance*, 20:783–797, 02 2020.

Lan Zhang, Per A Mykland, and Yacine Aït-Sahalia. A tale of two time scales. *Journal of the American Statistical Association*, 100:1394–1411, 12 2005.

Hao Zhu, Lu Bai, Lidan He, and Zhi Li. Forecasting realized volatility with machine learning: Panel data perspective. *Journal of Empirical Finance*, 73:251–271, 09 2023.

A Training, testing, and evaluation process

A.1 Loss functions and evaluation metrics

We used 2 loss functions in the training process: the Huber-Loss function on the training set, and the MSE on the validation set. We evaluate and compare each [**A-H-M**] model variant with 2 loss functions: the RMSE loss and the Quasi Likelihood (Q-LIKE) function. We also use the Overvaluation frequency additional metrics to inform us on a different view of model performance.

Huber Loss function Because of its superior robustness, we use the Huber Loss function as a loss function on the training set. The Huber loss function integrates both linear and quadratic scoring approaches. It uses hyperparameter delta δ which is tuned according to the data. The computed loss is linear (L1 loss) for values superior to δ, and quadratic (MSE Loss) for values inferior to δ. Huber Loss is particularly useful in regression tasks where data may contain outliers or non-standard distributions, ensuring stable and robust training of the model.

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for } |y - f(x)| \leq \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases}$$

Mean Squared Error (MSE) MSE measures the average of the squares of the errors. It is more sensitive to larger errors compared to MAE and is defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where y_i is the actual value, $\hat{y}_i$ is the predicted value, and n is the number of observations. It is particularly useful when large errors are more significant than smaller ones.

Quasi Likelihood Function This metric is often used in statistical models to approximate the likelihood when the exact likelihood is either unknown or difficult to compute. In a machine learning context, it serves as a means to assess the likelihood of the model forecasts given the data :

$$QLIKE = \ n^{-1}\sum_{t=1}^{n}\left(\log\left(\hat{y}_t^2\right) + y_t^2\hat{y}_i^{-2}\right)$$

where y_t is the actual value, $\hat{y}_t$ is the predicted value, and n is the number of observations.

Overvaluation Frequency function This metric helps us in evaluating the proportion of forecasts that were above the actual realized values for each model. It is defined as :

$$OF_{t,\bar{T}} = \frac{1}{T}\sum_{t=1}^{T} I_{\hat{y}_t > y_t}$$

where y_t is the actual value, $\hat{y}_t$ is the predicted value, and n is the number of observations.

A.2 Over-fitting prevention

In machine learning, over-fitting occurs when a model learns the training data too well, including its noise and outliers, which reduces its ability to perform well on unseen data. To prevent over-fitting, several techniques are used, such as dropout regularization, batch normalization, and weight decay. In machine learning, over-fitting is a common problem where a model performs well on training data but poorly on unseen data. Regularization techniques are essential for preventing over-fitting, allowing models to generalize better from the training data to unseen data. We will discuss three fundamental regularization techniques: dropout regularization, batch normalization, and weight decay.

Dropout Regularization Dropout regularization is a stochastic technique to prevent over-fitting in neural networks. It works by randomly deactivating a subset of neurons in a layer during each training iteration, which forces the network to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.
For a fully connected layer with input vector $\mathbf{x}$ and output vector $\mathbf{y}$, without dropout, the relationship is given by:

$$\mathbf{y} = \phi(\mathbf{W}\mathbf{x} + \mathbf{b})$$

where ϕ is the activation function, $\mathbf{W}$ is the weight matrix, and $\mathbf{b}$ is the bias vector. When dropout is applied, this becomes:

$$\mathbf{y} = \phi((\mathbf{W} \odot \mathbf{D})\mathbf{x} + \mathbf{b})$$

where $\mathbf{D}$ is a diagonal matrix where each diagonal element D_{ii} is an independent Bernoulli random variable with probability p of being 1.
During training, dropout is applied, and during testing, it is not used but the weights are scaled by p to account for the reduced number of active neurons, which ensures that the expected sum of the inputs remains the same.

Batch Normalization Batch normalization (BN) is a technique to normalize the inputs of a layer to mitigate the problem of internal co-variate shift. BN standardizes the outputs of the previous layer by subtracting the batch mean and dividing by the batch standard deviation.
For a given feature x, BN transforms it as:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$y_i = \gamma\hat{x}_i + \beta$$

where

- m is the size of the mini-batch
- μ_B is the mini-batch mean
- σ_B^2 is the mini-batch variance
- ϵ is a small constant for numerical stability
- γ and β are learnable parameters of the model.

  Batch Normalization allows for higher learning rates and reduces the strong dependence on initialization.

Weight Decay Weight decay is a form of L2 regularization that discourages large weights in the model through a penalty on the loss function proportional to the sum of the squares of the weights. This leads to a simpler model and helps to prevent over-fitting.
The loss function with weight decay is given by:

$$L_{new} = L_{original} + \frac{\lambda}{2N} \sum_{i=1}^{N} w_i^2$$

where

- $L_{original}$ is the original loss without regularization
- w_i are the weights of the model
- N is the total number of weights
- λ is the regularization parameter.

During training, this has the effect of updating the weights as if they are decaying towards zero, hence the name weight decay.
We use these three techniques in combination to mitigate the risk of over-fitting and improve the model's generalization to new, unseen data.

A.3 Training and testing

Each [**A-H-M**] with different hyper-parameters was trained on the first training set D1 (with Huber loss), and then used to produce forecasts on the validation set D2. The final specification of each [**A-H-M**] model was selected based on the forecast performance on the validation set (with MSE), which are considered pseudo-out-of sample forecasts (Kelly and Xiu (2023)). For each [**A-H-M**] model, we find the best model through the tuning on D1 and D2, which is then re-trained on the full D1 + D2 set (60% of data). The training is done by minimizing the Huber Loss on D1, and using the MSE to compare and validate results on D2. We chose to use the Huber loss function on the training set because of its flexibility and adaptability in terms of scoring, in the sense that it provides us with the benefits of both an

L1 (MAE) and an L2 (MSE) scoring function. In particular, the flexibility provided by its δ hyper-parameter is well-suited for volatility series because they exhibit non-stationary, auto-correlation, and idiosyncratic jumps. The δ parameter is a way to allow the loss function itself to adapt to the volatility regime by switching from linear to quadratic scoring depending on the size of the error. The choice of using the MSE on the validation set was driven by its status as a standard for model evaluation on validation and test sets in the deep learning literature. Subsequently, we proceed to build the ensemble dataset as explained above, by making forecasts on the D3 dataset using the trained single models. The outputs from the single model's forecasts on D3 are then used as input for the ensemble models training (with Huber loss) on D3 with the actual realized values as the target. The final comparison of all our models evaluated on the (never seen by any model during training) D4 test set is performed using two functions that have been proved by Patton (2011) to be robust to volatility proxies: the Quasi Likelihood and the (Root) Mean-Squared Error function. Our choice of these two was purely motivated by a need for an unbiased and robust ranking of our models which demonstrates robustness to proxies, sampling frequency, or signal-to-noise ratio (Patton (2011)). Additionally, it seemed appropriate to use the Q-LIKE to have better comparability since one of our benchmarks is a statistical distribution likelihood maximizing model (CGARCH). It also allowed us to have a preliminary assessment of how well our models fitted the volatility distributions without directly being optimized or trained to do so.

A.4 Evaluation

To evaluate and compare our final models forecasting accuracy across assets and horizons, we supplement our performance metrics with 2 evaluation frameworks: the Model Confidence Set method from Hansen et al. (2003). As suggested by Brownlees et al. (2011) and the Multivariate Giacomini-White test from Borup et al. (2024). We succinctly discuss the two frameworks in this section.

