

Two-stage Volatility-Quantile models using deep learning density networks with application to Cryptocurrencies

Nia P. Chen

nia.chen.res@gmail.com

The University of Sydney

Jennifer S.K. Chan

The University of Sydney

Linh H. Nghiem

The University of Sydney

Research Article

Keywords: Two stage volatility-return models, density networks, LSTM, quantile and expectile regression, Cryptocurrency

Posted Date: April 21st, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-9262246/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

Two-stage Volatility-Quantile models using deep learning density networks with application to Cryptocurrencies

Abstract: Recently, deep learning density networks have become the standard baselines for financial time-series forecasting. In this paper, we introduce density networks structured using the two-stage volatility to return models, in which the volatility forecast are treated as an input to the scale parameter of the second stage return model. The models employ a negative log-likelihood loss for both stages, which is constructed from the Weibull or a generalized beta distribution for stage one and normal (Gaussian) or Student- t for stage two. VaR is computed directly from the fitted returns distribution using mean regression. However, it is well known that effects on the mean can differ substantially to those on the extreme quantiles, the focus for risk management in investment portfolios. Hence, nonparametric quantile and expectile regressions are considered by extending the return loss to asymmetric Laplace and Gaussian proxy distributions. The time varying variance from first stage weights the observations inversely to their variances. Our empirical examination focuses on daily returns and Parkinson volatility measures of Bitcoin and Ripple. In comparing with the parametric VaR, the nonparametric quantiles and expectiles shows great adaptiveness and accuracy to observed data.

Keywords: Two stage volatility-return models, density networks, LSTM, quantile and expectile regression, Cryptocurrency

1 Introduction

In the intricate domain of financial analysis, the quantification of risk and return constitutes a central objective. Returns, typically expressed as percentages, represent the proportional gain or loss relative to the initial investment and serve as an indicator for evaluating the investment performance. Return maximization alone is insufficient for informed decision-making, prudent investment management also necessitates the minimization of uncertainty, conventionally measured by the variance of returns. Variance not only captures how unpredictable or inconsistent the investment's outcomes would be but also constitutes a critical input in risk-adjusted performance metrics such as the Sharpe ratio, which evaluates excess returns relative to the level of volatility assumed. Within this framework, a higher expected return accompanied by lower variance signifies a superior portfolio risk–return efficiency, reflecting an investor's capacity to achieve optimal performance under constrained risk exposure.

Numerous advancements were made in volatility modeling, which started with the Auto-Regressive Conditionally Heteroskedastic (ARCH) model by Engle (1982) and the Generalized ARCH (GARCH) model by Bollerslev (1986). Using daily closing prices, the GARCH models estimate volatility as latent processes behind the observed return data via a deterministic function of the lagged squared innovations and their lagged variances. Alizadeh et al. (2002), Chou (2005) and Brandt and Jones (2006) argued that GARCH-type models are less efficient as they neglect some useful information within a reference period. Moreover, being modeled as a latent deterministic process in a GARCH-type model using returns, the volatility dynamic is not adequately captured stochastically. Instead of treating volatility as a latent process, some researchers endeavored to model volatility directly. The Conditional Auto-Regressive Range (CARR) model (Chou, 2005) was introduced to capture the dynamics of volatility directly via some volatility measures instead of returns. Classical volatility measures such as the open-to-close (CO) employ squared logarithmic returns based on the daily opening and closing prices. However, the efficiency of the daily volatility measures can be improved utilizing more information, such as the daily highest and lowest prices. Parkinson (1980) proposed Parkinson (PK) variance measure, which is calculated from daily high-low range and is more efficient than the classic volatility measure defined as the square of daily

return.

Apart from modeling the volatility, we can also model the return of the assets which provides the estimate for optimizing the sharp ratio of a portfolio. Volatility has been an important component in forecasting financial return as it can provide the mean or quantile regression models for returns with a time varying scale parameter. This motivates the development of a two-stage volatility–return framework, in which the first stage estimates volatility using a CARR model, and the second stage incorporates the fitted volatilities into the return model to account for heteroskedasticity (Tan et al., 2022; Savage and Chan, 2025). Unlike the conventional GARCH framework, which relies solely on return information, the two-stage model utilises both return and volatility data, treating them as two interrelated stochastic processes.

For the stage two return models, the simple Autoregressive Moving-Average (ARMA) specification is widely used to capture short-term linear dependence in time series. Box et al. (2015) provided the general ARMA framework with assumption of independent and identically distributed Normal errors. Subsequent finance applications including Campbell et al. (1998) emphasized ARMA modeling as a basic tool for return predictability tests. To capture the excess kurtosis observed in financial return distributions, the classical ARMA model with Gaussian innovations has been extended to capture Student- t innovations. Lastly, the two-stage CARR-return model of Tan et al. (2022) and Savage and Chan (2025) adopted an AR specification and normal and Student- t innovations for the stage two return model.

However, as markets evolve and financial instruments become more complex, the need for more advanced models to capture the dynamics of returns becomes evident. Distribution quantiles offer an important perspective in the assessment of financial risk by concentrating on the distribution’s tail behavior. Although extreme tail events are rare, they are often responsible for the most severe financial losses. Regression applied directly to the percentage or probability of loss particularly for extreme cases facilitates a more precise evaluation of the risk and yields insights beyond those captured by the conventional mean based frameworks. Emerging from this, it necessitates the concept of Value at Risk (VaR) which is defined as a quantile of the return distribution. This framework refines the understanding of tail risks by focusing not just on the mean behavior but on the entire distribution of returns. Subsequently, it reveals any level of tail risk giving the worst loss expected for a portfolio over time for a chosen confidence level. Jorion (1997) discussed the parametric VaR method which assumes returns are normally distributed and computes VaR via mean and standard deviation based on the distribution assumption. However, the problem of two stage mean regression is it needs a correct distribution assumptions to ensure the tail risk forecast is unbiased. More importantly, these two stage means based on mean regression may fail to capture the dynamics at extreme quantiles efficiently.

Koenker and Bassett Jr (1978) proposed quantile regression using an asymmetric L_1 loss function and it has gained prominence in recent years. This framework is nonparametric in nature because it models different quantiles of the return distribution independently. If the weights are symmetric (equal between positive and negative errors), it is called median regression and it converges to mean regression if the distribution is symmetric. Within this framework, Engle and Manganelli (2004) proposed the conditional autoregressive VaR (CAViaR) model, which models the VaRs or quantiles of the return distribution directly and captures the quantile persistence using lagged volatility and expected quantile. To minimize the asymmetric L_1 loss function, they assume a proxy Asymmetric Laplace (AL) distribution with constant variance. However, the constant variance assumption for highly volatile assets is often inadequate to capture the heavy tails and volatility clustering observed in returns.

There are different modeling strategies to capture a time-varying variance. Gerlach et al. (2011) reviews the CAViaR model within a Bayesian framework, employing the AL as a proxy distribution for the return quantile process and allowing the scale parameter to evolve over time through GARCH-type dynamics. Taylor (2019) used AL as the conditional distribution for VaR where the scale parameter of AL is taken to be the time-varying conditional expectation of the tick loss function of weighted errors. In a related development, Iacopini et al. (2022) proposed a Bayesian multivariate quantile regression approach that applied multivariate AL distribution approximated by multivariate normal mixture, with the scale param-

eters linked directly to time-varying stochastic volatility models. Our framework implements a two-stage approach to quantile modeling with a time-varying scale parameter. In the first stage, we model the variance to get the estimated time-varying volatility which will be used as the scale parameter later in a separate quantile model. While [Chen et al. \(2025\)](#) also proposed a two-stage volatility to quantile model, their approach differs as they use Bayesian methodology and include the estimated volatility from stage one as a covariate in the CAViaR function in stage two, whereas we assign the volatility estimates directly to the scale parameter of the AL distribution. More importantly, our two-stage model models two data sources using two different models.

Development of traditional statistical models to capture some complex hidden patterns is challenging due to the substantial increase in the number of parameters. Artificial neural network (ANN) or simply neural network (NN) emerges as an optimal method for financial time series due to its flexibility in modeling the complex dynamics and use of stochastic gradient descent in training parameters. When combining with statistical models, NN offers remarkable capacity for linear and non-linear mapping, surpassing the constraints of rigid model structures in traditional statistical models ([Zhang, 2003](#)). A feedforward NN (FNN; also called Multi-Layer Perceptron) is one of the earliest NN architectures, consisting of an input layer, one or more hidden layers between the input and output layers of artificial neurons and nonlinear activation functions to capture non-linear covariate effects. [Dhamija and Bhalla \(2010\)](#) compared the GARCH models with the FNN models and found out that in out-of-sample forecasting, FNNs can beat the statistical model. [Shahwan and Said \(2012\)](#) compared the Bayesian FNN models with ARIMA models and concluded that FNN models outperform ARIMA models for high-dimensional chaotic data. NNs excel generalized linear models (GLMs) in learning empirically the intricate logic between input features and output targets throughout the training process.

