Autoregression vs. Neural Networks: Forecasting Stock Implied Volatility

An in-depth analysis of techniques to predict implied volatility.

Hephaestus Applied Artificial Intelligence Association

MINERVA Investment Managment Society

Authors:

Member	Role
Leonardo Cremona	Head
Francesca Casillo	Member
Micaela Contini	Member
Francesco Sassi	Member
Anna Maruccio	Member

Milan, February 13, 2025

Contents

1 | Introduction

1.1 | Motivations

Volatility is among the most important measures used in finance: it captures how risky an investment is, providing fundamental information when making decisions involving financial securities. Volatility prediction is therefore extremely significant for any aspect of finance, from portfolio construction to asset pricing. Indeed, one of the most practical applications of predicted volatility is options pricing: the most common model used is the Black-Scholes, which uses Implied Volatility (IV) to price derivatives. Implied volatility, as the name suggests, is the market's expectation of future volatility of an underlying asset, generally 30-days forward, and is closely related to the price of options.

In this paper, we took a quantitative approach to predict implied volatility, without performing any sentimental analysis nor factoring in market perceptions of exogenous qualitative shocks. Therefore, **we assume that the value and behavior of future risk is explained fully by historical volatility**. Although this approach may seem simplistic in periods in which new crucial information is released to the public, it yielded surprising results, thus confirming the assumption that, at least partially, future implied volatility is explained by historical volatility.

1.2 | Autoregression vs Neural Networks

When forecasting stock implied volatility, analysts often compare two primary modeling paradigms: traditional autoregressive volatility models and neural network–based methods. Traditional models—such as GARCH, EGARCH, GJR-GARCH, FIGARCH, and APARCH—employ a parametric framework in which past volatility is linked to future volatility through a predetermined functional form. These models incorporate assumptions about the error term's distribution, typically Gaussian or Student's t, and use specific parameters to capture distinct phenomena. For example, EGARCH and GJR-GARCH are designed to account for asymmetric or "leverage" effects, where negative shocks to returns have a larger impact on future volatility than positive shocks. FIGARCH extends the model to capture long memory effects, while APARCH provides additional flexibility regarding how return deviations influence volatility. This explicit parameterization allows for direct financial interpretation—such as understanding the speed at which shocks decay—but it also limits the models' ability to capture complex, higher-order nonlinear relationships inherent in financial time series data.

In contrast, neural network models like LSTM, MLP, and Sigma Cell eschew strict parametric forms in favor of a data-driven approach. These nonparametric models learn patterns directly from historical data, enabling them to approximate highly nonlinear functions and capture intricate interactions that traditional models might overlook. LSTM networks, in particular, are well suited for sequential data, as they are designed to handle long-term dependencies and evolving patterns over time. Sigma Cell variants, with their specialized gating mechanisms, further enhance the ability to model cyclical or regime-switching dynamics. Although neural networks often achieve superior predictive accuracy by uncovering subtle patterns within complex datasets, their inner workings typically lack the straightforward interpretability of autoregressive models; individual parameters in a neural network do not readily translate into clear financial meanings.

The differences between these approaches extend to their data requirements and computational complexity. Autoregressive volatility models generally require smaller datasets to estimate parameters reliably, provided the underlying conditions of stationarity and stability hold. Their estimation routines, commonly based on maximum likelihood methods, are computationally efficient, making them well suited for applications like risk management where daily or monthly data are available. Neural networks, however, usually demand extensive datasets to effectively learn the underlying dynamics, particularly when deeper architectures are involved. Training such networks often requires significant computational resources—frequently involving GPUs—and careful tuning of various hyperparameters, including learning rate, network depth, and regularization. Moreover, while neural networks adapt flexibly to new patterns in data, they are prone to overfitting if the available data are insufficient or exhibit high levels of nonstationarity.

In capturing nonlinear and asymmetric effects, each approach has its own merits. Autoregressive volatility models incorporate built-in mechanisms to handle asymmetry. Models like EGARCH and GJR-GARCH explicitly model the leverage effect, ensuring that negative shocks produce a disproportionate increase

in volatility compared to positive shocks. FIGARCH, by accounting for long memory, and APARCH, through its flexible treatment of return deviations, both demonstrate how these models can be tailored to capture specific characteristics of volatility dynamics. Nevertheless, because these features are embedded in predefined structures, the extent to which they can model more complex or higher-order nonlinearities remains limited. Neural networks, by contrast, learn such effects implicitly. Their ability to approximate complex functions enables them to capture asymmetries, nonlinear dependencies, and even regime shifts without the need for predetermined functional forms. Recurrent architectures like LSTMs are particularly adept at learning memory effects from sequential data, making them valuable for short-term forecasting and high-frequency trading scenarios where market conditions can change rapidly.

The choice between these two modeling frameworks often depends on the specific application and practical considerations. Traditional autoregressive models have long been favored by financial institutions and academic researchers due to their historical track record, regulatory acceptance, and the clarity of their parameter interpretations. They are widely used in risk management tasks, such as Value at Risk (VaR) calculations and stress testing, where understanding the reasons behind volatility shifts is as important as predicting the shifts themselves. On the other hand, neural network models shine in environments characterized by high-frequency data or when additional data sources—such as fundamental indicators, market sentiment, or unstructured information from news and social media—are incorporated into the analysis. In these complex data environments, the enhanced flexibility and pattern recognition capabilities of neural networks often yield superior predictive performance, even though this comes at the expense of model transparency.

Practical considerations further influence the choice between these paradigms. The interpretability of autoregressive models makes them particularly appealing in situations where transparency is essential, such as when presenting results to regulators or corporate boards. In contrast, if the primary objective is maximizing predictive accuracy, especially in high-dimensional or high-frequency settings, the data-driven approach of neural networks may be preferable despite its "black box" nature. However, the higher computational demands and potential for overfitting in neural network models require rigorous model risk management practices, such as cross-validation and robust uncertainty quantification, in contrast to the well-established statistical tests available for GARCH-family models. As a result, many practitioners opt to explore both methods—sometimes even integrating them—to strike an optimal balance between interpretability and forecasting performance.

Ultimately, the optimal strategy for predicting stock implied volatility depends on factors such as the quality and quantity of available data, computational resources, and the specific forecasting objectives at hand. In our analysis,we look exclusively at the raw prediction power of these two modeling approaches, from which the reader can pull their own conclusions for their specific use case.

1.3 | Data Sets

1.3.1 | Data Source and Collection

For our analysis, we sourced historical stock price data using the `yfinance` Python library, which provides access to financial market data through Yahoo Finance. The library allows efficient retrieval of adjusted closing prices over a specified time span. Given our focus on implied volatility forecasting, we required a robust dataset covering various financial instruments, including equity indices, commodities, and currency pairs.

To ensure consistency across assets, we developed a function to retrieve historical prices for a given ticker symbol over a predefined time span. The function `get ticker data()` utilizes `yfinance` to fetch daily adjusted closing prices over a specified number of years. This method was chosen to ensure uniformity in the representation of price series across different tickers, avoiding inconsistencies that may arise from using different pricing conventions (e.g., closing price vs. adjusted closing price). The adjusted closing price was specifically selected to account for corporate actions such as dividends and stock splits, ensuring a more accurate measure of price changes. The tickers on which volatility was calculated and forecasted are the following: `GSPC`, `IXIC`, `FTSE`, `FCHI`, `GDAXI`, `FTSEMIB.MI`, `AXJO`, `HSI`, `N225`, `NSEI`, `BTC-USD`, `GC=F`, `EURUSD=X`, `EURGBP=X`.

1.3.2 | Computing Volatility from Stock Prices

A crucial step in our methodology was the computation of historical volatility from the retrieved price series. Instead of directly using implied volatility tickers from financial sources, we opted to calculate historical volatility ourselves. The primary reason for this choice was data availability: not all tickers in our study had implied volatility readily accessible. By standardizing our volatility computation, we ensured a uniform approach across all assets, thereby eliminating discrepancies that could arise from missing or inconsistent implied volatility data.
To compute historical volatility, we followed a two-step approach:

1. **Log Returns Calculation**:
   - Given a time series of adjusted closing prices, we computed log returns using the following formula:
     $$r_t = \ln(P_t) - \ln(P_{t-1}) \tag{1.1}$$
     where P_t represents the adjusted closing price at time t, and r_t is the log return.
   - This transformation ensures that price changes are expressed in relative terms, making comparisons across different assets more meaningful.
2. **Historical Volatility Calculation**:
   - Using the computed log returns, we calculated historical volatility as the rolling standard deviation of returns over a predefined window (30 days in our case). The formula used is:
     $$\sigma_t = \sqrt{252} \times \text{std}(r_t) \tag{1.2}$$
   - Here, the standard deviation is computed over a rolling window, and the factor $\sqrt{252}$ annualizes the volatility, assuming 252 trading days in a year. This approach aligns with standard industry practices for volatility estimation.

The functions `compute returns()` and `compute historical volatility()` implemented this methodology efficiently. Our approach ensured that we derived volatility measures consistently across all tickers, providing a solid foundation for comparing autoregressive models and neural networks in forecasting implied volatility.

By standardizing our data preprocessing pipeline, we mitigated inconsistencies in volatility data availability and ensured fair model evaluation across multiple financial instruments. This preprocessing step was instrumental in maintaining methodological rigor and reproducibility in our analysis.