Model Confidence set The MCS is a statistical method for model comparison. It provides a set of models from a larger collection, with a certain level of confidence that the set includes the best model according to a chosen loss function.
Given a set M^0 containing a finite number m_0 of models, indexed by $i = 1, \ldots, m_0$, the loss associated with model i in period t is denoted by L_{it}. The relative performance of models i and j in period t is given by:

$$d_{ijt} = L_{it} - L_{jt}$$

where $\mathbb{E}[d_{ijt}]$ is assumed to be finite and constant over t for all $i, j \in M^0$.
The set of superior models, denoted by M^*, is defined as:

$$M^* = \{i \in M^0 : \mu_{ij} < 0 \text{ for all } j \in M^0\}$$

where $\mu_{ij} = \mathbb{E}[d_{ijt}]$. The MCS algorithm determines M^* through a sequence of significance tests, eliminating significantly inferior models.
The hypotheses tested are of the form:

$$H_0^M : \mu_{ij} = 0 \text{ for all } i, j \in M$$

where $M \subseteq M^0$, against the alternative $H_A^M : \mu_{ij} \neq 0$ for some $i, j \in M$. The MCS procedure aims to construct a subset $M_\alpha^* \subseteq M^0$ that contains all of M^* with a pre-specified coverage probability $1 - \alpha$, based on an equivalence test θ_M and an elimination rule e_M. The equivalence test θ_M tests H_0^M for different M. The elimination rule e_M removes models statistically inferior, reducing M until it only contains indistinguishable models in terms of performance. The test statistic for this test is the well-known test used in Diebold and Mariano (2002) :

$$t_{ij} = \frac{\mu_{ij}}{\sqrt{var(d_{ij})}}$$

Where $var(d_{ij})$ is the estimate variance of d_{ij}.

The final set M_α^* is a model confidence set containing the best-performing models with high probability 1 - α , offering a statistically robust selection method. In other words, the MCS contains all superior models with a given probability (its coverage probability) under certain assumptions. Bootstrap methods are often employed for practical implementation of the MCS procedure, especially when the number of models is large.

Multivariate Giacomini-White The Multivariate Giacomini-White test extends the framework of Giacomini and White (2006) to a multivariate setting for real-time assessment and identification of state dependencies in predictability. This test compares the predictive ability of multiple forecasting methods under varying conditions.
Consider a setting with $p + 1$ forecasting methods ($p \geq 1$), indexed by $i = 1, \ldots, p + 1$. The forecast of the target variable $y_{t+\tau}$ at time t using the i-th method is denoted as:

$$f_{ti+\tau} = \alpha_{ti} + \beta_{it} x_{it}$$

where α_{ti} and β_{it} are parameters estimated from the data, and x_{it} is the vector of predictors. A rolling window forecasting scheme is assumed for estimation.
The Multivariate Giacomini-White test statistic assesses whether these $p + 1$ methods have equal predictive ability. It does this by evaluating the forecasting performance of each method against a loss function $L_{t+\tau}$, which measures the prediction error for the forecast $f_{ti+\tau}$. The test can be seen as a generalization of the Diebold-Mariano test that measures the Conditional Predictive Ability instead of the Unconditional Predictive Ability. The test, like the Diebold-Mariano variant, measures the statistical significance of the differences between the two models' forecasts. It is an asymptotic 2 test.
The test statistic in the Multivariate Giacomini-White test plays a crucial role in determining the statistical significance of differences in predictive abilities across various forecasting methods.

- **Loss Function**: A loss function L is defined to quantify the prediction error of each forecast. This could be mean squared error, absolute error, or any other metric appropriate for the analysis.

- **Forecast Error**: The forecast error at time t for the horizon τ is the difference between the forecast $f_{ti+\tau}$ and the actual observed value $y_{t+\tau}$.
- **Loss Differential**: The loss differential for methods i and j at time t is computed as $d_{ijt} = L(y_{t+\tau}, f_{ti+\tau}) - L(y_{t+\tau}, f_{tj+\tau})$.
- **Test Statistic Computation**: Aggregate these differentials over time to form the test statistic. This involves creating a vector of loss differentials and using it to form a multivariate statistic.
- **Statistical Significance**: Perform hypothesis testing using the test statistic to determine if there is a statistically significant difference in predictive performance between the forecasting methods.

B Mathematical intuition behind the Transformer

As seen in Thickstun (2019), a transformer's encoder block as used in this study is a parameterized function class $f_\theta : \mathbb{R}^{p\times d} \to \mathbb{R}^{p\times d}$. If $x \in \mathbb{R}^{p\times d}$ then $f_\theta(x) = z$ where

$$
\begin{aligned}
Q^{(h)}(x_i) = W_{h,q}^T x_i, \quad K^{(h)}(x_i) = W_{h,k}^T x_i, \quad V^{(h)}(x_i) = W_{h,v}^T x_i, \quad & W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d\times k}, \\
\alpha_{ij}^{(h)} = \operatorname{softmax}\left(\frac{Q^{(h)}(x_i)^T K^{(h)}(x_j)}{\sqrt{k}}\right), & \\
\mathbf{u_i'} = \sum_{h=1}^{H} W_{c,h}^T \sum_{j=1}^{p} \alpha_{ij}^{(h)} V^{(h)}(\mathbf{x_j}), & W_{c,h} \in \mathbb{R}^{k\times d}, \\
\mathbf{u_i} = \operatorname{LayerNorm}(\mathbf{x_i} + \mathbf{u_i'}; \gamma_1, \beta_1), & \gamma_1, \beta_1 \in \mathbb{R} \\
\mathbf{z_i'} = W_2^T \operatorname{ReLU}(W_1^T \mathbf{u_i}), & W_1 \in \mathbb{R}^{d\times m}, \quad W_2 \in \mathbb{R}^{m\times d}, \\
\mathbf{z_i} = \operatorname{LayerNorm}(\mathbf{u_i} + \mathbf{z_i'}; \gamma_2, \beta_2). & \gamma_2, \beta_2 \in \mathbb{R}
\end{aligned}
$$

The notation $\operatorname{softmax}_j$ indicates we take the softmax over the d-dimensional vector indexed by j. The LayerNorm function is defined for $z \in \mathbb{R}^k$ by

$$
\operatorname{LayerNorm}(\mathbf{z}; \gamma, \beta) = \gamma \frac{(\mathbf{z} - \mu_z)}{\sigma_z} + \beta, \qquad \gamma, \beta \in \mathbb{R}^k.
$$

where

$$
\mu_z = \frac{1}{k}\sum_{i=1}^{k} \mathbf{z_i}, \quad \sigma_z = \sqrt{\frac{1}{k}\sum_{i=1}^{k} (\mathbf{z_i} - \mu_z)^2}.
$$

The parameters θ to be learned from our data consist of the values contained in the weight matrices W along with the LayerNorm parameters γ and β, all indicated on the right-hand side. The input $x \in \mathbb{R}^{p\times k}$ is a collection of d time-series (features) of length p. In our case, d = 1 since we only use a series of past realized volatilities as input.

Table 1: Descriptive statistics of annualized returns and volatility series

Series	Statistic	S&P 500	NASDAQ	RUSSEL 2000	GOLD	OIL
Volatility	count	4532	4532	4532	4262	3961
	mean	1.98	2.15	3.03	1.83	3.70
	std	2.22	1.54	3.86	2.00	3.33
	min	0.24	0.21	0.48	0.05	0.40
	25%	1.06	1.29	1.69	1.08	2.41
	50%	1.45	1.77	2.21	1.46	3.20
	75%	2.20	2.53	3.20	2.04	4.25
	max	75.09	25.75	165.24	75.64	170.427
	median	1.45	1.78	2.21	1.46	3.21
	range	74.85	25.54	164.76	75.58	170.02
	skew	12.46	4.52	20.54	19.10	32.35
	kurtosis	308.08	38.13	743.94	601.57	1585.86
	Hurst Exponent	**0.40**	**0.35**	**0.43**	**0.42**	**0.47**
	Range	1990 - 2023	1990 - 2023	1990 - 2023	2000 - 2023	2000 - 2023
	ADF test	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	JB test	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
Returns	mean	0.08	0.11	0.076	0.09	0.07
	std	3.01	3.41	3.86	2.80	6.60
	min	-32.16	-33.13	-38.80	-24.74	-71.11
	25%	-1.06	-1.37	-1.78	-1.23	-3.14
	50%	0.18	0.24	0.22	0.12	0.29
	75%	1.44	1.83	2.11	1.59	3.35
	max	27.61	28.12	22.62	21.78	80.54
	median	0.17	0.25	0.22	0.12	0.29
	range	59.78	61.25	61.42	46.52	151.66
	skew	-0.51	-0.40	-0.57	-0.34	0.05
	kurtosis	12.94	7.40	7.78	5.22	17.68

Here we see the relevant descriptive statistics of the two kinds of series used in this study: Annualized realized volatility and returns