Basic FNNs are further advanced to increase their flexibility and adaptiveness. Popular extensions include deep learning NN (DLNN) with multiple hidden layers to reshape the data through hierarchically interconnected layers of neurons ([LeCun et al., 2015](#)). Through this high-level abstraction of data, they automatically learn to extract salient features of the input data. With the rapid progression in NNs, Recurrent NNs (RNNs) emerged as a prominent tool for sequence predictions due to their inherent capacity to remember previous inputs in their hidden layers. However, RNNs faced challenges like vanishing gradient problems, prompting the development of Long Short-Term Memory (LSTM) networks ([Hochreiter and Schmidhuber, 1997](#)). LSTMs, with their specialized cell structures, offer enhanced memory and learning capabilities, making them apt for tasks involving sequences and time series data.

NNs can be viewed as some complicated and dynamic regression models as they bear some similarities to regression models in which regression parameters are estimated to optimize certain objectives or loss functions. From a statistical modeling perspective, the common mean square error (MSE) loss function is applied to non-parametric regression, essentially matching the first-order moment in the L_2 norm ([Hastie et al., 2009](#)) which corresponds to normal distribution for symmetric data. This assumption fails to capture, for example, the right skewed volatility measures, even though non-linear activation functions such as ReLU can be adopted to ensure positivity. More importantly, the MSE loss function fails to provide higher order moment parameters such as uncertainties of the target variable. Uncertainty quantification has received growing interest in NNs ([Amini et al., 2020](#); [Wong et al., 2025](#)). To tackle this shortcoming, density networks were proposed, adopting negative log likelihood (NLL) as the loss function to incorporate different distribution assumptions. In this study, we investigate conditional density modeling and forecasting within the NN framework, addressing their inherent complexities.

Financial time series modeling using NN is not an novel idea now. Back to 1991, [Specht et al. \(1991\)](#) proposed the General Regression Neural Network (GRNN) which is close to a kernel regression but implemented in a NN structure. [Kanas \(2001\)](#) further used the GRNN to model the returns for stock market. NNs are applied to capture different statistical modeling ideas. [Donaldson and Kamstra \(1997\)](#) proposed the ANN-GARCH model, by adding a FNN into the GARCH function to allow for nonlinear dynamics in the conditional volatility. [Schittenkopf et al. \(2000\)](#) proposed the recurrent normal mixture density

network where they used FNNs to model the first four order moments to capture heteroscedasticity and showed to perform better than GARCH model. Advancing in network architecture, [Ramos-Pérez et al. \(2021\)](#) proposed hybrid models that use GARCH outputs as feature inputs to multi-transformer architectures in volatility forecast, while highlighting the increasing shift toward hybrid approaches that combine the classical volatility models with the adaptability of deep learning. For quantile regression, [Xu et al. \(2016\)](#) applied a single layer FNN to model the quantile autoregression structure, capturing nonlinearity of return quantiles more effectively. To ensure smoothness of the asymmetric L_1 loss function, they replaced it with the Huber norm. Similarly, [Jiang et al. \(2017\)](#) employed an asymmetric least squares loss within the single layer NN framework to model conditional expectile and added a penalty term to control the trade off with the asymmetric error term. [Chen et al. \(2025\)](#) employed the FNN-CAViaR hybrid model for quantile regression where the CAViaR component to capture persistence and NN component to capture other feature effects are combined in the mean function for a better model interpretation. This paper adopts the density networks, enabling the modeling of volatility in stage one and returns, and quantiles in stage two within a unified framework.

The aim of this paper is threefold. First, we propose density networks which integrates a two-stage volatility-return models with LSTMs. The inaugural stage, which resonates with the statistical CARR model, is dedicated to modeling volatility. The loss function we consider in this stage will be the Weibull and GB2 NLLs. Successively, the output from this initial stage serves as an input to the subsequent stage. For this second stage, we model the return with the NLL loss function based on normal and Student- t distributions. Second stage return forecasts are used to derive quantile predictions under specific distributional assumptions, allowing for parametric estimation of conditional quantiles. We employ a fixed volatility scaling parameter to calibrate the magnitude of model-implied volatilities prior to second stage modeling. This is a novel extension in the literature integrating the two stage volatility-return model to the NN framework, enhancing the modeling capacity tremendously.

Our second contribution generalizes the second-stage to quantile regression model via an AL distribution with time-varying scale parameters. This generalization, which encapsulates the modeling of return and quantiles, makes quantile estimates more accurate to reflect quantile specific dynamics and volatility clustering. To assess performance, we conduct a comprehensive comparison across competing models to estimate quantiles: *single stage quantile regression*; *two stage mean regression*; and *two stage quantile regression*, and identify the most effective model specifications. We note that one stage models fix the scale parameter whereas two-stage models allow the scale parameter evolves over time, informed by the first-stage volatility estimates. Finally, their quantile forecasts are visualized to illustrate the practical differences in capturing risk dynamics.

Our third contribution is to demonstrate the applicability of the proposed network models to study the cryptocurrency market. Cryptocurrencies are trading 24/7 and were established in late 2000s. They become popular investment tools recently among investors and financial institutes and hence, attract much academic attention in the modeling of financial risk. They are known to be quite different from the traditional stock market due to their heightened volatility caused by factors such as lower liquidity, speculative trading, sensitivity to news events and so on ([Phillip et al., 2018](#); [Tan et al., 2020](#)). Hence, the more complex deep learning model is also suitable in capturing the complex hidden patterns of cryptocurrencies. Modeling cryptocurrencies using deep learning NN is not a new topic where [Ji et al. \(2019\)](#) suggested different features and architectures such as deep learning NN, LSTM, and convolution NN for Bitcoins modeling. [Nasirtafreshi \(2022\)](#) studied the Bitcoin price forecasting using LSTM demonstrating the effectiveness of recurrent deep learning models in capturing the persistence and nonlinear behavior in cryptocurrencies. However, these models considered only a single stage model and so, our proposed two-stage CARR-return models utilizing volatility and return information in each stage is a break-through.

The paper is structured as follows. In section 2, we introduce some basic concept behind the NN model including LSTM, and explain in detail the variance, return and quantile regression model. Section 3 discusses the return and variance properties of Bitcoin and Ripple used in the paper along with features

used for Bitcoin and Ripple modeling. The forecast result and model comparison are presented in section 4. Lastly in section 5, we conclude the study and suggest the future studies.

2 Proposed Model

In this section, we describe our two-stage model structure in modeling variance and mean/quantile of returns. To that end, we first introduce the return and variance of return measures as model input. For a price series of financial asset, let C_t, H_t, L_t be the closing, high, and low price at day t , respectively, for $t = 1, \dots, N$. The return Y_t and PK variance V_t measures at day t is then calculated as

$$Y_t := \log \frac{C_t}{C_{t-1}} = \log C_t - \log C_{t-1}, \quad (1)$$

and

$$V_t = \frac{(\log H_t - \log L_t)^2}{4 \log 2}, \quad (2)$$

respectively. We propose a two-stage framework to model these two measures as depicted in Figure 1. In the following subsections, we describe each stage of the model in detail.

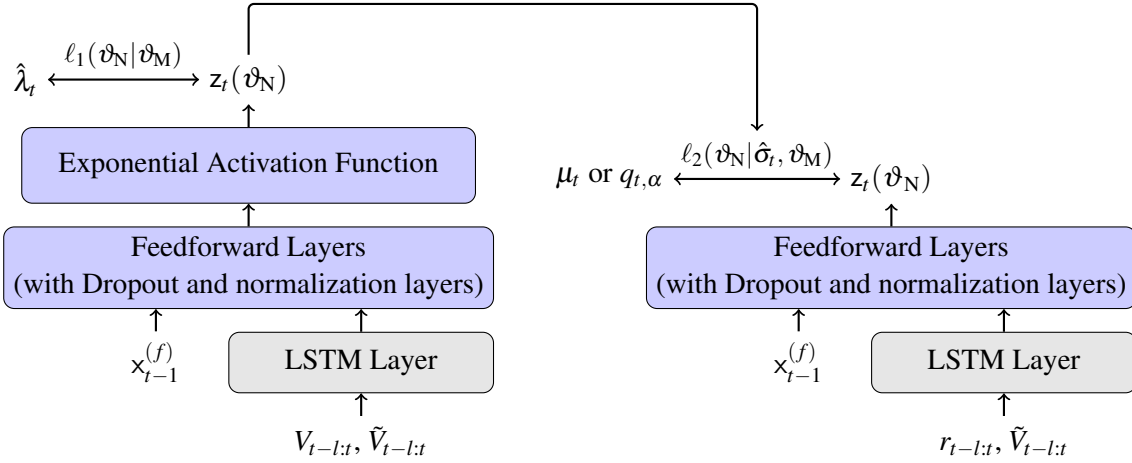


Figure 1: Two stage Neural Network model flowchart. For single stage direct quantile regression model, only the right panel is applied and the top description is replaced by $q_{t,\alpha} \xleftrightarrow{\ell_{2,AL}(\vartheta_N | \vartheta_M)} z_t(\vartheta_N)$ where $\vartheta_M = \{\sigma\}$.