2 | Autoregression

In this section, we will go over not only the implementation of these models and the intuition behind them, as well as some of the results, but we will also look at some of the historical context behind each model and how they build upon each other. It will also become clear that there isn't a one size fits all for these models, certain models are suitable for more stable markets, and others for more volatile markets. Finally, before we start, the reader should keep in mind that in a real world setting, these models are paired with extra parameters to model market sentiment as well as policy decisions among many other indicators, and they are not the end all be all for volatility prediction.

2.0 | The ARCH Model for Predicting Stock Implied Volatility

Although not considered in our analysis, we take a brief look at the root of the auto-regressive models for context. The AutoRegressive Conditional Heteroskedasticity (ARCH) [4] model is a cornerstone of modern time series analysis, particularly in financial econometrics where volatility modeling is of prime importance. Originally developed by Robert F. Engle in 1982, the ARCH model was introduced to capture the phenomenon of volatility clustering—periods of high volatility often cluster together, followed by periods of relative calm. This model has since inspired a range of extensions (such as GARCH, EGARCH, and others) but remains fundamental to understanding how past data can inform estimates of current volatility. In stock markets, volatility can be estimated from market prices of options (implied volatility), and ARCH-based methods provide a rigorous framework for forecasting this implied volatility over time.

2.0.1 | Brief Historical Context

Prior to Engle's seminal work, econometric models typically assumed constant variance (homoskedasticity) in regression disturbances. This assumption, although mathematically convenient, often failed to capture the real-world fluctuation patterns in financial markets. By observing empirical data, Engle noticed that large price changes in financial asset returns are more likely to be followed by further large changes (regardless of sign), while small changes tend to follow small changes—a phenomenon referred to as volatility clustering.
In his 1982 paper, "Autoregressive Conditional Heteroskedasticity with Estimates of the Variance of U.K. Inflation," Engle introduced the ARCH(1) model as a new way to describe time-varying volatility. Heteroskedasticity means that variances are not constant over time; instead, each period's variance depends on past observations. This framework revolutionized econometric modeling of volatility, earning Engle the Nobel Prize in Economics in 2003.

2.0.2 | The Intuition Behind ARCH

"Conditional heteroskedasticity" implies that the variance of the current error term (or innovation) depends on the magnitude of previous error terms. In simpler terms:

> Volatility today (i.e., this period's conditional variance) depends on the magnitude of deviations (squared) from the mean in previous periods.

This dependence on past squared errors is what enables the ARCH model to capture volatility clustering. When large shocks (positive or negative) occur, the memory of these shocks influences future volatility estimates.
For financial data such as stock returns or implied volatilities derived from options, this approach is particularly compelling.

2.0.3 | The Mathematical Formulation of ARCH

The simplest version, **ARCH(1)**, may be introduced in a univariate time series context. Let $\{r_t\}$ represent the returns or innovations (e.g., deviations of returns from their mean) at time t. We often write:

$$r_t = \sigma_t \varepsilon_t \tag{2.1}$$

where:

- ε_t is a white noise process with zero mean and unit variance ($\varepsilon_t \sim$ i.i.d. $(0, 1)$).

- σ_t is the time-varying standard deviation (volatility) of r_t.

ARCH models specify that σ_t^2, the conditional variance of r_t, depends on past squared values of the series. Specifically, an ARCH(p) model can be written as:

$$\sigma_t^2 = \omega + \alpha_1 r_{t-1}^2 + \alpha_2 r_{t-2}^2 + \cdots + \alpha_p r_{t-p}^2 \tag{2.2}$$

where:

- $\omega > 0$ is a constant term.
- $\alpha_i \geq 0$ are the coefficients measuring the influence of past squared innovations (or returns).

In the simplest **ARCH(1)** case, we have just one lag:

$$\sigma_t^2 = \omega + \alpha_1 r_{t-1}^2. \tag{2.3}$$

2.0.4 | Estimation and Forecasting

Estimation of ARCH models typically uses maximum likelihood estimation (MLE). Given a set of observed returns $\{r_t\}$:

1. **Initial guess:** Start with initial parameter values ω and α_i.
2. **Iterative optimization:** Use a numerical optimization algorithm (e.g., BFGS, Newton-Raphson) to find the parameter values that maximize the likelihood of observing the actual return series.
3. **Conditional variance updates:** For each time step t, the conditional variance σ_t^2 is computed from its previous values, and the likelihood is calculated.

2.0.5 | Limitations

Limitations:

- Parameter proliferation in high-order models.
- Does not account for leverage effects.
- May not capture long memory effects efficiently.

2.1 | GARCH: The Next Generation of Volatility Models

Building on the foundations laid by the ARCH model, the Generalized AutoRegressive Conditional Heteroskedasticity (GARCH) framework was introduced by Tim Bollerslev in 1986. GARCH extends the original ARCH formulation by allowing not only past squared innovations but also past conditional variances to affect the current level of volatility. This relatively simple enhancement often provides a more efficient and realistic characterization of volatility in financial markets—especially useful for modeling and predicting stock implied volatility.
The GARCH model (and its many variations) has since become the *de facto* workhorse for financial volatility modeling, although workhorse does not necessarily imply the most accurate, but just the most practical and widely used. Its success is largely due to its ability to capture long-memory volatility persistence with fewer parameters, facilitating more stable and tractable forecasts of time-varying volatility for assets and their implied volatilities.

2.1.1 | GARCH Basics and Intuition

Recall that the essential idea of ARCH is:

> **Today's volatility depends on the magnitude of past squared returns.**

The **GARCH**[2] framework supplements this by adding a dependence on past *volatility levels* themselves. In other words, where ARCH conditions today's variance on *only* the squared innovations from previous periods, GARCH conditions it on *both* those squared innovations *and* yesterday's (and possibly earlier days') estimated variance.
This addition of a feedback loop from past volatility estimates achieves two main benefits:

1. **Efficiency :** Instead of requiring many lags of squared returns, GARCH can achieve a long memory effect with fewer parameters.
2. **Realistic Volatility Dynamics:** Empirical evidence shows volatility often reacts not just to extreme price changes but also to the lingering impact of higher or lower overall volatility regimes.

2.1.2 | Mathematical Formulation

The most commonly used GARCH specification in practice (and the one we've implemented) is the **GARCH(1,1)** model, which can be viewed as a direct extension of the ARCH(1) idea. Let $\{r_t\}$ represent the return process (or innovations) at time t. We write:

$$r_t = \sigma_t \varepsilon_t,$$

where:

- ε_t is white noise with zero mean and unit variance,
- σ_t is the time-varying standard deviation of r_t.

The **GARCH(1,1)** conditional variance specification is:

$$\sigma_t^2 = \omega + \alpha_1 \, r_{t-1}^2 + \beta_1 \, \sigma_{t-1}^2,$$

where

- $\omega > 0$ is a constant term,
- $\alpha_1 \geq 0$ measures the influence of last period's squared innovation on current volatility (similar to ARCH),
- $\beta_1 \geq 0$ measures the influence of last period's volatility level on current volatility.

For a general **GARCH**(p, q) model, the conditional variance depends on p lags of σ^2 and q lags of r^2:

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

Key Properties and Constraints

1. **Positivity:** $\omega > 0$, and typically $\alpha_i, \beta_j \geq 0$ to ensure the variance remains nonnegative.
2. **Stationarity:** For a GARCH(1,1), a common stationarity condition is $\alpha_1 + \beta_1 < 1$. This helps ensure that volatility does not explode to infinity.
3. **Persistence:** A large value of $\alpha_1 + \beta_1$ close to 1 implies high persistence of volatility shocks; i.e., volatility remains elevated for an extended period following a large market move.

2.1.3 | Implementation

For all the ARCH based models, we used the arch library in python, a comprehensive and versatile tool for implementing these types of models.
The definition is straight forward,

```
model = arch_model(returns, vol='GARCH', p=1, q=1, dist='normal')
```

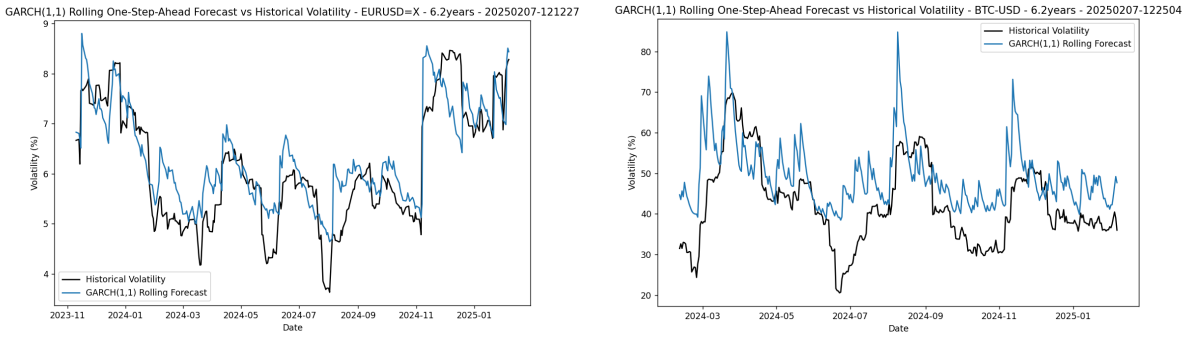
Then we used a rolling forecast method to create a graph of predictions. The number of days ahead that these models can predicted is called the forecast horizon, and tends to be between 1 and 5 days. However, to create a graph of predictions, we must necessarily use a one day forecast horizon, or their predictions would over lap and would be messy on the graph, as well as the evaluation being orders of magnitude more complicated.
The rolling forecast method allows for each point on the graph to be a one day ahead prediction, using a rolling window of past data to inform predictions. What it does is it trains the model on past data (the window) up to a time t, making parameter estimation more precise, and then predicts the value for time t+1, which is then graphed. The function is then called again to predict time t+2, but the data from

the previous day is not the data the model predicted, but rather the realised volatility of the previous day. Longer windows (500+) allow for smoother curves and less sensitive to short term fluctuations, however can be slow to react to sudden market changes. Shorter windows (i.e., 30 days) are much more responsive, but tend to be very jumpy. In our implementation, we used an initial window of around 5 years (1290 days) to match the training data of the neural network models, and then switched to a 30 day rolling window for the "test" portion of the data to mimic testing a trained neural network. This has the advantages of allowing the model to be more resilient to short term fluctuations, but still responsive to sudden market changes. However, as you will see in the following section, these models remain very jumpy