Table 2: Hyperparameters tested

(a) Model Hyperparameters tested

Model	Hyperparameter	Tested values
Transformer	Attention Heads	2 – 4 - 8
	Hidden dimension size	1 – 4 – 8 – 16 – 32 – 64 – 128 – 256 – 512 - 1024
	Query, Key and Value Dimension	8 – 16 – 32 – 64
	Activation functions	ReLu, tanH, sigmoid
LSTM	Hidden Size	1 – 2 – 4 – 8
	Input Feed Forward dimensions	1 – 4 – 8 – 16 – 32 – 64 – 128 – 256 – 512 - 1024
	LSTM layers	1 – 2 – 4 – 8
	Activation functions	ReLu, tanH, sigmoid
XGBOOST	max_depth	2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10
	max_leaf_nodes	2 - 4 - 8 - 16 - 32
	num_estimators	50 - 60 - 70 - 80 - 100 - 200 - 150 - 500
	base learner	linear regressor, tree
	criterion	gini, entropy, log_loss
NGBOOST	Gradient	Natural gradient - Ordinary Gradient
	Distribution	Normal - Student T
	num_estimators	50 - 60 - 70 - 80 - 100 - 200 - 150 - 500
	base learner	linear regressor, tree
	criterion	gini, entropy, log_loss

(b) Training procedure hyperparameters tested

Hyperparameter	Tested values
Learning Rate	0.0001 – 0.001 – 0.01 – 0.1 – 1
Gradient Clipping value	0.5 – 1 – 2 - 5 – 10 – 25 – 50
Batch size	16 – 32 – 64 - 128
Optimizers	Adam, SGD
Scheduler's step size	1 – 3 – 5
L1 Weight Regularization	0 - 0.01
L2 Weight Regularization	0 - 0.01
Weight_decay	0.0001 - 0.000001
Num Epochs	100 - 200 - 500 - 1000 - 5000
Input Sequence Length	5 - 10 - 20 -30
Optimization Algorithm	Adaptive Moment Estimation - Stochastic Gradient Descent

Subfigure **(a)** of this table shows us all the hyperparameters tested for every architecture studied. The first column displays the architectures in each cell, the second column shows the name of the hyperparameters tested, and the third column shows us the values tested. Subfigure **(b)** shows us all the training hyperparameters that we tested on all our models.

Table 3: Final hyperparameters retained for A-60 models after grid-search

Model	Hyperparameter	S&P500	NASDAQ	RUSSEL	GOLD	OIL
Transformer	Attention Heads	8	8	8	4	4
	Hidden dimension size	256	256	128	128	128
	Query, Key and Value Dimension	32	32	32	32	32
	Activation functions	sigmoid	sigmoid	sigmoid	sigmoid	sigmoid
LSTM	Hidden Size	8	8	8	4	4
	Input Feed Forward dimensions	1	1	1	1	1
	LSTM layers	1	1	1	1	1
	Activation functions	ReLu	ReLu	ReLu	ReLu	ReLu
XGBOOST	max_depth	5	5	5	5	5
	max_leaf_nodes	8	8	8	8	8
	num_estimators	500	500	500	500	500
	base learner	Tree	Tree	Tree	Tree	Tree
	criterion	Entropy	Entropy	Entropy	Entropy	Entropy
NGBOOST	Gradient	4	4	4	4	4
	Distribution	Student T	Student T	Student T	Student T	Student T
	num_estimators	500	500	500	500	500
	base learner	Tree	Tree	Tree	Tree	Tree
	criterion	gini	gini	gini	gini	gini

Table 4: Out of sample forecasting results across different horizons for the S&P500

Horizon	Model	RMSE in bps	Quasi-likelihood	OF in %
h-5	**HAR**	35.5	13.7	53.5
	CGARCH	42.9	33.0	78.2
	XgBoost	30.0	10.7	62.4
	NgBoost	30.8	16.5	54.5
	LSTM	37.6	11.8	48.5
	Transformer	37.5	11.7	47.5
	X-N-L-T	62.6	45.8	57.4
	XG CGARCH	114.3	51.5	0.0
	NG CGARCH	54.9	50.8	80.2
	LSTM CGARCH	161.2	216.2	98.0
	TRANS CGARCH	76.9	83.7	93.1
	X-N-L-T CGARCH	106.0	44.1	0.0
h-10	**HAR**	33.2	14.4	55.4
	CGARCH	43.8	37.6	80.2
	XgBoost	27.8	11.9	61.4
	NgBoost	28.3	13.0	57.4
	LSTM	33.5	11.1	46.5
	Transformer	31.9	14.2	62.4
	X-N-L-T	33.6	11.4	46.5
	XG CGARCH	58.3	45.5	35.6
	NG CGARCH	40.4	17.5	31.7
	LSTM CGARCH	36.0	14.1	42.6
	TRANS CGARCH	33.5	14.3	55.4
	X-N-L-T CGARCH	39.8	16.6	29.7
h-15	**HAR**	33.0	15.0	61.0
	CGARCH	47.0	43.2	84.0
	XgBoost	22.0	12.8	67.0
	NgBoost	21.4	21.9	66.0
	LSTM	33.1	12.8	52.0
	Transformer	31.1	14.5	67.0
	X-N-L-T	36.9	28.9	30.0
	XG CGARCH	36.4	13.8	40.0
	NG CGARCH	50.6	36.0	28.0
	LSTM CGARCH	144.2	56.3	3.0
	TRANS CGARCH	39.4	18.4	47.0
	X-N-L-T CGARCH	51.7	36.5	85.0
h-20	**HAR**	30.4	15.1	63.6
	CGARCH	48.1	45.9	86.9
	XgBoost	25.5	12.3	70.7
	NgBoost	21.6	17.0	63.6
	LSTM	31.6	12.9	53.5
	Transformer	30.0	11.7	54.5
	X-N-L-T	34.7	23.5	25.3
	XG CGARCH	29.7	16.6	78.8
	NG CGARCH	43.5	34.1	89.9
	LSTM CGARCH	77.6	68.6	86.9
	TRANS CGARCH	26.8	12.4	50.5
	X-N-L-T CGARCH	95.6	80.3	53.5
h-60	**HAR**	36.8	37.3	84.9
	CGARCH	78.3	110.4	96.8
	XgBoost	26.7	22.1	79.6
	NgBoost	25.3	30.7	74.2
	LSTM	30.3	22.8	64.5
	Transformer	27.9	19.4	67.7
	X-N-L-T	35.0	25.8	74.2
	XG CGARCH	50.5	54.2	63.4
	NG CGARCH	40.0	42.8	88.2
	LSTM CGARCH	73.6	84.0	94.6
	TRANS CGARCH	45.3	49.5	94.6
	X-N-L-T CGARCH	43.3	44.2	47.3

This table shows us, for the S&P500, the out-of-sample performance metrics scored by our models for every horizon. The displayed metrics from left to right are : **the Root-Mean Square Error, the Quasi Likelihood, and the Overvaluation Frequency**

Table 5: Out of sample forecasting results across different horizons for NASDAQ

Horizon	Model	RMSE in bps	Quasi-likelihood	OF in %
h-5	**HAR**	35.2	8.6	51.5
	CGARCH	36.9	11.6	66.3
	XgBoost	31.9	7.5	51.5
	NgBoost	30.9	10.9	50.5
	LSTM	34.8	7.3	43.6
	Transformer	35.5	7.9	49.5
	X-N-L-T	40.0	10.4	54.5
	XG CGARCH	42.2	13.0	58.4
	NG CGARCH	41.3	13.1	55.4
	LSTM CGARCH	38.3	10.2	56.4
	TRANS CGARCH	49.2	21.6	49.5
	X-N-L-T CGARCH	57.5	28.8	25.7
h-10	**HAR**	36.5	9.3	54.5
	CGARCH	38.7	13.5	69.3
	XgBoost	32.1	7.8	52.5
	NgBoost	35.4	11.0	49.5
	LSTM	36.2	8.3	48.5
	Transformer	36.0	7.9	47.5
	X-N-L-T	40.2	9.9	46.5
	XG CGARCH	42.6	15.5	67.3
	NG CGARCH	41.1	16.0	73.3
	LSTM CGARCH	39.1	14.5	71.3
	TRANS CGARCH	44.1	19.8	77.2
	X-N-L-T CGARCH	50.3	19.2	37.6
h-15	**HAR**	36.0	9.4	59.0
	CGARCH	39.1	14.9	74.0
	XgBoost	30.7	7.3	53.0
	NgBoost	34.9	8.7	50.0
	LSTM	37.4	7.9	40.0
	Transformer	35.5	7.8	47.0
	X-N-L-T	40.1	10.7	35.0
	XG CGARCH	30.2	6.9	55.0
	NG CGARCH	50.4	19.6	46.0
	LSTM CGARCH	33.0	10.0	71.0
	TRANS CGARCH	45.5	20.4	77.0
	X-N-L-T CGARCH	34.5	10.9	51.0
h-20	**HAR**	34.2	8.9	62.6
	CGARCH	39.0	15.7	78.8
	XgBoost	31.3	7.9	58.6
	NgBoost	31.2	10.6	54.5
	LSTM	32.8	7.8	61.6
	Transformer	33.3	7.1	49.5
	X-N-L-T	64.0	77.4	1.0
	XG CGARCH	30.4	7.7	59.6
	NG CGARCH	30.8	7.0	57.6
	LSTM CGARCH	70.2	41.0	94.9
	TRANS CGARCH	50.4	25.0	87.9
	X-N-L-T CGARCH	117.2	84.0	98.0
h-60	**HAR**	35.0	13.5	72.0
	CGARCH	59.0	35.0	91.4
	XgBoost	31.1	9.2	61.3
	NgBoost	33.1	14.0	58.1
	LSTM	37.7	14.5	68.8
	Transformer	33.2	10.4	55.9
	X-N-L-T	35.5	14.1	73.1
	XG CGARCH	54.5	30.9	89.2
	NG CGARCH	62.9	38.6	95.7
	LSTM CGARCH	112.3	82.7	98.9
	TRANS CGARCH	55.4	62.0	35.5
	X-N-L-T CGARCH	67.0	62.8	12.9