2.1 First Stage - Variance Modeling

To model volatility, Chou (2005) proposed the CARR model and claimed that range-based volatility estimators are more efficient than return-based GARCH model estimators. For the observed PK variance V_t at time t , the general CARR(h, k) model is specified as

$$\begin{aligned} V_t &= \lambda_t \varepsilon_t, \quad t = 1, 2, \dots, T, \\ \lambda_t &= \delta_0 + \sum_{i=1}^h \delta_{1i} V_{t-i} + \sum_{j=1}^k \delta_{2j} \lambda_{t-j}, \end{aligned} \quad (3)$$

where $\lambda_t = E(V_t)$ is the conditional expected measures at time t . This expectation is modeled recursively based on h lagged observed variances V_{t-i} and k lagged mean λ_{t-j} , with the parameter $\delta_0, \delta_{1i}, \delta_{2j} > 0$ for $i = 1, \dots, h$ and $j = 1, \dots, k$. Taking the density network approach, $\varepsilon_t \in \mathbb{R}^+$ is assumed to follow some distributions defined on the positive real line with mean one independent of λ_t . This model structure using variance measures, with long and short term persistence, is more efficient than GARCH model (Chou, 2005). Following the idea, we adopt the variance measures as target variables. Moreover, LSTM

within density networks can incorporate different persistence measures (such as lagged PK in Eq. (2) and GARCH output; see Figure 1) and provides further flexibility by allowing nonlinearity and capturing long memory, regime changes and nonlinear interactions for long and noisy series. Regarding distribution for ε_t , [Chan et al. \(2019\)](#) provided a list and we adopt generalized Beta type II (GB2) ([McDonald and Xu, 1995](#)) and consider Weibull as its special case.

Specifically, the GB2 distribution is characterized by three shape parameters κ, ρ_1, ρ_2 and a scale parameter $\eta_t > 0$, and hence it allows more flexible shape of the distribution. If V_t follows a GB2($\kappa, \eta_t, \rho_1, \rho_2$) distribution, the probability density function (pdf) is given by

$$f_{\text{GB2}}(V_t | \kappa, \eta_t, \rho_1, \rho_2) = \frac{\frac{\kappa}{\eta_t} \left(\frac{V_t}{\eta_t}\right)^{\kappa\rho_1-1}}{B(\rho_1, \rho_2) \left[1 + \left(\frac{V_t}{\eta_t}\right)^\kappa\right]^{\rho_1+\rho_2}}, \quad (4)$$

where $B(\cdot, \cdot)$ denotes the beta function. Regression is applied to the mean of the distribution given by

$$E(V_t) = \hat{\lambda}_{t;\text{GB2}}(\vartheta_N) = \frac{\eta_t(\vartheta_N)B(\rho_1 + 1/\kappa, \rho_2 - 1/\kappa)}{B(\rho_1, \rho_2)} \Rightarrow \eta_t(\vartheta_N) = \frac{\hat{\lambda}_{t;\text{GB2}}(\vartheta_N)B(\rho_1, \rho_2)}{B(\rho_1 + 1/\kappa, \rho_2 - 1/\kappa)} \quad (5)$$

where ϑ_N is the regression/network parameters for the input features $\mathbf{x}_{i:t-l:t}$ described in section 3.2. The first stage NLL for all model parameters given an independent sample $V_1, \dots, V_N$ taken from this distribution is

$$\begin{aligned} \ell_{1;\text{GB2}}(\eta_t(\vartheta_N), \kappa, \rho_1, \rho_2 | V_{1:N}) &= -\log(\kappa) + \log(B(\rho_1, \rho_2)) \\ &\quad - \frac{1}{N} \sum_{t=1}^N \left[(\kappa\rho_1 - 1) \log(V_t) - \kappa\rho_1 \log(\eta_t(\vartheta_N)) - (\rho_1 + \rho_2) \log \left\{ 1 + \left(\frac{V_t}{\eta_t(\vartheta_N)} \right)^\kappa \right\} \right] \end{aligned} \quad (6)$$

where $\eta_t(\vartheta_N)$ is given in Eq (5). The Weibull distribution is a special case of GB2 distribution when $\rho_1 = 1$ and $\rho_2 \rightarrow \infty$. For the Weibull distribution with the scale parameter η_t and shape parameter $\kappa > 0$, the pdf is given by

$$f_{\text{Wei}}(V_t | \eta_t, \kappa) = \frac{\kappa}{\eta_t} \left(\frac{V_t}{\eta_t}\right)^{\kappa-1} \exp \left[- \left(\frac{V_t}{\eta_t}\right)^\kappa \right], \quad (7)$$

where $\Gamma(\cdot)$ is the gamma function and again, a regression is assigned to the expected value

$$E(V_t) = \hat{\lambda}_{t;\text{wei}}(\vartheta_N) = \eta_t \Gamma \left(1 + \frac{1}{\kappa} \right) \Rightarrow \eta_t(\vartheta_N) = \hat{\lambda}_{t;\text{wei}}(\vartheta_N) / \Gamma(1 + \kappa^{-1}). \quad (8)$$

The NLL ℓ_1 for (η_t, κ) from an independent sample $V_1, \dots, V_N$ taken from this distribution is

$$\ell_{1;\text{Wei}}(\eta_t(\vartheta_N), \kappa | V_{1:N}) = -\log \left(\frac{\kappa}{\eta_t(\vartheta_N)} \right) - \frac{1}{N} \sum_{t=1}^N \left[(\kappa - 1) \log \left(\frac{V_t}{\eta_t(\vartheta_N)} \right) - \left(\frac{V_t}{\eta_t(\vartheta_N)} \right)^\kappa \right], \quad (9)$$

where $\eta_t(\vartheta_N)$ is given in Eq. (8).

2.2 Second Stage - mean and quantile models for returns

In this section, we discuss the second stage return model which takes the predicted variance $\hat{\lambda}_t$ from first stage as a time-varying scale parameter of the normal or Student- t distribution when modeling return mean, and the AL distribution when estimating conditional quantiles. In this way, the volatility clustering can be captured more efficiently using estimates directly from variance measures.

2.2.1 Mean regression model for returns

Return models that utilize normal distribution are fundamental in financial econometrics for capturing the behavior of asset returns. The normal distribution is widely used due to its mathematical simplicity and the assumption of constant variance, providing a baseline model for risk and return analysis. However, financial returns often exhibit heavier tails and higher kurtosis than the normal distribution can capture. To address this, the Student- t distribution is employed, which allows for heavier tails and thus better accommodates extreme returns and volatility clustering. The key consideration when applying the Student- t distribution in return modeling is the estimation of the degrees of freedom parameter ν , which governs the tail thickness of the distribution. Section 2.4.2 provides details of estimating the ν parameter.

Another consideration is the scale adjustment of λ_t (Savage and Chan, 2025; Tan et al., 2022) by multiplying it to a scale adjustment factor ϕ when estimating $\text{Var}(Y_t)$. By setting ϕ as a parameter to be estimated, this allows flexibility for λ_t to capture only the volatility trend precisely. This is particularly applicable to the two-stage variance-quantile model when estimating the variance of a skew distribution is more challenging. Hence, the final variance specification is $\text{Var}(Y_t) = \phi \lambda_t$. For the variance of a t -distribution, it takes the form $\text{Var}(Y_t) = \nu \sigma_t^2 / (\nu - 2)$. Thereby, the scale parameter of the t -distribution is

$$\hat{\sigma}_t = \sqrt{\phi \hat{\lambda}_t (\nu - 2) / \nu} \quad (10)$$

and normal distribution corresponds to the t distribution with $\nu \rightarrow \infty$. Numerical experiment in Savage and Chan (2025) and Tan et al. (2022) showed that usually $\phi \simeq 1$ for both normal and t return distributions when applying mean regression, confirming the efficiency of stage one in estimating the variance of returns. Hence, we set $\phi = 1$ for mean regression.