2.1.4 | Estimation and Forecasting

The estimation procedure for GARCH is similar to ARCH:

1. **Maximum Likelihood Estimation (MLE):** Parameters ω, α_i, and β_j are typically estimated by maximizing the likelihood function under the assumption that $\{r_t\}$ (conditional on σ_t^2) follows a certain distribution—often Gaussian, but sometimes Student's t for fatter tails.
2. **Iterative Methods:** As with ARCH, practitioners use iterative optimization techniques (e.g., BFGS, Newton-Raphson) to find parameter estimates.
3. **Forecasting Volatility:** After fitting, we use the parameter estimates to forecast future volatility:
   - A one-step-ahead variance forecast in GARCH(1,1) is
   $$\hat{\sigma}_{t+1}^2 = \hat{\omega} + \hat{\alpha}_1 r_t^2 + \hat{\beta}_1 \hat{\sigma}_t^2.$$
   - Multi-step forecasts generally converge to the long-run average variance, given by
   $$\frac{\hat{\omega}}{1 - \hat{\alpha}_1 - \hat{\beta}_1},$$
   if $\hat{\alpha}_1 + \hat{\beta}_1 < 1$.



((a)) GARCH Best Case; MAE=0.517

((b)) GARCH Worst Case; MAE=11.47

Figure 2.1: Best and Worst case for GARCH

2.1.5 | Why GARCH Might Be Preferred to ARCH

1. **Efficiency :**
   A GARCH(1,1) can effectively capture long-memory effects in volatility with just three parameters (ω, α_1, and β_1). By contrast, an ARCH(p) might need many more parameters (and lags) to produce a comparable volatility autocorrelation profile.
2. **Adaptive Memory:**
   GARCH allows volatility to be influenced by its own past levels. This "memory of volatility" aligns well with observed financial time series, where volatility states tend to persist even if there are no large shocks in subsequent periods.
3. **Broad Applicability:**
   GARCH-based techniques are widely used across equity, fixed-income, FX, and commodities markets, making the model a standard tool in quantitative finance.

2.1.6 | Advantages and Limitations

Advantages:

1. **Efficient Parameterization:** GARCH(1,1) uses fewer parameters yet can capture significant volatility persistence.
2. **Widely Researched and Validated:** Decades of empirical studies show GARCH typically offers strong in-sample and out-of-sample performance for many financial assets.
3. **Flexible Extensions:** Numerous variants (EGARCH, TGARCH, IGARCH, etc.) address asymmetry, leverage effects, or other real-world market features.

Limitations:

1. **Symmetry in Volatility Shocks:** The basic GARCH(1,1) does not differentiate between positive and negative returns (it uses squared returns). Real markets often exhibit "leverage effects," where negative shocks have a stronger impact on future volatility than positive shocks.
2. **Model Misspecification:** If the true volatility process has structural breaks or regime shifts (e.g., different volatility regimes), a simple GARCH(1,1) may be inadequate.
3. **Distributional Assumptions:** Assuming normally distributed errors might underestimate tail risk. More sophisticated versions (e.g., GARCH with t-distributed errors) can address heavier tails.

2.2 | EGARCH: Capturing Asymmetric Volatility Effects

While GARCH models offer an efficient way to capture volatility clustering, they often assume that positive and negative shocks have a *symmetric* effect on future volatility. In practice, many financial markets exhibit the so-called **leverage effect**: negative returns (e.g., price drops) tend to increase future volatility more than positive returns of the same magnitude. To accommodate this asymmetry, Daniel B. Nelson introduced the **Exponential GARCH (EGARCH)**[6] model in 1991, providing a more flexible framework that captures differential impacts of shocks and still maintains a tractable form for forecasting volatility—particularly relevant when modeling implied volatility in equities.

2.2.1 | Key Idea and Model Specification

Instead of modeling σ_t^2 linearly (as in GARCH) or modeling it strictly via past squared returns (as in ARCH), EGARCH works with the *logarithm* of the conditional variance. A standard **EGARCH(1,1)** can be written as:

$$\log\left(\sigma_t^2\right) = \omega + \beta \log\left(\sigma_{t-1}^2\right) + \alpha\Big(\big|z_{t-1}\big| - E\big[\big|z_{t-1}\big|\big]\Big) + \gamma\, z_{t-1},$$

where:

- σ_t^2 is the conditional variance of r_t.
- $z_{t-1} = \frac{r_{t-1}}{\sigma_{t-1}}$ are the standardized residuals (shocks from the previous period).
- $E\big[\big|z_{t-1}\big|\big]$ is the expected value of the absolute standardized shock.
- ω, α, β, and γ are parameters to be estimated.

This formulation introduces several unique features:

1. **Log of Variance:** Modeling $\log\left(\sigma_t^2\right)$ rather than σ_t^2 directly *automatically* ensures $\sigma_t^2 > 0$ without imposing positivity constraints on the parameters.
2. **Magnitude of Shocks:** The term $\big|z_{t-1}\big| - E\big[\big|z_{t-1}\big|\big]$ measures how large the absolute shock is, relative to its average magnitude. This captures the size effect on volatility. For a standard normal distribution (as in our case) the expectations is $\sqrt{2/\pi}$, but it can differ base on the assumed distribution.

3. **Sign of Shocks (Leverage Effect):** The coefficient γ multiplies z_{t-1}, thereby incorporating the *sign* of the innovation. If $\gamma < 0$, negative shocks increase future volatility more than positive shocks of the same magnitude, reflecting the leverage effect commonly seen in equities.

4. **Persistence:** The parameter β plays a role analogous to β_1 in GARCH(1,1), governing how slowly or quickly volatility reverts to its long-term average.

2.2.2 | Asymmetry and Leverage Effect

A hallmark of EGARCH is its built-in capacity to model asymmetry. Specifically:

- **Symmetric Case:** If $\gamma = 0$, then the model becomes symmetric in terms of volatility response to positive or negative shocks.
- **Leverage Effect:** If $\gamma < 0$, negative shocks have a stronger impact on future volatility than positive shocks of the same magnitude.

In equity markets, negative price movements often accompany rising leverage ratios for firms, thus heightening perceived risk and volatility. EGARCH captures this mechanism by directly incorporating the sign and magnitude of shocks into the conditional variance equation.

2.2.3 | Implementation

For the EGARCH model, the only thing that changes is the model definition

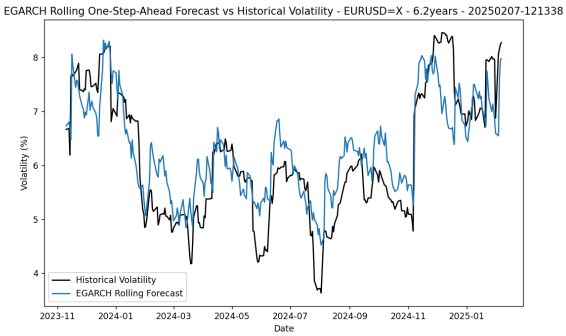
```
model = arch_model(returns, vol='EGARCH', p=1, q=1, dist='normal')
```

The rolling forecast and horizon stay the same, refer back to subsection 2.1.4 if re clarification is needed. All the parameter estimations are handled by the library.

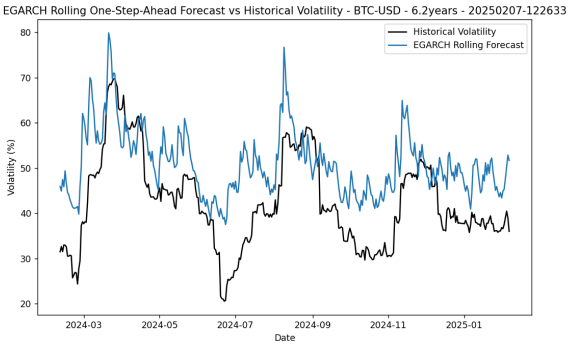
2.2.4 | Estimation and Forecasting

Estimation of EGARCH typically proceeds via **Maximum Likelihood Estimation (MLE)**, assuming a probability distribution for the standardized residuals z_t. Common distributions include:

- **Normal Distribution:** Often a baseline assumption, though it may underestimate tail risk.
- **Student's t Distribution:** Provides heavier tails for large, infrequent shocks.