This table shows us, for the NASDAQ, the out-of-sample performance metrics scored by our models for every horizon. The displayed metrics from left to right are: **the Root-Mean Square Error, the Quasi Likelihood, and the Overvaluation Frequency**

Table 6: Out of sample forecasting results across different horizons for the RUSSEL

Horizon	Model	RMSE in bps	Quasi-likelihood	OF in %
h-5	**HAR**	58.2	10.4	61.4
	CGARCH	63.1	9.2	37.6
	XgBoost	53.7	8.3	64.4
	NgBoost	56.5	14.7	57.4
	LSTM	64.6	9.7	38.6
	Transformer	59.0	8.7	40.6
	X-N-L-T	58.1	9.4	57.4
	XG CGARCH	57.8	8.6	53.5
	NG CGARCH	58.9	12.3	72.3
	LSTM CGARCH	58.8	9.2	54.5
	TRANS CGARCH	58.2	8.8	52.5
	X-N-L-T CGARCH	58.3	9.2	55.4
h-10	**HAR**	59.1	10.1	58.4
	CGARCH	63.9	9.5	36.6
	XgBoost	57.2	12.0	61.4
	NgBoost	57.1	12.4	48.5
	LSTM	62.6	8.6	37.6
	Transformer	60.6	8.8	34.7
	X-N-L-T	56.4	7.7	50.5
	XG CGARCH	82.7	15.5	66.3
	NG CGARCH	59.5	11.8	66.3
	LSTM CGARCH	61.3	12.9	67.3
	TRANS CGARCH	57.4	11.0	69.3
	X-N-L-T CGARCH	154.5	52.0	12.9
h-15	**HAR**	58.8	10.1	60.0
	CGARCH	60.7	8.4	42.0
	XgBoost	54.0	9.6	63.0
	NgBoost	54.6	13.9	58.0
	LSTM	60.8	8.6	42.0
	Transformer	59.3	8.2	42.0
	X-N-L-T	58.5	7.3	44.0
	XG CGARCH	54.9	11.2	77.0
	NG CGARCH	60.2	9.3	60.0
	LSTM CGARCH	83.7	29.0	23.0
	TRANS CGARCH	58.4	9.9	63.0
	X-N-L-T CGARCH	61.4	13.8	79.0
h-20	**HAR**	52.6	9.1	63.6
	CGARCH	53.1	7.3	47.5
	XgBoost	49.6	8.2	66.7
	NgBoost	51.5	13.2	58.6
	LSTM	54.5	7.3	41.4
	Transformer	56.6	8.4	33.3
	X-N-L-T	51.5	10.3	75.8
	XG CGARCH	63.2	18.9	83.8
	NG CGARCH	77.4	27.5	88.9
	LSTM CGARCH	76.7	27.4	87.9
	TRANS CGARCH	82.2	31.6	90.9
	X-N-L-T CGARCH	62.3	15.5	65.7
h-60	**HAR**	52.0	14.8	75.3
	CGARCH	50.9	13.8	73.1
	XgBoost	42.4	8.5	72.0
	NgBoost	45.4	18.1	60.2
	LSTM	50.1	10.7	50.5
	Transformer	51.9	7.6	45.2
	X-N-L-T	40.9	7.1	64.5
	XG CGARCH	74.0	28.2	93.5
	NG CGARCH	49.9	13.2	74.2
	LSTM CGARCH	49.9	13.8	84.9
	TRANS CGARCH	58.3	18.6	92.5
	X-N-L-T CGARCH	78.7	28.7	92.5

This table shows us, for the RUSSEL, the out-of-sample performance metrics scored by our models for every horizon. The displayed metrics from left to right are: **the Root-Mean Square Error, the Quasi Likelihood, and the Overvaluation Frequency**

Table 7: Models out-of-sample forecasting results across different horizons for gold

Horizon	Model	RMSE in bps	Quasi-likelihood	OF in %
h-5	HAR	31.0	10.8	74.7
	CGARCH	43.2	28.7	90.5
	XgBoost	31.1	10.5	70.5
	NgBoost	33.5	11.1	65.3
	LSTM	29.8	7.0	58.9
	Transformer	30.1	6.1	43.2
	X-N-L-T	30.7	7.3	50.5
	XG CGARCH	36.2	16.5	21.1
	NG CGARCH	30.7	8.8	55.8
	LSTM CGARCH	43.5	45.2	7.4
	TRANS CGARCH	30.0	8.8	65.3
	X-N-L-T CGARCH	31.8	7.8	45.3
h-10	HAR	31.7	11.6	73.4
	CGARCH	40.4	25.5	92.6
	XgBoost	31.4	10.6	74.5
	NgBoost	43.3	20.2	67.0
	LSTM	30.8	8.2	63.8
	Transformer	30.5	6.7	48.9
	X-N-L-T	32.4	8.3	39.4
	XG CGARCH	32.1	6.6	34.0
	NG CGARCH	38.2	18.4	19.1
	LSTM CGARCH	33.3	8.3	25.5
	TRANS CGARCH	33.8	8.2	27.7
	X-N-L-T CGARCH	43.8	39.7	7.4
h-15	HAR	32.2	12.4	78.7
	CGARCH	41.4	27.0	93.6
	XgBoost	31.8	11.2	75.5
	NgBoost	37.0	16.9	71.3
	LSTM	30.9	7.2	55.3
	Transformer	30.9	10.0	72.3
	X-N-L-T	32.9	7.3	33.0
	XG CGARCH	34.7	9.0	21.3
	NG CGARCH	43.5	29.7	0.0
	LSTM CGARCH	32.4	6.8	39.4
	TRANS CGARCH	36.4	11.4	10.6
	X-N-L-T CGARCH	30.5	10.3	72.3
h-20	HAR	32.5	12.8	81.7
	CGARCH	43.6	28.6	93.5
	XgBoost	32.2	11.2	71.0
	NgBoost	41.9	19.6	71.0
	LSTM	30.7	6.4	54.8
	Transformer	30.8	9.5	71.0
	X-N-L-T	30.4	7.4	61.3
	XG CGARCH	38.6	15.2	5.4
	NG CGARCH	31.2	10.2	67.7
	LSTM CGARCH	36.5	10.8	14.0
	TRANS CGARCH	36.5	11.1	18.3
	X-N-L-T CGARCH	31.1	7.2	51.6
h-60	HAR	31.6	18.0	88.5
	CGARCH	58.6	49.7	95.4
	XgBoost	32.2	13.1	77.0
	NgBoost	71.9	17.0	72.4
	LSTM	26.7	9.6	65.5
	Transformer	25.1	9.4	74.7
	X-N-L-T	34.3	19.8	78.2
	XG CGARCH	49.1	33.7	82.8
	NG CGARCH	109.7	105.3	93.1
	LSTM CGARCH	75.4	64.6	93.1
	TRANS CGARCH	27.0	6.8	40.2
	X-N-L-T CGARCH	183.8	168.0	92.0

This table shows us, for gold, the out-of-sample performance metrics scored by our models for every horizon. The displayed metrics from left to right are: **the Root-Mean Square Error, the Quasi Likelihood, and the Overvaluation Frequency**

Table 8: Models out-of-sample forecasting results across different horizons for oil