The log-likelihood function of t -distribution for the sample $y_{1:N}$ is given by

$$\begin{aligned} \ell_{2,T}(\mu_t(\vartheta_N), \nu | \sigma_t, y_{1:N}) &= \frac{1}{N} \sum_{t=1}^N \frac{1}{2} \log(\pi) + \frac{1}{2} \log(\nu) + \log(\sigma_t) - \log \Gamma\left(\frac{1}{2} + \frac{1}{2}\nu\right) + \\ &\quad \log\left(\Gamma\left(\frac{\nu}{2}\right)\right) + \frac{\nu+1}{2} \log\left(1 + \frac{(y_t - \mu_t(\vartheta_N))^2}{\sigma_t^2 \nu}\right), \end{aligned} \quad (11)$$

where the scale parameter σ_t is estimated by $\hat{\sigma}_t$ using $\hat{\lambda}_t$ in Eq. (10), $\mu_t(\vartheta_N)$ is the mean return when regression is applied, ϑ_N is the regression/network parameters for features $x_{t-1}^{(f)}$, ν is the degrees of freedom of the t -distribution, and $\Gamma(\cdot)$ is the gamma function.

When the *stage two* return model adopts a symmetric normal or t distribution with CDF $F(\cdot)$, the $\alpha \in (0, 1)$ level condition quantile is given by:

$$q_{t;\alpha} = \mu_t(\vartheta_N) + \hat{\sigma}_t F^{-1}(\alpha), \quad (12)$$

where $F(\cdot)$ is the CDF of normal or t distribution. This approach demonstrate an parametric way of modeling quantiles from mean regression with symmetric distribution assumptions.

2.2.2 Quantile regression model for returns

On the other hand, quantile can be modeled directly based on asymmetric loss functions. Quantile $q_{t;\alpha}$ for the return distribution Y_t at time t is the solution to the first order optimization function

$$q_{t;\alpha} = \underset{x}{\operatorname{argmin}} E[(\alpha - \mathbf{1}(Y_t < x))(Y_t - x)]. \quad (13)$$

where $\mathbf{1}(\cdot)$ is an indicator function, equals to one when Y_t is less than $q_{t;\alpha}$ and zero otherwise. Engle and Manganelli (2004) proposed the CAViaR model to estimate the conditional VaR/quantile. The general

formula for CAViaR model, which is similar to CARR model, is given by

$$q_{t;\alpha} = \beta_0 + \sum_{i=1}^h \beta_{1,i} q_{t-i;\alpha} + \sum_{j=1}^k \beta_{2,j} V_{t-j} \quad (14)$$

where $\beta = \{\beta_0, \beta_{1,1:h}, \beta_{2,1:k}\}$ are the parameters to be estimated from the CAViaR model. Again, this efficient model illustrates the importance of persistence terms. The LSTM cell state, described in Section 2.3.1, functions as the autoregressive memory component, effectively capturing the long-term temporal dependencies analogous to the lagged terms in the CAViaR specification. To extend CAViaR model structure into LSTM, we incorporate look back windows of lagged returns and fitted GARCH estimates as input features (see Figure 1).

Since quantile is the solution to the L_1 loss in Eq. (13), the AL distribution can be used as a proxy distribution in the CAViaR modeling since minimizing L_1 loss is equivalent to maximizing the log-likelihood for a location parameter of an AL distribution. The pdf of AL distribution with mode $q_{t;\alpha}$, scale $\sigma^2 > 0$ and skewness $\alpha \in (0, 1)$ can be written as:

$$f_{\text{AL}}(y_t | q_{t;\alpha}, \sigma, \alpha) = \frac{\alpha(1-\alpha)}{\sigma} \exp \left\{ -\frac{y_t - q_{t;\alpha}}{\sigma} [\alpha - \mathbf{1}(y_t \leq q_{t;\alpha})] \right\}. \quad (15)$$

The mean μ_t and variance σ^2 of the distribution are given by

$$\mu_t = q_{t;\alpha} + \frac{1-2\alpha}{\alpha(1-\alpha)} \sigma, \quad \text{and} \quad \text{Var}(Y_t) = \frac{1-2\alpha+2\alpha^2}{\alpha^2(1-\alpha)^2} \sigma^2. \quad (16)$$

Although the scale parameter σ of the AL distribution used for quantile regression is often fixed (Chen et al., 2025), one may consider alternately a time-varying scale parameter σ_t that evolves with market conditions. Under this setting, there are two possible approaches for generating quantile forecasts: (1S) *single stage direct quantile regression* using a fixed σ ; or (2S) *two stage direct quantile regression* incorporate time-varying variance based on the first-stage estimate, $\hat{\sigma}_t$. Similar to Eq. (10), the scale parameter of the AL distribution is

$$\hat{\sigma}_t = \sqrt{\phi \hat{\lambda}_t (\alpha^2(1-\alpha)^2) / (1-2\alpha+2\alpha^2)} \quad (17)$$

where ϕ is a parameter to be estimated. This design lets the quantile model framework learn both the dynamics of variance and the conditional quantile structure of returns in a coherent way. The NLL for the sample $y_{1:N}$ given $\sigma_{1:N}$ from stage one (two stage) is

$$\ell_{2;\text{AL}}(q_{t;\alpha}(\vartheta_N), s | \hat{\sigma}_{1:N}, y_{1:N}) = -\log(\alpha(1-\alpha)) + \frac{1}{N} \sum_{t=1}^N \log \hat{\sigma}_t + \frac{(y_t - q_{t;\alpha}(\vartheta_N))}{\hat{\sigma}_t} [\alpha - \mathbf{1}(y_t \leq q_{t;\alpha}(\vartheta_N))]. \quad (18)$$

For the single stage case, the constant scale parameter σ needs to be estimated but there is no scaling factor ϕ . Lastly, we note that apart from the regression/network parameters ϑ_N , there are also non-regression model/distribution parameters referring to the NLL loss function in Eq. (9), (6), (11) and (18) for the case of Weibull, GB2, t and AL distributions respectively. They are summarized below:

$$\text{Wei: } \vartheta_M = \{\kappa\}; \text{ GB2: } \vartheta_M = \{\kappa, \rho_1, \rho_2\}; \text{ N: } \vartheta_M = \{\}; \text{ T: } \vartheta_M = \{\nu\}; \text{ AL 2S: } \vartheta_M = \{\phi\}; \text{ AL 1S: } \vartheta_M = \{\sigma\} \quad (19)$$

where N and T represent normal and t distributions. We also note that network and model parameters $(\vartheta_N, \vartheta_M)$ are estimated separately for each stage.

2.3 Neural Network Model Framework

In this section, we discuss our NN architecture consisting of FNN and LSTM as structured in Figure 1 which also explains how we input the additional features into our model.

The FNN model (Rosenblatt, 1958) is a fundamental type of NN where connections between nodes do not form cycles. Instead, data flows in one direction from input nodes through hidden layers to output nodes without any feedback loops. For sequential data where temporal dependencies are crucial, LSTM networks (Hochreiter and Schmidhuber, 1997) gives a more advanced solution. LSTMs are a type of RNN designed to handle long-range dependencies and mitigate issues related to vanishing and exploding gradients. In a LSTM, data flows from the input sequence through input, forget, and output gates into the cell state and hidden state, which then pass forward through time steps and layers to produce the final output.

From Figure 1, we model, in stage one, the PK variance measure V_t using input features $x_{i;t-l:t} = (V_{t-l:t}, \tilde{V}_{t-l:t}, x_{t-1}^{(f)})$ including the look back window (size l) of Parkinson variances $V_{t-l:t}$ and predicted GARCH outputs $\tilde{V}_{t-l:t}$ that are first passed into the LSTM layers. Then the output from LSTM together other feature vector $x_{t-1}^{(f)}$ are passed to the FNN layers to produce output neurons. For stage two, the architecture is similar. The look back window of daily log returns $y_{t-l:t}$ and GARCH outputs $\tilde{V}_{t-l:t}$ are first passed to the LSTM layers. Outputs join with features $x_{t-1}^{(f)}$ are again passed to the FNN layers to produce output neurons.