((a)) EGARCH Best Case; MAE=0.53



((b)) EGARCH Worst Case; MAE=11.3

Figure 2.2: Best and Worst case for EGARCH

The log-likelihood function is maximized using iterative algorithms (e.g., BFGS or other quasi-Newton methods). Once estimated, forecasting follows similarly to GARCH. For instance, the one-step-ahead forecast for $\log(\sigma^2_{t+1})$ is obtained by plugging in the latest estimates and data into the EGARCH equation. Multi-step forecasts converge toward a long-run equilibrium implied by the parameters (assuming stationarity conditions are met).

2.2.5 | Advantages and Limitations

Advantages

1. **Positivity Guaranteed:** The log specification means no ad-hoc constraints on parameters to keep variance positive.
2. **Captures Leverage Effects:** EGARCH explicitly incorporates the sign of shocks, making it well-suited for equity markets.
3. **Flexible Functional Form:** Taking the log of variance allows a more flexible and potentially smoother dynamic response compared to linear GARCH.

Limitations

1. **Complexity:** Estimation is more involved than standard GARCH, and convergence can be sensitive to starting values.
2. **Interpretation of Parameters:** The interpretation of ω and γ can be less intuitive than in linear GARCH models.
3. **Potential Overfitting:** Like any model with additional parameters, EGARCH can overfit if not carefully regularized or validated, especially for smaller sample sizes.

2.3 | GJR-GARCH: Accounting for Leverage in a Threshold Framework

The GJR-GARCH[5] model, named after Glosten, Jagannathan, and Runkle (1993), builds upon the GARCH framework by introducing an additional leverage term to capture the asymmetric effects specifically tied to negative shocks. This enhancement is particularly relevant in equity markets, where downward price movements often trigger stronger volatility reactions than upward movements of similar magnitude.

2.3.1 | Key Idea and Model Specification

A standard **GJR-GARCH(1,1)** model can be written as:

$$\sigma_t^2 = \omega + \alpha_1 \varepsilon_{t-1}^2 + \gamma_1 \varepsilon_{t-1}^2 I\{\varepsilon_{t-1} < 0\} + \beta_1 \sigma_{t-1}^2,$$

where:

- ε_{t-1} are the residuals (innovations) from the mean equation, often $r_{t-1} - \mu$,
- σ_{t-1}^2 is the conditional variance from the previous period,
- $I\{\varepsilon_{t-1} < 0\}$ is an *indicator function* that equals 1 if $\varepsilon_{t-1} < 0$ and 0 otherwise,
- $\omega > 0$ is a constant,
- $\alpha_1 \geq 0$ measures the influence of the previous period's squared shock (similar to standard GARCH),
- γ_1 captures the *additional* impact of negative shocks on future volatility,
- $\beta_1 \geq 0$ measures the persistence of past volatility levels.

Interpretation of the Indicator Function. The term $\gamma_1 \varepsilon_{t-1}^2 I\{\varepsilon_{t-1} < 0\}$ effectively adds extra weight to squared shocks when they are negative. Thus, a negative shock of a given magnitude will increase next period's volatility more than a positive shock of the same magnitude would.

2.3.2 | Asymmetry and Leverage Effect

By explicitly modeling the difference between negative and positive returns, GJR-GARCH is able to capture the **leverage effect**—a phenomenon often observed in equity markets where downward price movements increase financial leverage in firms and may result in sharper volatility spikes. This asymmetry term is critical for analysts and traders who rely on realistic volatility estimates to manage downside risk.

2.3.3 | Implementation

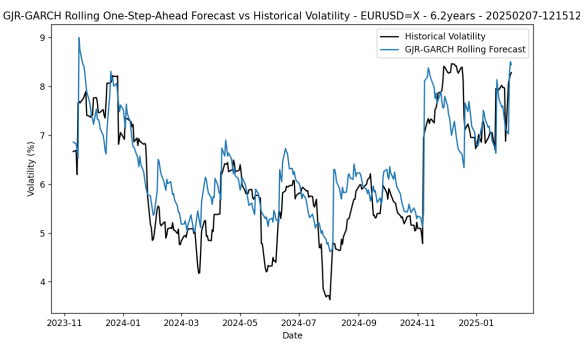
As per the previous models, the same rolling forecast method and windows are used, and the model is defined using the arch library as follows

```
model = arch_model(returns, vol='GARCH', p=1, o=1, q=1, dist='normal')
```

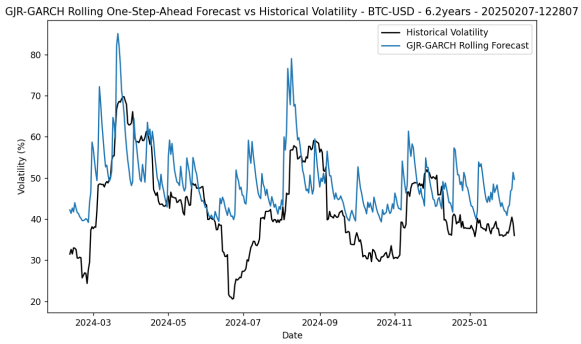
where the o parameter here represents γ_1, and is set to one to include it in the model (the real value is then estimated by the library)

2.3.4 | Estimation and Forecasting

- **Estimation:** GJR-GARCH parameters are typically estimated via **Maximum Likelihood Estimation (MLE)** under an assumed distribution for the errors (e.g., Normal or Student's t). Optimization algorithms like BFGS or other gradient-based methods are employed.
- **Forecasting:** After estimation, one-step-ahead forecasts of σ^2_{t+1} are obtained by plugging the latest values into the GJR-GARCH equation. Multi-step forecasts can be derived by iteratively applying the conditional variance recursion, though they may converge to a long-run variance if stationarity conditions (like $\alpha_1 + \gamma_1/2 + \beta_1 < 1$ in the GJR-GARCH(1,1) case) are satisfied.



((a)) GJR-GARCH Best Case; MAE=0.5151



((b)) GJR-GARCH Worst Case; MAE=11.78

Figure 2.3: Best and Worst case for GJR-GARCH

2.3.5 | Advantages and Limitations

Advantages

1. **Captures Asymmetry Explicitly:** By including an indicator for negative shocks, GJR-GARCH naturally handles leverage effects.
2. **Parsimonious Extension of GARCH:** It only adds one parameter (γ_1) to the standard GARCH(1,1) while delivering significantly improved realism for equity markets.
3. **Flexible Implementation:** Many software libraries (`arch`, `rugarch`, etc.) directly support GJR-GARCH, making it straightforward to implement.

Limitations

1. **Binary Threshold:** The indicator function is a sharp cutoff at zero, which may not always capture other subtle asymmetries (e.g., news-driven or regime-dependent effects).
2. **Possible Overfitting:** Adding more parameters (for instance, multiple thresholds) can complicate estimation, especially with limited data.
3. **Distributional Assumptions:** If large outliers or heavy tails are present, practitioners may still need to consider Student's t or other distributions for the error term.

2.4 | FIGARCH: Modeling Long Memory in Volatility

While GARCH-type models capture volatility clustering and persistence, many financial time series exhibit even longer-lived volatility dependencies, often referred to as "long memory." The **Fractionally Integrated GARCH (FIGARCH)**[1] model, introduced by Baillie, Bollerslev, and Mikkelsen (1996), extends the GARCH framework to accommodate fractional integration in the volatility process, thereby capturing these protracted dependencies more effectively.

2.4.1 | Key Idea and Model Specification

A general **FIGARCH**(p, d, q) model can be viewed as a fractional integration of the GARCH process. Typically, the model is written using lag operators. Let L be the lag operator ($Lx_t = x_{t-1}$). The FIGARCH process for the conditional variance σ_t^2 often takes the form:

$$\Big(1 - \beta(L)\Big)\,\sigma_t^2 \;=\; \omega + \Big[1 - \phi(L)\,(1-L)^d\Big]\,\varepsilon_t^2,$$

where:

- $\varepsilon_t = r_t - \mu$ (the residual or innovation from the mean equation),
- σ_t^2 is the conditional variance,
- Lag Polynomials $\beta(L)$ and $\phi(L)$: Represent the weighted impact of past variances and shocks, respectively. They are computed as

$$\beta(L) \;=\; \beta_1\,L + \beta_2\,L^2 + \cdots, \quad \phi(L) \;=\; \phi_1\,L + \phi_2\,L^2 + \cdots.$$

- Fractional Differencing Operator $(1-L)^d$: Defined using a binomial series expansion:

$$(1-L)^d \;=\; \sum_{k=0}^{\infty} \binom{d}{k} (-L)^k, \quad \text{with} \quad \binom{d}{k} \;=\; \frac{\Gamma(d+1)}{\Gamma(k+1)\,\Gamma(d-k+1)}.$$

  This operator allows even old shocks to have a diminishing yet persistent effect on today's volatility.

- d is the fractional differencing parameter, taking values in $0 \le d \le 1$.