Horizon	Model	RMSE in bps	Quasi-likelihood	OF in %
h-5	**HAR**	64.8	5.9	63.6
	CGARCH	91.7	13.6	6.8
	XgBoost	62.3	5.7	65.9
	NgBoost	65.0	7.9	59.1
	LSTM	67.4	5.4	48.9
	Transformer	67.9	5.8	53.4
	X-N-L-T	92.4	26.0	39.8
	XG CGARCH	82.1	11.4	54.5
	NG CGARCH	67.1	6.4	52.3
	LSTM CGARCH	70.5	6.8	61.4
	TRANS CGARCH	73.4	8.0	53.4
	X-N-L-T CGARCH	88.6	26.5	45.5
h-10	**HAR**	64.7	5.8	58.0
	CGARCH	92.0	14.1	5.7
	XgBoost	57.8	5.3	65.2
	NgBoost	65.6	7.4	62.5
	LSTM	68.8	5.9	40.9
	Transformer	67.3	5.7	44.3
	X-N-L-T	71.7	8.2	63.6
	XG CGARCH	73.2	6.9	44.3
	NG CGARCH	70.6	6.1	42.0
	LSTM CGARCH	73.5	6.6	35.2
	TRANS CGARCH	71.2	6.1	35.2
	X-N-L-T CGARCH	80.7	10.0	44.3
h-15	**HAR**	65.5	6.1	57.5
	CGARCH	88.5	12.1	11.5
	XgBoost	60.9	5.7	64.4
	NgBoost	70.4	7.7	56.3
	LSTM	69.1	5.9	43.7
	Transformer	67.0	5.8	51.7
	X-N-L-T	83.5	13.7	55.2
	XG CGARCH	81.4	12.3	69.0
	NG CGARCH	65.2	6.5	56.3
	LSTM CGARCH	72.7	6.3	33.3
	TRANS CGARCH	66.1	5.7	48.3
	X-N-L-T CGARCH	82.8	12.6	49.4
h-20	**HAR**	66.8	6.3	61.6
	CGARCH	84.4	10.2	11.6
	XgBoost	105.8	10.0	69.3
	NgBoost	299.9	24.9	65.1
	LSTM	70.2	5.8	41.9
	Transformer	66.1	5.9	57.0
	X-N-L-T	68.9	5.6	43.0
	XG CGARCH	73.6	7.6	53.5
	NG CGARCH	74.5	7.2	33.7
	LSTM CGARCH	70.4	8.2	57.0
	TRANS CGARCH	73.8	6.5	37.2
	X-N-L-T CGARCH	68.7	8.0	57.0
h-60	**HAR**	47.7	7.6	77.5
	CGARCH	44.9	4.3	42.5
	XgBoost	43.0	5.7	63.7
	NgBoost	56.5	8.4	60.0
	LSTM	43.0	5.3	60.0
	Transformer	44.7	6.0	68.8
	X-N-L-T	47.8	7.2	65.0
	XG CGARCH	52.4	8.2	45.0
	NG CGARCH	96.4	23.9	95.0
	LSTM CGARCH	53.4	8.6	71.2
	TRANS CGARCH	44.8	4.0	45.0
	X-N-L-T CGARCH	103.4	23.7	75.0

This table shows us, for Crude Oil, the out-of-sample performance metrics scored by our models for every horizon. The displayed metrics from left to right are : **the Root-Mean Square Error, the Quasi Likelihood, and the Overvaluation Frequency**

Table 9: MGW different predictive ability test results against benchmark

		S&P500		NASDAQ		RUSSEL		GOLD		OIL	
		HAR	CGARCH	HAR	CGARCH	HAR	CGARCH	HAR	CGARCH	HAR	CGARCH
Horizon											
h-5	**XgBoost**	**YES**	**YES**	YES	YES	YES	YES	YES	YES	YES	YES
	NgBoost	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	LSTM	NO	YES	YES	YES	NO	NO	YES	YES	NO	YES
	Transformer	NO	YES	NO	NO	NO	NO	YES	YES	NO	YES
	X-N-L-T	YES	NO	NO	NO	NO	NO	NO	YES	YES	NO
	XG CGARCH	YES	YES	YES	NO	YES	YES	YES	YES	YES	NO
	NG CGARCH	YES	YES	NO	YES	YES	NO	YES	YES	YES	YES
	LSTM CGARCH	YES	YES	NO	NO	NO	NO	YES	NO	NO	YES
	TRANS CGARCH	YES	YES	NO	NO	NO	NO	NO	YES	NO	YES
	X-N-L-T CGARCH	YES	YES	YES	NO	NO	NO	NO	YES	YES	NO
h-10	**XgBoost**	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	NgBoost	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	LSTM	NO	YES	YES	YES	NO	YES	YES	YES	NO	YES
	Transformer	NO	YES	NO	YES	NO	NO	YES	YES	NO	YES
	X-N-L-T	NO	YES	NO	NO	YES	YES	NO	YES	YES	YES
	XG CGARCH	YES	NO	YES	YES	YES	YES	YES	YES	YES	YES
	NG CGARCH	NO	NO	YES	YES	YES	YES	NO	YES	YES	YES
	LSTM CGARCH	NO	YES	YES	NO	YES	NO	NO	YES	NO	YES
	TRANS CGARCH	NO	YES	YES	YES	NO	NO	NO	YES	NO	YES
	X-N-L-T CGARCH	NO	YES	YES	YES	NO	NO	YES	YES	YES	NO
h-15	**XgBoost**	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	NgBoost	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	LSTM	NO	YES	NO	YES	NO	NO	YES	YES	NO	YES
	Transformer	NO	YES	NO	YES	NO	NO	YES	YES	NO	YES
	X-N-L-T	NO	YES	NO	NO	YES	NO	NO	YES	YES	NO
	XG CGARCH	YES	YES	YES	YES	NO	YES	YES	YES	YES	YES
	NG CGARCH	YES	YES	YES	NO	NO	YES	YES	NO	YES	YES
	LSTM CGARCH	YES	YES	NO	YES	YES	YES	NO	YES	NO	YES
	TRANS CGARCH	NO	YES	YES	NO	NO	NO	NO	YES	NO	YES
	X-N-L-T CGARCH	YES	NO	NO	NO	NO	NO	YES	YES	YES	NO
h-20	**XgBoost**	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	NgBoost	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	LSTM	NO	YES	YES	YES	NO	NO	YES	YES	NO	YES
	Transformer	NO	YES	NO	YES	NO	YES	YES	YES	NO	YES
	X-N-L-T	NO	YES	YES	YES	NO	NO	YES	YES	NO	YES
	XG CGARCH	NO	YES	NO	YES	YES	YES	YES	YES	YES	NO
	NG CGARCH	YES	YES	YES	YES	YES	YES	NO	YES	YES	YES
	LSTM CGARCH	YES	YES	YES	YES	YES	YES	NO	YES	YES	NO
	TRANS CGARCH	NO	YES	YES	YES	YES	YES	NO	YES	NO	YES
	X-N-L-T CGARCH	YES	YES	YES	YES	NO	NO	YES	YES	NO	NO
h-60	**XgBoost**	YES	YES	YES	YES	YES	YES	YES	YES	YES	NO
	NgBoost	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	LSTM	YES	YES	YES	YES	NO	NO	YES	YES	YES	NO
	Transformer	YES	YES	NO	YES	NO	NO	YES	YES	NO	NO
	X-N-L-T	NO	YES	NO	YES	YES	YES	NO	YES	NO	NO
	XG CGARCH	YES	YES	YES	YES	YES	YES	YES	YES	NO	YES
	NG CGARCH	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	LSTM CGARCH	YES	NO	YES	YES	NO	NO	YES	YES	NO	NO
	TRANS CGARCH	YES	YES	YES	NO	YES	YES	YES	YES	NO	NO
	X-N-L-T CGARCH	NO	YES	YES	NO	YES	YES	YES	YES	YES	YES

In this table, we're able to see for each forecasting horizon (first column) and each model at each horizon (second column), what is the answer to the question: "**Does the model in row have a different predictive ability than the benchmark at this horizon?**" obtained from the p-value of the MGW test. With a threshold at 1%, we're able to say for example that for the **h-5** horizon, the **XgBoost** model has a statistically different predictive ability than both our benchmarks for the **S&P500**.