2.3.1 LSTM layers

Our NN architecture begins with LSTM. Let $\mathcal{H}_d > 0$ be the constant size of the hidden dimension of the LSTM framework and $\mathcal{H}_l > 0$ be the number of the LSTM hidden layers. At start, the input is $x_{1:t}^{(0)} = (V_{t-l:t}, \tilde{V}_{t-l:t}) \in \mathbb{R}^{2l}$ for stage one variance model, and $x_{2:t}^{(0)} = (y_{t-l:t}, \tilde{V}_{t-l:t}) \in \mathbb{R}^{2l}$ for stage two return (mean or quantile) model. In any hidden layer $h = 1, \dots, \mathcal{H}_l$, let $x_t^{(h-1)} \in \mathbb{R}^{\mathcal{H}_d}$ (dropping the state subscript to denote any state model in general) be the output neuron in layer $h-1$ which is also the input neuron in layer h . The concatenated input $[x_{s;t-l:t}^{(h-1)}, x_{j;t-l-1:t-1}^{(h)}] \in \mathbb{R}^{d^{(h-1)}+d^{(h)}}$ where $x_{t-l-1:t-1}^{(h)} \in \mathbb{R}^l$ is the hidden state vector from the previous time point, $d^{(0)} = 2l$, $d^{(h)} = \mathcal{H}_d$, $h = 1, \dots, \mathcal{H}_l$ passes through the input gate (i), forget gate (f), and output gate (o) of LSTM given by

$$\begin{aligned} \text{Input gate} &: i_t^{(h)}(W^{(i,h)}, b^{(i,h)}) = f_i \left(W^{(i,h)} [x_{t-l:t}^{(h-1)}, x_{t-l-1:t-1}^{(h)}] + b^{(i,h)} \right), \\ \text{Forget gate} &: f_t^{(h)}(W^{(f,h)}, b^{(f,h)}) = f_f \left(W^{(f,h)} [x_{t-l:t}^{(h-1)}, x_{t-l-1:t-1}^{(h)}] + b^{(f,h)} \right), \\ \text{Output gate} &: o_t^{(h)}(W^{(o,h)}, b^{(o,h)}) = f_o \left(W^{(o,h)} [x_{t-l:t}^{(h-1)}, x_{t-l-1:t-1}^{(h)}] + b^{(o,h)} \right). \end{aligned}$$

where f_i, f_f, f_o are the activation functions for the input, forget and output gate which are all set to the sigmoid function, $\text{sigmoid}(x) = (1 + e^{-x})^{-1}$. Next, the candidate memory cell ($\tilde{c}$) is updated to give the cell state and hidden state as below:

$$\begin{aligned} \text{Candidate update} &: \tilde{c}_t^{(h)}(W^{(c,h)}, b^{(c,h)}) = f_m \left(W^{(c,h)} [x_{t-l:t}^{(h-1)}, x_{t-l-1:t-1}^{(h)}] + b^{(c,h)} \right), \\ \text{Cell state} &: c_t^{(h)} = f_t^{(h)} \odot c_{t-1}^{(h)} + i_t^{(h)} \odot \tilde{c}_t^{(h)}, \\ \text{Hidden state} &: x_t^{(h)} = o_t^{(h)} \odot f_c(c_t^{(h)}). \end{aligned}$$

where $\odot$ denotes element wise multiplication, the h -th hidden layer state cells $i_t^{(h)}, f_t^{(h)}, o_t^{(h)}, \tilde{c}_t^{(h)}, c_t^{(h)} \in \mathbb{R}^{d^{(h)}}$, the weight matrices $W^{(i,h)}, W^{(f,h)}, W^{(o,h)}, W^{(c,h)} \in \mathbb{R}^{d^{(h)} \times (d^{(h-1)}+d^{(h)})}$, the bias vectors $b^{(i,h)}, b^{(f,h)}, b^{(o,h)}, b^{(c,h)} \in \mathbb{R}^{d^{(h)}}$, and f_m and f_c are the hyperbolic tangent function $\tanh(x) = (e^x - e^{-x}) / (e^x + e^{-x})$. At time $t = 0$, the initial value of the cell state $c_0 = 0$. The LSTM network parameter for hidden layer h are

$$W^{(h)} = \{W^{(i,h)}, W^{(f,h)}, W^{(o,h)}, W^{(c,h)}\} \in \mathbb{R}^{4d^{(h)} \times (d^{(h-1)}+d^{(h)})} \text{ and } b^{(h)} = \{b^{(i,h)}, b^{(f,h)}, b^{(o,h)}, b^{(c,h)}\} \in \mathbb{R}^{4d^{(h)}}. \quad (20)$$

2.3.2 Feedforward layers

The LSTM output $x_t^{(\mathcal{H}_d)}$ is then concatenated with additional features $x_t^{(f)} = (x_{1:t}, x_{2:t}, \dots, x_{n:t}) \in \mathbb{R}^n$ (see Appendix A for details) and passed to FNN layers as below:

$$\begin{aligned}\tilde{x}_t(\tilde{W}_1, \tilde{b}_1) &= f_{\text{iden}}(\tilde{W}_1[x_{(t-1)}^{(f)}, x_{t-l:t}^{(\mathcal{H}_d)}] + \tilde{b}_1), \\ z_t(\tilde{W}_2, \tilde{b}_2) &= f_{\text{out}}(\tilde{W}_2 \tilde{x}_t + \tilde{b}_2),\end{aligned}\tag{21}$$

where the weight matrices $\tilde{W}_1 \in \mathbb{R}^{\mathcal{H}_d \times (n + \mathcal{H}_d \times l)}$, $\tilde{W}_2 \in \mathbb{R}^{\mathcal{H}_d}$, and the bias vectors $\tilde{b}_1 \in \mathbb{R}^{\mathcal{H}_d}$ and $\tilde{b}_2 \in \mathbb{R}$.

The batch normalization layer and the dropout layer are assigned following the first hidden layer of FNN and preceding the output layer. In our case study, although Figure 5 shows that the cross-correlations between features and targets (Return and Parkinson variance) are relatively weak, we retain a linear output mapping as a benchmark, that is f_{iden} is an identity function. In settings with low linear signal, establishing a linear baseline is essential before introducing nonlinearities that may overfit noise. This also ensures parsimony and improves model interpretability. However, the output neuron for variance model is $z_t = \lambda_t > 0$, the exponential function $f_{\text{out}}(x) = \exp(x)$ is applied to positivity and it is linear for return model.

Lastly, the vector of all network parameters is

$$\vartheta_N = \{W^{(1:\mathcal{H}_1)}, b^{(1:\mathcal{H}_1)}, \tilde{W}_1, \tilde{W}_2, \tilde{b}_1, \tilde{b}_2\}\tag{22}$$

and the total number of network parameters is

$$\mathcal{H}_d \times (n + \mathcal{H}_d \times l + 1) + \mathcal{H}_d + 1 + \sum_{h=1}^{\mathcal{H}_d} 4d^{(h)}(d^{(h-1)} + d^{(h)} + 1)$$

where $d^{(0)} = 2l$. The output neuron z_t in Eq. (21) is mapped to different parameters for different stage and models. For the stage one variance model, $z_t = \lambda_t$, for the stage two mean return model, $z_t = \mu_t$; and for the stage two quantile return model, $z_t = q_{\alpha,t}$.

Besides the network parameters, we have another parameter vector which represents the set of NN tuning hyper-parameter which includes, apart from $\mathcal{H}_d, \mathcal{H}_l, l$, the Batch Size $\mathcal{B}$ which indicates how many samples are processed before updating weights; Dropout Rate $\mathcal{R}$ which is the fraction of neurons randomly deactivated during training; and Learning Rate $\mathcal{L}$ which controls how large each update step is. Therefore, the vector of hyper-parameter ϑ_T is:

$$\vartheta_T = (\mathcal{B}, \mathcal{R}, \mathcal{H}_d, \mathcal{H}_l, \mathcal{L}, l).\tag{23}$$

See section 2.4.1 for details.

2.4 Model selection process with parameter tuning and optimization

We remark that two parameter types, namely the model parameters ϑ_M in Eq. (19) and network parameters ϑ_N in Eq. (22) define the density network loss function ℓ whereas the architecture tuning ϑ_T parameters to define the NN model structure. Hence, the vector of all parameters $\theta = (\vartheta_M, \vartheta_N, \vartheta_T)$ for the two stage models. Different parameters are estimated/optimized using different methods at different steps, for example, the hyperparameters ϑ_T as well as model parameter ϑ_M for GB2 model are searched over while those for Weibull and AL models are optimized. We present the steps in estimating these parameters.

This section provides a detail explanation on the model selection process as well as parameter optimization. Table 1 summarizes the process.