A commonly used parsimonious version is the **FIGARCH(1, d, 1)** model. The key parameter is d, which controls the degree of fractional integration:

- $d = 0$: The model reduces to GARCH.
- $0 < d < 1$: Indicates long memory, with autocorrelations of volatility decaying more slowly.
- $d = 1$: Often considered non-stationary, so typically d must be strictly less than 1 for stability.

Long Memory Interpretation. A positive d close to 1 suggests that a shock to volatility dies out very gradually, implying high persistence. If d is small (but greater than 0), the memory is still longer than standard GARCH, but not extreme.

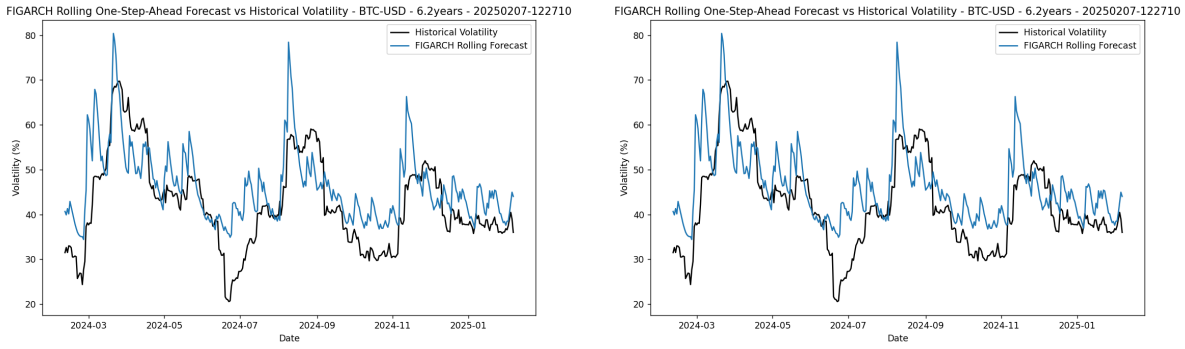
2.4.2 | Implementation

Same as other models, it is defined using the arch library, and the other parameters are estimated automatically

```
am = arch_model(returns, vol='FIGARCH', p=1, q=1, dist='normal')
```

2.4.3 | Estimation and Forecasting

- **Estimation:** Estimating FIGARCH involves maximum likelihood or quasi-maximum likelihood methods, with specialized optimization routines to handle the fractional differencing term. Convergence can be more challenging compared to standard GARCH, and starting values for d can impact the final estimates.
- **Forecasting:** Once fitted, one-step-ahead forecasts of σ^2_{t+1} are computed similarly to GARCH-type models, except the volatility recursion includes the fractional differencing component. Multi-step forecasts reflect the slowly decaying effect of past volatility shocks, yielding longer-lasting impacts in the forecast path than a standard GARCH model would.



((a)) FIGARCH Best Case; MAE=0.648

((b)) FIGARCH Worst Case; MAE=8.78

Figure 2.4: Best and Worst case for FIGARCH

2.4.4 | Advantages and Limitations

Advantages

1. **Captures Long Memory:** FIGARCH can model slowly decaying volatility autocorrelations that standard GARCH-type models might miss.
2. **More Realistic in Certain Markets:** For assets known to exhibit persistence over long horizons (e.g., certain commodity, FX, or equity markets), FIGARCH may yield improved fit and forecasts.
3. **Adaptable Framework:** The model generalizes GARCH, retaining much of its structure while allowing for fractional integration.

Limitations

1. **Complex Estimation:** Fitting FIGARCH requires more sophisticated numerical methods and can be sensitive to starting values.
2. **Interpretation of d:** While d signifies the degree of long memory, it can be less intuitive to interpret and validate compared to simpler GARCH parameters.
3. **Higher Computational Cost:** The fractional differencing terms increase the computational complexity, potentially slowing down forecasting for large datasets or real-time applications.

2.5 | APARCH: Asymmetric Power ARCH

Observing that volatility series often exhibit both asymmetry and nonlinear relationships with past shocks, researchers sought a unified model that could address these features more flexibly than standard GARCH or even some of its asymmetric variants.The **Asymmetric Power ARCH (APARCH)**[3] model, introduced by Ding, Granger, and Engle (1993), generalizes the ARCH/GARCH framework by allowing for a power transformation of the conditional standard deviation and explicitly modeling asymmetry. Through its flexible parameterization, APARCH can capture various empirical features of financial volatility, such as heavy tails, skewness, and different types of leverage effects.

2.5.1 | Key Idea and Model Specification

A general **APARCH(p, q)** model specifies the conditional standard deviation raised to a power δ (hence the term "power ARCH"). In the simplest APARCH(1,1) form, the model can be expressed as:

$$\sigma_t^\delta = \omega + \alpha\Big(|\varepsilon_{t-1}| - \gamma\varepsilon_{t-1}\Big)^\delta + \beta\,\sigma_{t-1}^\delta,$$

where:

- σ_t is the conditional standard deviation (not variance) of r_t,
- $\varepsilon_t = r_t - \mu$ is the innovation or residual,
- $\delta > 0$ is the power parameter,
- γ is the asymmetry or leverage parameter,
- ω, α, β are nonnegative parameters (with $\omega > 0$ typically).

Role of δ. By allowing $\delta \neq 2$, APARCH can better capture deviations from the standard GARCH assumption (which implicitly uses $\delta = 2$). For instance, δ might be estimated to be closer to 1, which can reflect absolute deviations rather than squared deviations.

Asymmetry via γ. If $\gamma \neq 0$, then positive and negative shocks do not affect volatility in the same way. Typically, $\gamma > 0$ indicates that negative shocks (when $\varepsilon_{t-1} < 0$) have a larger effect on volatility than positive shocks of the same magnitude.

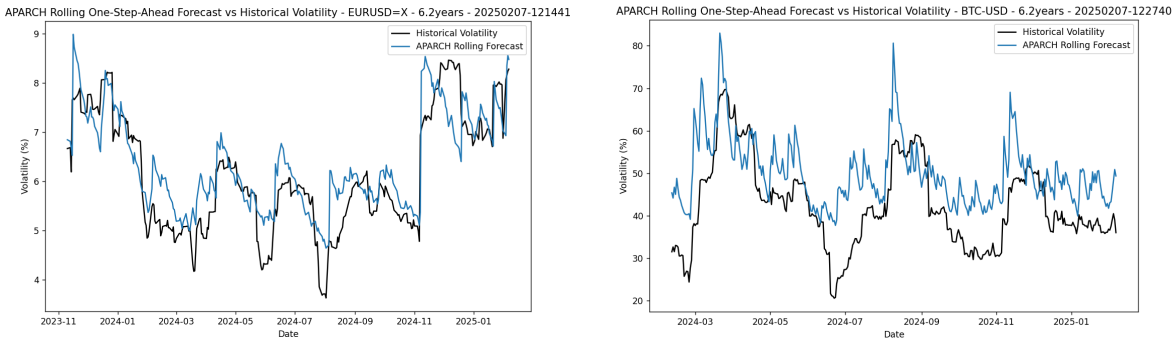
2.5.2 | Implementation

As for the other models, this one is implemented using the arch library, and the variables are estimated by the function

```
am = arch_model(returns, vol='APARCH', p=1, q=1, dist='normal')
```

2.5.3 | Estimation and Forecasting

- **Estimation:** As with other GARCH-type models, APARCH parameters are often estimated via **Maximum Likelihood Estimation (MLE)**, assuming a chosen distribution for the error terms (e.g., Normal or Student's t). Numerical optimization routines (e.g., BFGS) are employed to find the set of parameters $(\omega, \alpha, \beta, \gamma, \delta)$ that best fits the data.
- **Forecasting:** Once fitted, one-step-ahead forecasts of σ_{t+1}^δ are computed by plugging in the last observed residual and standard deviation. Multi-step forecasts can be derived by iterating the APARCH recursion, similar to GARCH-based models. The resulting σ_t can then be squared and annualized if the objective is to forecast variance in an annualized scale.



((a)) APARCH Best Case; MAE=0.518

((b)) APARCH Worst Case; MAE=11.24

Figure 2.5: Best and Worst case for APARCH

2.5.4 | Advantages and Limitations

Advantages

1. **Flexible Functional Form:** The power parameter δ introduces a wide range of possible dynamic behaviors.
2. **Direct Asymmetry Control:** APARCH explicitly separates the impact of negative shocks from positive shocks through γ.
3. **Unified Framework:** It generalizes various ARCH/GARCH family models under one umbrella, offering a potentially better fit for data with complex volatility structures.

Limitations

1. **Increased Complexity:** More parameters (γ, δ) can make convergence slower and estimation more prone to local optima.
2. **Interpretation Challenges:** δ and γ can be less intuitive to interpret compared to standard GARCH parameters.
3. **Overfitting Risk:** As with other extended GARCH models, added flexibility can lead to overfitting if the sample size is not sufficiently large.

3 | Neural Networks

3.1 | MLP

The Multi-Layer Perceptron (MLP) is a class of feedforward artificial neural networks that consists of multiple layers of interconnected neurons. It is made up of an input layer, one or more hidden layers, and an output layer, where each layer is fully connected to the next through weighted connections. Mathematically, an MLP with one hidden layer can be expressed as:

$$h^{(1)} = \phi(W^{(1)}x + b^{(1)}) \tag{3.1}$$

$$y = \phi_o(W^{(2)}h^{(1)} + b^{(2)}) \tag{3.2}$$

where:

1. x is the input vector,
2. $W^{(1)}$ and $b^{(1)}$ are the weight matrix and bias for the hidden layer,
3. $h^{(1)}$ is the hidden layer output after applying activation function ϕ,
4. $W^{(2)}$ and $b^{(2)}$ are the weight matrix and bias for the output layer,
5. y is the final output, and ϕ_o is the activation function for the output layer.