Table 10: Model confidence set : best models at 99.9% confidence level

	S&P500	**NASDAQ**	**RUSSEL 2000**	**GOLD**	**OIL**
h-5	HAR XgBoost NgBoost LSTM Transformer NG CGARCH	HAR XgBoost LSTM Transformer X-N-L-T NG CGARCH	CGARCH XgBoost NgBoost Transformer LSTM CGARCH TRANS CGARCH X-N-L-T CGARCH	XgBoost NgBoost LSTM Transformer X-N-L-T TRANS CGARCH X-N-L-T CGARCH	HAR XgBoost NgBoost LSTM Transformer NG CGARCH LSTM CGARCH
h-10	XgBoost NgBoost LSTM Transformer X-N-L-T NG CGARCH	HAR XgBoost NgBoost LSTM Transformer X-N-L-T	HAR CGARCH XgBoost NgBoost LSTM Transformer X-N-L-T NG CGARCH	XgBoost NgBoost LSTM Transformer X-N-L-T XG CGARCH	HAR XgBoost NgBoost LSTM Transformer XG CGARCH NG CGARCH TRANS CGARCH
h-15	HAR XgBoost NgBoost LSTM Transformer XG CGARCH	XgBoost NgBoost LSTM Transformer XG CGARCH NG CGARCH	CGARCH XgBoost NgBoost LSTM Transformer X-N-L-T NG CGARCH TRANS CGARCH	XgBoost NgBoost LSTM Transformer X-N-L-T NG CGARCH LSTM CGARCH	HAR XgBoost NgBoost LSTM Transformer NG CGARCH LSTM CGARCH TRANS CGARCH X-N-L-T CGARCH
h-20	HAR XgBoost NgBoost LSTM Transformer XG CGARCH NG CGARCH TRANS CGARCH	HAR XgBoost NgBoost LSTM Transformer XG CGARCH NG CGARCH	HAR CGARCH XgBoost NgBoost LSTM Transformer X-N-L-T NG CGARCH	XgBoost NgBoost LSTM Transformer X-N-L-T NG CGARCH X-N-L-T CGARCH	HAR XgBoost NgBoost LSTM Transformer X-N-L-T NG CGARCH
h-60	XgBoost NgBoost LSTM Transformer X-N-L-T NG CGARCH	HAR XgBoost NgBoost Transformer X-N-L-T NG CGARCH	XgBoost NgBoost LSTM Transformer X-N-L-T NG CGARCH	XgBoost NgBoost LSTM Transformer TRANS CGARCH NG CGARCH	HAR CGARCH XgBoost NgBoost LSTM Transformer X-N-L-T XG CGARCH NG CGARCH LSTM CGARCH TRANS CGARCH

Table 10 shows us, for every **[A-H]** problem , the set of models for which the predictive abilities are not only the best, but also statistically indistinguishable from each other with a **99%** confidence level according to the MCS test. For example, the MCS results tells us that **for forecasting the 15 days ahead realized volatility of GOLD, the subset of best models with identical predictive ability is composed of the XgBoost, The NgBoost, LSTM, the Transformer, the X-N-L-T, the NG GARCH, and the LSTM GARCH** with a 99% confidence level