Model	Stage	Dist.	ϑ_M	Search method step 1	Interactive method step 2
Variance (λ_t)	1	Weibull	κ	ϑ_T	ACI: NR (ϑ_M), Adam (ϑ_N)
		GB2	κ, ρ_1, ρ_2	ϑ_T, ϑ_M	Adam (ϑ_N)
Return Mean (μ_t)	2	N	-	ϑ_T	Adam (ϑ_N)
		T	ν	ϑ_T	ACI :MM (ϑ_M), Adam (ϑ_N)
Return Quantile ($q_{t;\alpha}$)	2	AL	φ	$\vartheta_T, \vartheta_M^0$	ACI: NR (ϑ_M), Adam (ϑ_N)
	1	AL	σ	$\vartheta_T, \vartheta_M^0$	ACI: NR (ϑ_M), Adam (ϑ_N)

Table 1: Procedures for parameter optimization for all models. ϑ_M^0 denotes starting values. ϑ_M^0 for Weibull distribution are obtained from Bayesian CARR(1,1) model and for t distribution is obtained from sample kurtosis.

2.4.1 Step 1: Hyperparameter estimation

Hyperparameter setting and tuning

Firstly, since NNs typically run on mini-batches to reduce memory traces and allow parallel running to shorten the training times, batch size $\mathcal{B}$ indicates size of each mini-batches. Larger $\mathcal{B}$ leads to a shorter training time, higher memory usage, and possibly lower accuracy. The batch size range we consider is (16,64). Dropout rate, $\mathcal{R}$ is used to prevent overfitting. The rate refers to the percentage of the neuron being deactivate. The searched range for the dropout rate is (0.2,0.5).

Next, for LSTM, the number of hidden layers $\mathcal{H}_l$ is an important parameter that describes the architecture of a LSTM. If the data have large number of features, $\mathcal{H}_l$ is suggested to be 3 to 5. However, a better loss function can increase the network power and potentially reduce $\mathcal{H}_l$. Therefore, we search the number of layers in a range of $\mathcal{H}_l \in \{1, \dots, 5\}$. Another important hyperparameter is $\mathcal{H}_d$ which relates to the number of neurons per hidden layer and is assumed constant for convenient. The number of hidden dimension $\mathcal{H}_d$ is fixed for each layer for simplicity ranging from 32 to 128. For the learning rate, a large $\mathcal{L}$ allows the model to learn faster, at the cost of skipping the optimal weights and arriving at a sub-optimal set of weights. The learning rate is searched in the range (0.00001,0.1). Besides, the searched range for the window size l is (2,30).

Because the modeling of persistence of time series is often considered more important, the next FNN architecture is kept simple with one hidden layer. We assign a batch normalization layer followed by a dropout layer after the first hidden layer in the FNN part after the LSTM layers to mitigate the challenges of overfitting and ensure stability and efficiency of the learning process. Lastly, we choose the efficient optimizer Adam (Kingma and Ba, 2015) to address the vanishing or exploding gradients issue when estimating the network parameters.

These architecture hyperparameters ϑ_T denoted in Eq. (23) are searched by using the Optuna package in Python. Apart the optimizer Optuna used in this paper, Python libraries also facilitate other hyperparameter optimization, including KerasTuner, Ray Tune, and Hyperopt and so on. Due to the lightweight architecture and ease of automation, Optuna is chosen. Moreover, it requires only a simple objective function that returns a scalar metric to guide the optimization. For parameter selection, we adopt the Tree-structured Parzen Estimator (TPESampler). This sampler models “good” vs “bad” trials separately, then proposes new hyperparameters that are likely to improve your objective. Furthermore, Optuna includes an efficient pruning mechanism that stops bad trials early based on intermediate results. In conclusion, tuning for these hyperparameters ϑ_T (while fixing optimizer to Adam) using Optuna is very important to provide good architectural environment to run the LSTMs and FNNs.

To get the reasonable hyperparameter values of ϑ_T , we sample simultaneously $\mathcal{M} = 8000$ sets of possible ϑ_T values using Optuna (Akiba et al., 2019) package in Python. The best set of tunable hyperparameter is selected based on the lowest validation loss *alone* except for three cases. In the first case, the shape parameters ϑ_M of the stage one variance models with GB2 distribution are searched together. Next for

quantile model with AL distribution, the search also includes $\vartheta_M = \{\varphi\}$ for two-stage and $\vartheta_M = \{\sigma\}$ for single stage but they are used only as initial values $\vartheta_M^{(0)}$ for the NR method in step 2 because the estimation procedures in step 2 are sensitive to initial values of φ and σ which vary with ϑ_T . For Weibull and t distributions, initial values are less dependent on ϑ_R . Hence the initial value is estimated independently using CARR(1,1) model for Weibull distribution whereas for t distribution, we used a method-of-moment approach described in Eq. (24) below to replace the model mean μ_t and SD σ_t estimates with sample mean $\bar{y}$ and sample SD s .

2.4.2 Step 2: Model parameter estimation and model selection

Model parameter optimization

By fixing the ϑ_T searched in the previous step, we are using this set of hyperparameter to estimate the optimal model ϑ_M and network ϑ_N parameters. Specifically, for the variance model with Weibull NLL, return model with t NLL and direct quantile models with AL NLL, we estimate $\vartheta_M|\vartheta_N, \vartheta_T$ and $\vartheta_N|\vartheta_M, \vartheta_T$ iteratively until ϑ_M converges. These procedures give rise to the alternating conditional iterative (ACI) estimation method as illustrated in Figure 2. In each iterative loop, the model parameters ϑ_M can be estimated using different methods. For example, it can be optimized alone (outside stochastic gradient descend (SGD) of NN) using optimizers such as Newton-Raphson (NR) condition on network parameters ϑ_N and hyperparameter ϑ_T .

Figure 2 shows, the ACI method using Bayesian CARR model or MM to generate initial values ϑ_M^0 for the variance (using Weibull) and return (using t) models. Initial values for quantile models are searched with the tunable hyperparameters in step 1. Based on the initial parameter ϑ_M^0 and the hyperparameters ϑ_T from step 1, we can run the NN to get the first set of weight and bias ϑ_N^0 and the output z_t . The NN is trained for 100 epochs and the optimal parameters θ is selected which minimizing the combined validation and test loss. For the next step, given z_t and ϑ_N^0 , the model parameter ϑ_M is optimized based on the NN output and upgrades to the ϑ_M^1 . We repeat this process until the difference between ϑ_M^i and ϑ_M^{i+1} is less than 0.001 where i denotes any iteration.

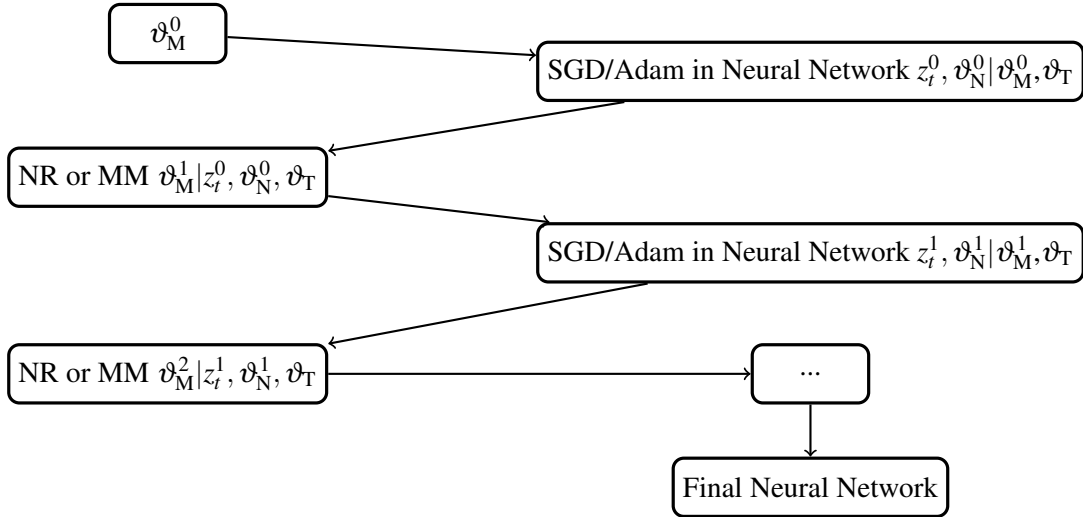


Figure 2: Alternating conditional iterative estimation method of $\vartheta_M|\vartheta_N, \vartheta_T$ and $\vartheta_N|\vartheta_M, \vartheta_T$ with fixed tuning parameter ϑ_T . NR method is used in Weibull model, and one and two-stage AL models whereas moment match (MM) method is used in t model. GB2 model does not use this method; instead $(\vartheta_M, \vartheta_T)$ are search together.