3.1.1 | Training and Loss Function

The MLP is trained using backpropagation, an algorithm that computes gradients through the chain rule and updates weights through Adam optimization. The loss function for the regression task is:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}i)^2 \tag{3.3}$$

The weights are updated iteratively using gradient descent, ensuring the network learns to minimize the error.

3.1.2 | Advantages vs. Disadvantages of MLP

Advantages:

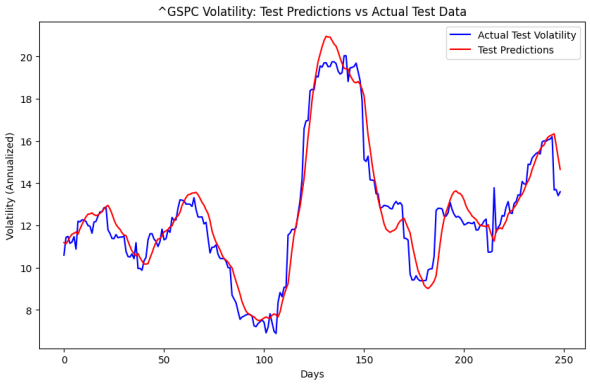
- **Universal Approximation:** MLPs can approximate any continuous function given enough hidden neurons.
- **Feature Learning:** They can learn feature representations automatically, without requiring hand-crafted features.
- **Parallel Computation:** Efficiently implemented on modern GPUs, allowing scalable training.

Disadvantages:

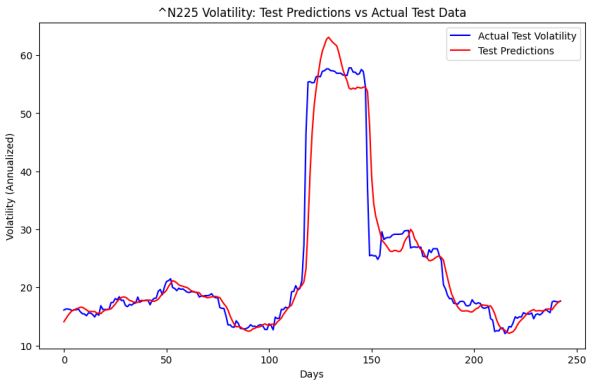
- **Prone to Overfitting:** Without regularization techniques such as dropout or weight decay, MLPs can memorize training data instead of generalizing.
- **Difficulty in Handling Sequential Data:** Unlike RNNs, MLPs lack memory mechanisms, making them not optimal for tasks involving temporal dependencies.
- **Hyperparameter Sensitivity:** Performance depends on parameters such as learning rate, number of layers, and activation functions, which require extensive tuning.

3.1.3 | Model Training and Evaluation

The model was trained for 50 epochs with batch size of 32. Two hidden layers were used, the first with 64 neurons and the second with 32. To evaluate the model, the metrics used were the Mean Squared Error (MSE), the Mean Absolute Error (MAE), the R-squared (R^2), and Mean Squared Log Error (MSLE). These metrics provide a comprehensive assessment of the model's predictive accuracy and its ability to capture the underlying volatility dynamics. The results are visualized using plots that show the actual vs. predicted volatility over time, providing insights into the model's performance across different tickers.



((a)) MLP Best Case; MAE=0.701245



((b)) MLP Worst Case; MAE=1.795485

Figure 3.1: Best and Worst case for MLP

3.2 | Long Short-Term Memory (LSTM) Networks

Financial markets are highly dynamic, with volatility playing a crucial role in investment decisions, risk management, and portfolio optimization. Traditional autoregressive models, such as GARCH-based approaches, have been widely used to model and forecast volatility. While effective, these models operate under strict assumptions, such as linearity and stationarity, which may limit their ability to capture complex market behaviors.

Deep learning models, particularly Long Short-Term Memory (LSTM) networks, provide a promising alternative by leveraging temporal dependencies and learning from sequential data patterns without requiring strong parametric assumptions. LSTMs are specifically designed to retain long-term dependencies, making them well-suited for capturing volatility clusters and structural breaks observed in financial time series. Given the success of deep learning in various forecasting tasks, our goal was to assess the performance of LSTMs in forecasting implied volatility and compare them against traditional autoregressive models.

3.2.1 | Introduction to LSTM for Volatility Prediction

Recurrent Neural Networks (RNNs) are well-suited for time series forecasting due to their ability to capture sequential dependencies. However, traditional RNNs suffer from the vanishing gradient problem, limiting their ability to learn long-term dependencies. To address this limitation, Long Short-Term Memory (LSTM) networks introduce memory cells and gating mechanisms that regulate information flow, making them highly effective for financial time series forecasting.

In this project, we implemented an LSTM model to predict the implied volatility of various financial instruments, including equity indices, commodities, and currencies. Our LSTM model was trained on historical volatility computed from adjusted closing prices. As discussed earlier in the paper, our data preprocessing approach included retrieving adjusted closing prices, computing log returns, calculating historical volatility, normalizing the data, and performing a train-test split. This standardized preprocessing ensured consistency across all tickers and prepared the data effectively for model training.

3.2.2 | LSTM Model Architecture

The LSTM model was implemented using Keras and structured as follows:

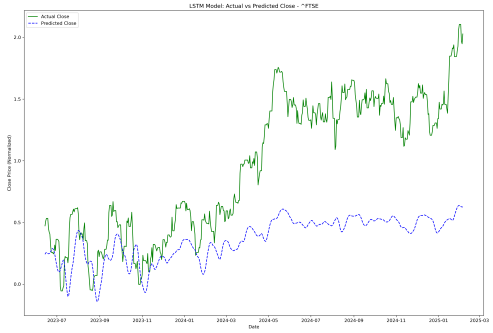
- **Input Layer**: Takes a sequence of past volatility values.
- **LSTM Layers**: Capture temporal dependencies in the data.

- **Dropout Layers**: Reduce overfitting by randomly dropping neurons during training.
- **Dense Layer**: Produces the final predicted volatility value.

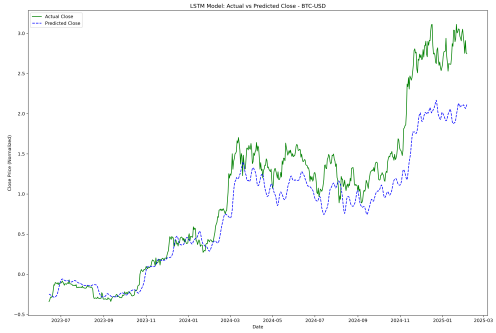
The model was compiled using the **Adam optimizer** with **mean squared error (MSE) loss function**, and trained using early stopping to prevent overfitting.

3.2.3 | Model Training and Evaluation

The model was trained for 10 epochs with a batch size of 64. Predictions were compared against actual values using common evaluation metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R^2). These metrics provided insight into the model's performance in capturing volatility patterns and predicting future values.
Figure 3.2 illustrates the actual and predicted close prices for the FTSE index and Bitcoin (BTC-USD) using the LSTM model. The model captures the general trend in volatility, particularly during stable market conditions, but some deviations are noticeable during periods of high market fluctuation. The FTSE index shows a relatively smooth prediction alignment, whereas BTC-USD exhibits a higher degree of volatility, making it more challenging for the model to forecast extreme price movements. These visualizations provide insight into the model's predictive capabilities across different asset classes.



((a)) LSTM Model: Actual vs. Predicted Close for FTSE Index



((b)) LSTM Model: Actual vs. Predicted Close for BTC-USD

Figure 3.2: LSTM Model Performance on Selected Tickers

3.2.4 | Improved LSTM Model

An improved LSTM model was trained with **50 epochs** and **dropout regularization (0.5)** to enhance generalization. Early stopping was applied to halt training when validation loss stopped improving. The improved model exhibited better predictive performance, with reduced MSE and higher R^2 scores compared to the basic LSTM model.

3.2.5 | Conclusion

LSTMs offer a powerful approach for forecasting implied volatility, leveraging sequential dependencies in historical volatility data. Compared to traditional autoregressive models, LSTMs provide the flexibility to learn nonlinear patterns, making them a promising tool for financial time series forecasting. The incorporation of advanced techniques, such as dropout regularization and early stopping, further enhances model robustness.

3.3 | σ-Cell RLTV

The σ-Cell RNN model [7] introduces a novel hybrid approach to volatility forecasting by integrating Recurrent Neural Networks (RNNs) with the principles of Generalized Autoregressive Conditional Heteroskedasticity (GARCH) models. This model leverages the dynamic capabilities of RNNs to capture complex patterns in financial time series data while incorporating the time-varying volatility estimation of GARCH models. It addresses the limitations of traditional econometric models, such as their inability to capture nonlinear dependencies and their reliance on fixed parameters.