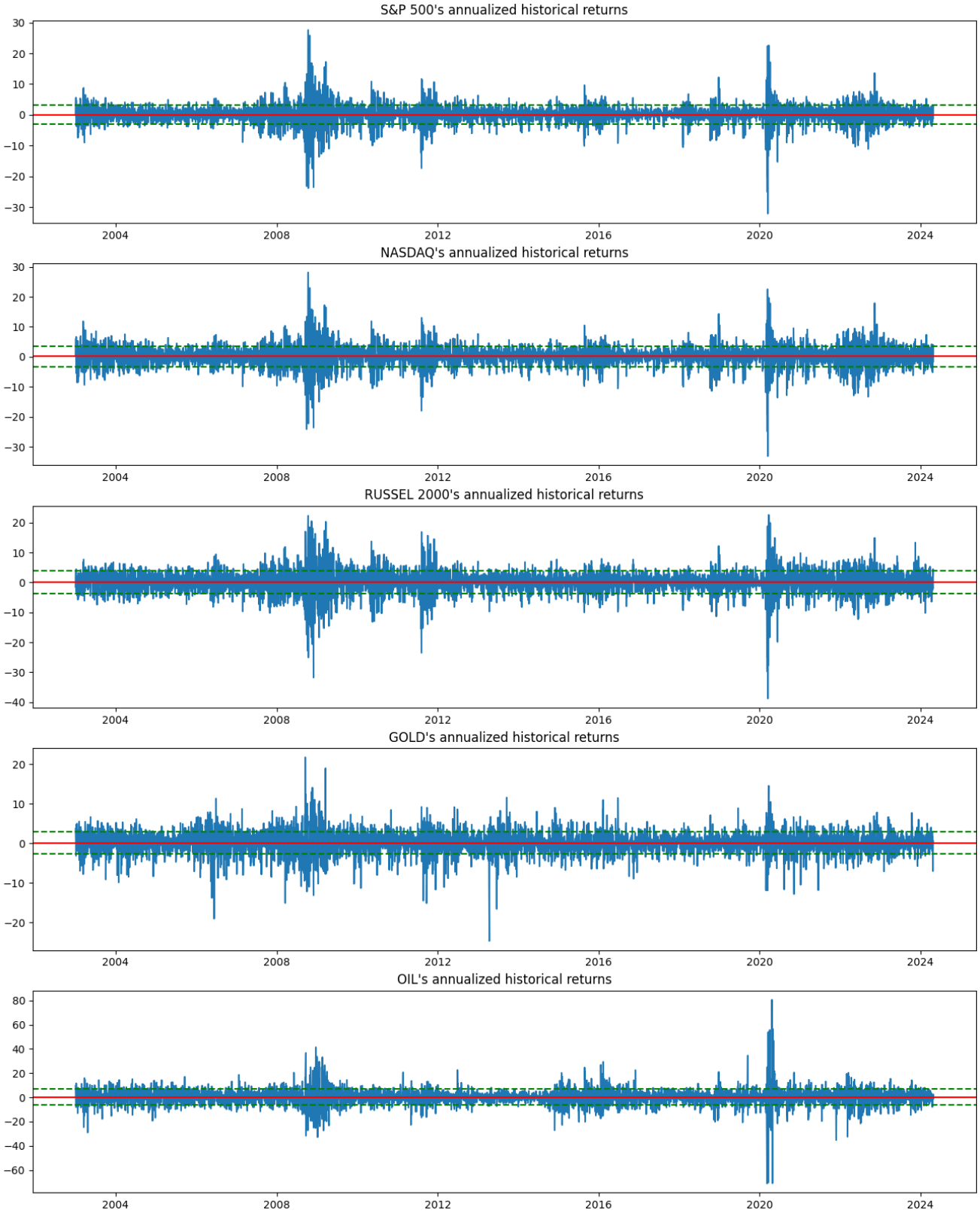


Figure A.1: Historical daily returns

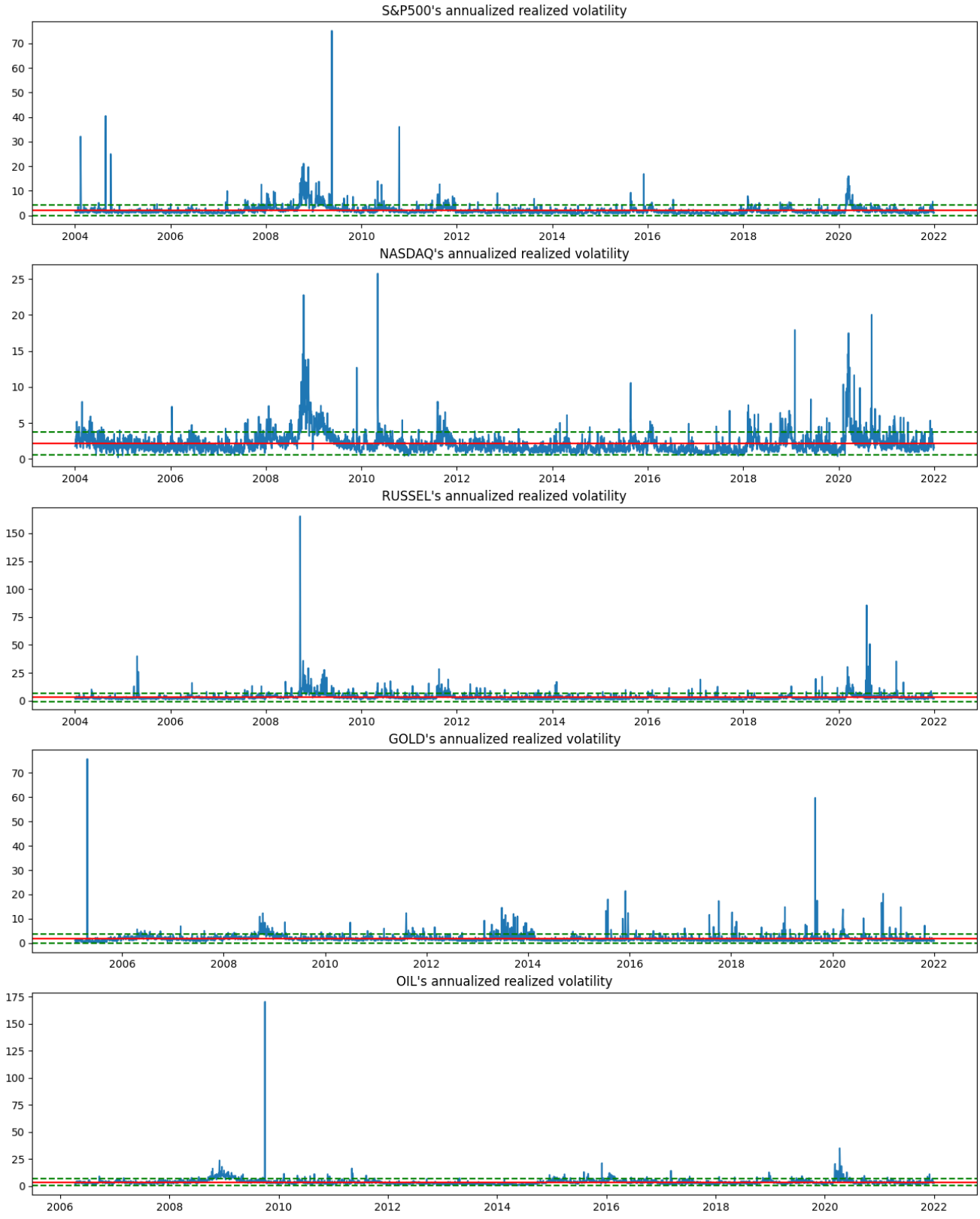


Figure A.2: Historical realized volatility of daily returns

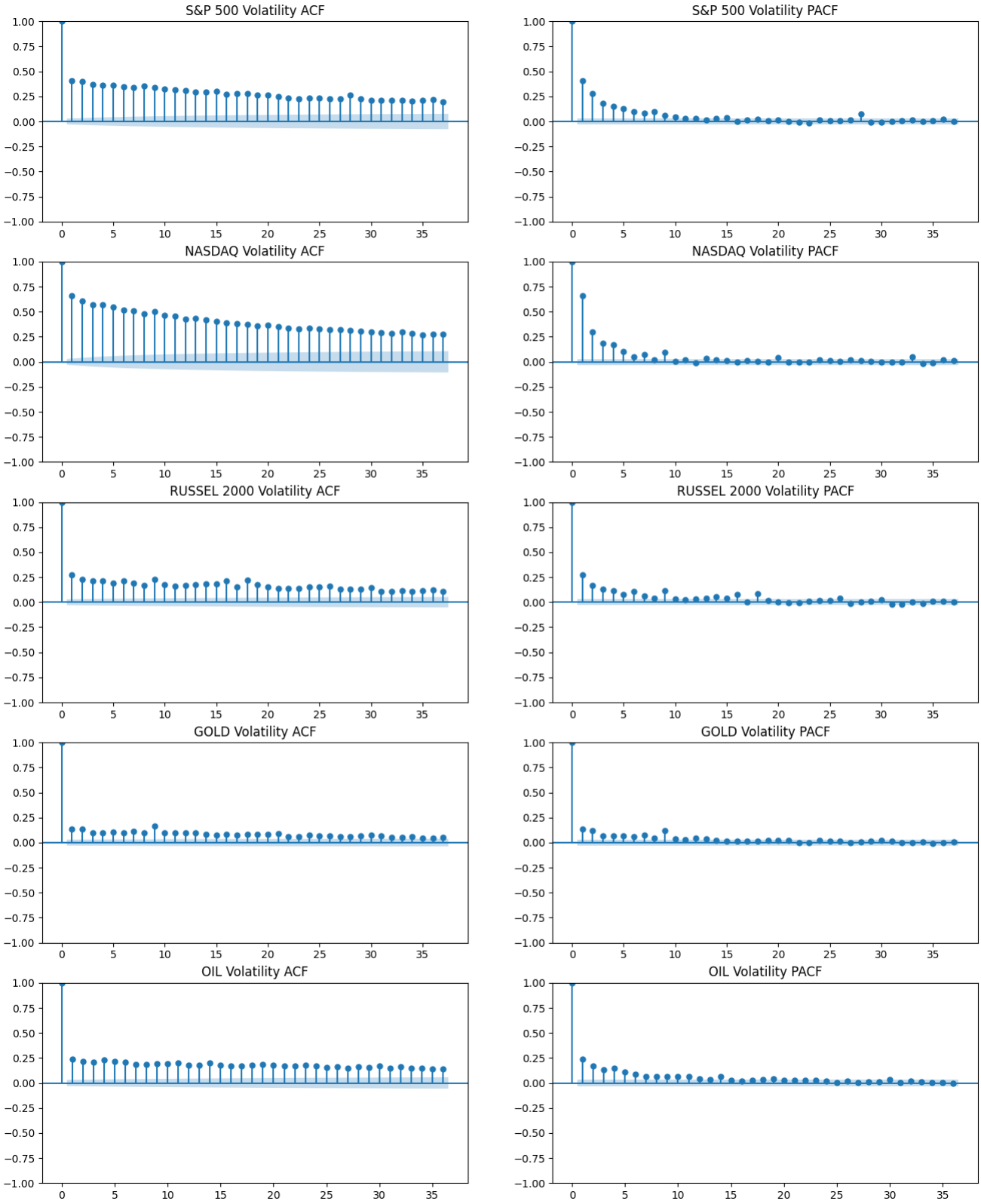


Figure A.3: Realized Volatility Auto-correlation and Partial Auto-correlation functions

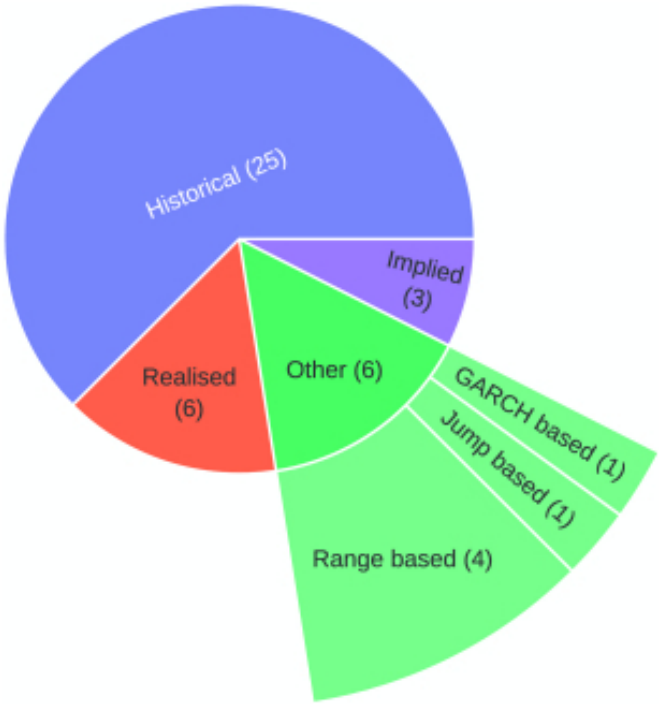


Figure A.4: Image from Ge et al. (2023a) showing the types of volatility definitions used by the papers in their literature review