Variance Models - First stage

The two loss function considered for the variance models are $\ell_{1;\text{Wei}}(\eta_t(\vartheta_N), \kappa|V_{1:N})$ in Eq. (9) and $\ell_{1;\text{GB2}}(\eta_t(\vartheta_N), \kappa, \rho_1, \rho_2|V_{1:N})$ in Eq. (6). For $\ell_{1;\text{Wei}}$, the NR method estimates the shape parameter $\vartheta_M = \{\kappa\}$

(see Eq. (19)) by iteratively refines the estimate using the first and second order derivatives of the NLL, allowing fast local convergence. By cross checking the `newton-cg` function (Virtanen et al., 2020) in python and manual calculation of NR, both give similar optimized κ value based on the dataset.

For $\ell_{1;GB2}$, GB2 distribution has three shape parameters to optimize, which complicates the model structure considerably, making it challenging for the optimizer to locate and converge to optimal solution. We find that most of the optimizers including NR cannot converge. Therefore, we use the search method over the space of $\vartheta_M = \{\kappa, \rho_1, \rho_2\}$ together with the tuning hyperparameter ϑ_T base on the lowest validation loss by using the `optuna` package in python.

Return Model - Second stage

For the return model, $\ell_{2, \text{Norm}}$ and $\ell_{2, T}$ are considered to be the loss function for the density network. For $\ell_{2, \text{Norm}}(\mu_t(\vartheta_N) | \hat{\sigma}_t)$, no additional model parameter need to be optimized. However, for $\ell_{2, T}(\mu_t(\vartheta_N), \nu | \hat{\sigma}_t)$, similar to the case of $\ell_{1, \text{Wei}}$, we first estimate ν through conditional iterations using NR method. However, the optimization gives $\nu < 2$, implying a distribution with heavily tails and undefined variance, preventing the calculation of parameteric quantiles. Moreover, using such estimated parameter gives large and hence poor performance metrics such as MSE and MAE showing that such ν estimate is numerically unstable. Instead, the method of moment (MM) approach is adopted as below

$$\nu = \frac{6}{\mathcal{K}} + 4 \quad \text{where} \quad \mathcal{K} = \frac{1}{N} \sum_{t=1}^N \left[\frac{y_t - \mu_t}{\hat{\lambda}_t^{1/2}} \right]^4 - 3 \quad (24)$$

by matching the excess kurtosis $6/(\nu - 4)$ of t distribution to the excess kurtosis $\mathcal{K}$ calculated using mean μ_t and variance $\hat{\lambda}_t$ estimates. We update $\nu | \mu_t, \hat{\lambda}_t$ and $\mu_t | \nu, \hat{\lambda}_t$ iteratively for 100 loops, check the validation loss across the epochs, and choose the ν value based on the lowest average validation loss of the loop.

Quantile Model

For the two stage direct quantile model, the loss function $\ell_{2, \text{AL}}(q_{t; \alpha}(\vartheta_M) | \hat{\sigma}_t)$ includes the additional model parameter $\vartheta_M = \{\varphi\}$ as in Eq. (19) is estimated using NR optimizer in the conditional iterative estimation method in Figure 2. For the single stage direct quantile model, the loss function $\ell_{2, \text{AL}}(q_{t; \alpha} | \alpha, \sigma)$ includes optimizable scale parameter σ . By using the Conditional interacting estimation method, the σ is optimized in each interaction using the NR method.

2.4.3 Step 3: Repeats to ensure model stability

Because of the stochastic nature in training, different network parameters ϑ_N can be estimated based on the same selected parameter set $(\hat{\vartheta}_M, \hat{\vartheta}_T) \in \hat{\theta}$. Hence, we rerun 10 repeats of each model using 10 different seeds but fixing $(\hat{\vartheta}_M, \hat{\vartheta}_T) \in \hat{\theta}$ to ensure that the network parameters ϑ_N are stable. We then select the best network parameters ϑ_N^* such that $\theta^* = (\hat{\vartheta}_M, \hat{\vartheta}_T, \vartheta_N^*)$ gives the lowest *validation and test loss*. The whole selection procedures ensure good model training and forecast performance.

3 Empirical Application

We analyze Bitcoin, the most outstanding cryptocurrencies widely used around the world. We utilize the exchange rate of 1 BTC to USDT where USDT is a stable-coin designed to maintain parity with the U.S. dollar. According to the data from [CoinMarketCap.com](https://coinmarketcap.com), USDT accounts for 86.8% of Bitcoin transactions involving U.S. dollar-pegged cryptocurrencies (including USD, USDC, and USDT).

For comparative purposes, we extend our analysis to include a second cryptocurrency. Since we aim to select a cryptocurrency that does not share too many properties with Bitcoin—unlike Ethereum, which is another blockchain cryptocurrency. Based on the market capital price provided from [CoinMarketCap](https://coinmarketcap.com).

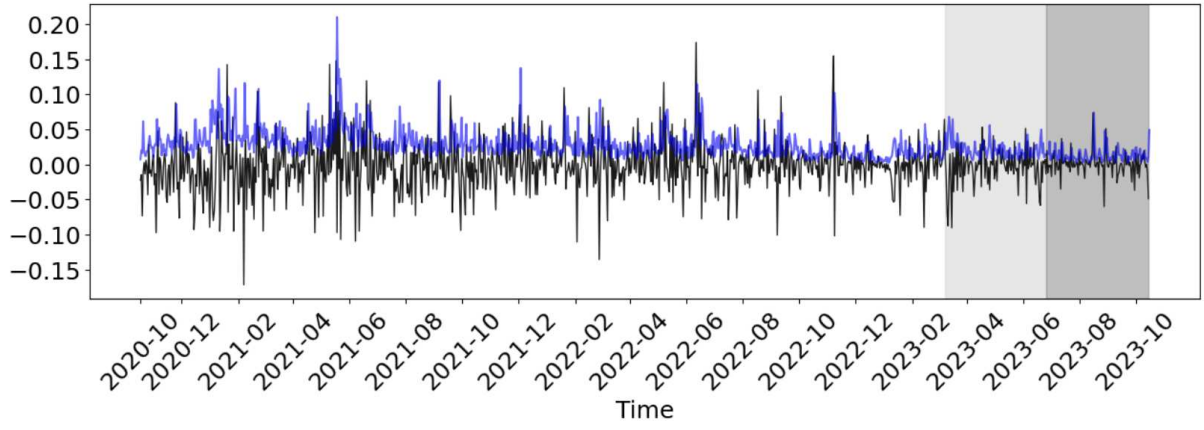
com, the top 5 cryptocurrency are Bitcoin, Ethereum, Tether, BinanceCoin and Ripple. Among those, Bitcoin, Ethereum and BinanceCoin are blockchains, while Tether is not a native cryptocurrency as it is built directly into the blockchain's infrastructure itself. Therefore, we choose XRP (Ripple), the largest digital payment protocol, for our empirical studies.

3.1 Target Data

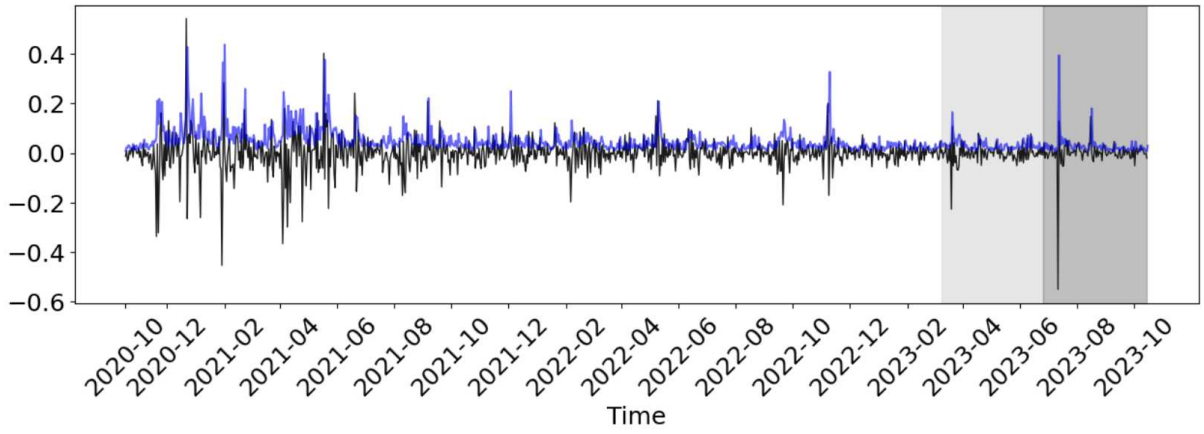
The Bitcoin historical daily data is collected from investing.com span between 17 October 2020 to 16 October 2023 with $T^* = 1095$. The training period are sampled from 17 October 2020 to 13 March 2023. This period includes $T = 876$ days. Further $W = T^* - T = 219$ days from 14 March 2023 is the validation and testing period where validation period consist 109 days and testing period has 110 days.