The core of the model is the σ-Cell RNN, implemented as a custom PyTorch module. The model consists of a time-varying parameter network, a residual RNN layer, and an output layer. The RNN layer is implemented using a Gated Recurrent Unit (GRU), which is well-suited for capturing sequential dependencies in time series data. The GRU is defined by the following equations:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{3.4}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{3.5}$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \tag{3.6}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{3.7}$$

Where

1. z_t is the update gate, with x_t being the input at time t, h_{t-1} the hidden state from the previous time step, and σ the sigmoid function.
2. r_t is the reset gate
3. $\tilde{h}_t$ is the candidate hidden state
4. h_t is the final hidden state at time t.

The GRU's ability to selectively retain or discard information from previous time steps makes it particularly effective for modeling financial time series.
The forward pass of the σ-Cell RNN is defined by the following equations:

$$w_t = \tilde{\phi}(W x_{t-1} + b) \tag{3.8}$$

$$W_{s,t} = \pi_1(w_t), \quad W_{r,t} = \pi_2(w_t) \tag{3.9}$$

$$h_t = \text{GRU}(x_{t-1}, h_{t-1}) \tag{3.10}$$

$$G(h_t) = \phi(h_t W_{ho} + b_o) \tag{3.11}$$

$$\tilde{x}_{t-1} = x_{t-1} - G(h_t) \tag{3.12}$$

$$\tilde{\sigma}_t^2 = \phi(\tilde{\sigma}_{t-1}^2 W_{s,t} + \tilde{x}_{t-1}^2 W_{r,t} + b_h) \tag{3.13}$$

$$\sigma_t^2 = \phi_o(\tilde{\sigma}_t^2 W_o + b_o) \tag{3.14}$$

Here

1. w_t represents the time-varying weights
2. $W_{s,t}$ and $W_{r,t}$ are the weights for the historical variance and residual error, respectively
3. h_t is the hidden state computed by the GRU
4. σ_t^2 is the estimated conditional volatility at time t
5. ϕ and ϕ_o are activation functions, typically ReLU.

3.3.1 | Training and Loss Function

The model is trained using the Adam optimizer, with a learning rate scheduler to adjust the learning rate during training. As a loss function, we used the Mean Squared Error (MSE), which measures the squared difference between the predicted and actual volatility values, and is defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.15}$$

where y_i is the actual volatility and $\hat{y}_i$ is the predicted volatility. The MSE loss is computed using PyTorch's `nn.MSELoss()` function, where the training loop iterates over the training data, computing the loss for each time step and updating the model's parameters using backpropagation. The validation loop evaluates the model's performance on the test data using the same MSE loss function.

3.3.2 | Advantages vs. Disadvantages for σ-Cell

Advantages:

- **Dynamic Volatility Modeling:** it incorporates time-varying parameters that adapt to changing market conditions, making it more flexible than traditional GARCH models, which use fixed parameters.
- **Integration of RNNs:** by using an RNN (GRU), the model can capture long-term dependencies in time series data
- **Residual Learning:** a residual learning mechanism is used to compute the difference between the input and the predicted value, which helps improve the accuracy of volatility predictions.
- **Scalability:** the model can be extended to handle multivariate time series.

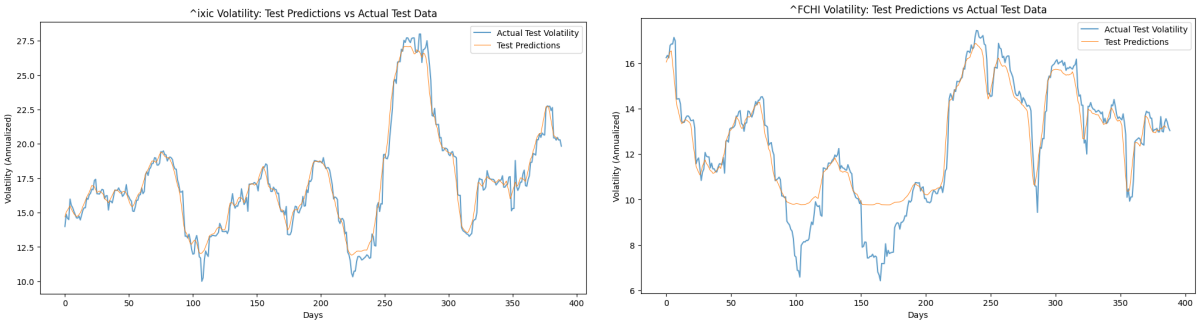
Disadvantages

- **Computational Complexity:** training the model requires significant computational resources, especially for large datasets.
- **Risk of Overfitting:** the model's flexibility and large number of parameters increase the risk of overfitting.
- **Sensitivity to Hyperparameters:** the performance depends heavily on the choice of hyperparameters (e.g., learning rate, hidden size, warm-up steps), which may require extensive tuning.
- **Training Instability:** the model can suffer from exploding gradients during training, especially if the data is not properly scaled or if the learning rate is too high.
- **Dependence on Initialization:** the model's performance can be sensitive to the initialization of weights and hidden states, which may lead to inconsistent results across different runs.

3.3.3 | Model Evaluation

The performance of the σ-Cell RNN model is evaluated using a test dataset, with predictions made after a warm-up period [1] to allow the model's internal states to stabilize. The predicted volatility is compared to the actual volatility using metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R^2), and Mean Squared Log Error (MSLE). These metrics provide a comprehensive assessment of the model's predictive accuracy and its ability to capture the underlying volatility dynamics. The results are visualized using plots that show the actual vs. predicted volatility over time, providing insights into the model's performance across different tickers.

[1] training warmup steps refer to an initial phase of training where the learning rate is gradually increased from a small value to the target learning rate (in our case 0.0001) over a specified number of steps or epochs



((a)) σ- Cell Best Case

((b)) σ- Cell Worst Case

Figure 3.3: Best and Worst case for σ- Cell

3.3.4 | Final Considerations

This novel approach of combining GARCH-like structures with RNN is solid but as outlined above, it comes with many limitations. Indeed, throughout the training across all tickers, the hidden state size had to be continuously adjusted to prevent overfitting, noting that a good compromise for most was a value of 32, but for some tickers like BTC-USD, it had to be increased to capture more complex patterns in a highly volatile environment. In addition to this, the random initialization states made it hard to obtain consistent results even across the same tickers. The model's predictive power is quite promising, but finding a way to stabilize results and states is crucial for determining its suitability across markets.

4 | Evaluation

In this section, we discuss the evaluation methods used for the models. The main metrics we looked at were:

- The mean absolute error (MAE)

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- The mean squared error (MSE)

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

- The mean log squared error (MSLE)

$$MSLE = \frac{1}{n}\sum_{i=1}^{n}(\log(1+y_i) - \log(1+\hat{y}_i))^2$$

We contemplated looking at other common model evaluation criteria (such as Akaike information criterion, Bayesian information criterion etc.), but as we mentioned in the introduction, we are considering exclusively the predicting power of these methods, and thus we are not looking to penalize said methods for extra parameter usage, something which these later criteria do.

4.1 | Autoregressive Models

Here are the results of the metrics for each model, averaged over each ticker:

Metric	Mean
MSE	41.21
MAE	3.501
MSLE	0.0548

Table 4.1: GARCH Model Metrics (4sf)

Metric	Value
MSE	39.37
MAE	3.545
MSLE	0.05539

Table 4.2: EGARCH Model Metrics (4sf)

Metric	Value
MSE	30.99
MAE	3.09
MSLE	0.04046

Table 4.3: FIGARCH Model Metrics (4sf)

Metric	Value
MSE	43.15
MAE	3.536
MSLE	0.05429

Table 4.4: APARCH Model Metrics (4sf)

Metric	Value
MSE	42.41
MAE	3.594
MSLE	0.05475

Table 4.5: GJR-GARCH Model Metrics (4sf)

As we can see, the model which performed the best was FIGARCH, with a mean MSE across all tickers of 31 vs the others of 39-43. Thus for the ARCH based models, on a wide scale, we crown FIGARCH winner.

4.2 | Neural Network Models

Metric	Mean
MSE	3.191
MAE	0.892
MSLE	0.005737

Table 4.6: MLP Model Metrics (4sf)

Metric	Value
MSE	0.2119
MAE	0.3357
MSLE	0.04252

Table 4.7: LSTM Model Metrics (4sf)

Metric	Value
MSE	0.4465
MAE	0.2351
MSLE	0.001823

Table 4.8: σ-Cell Model Metrics (4sf)

A comprehensive analysis leads to the conclusion that although the three models are suitable for predicting historical volatility, σ-Cell might have an edge over the other two, with a mean MSE slightly higher than that of LSTM but lower values on the other two metrics.

5 | Conclusions

To conclude, although these types of models are quite different and each has its own advantages and disadvantages, when considering pure prediction power based on past volatility data, LSTM and σ.Cell models seem to perform the best. However, this result does not preclude the existence of better models, and it might also vary depending on the parameters assigned to each of the models. Indeed, in some cases, the interpretability of model parameters is crucial, as understanding what each of the latter represents can be just as important as achieving high predictive accuracy. In other cases, maximizing performance may take precedence over interpretability, but this is necessarily our case since we are considering a metric that refers to market movements and is thus subject to various external factors, such as macroeconomic events, investor sentiment, and liquidity conditions.