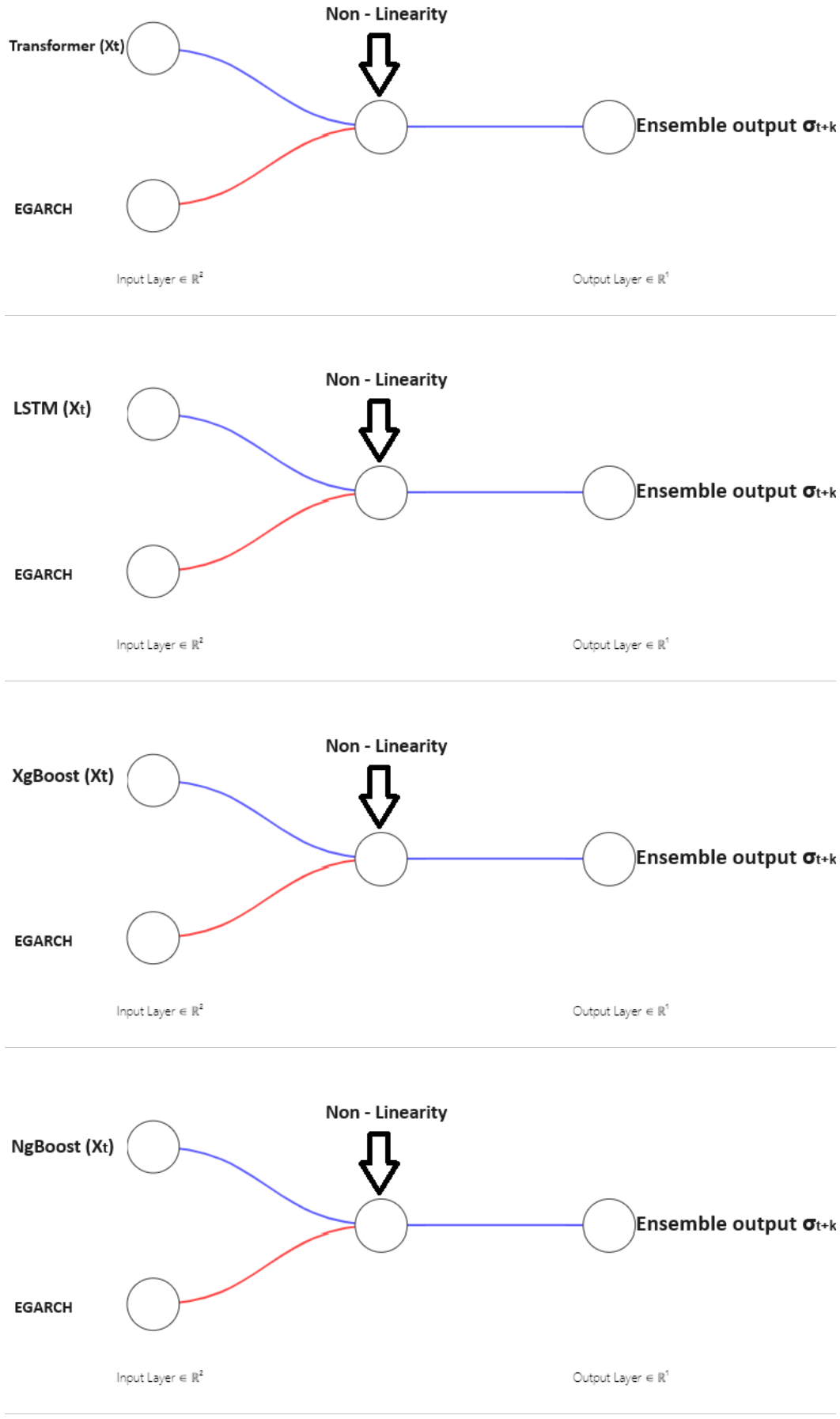


Figure A.5: CGARCH enhanced single models

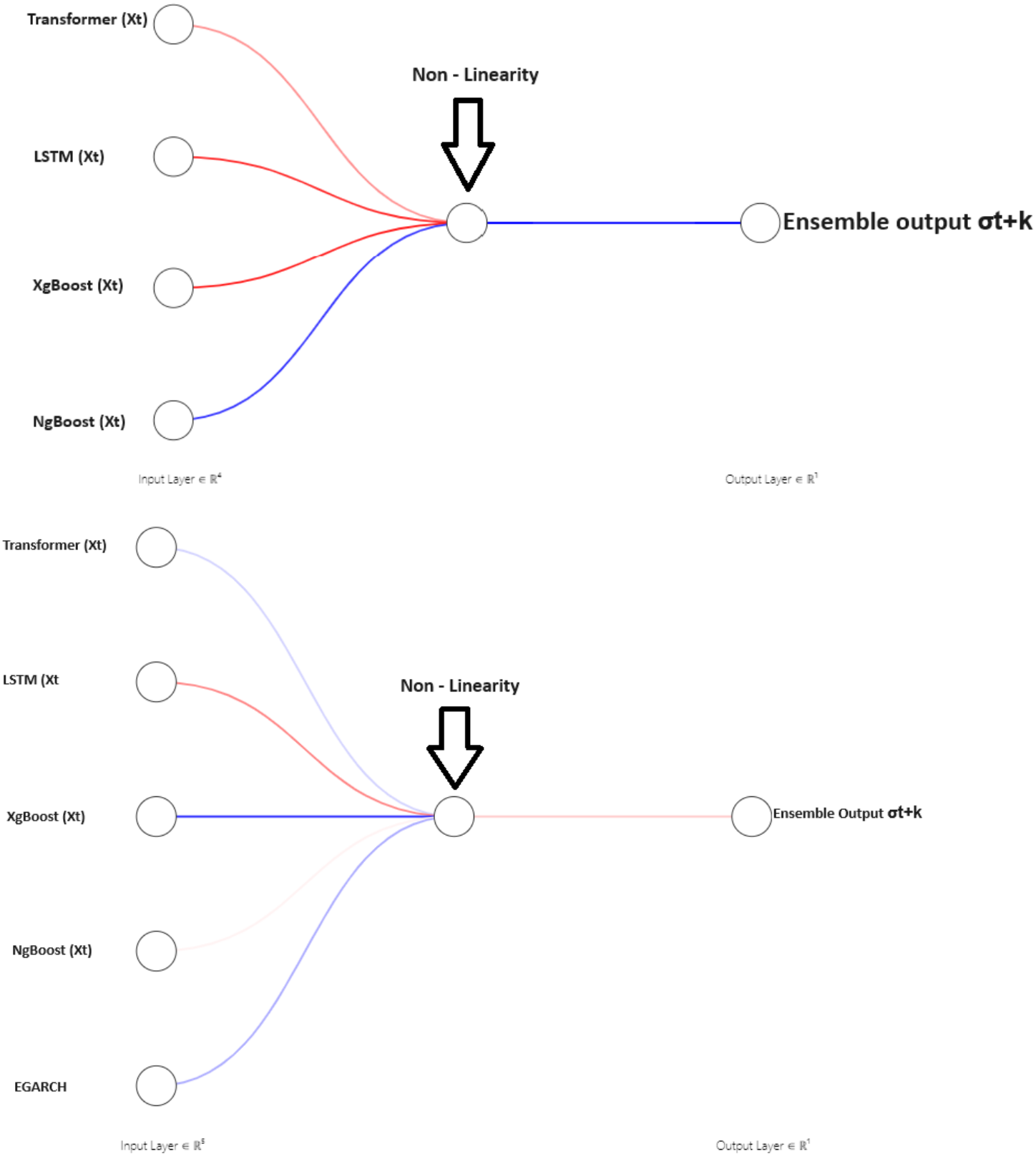


Figure A.6: X-N-L-T and CGARCH enhanced X-N-L-T

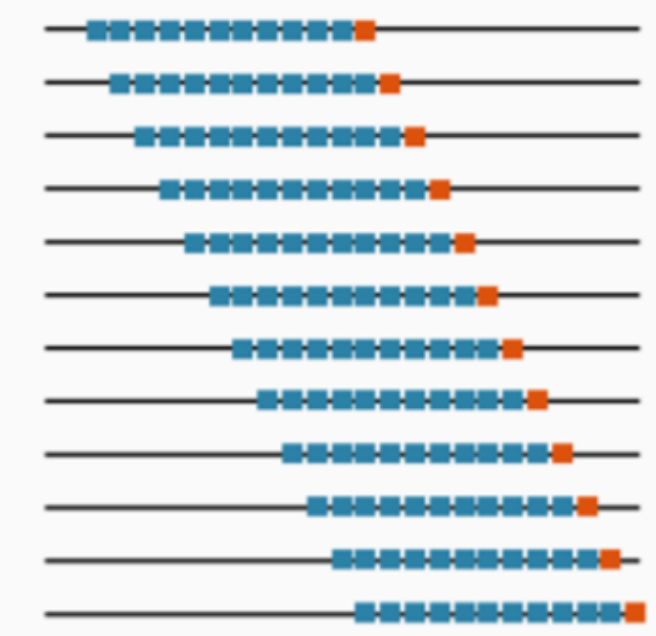


Figure A.7: Rolling Window procedure