Certain models, particularly those from the neural network family, may benefit from additional context, whereas traditional econometric models provide a clearer structure for understanding volatility dynamics. As a result, while LSTM and σ-Cell models demonstrate strong predictive performance on historical volatility data, their effectiveness may fluctuate when exposed to different market regimes or when additional explanatory variables are introduced. Therefore, the choice of model should be guided not only by its raw predictive accuracy but also by its adaptability to changing market conditions and its ability to integrate meaningful financial insights.

6 | References

[1] Richard T. Baillie, Tim Bollerslev, and Hans-Ole Mikkelsen. Fractionally integrated garch (figarch) model. 1996.

[2] Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity (garch). 1986.

[3] Zhongjun Ding, Clive W.J. Granger, and Robert F. Engle. Asymmetric power arch (aparch) model. 1993.

[4] Robert F. Engle. Autoregressive conditional heteroskedasticity with estimates of the variance of u.k. inflation. 1982.

[5] Lawrence R. Glosten, Ravi Jagannathan, and David E. Runkle. Threshold garch (gjr-garch) model. 1993.

[6] Daniel B. Nelson. Exponential garch (egarch) model. 1991.

[7] German Rodikov and Nino Antulov-Fantulin. Introducing the σ-cell: Unifying garch, stochastic fluctuations and evolving mechanisms in rnn-based volatility forecasting. 2023.

A | Appendix A: Model Metrics Across Tickers

model\ticker	GARCH(1,1)	EGARCH	FIGARCH	APARCH	GJR-GARCH	LSTM	MLP	Sigma
^GSPC	9.6498	9.4802	10.4224	9.2934	13.0663	0.118046	0.819399	0.0777
^IXIC	11.5175	12.0513	12.6611	11.5184	14.731	0.126354	0.905938	0.0176
BTC-USD	178.4363	163.2387	121.3544	167.4715	204.6019	0.09517	7.164814	0.1324
GC=F	3.0714	3.7565	4.1983	3.128	4.5747	0.439295	0.469238	0.1324
EURUSD=X	0.4072	0.413	0.6548	0.4111	0.4139	0.033144	0.087769	0.3801
^FTSE	6.6686	6.1865	2.668	6.678	6.5398	0.507347	0.419349	0.0138
^FCHI	12.3703	12.9384	9.2806	12.5109	14.9027	0.071262	0.438618	0.557
^GDAXI	10.1314	11.7206	5.7999	10.3493	13.1078	0.220456	0.480936	0.1116
FTSEMIB.MI	13.9387	12.9784	9.7848	13.8879	18.2419	0.208905	0.918205	0.0617
^AXJO	5.1511	5.809	5.2927	5.357	4.9726	0.183472	0.988807	0.0106
^HSI	23.2424	28.5075	30.1807	25.3799	23.0211	0.02886	2.713647	0.3645
^N225	138.5219	136.1688	101.3562	153.732	125.6113	0.673762	14.069915	3.8235
^NSEI	18.5798	19.7841	23.8213	19.5934	22.1138	0.216591	4.456345	0.1211
1308.T	145.2835	128.1889	96.3786	164.8405	127.8419	~~0.044626~~	10.738915	
Mean(4sf)	41.21	39.37	30.99	43.15	42.41	0.2119	3.191	0.4465
Mean(unrounde	41.21213571	39.37299286	30.98955714	43.15366429	42.41005	0.2119492857	3.190849643	0.4464615385

((a)) MSE

model\ticker	GARCH(1,1)	EGARCH	FIGARCH	APARCH	GJR-GARCH	LSTM	MLP	Sigma
^GSPC	2.319	2.3564	2.3182	2.3037	2.5295	0.25579	0.701245	0.1812
^IXIC	2.7198	2.8541	2.8042	2.7753	2.8719	0.256219	0.732764	0.1035
BTC-USD	11.4725	11.3129	8.7805	11.2435	11.7827	0.20036	2.024433	0.2051
GC=F	1.4826	1.6029	1.7476	1.4793	1.7824	0.504438	0.487389	0.2051
EURUSD=X	0.5168	0.5327	0.648	0.518	0.5151	0.145658	0.1777	0.3968
^FTSE	2.3311	2.2185	1.2689	2.3306	2.1094	0.59344	0.496528	0.0871
^FCHI	2.881	2.9906	2.3537	2.8981	2.9693	0.22042	0.473084	0.3078
^GDAXI	2.6735	2.9439	1.9549	2.7086	2.8474	0.363652	0.52082	0.171
FTSEMIB.MI	3.1543	3.0648	2.5179	3.1444	3.3887	0.394166	0.784365	0.1523
^AXJO	1.6864	1.8792	1.5223	1.6021	1.5841	0.345525	0.650007	0.0725
^HSI	2.7018	3.2308	3.4478	2.6895	2.8272	0.117805	0.947442	0.1984
^N225	6.1579	6.0083	5.5125	6.4611	6.1143	0.731009	1.795485	0.7803
^NSEI	2.8658	2.9768	3.1887	2.9062	3.2219	0.401835	1.228539	0.1948
1308.T	6.045	5.6522	5.1918	6.4487	5.7748	~~0.16975~~	1.468895	
Mean(4sf)	3.501	3.545	3.09	3.536	3.594	0.3357	0.892	0.2351
Mean	3.500535714	3.544578571	3.089785714	3.536364286	3.594192857	0.3357190714	0.8920497143	0.2350692308

((b)) MAE

model\ticker	GARCH(1,1)	EGARCH	FIGARCH	APARCH	GJR-GARCH	LSTM	MLP	Sigma
^GSPC	0.0414	0.0412	0.0432	0.0405	0.0505	0.041713	0.004532	0.0008
^IXIC	0.0283	0.03	0.0313	0.0288	0.0327	0.042434	0.002654	0.0001
BTC-USD	0.1073	0.1004	0.0662	0.1027	0.1125	0.009854	0.004393	0.0013
GC=F	0.0142	0.0182	0.018	0.0148	0.0203	0.058931	0.002041	0.0013
EURUSD=X	0.0085	0.0085	0.013	0.0085	0.0086	0.008564	0.001977	0.0091
^FTSE	0.0492	0.0458	0.0196	0.0492	0.047	0.121578	0.00363	0.0001
^FCHI	0.0642	0.0624	0.0455	0.0637	0.0689	0.01485	0.002703	0.0064
^GDAXI	0.0577	0.0593	0.033	0.058	0.0638	0.033529	0.002925	0.0013
FTSEMIB.MI	0.0613	0.0539	0.0404	0.0605	0.0671	0.036248	0.004304	0.0006
^AXJO	0.0264	0.0313	0.0249	0.0255	0.024	0.027937	0.006183	0.0001
^HSI	0.0199	0.027	0.0276	0.0187	0.0206	0.008521	0.003671	0.0002
^N225	0.1047	0.1116	0.0634	0.0993	0.0876	0.143113	0.011627	0.0014
^NSEI	0.0595	0.0705	0.0703	0.0611	0.0698	0.04033	0.020133	0.001
1308.T	0.1249	0.1154	0.07	0.1288	0.0931	~~0.007622~~	0.009539	
Mean(4sf)	0.05482	0.05539	0.04046	0.05429	0.05475	0.04252	0.005737	0.001823
Mean	0.05482142857	0.05539285714	0.04045714286	0.05429285714	0.05475	0.042516	0.005736571429	0.001823076923

((c)) MSLE

Figure A.1: Model Performance Metrics

B | Appendix B: Code

B.1 | Historical Volatility

Given the unavailability of volatility indexes for all tickers, we used the following code to compute the historical volatility:

Listing 1: Volatility Calculation

```
ADJ_COL = 'Adj Close'

def get_ticker_data(ticker, years_span, adj_col=ADJ_COL):
    end_date = datetime.datetime.today()
    start_date = end_date - datetime.timedelta(days=int(years_span * 365))
    try:
        data = yf.download(ticker,
                           start=start_date.strftime("%Y-%m-%d"),
                           end=end_date.strftime("%Y-%m-%d"),
                           interval='1d')
        if data.empty or adj_col not in data.columns:
            raise ValueError("No data returned or invalid column.")
        return data[adj_col]
    except Exception as e:
        raise ValueError(f"Error downloading data for {ticker}: {e}")

def compute_returns(price_series):
    returns = np.log(price_series).diff().dropna()
    returns = returns * 100 # rescaled
    return returns

def compute_historical_volatility(returns, window=30):
    """
    Computes the rolling historical volatility (annualized) using a specified window.
    (Calculated as the rolling standard deviation multiplied by  252 .)
    """
    hist_vol = returns.rolling(window=window).std() * np.sqrt(252)
    return hist_vol.dropna()
```

B.2 | Rolling forecast for GARCH models:

Listing 2: Rolling Forecast for GARCH Models

```
def compute_rolling_forecast(model_func, returns, initial_window=60):
    forecast_values = []
    forecast_dates = []
    for t in range(initial_window, len(returns)):
        sub_returns = returns.iloc[:t]
        try:
            forecast_series, _ = model_func(sub_returns, forecast_horizon=1)
            forecast_values.append(forecast_series.iloc[0])
            forecast_dates.append(returns.index[t])
        except Exception as e:
            forecast_values.append(np.nan)
            forecast_dates.append(returns.index[t])
    return pd.Series(data=forecast_values, index=forecast_dates)
```