Figure 3a and 3b shows the daily log return and variance (blue color) for Bitcoin and Ripple with shaded area representing the validation and test period. According to the return scale, ripple has larger variance during late 2020 to early 2021. They are also plotted in Figure 4 column 1. The daily log return of Ripple spiked on July 13, 2023 because a U.S. federal court ruled in the U.S. Securities and Exchange Commission versus Ripple (Labs Inc.) that XRP sales on exchanges to retail buyers were not securities, slashing regulatory risk and prompting major exchanges to relist the token. Top 2 rows of Table 2 highlight some important measures for the log return for both cryptocurrencies. Ripple has high kurtosis and negative skewness compared to Bitcoin which matches the result from the return series plot in Figure 3 where Ripple is more volatile. Both Bitcoin and Ripple have significant Shapiro-Wilk p-value indicating that the data is not normally distributed. Next two rows of Table 2 includes summary statistics for the Parkinson measures. Hurst exponent ([Hurst, 1951](#)) is included to evaluate the strength of long-term persistence for each time series. The Hurst exponent based on the raw return and variance series for both Bitcoin and Ripple are more than 0.5, which indicates persistence and that return variances of any coins has stronger persistence than its returns. Moreover, Bitcoin (returns and return variances) has stronger persistence than Ripple.

Figure 4 visualizes the autocorrelation function plot (ACF; column two) and partial autocorrelation function plot (PACF; column three) for return and $V_{PK,t}$ of both Bitcoin and Ripple respectively during the training period. ACF plots display short memory feature of both Bitcoin and Ripple daily return as they show no significant autocorrelation with value close to 0 after lag 0. However we do notice for Bitcoin, the ACF plot has lag 9 slightly above the shaded 95% confidence interval. This happens to the PACF of Bitcoin return as well. PACF is used to identify the AR term where it shows the correlation of the series with its lags after removing the effect of shorter lags. For Ripple return, there is no significant ACF and PACF as since the values are all within the confidence interval. For PK volatility $V_{PK,t}$, the ACF for both cryptocurrencies shows graduate decay and remains positive over many lags, indicating some memory in the dataset. Volatility, typically exhibits longer memory than return. The PACF plot drops sharply after lag two suggesting an AR(2) structure. In summary, Bitcoin (volatilities) has stronger persistence and AR property than Ripple (returns), both agree with Table 2.



(a) Bitcoin daily return and volatility



(b) Ripple daily return and volatility

— Daily Return — Daily Volatility

Figure 3: Daily return plot for the period between 17 October 2020 to 16 October 2023

Table 2: Summary statistics during the training period for Bitcoin and Ripple daily return and daily Parkinson measures. ‘pv’ represents p -value.

	Mean	Medium	SD	Kurtosis	Skewness	Hurst Exponent	Shapiro-Wilk pv
Bitcoin Return	-0.07×10^{-3}	-0.02×10^{-3}	0.04×10^{-3}	2.71	0.23	0.60	<0.0001
Ripple Return	-0.55×10^{-3}	-0.76×10^{-3}	0.07×10^{-3}	12.99	-0.16	0.54	<0.0001
Bitcoin V_{PK}	0.03	0.03	0.02	9.69	2.34	0.72	-
Ripple V_{PK}	0.01	0.00	0.01	89.08	8.36	0.68	-

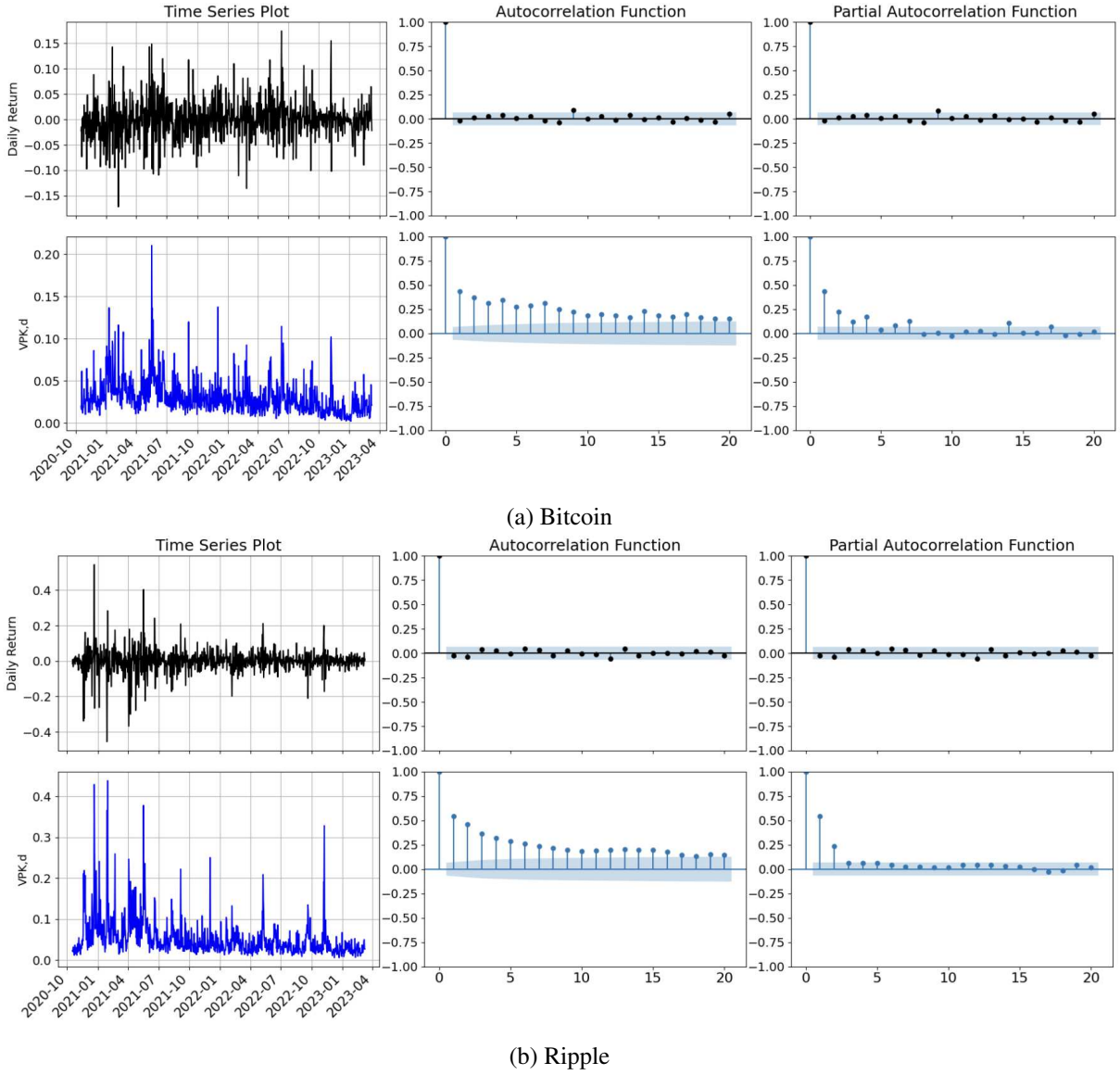


Figure 4: History, ACF and PACF plots for daily return (in black) and daily Parkinson measures (in blue) for BTC and Gold during the training period between 17 October 2020 to 13 March 2023

3.2 Features selection

For the LSTM for both volatility or return model, we use a look back window (lagged values) together with GARCH(1, 1) outputs to model the persistence. We do not include CARR outputs because GARCH model takes returns as input and so might present ability to capture additional volatility patterns.

For External features in FNN, Bitcoin and Ripple models use different sets due to the nature of the two cryptocurrency. Bitcoin is a blockchain based cryptocurrency. It operates based on blockchain technology, which records transactions on a decentralized, immutable ledger maintained by a network of nodes through a consensus mechanism such as proof-of-work. Majority features used for Bitcoin are related to blockchain data. Details of the features are provided in Appendix A.

By adopting non-blockchain features such as ‘Cost per transaction’ into the Bitcoin modeling, it helps to capture the cross-asset and macro linkage which blockchain feature might miss. We conduct ACF periodicity check to Bitcoin features, and notice that ‘average block size’, ‘cost per transaction’, ‘number of transaction per block’, ‘number of transaction’ and ‘number of unique address’ display weekly periodic pattern. These 7 days period is possible due to human activities. However, experience from [Chen et al.](#)

(2025) shows that the periodicity from the features does not seem to influence the volatility and mean return model. However, the weekly period has greater influences on quantile regression. Therefore, for the second stage quantile regression, the weekly periodicity of the above five Bitcoin features is removed by using STL decomposition function in python. See Section 3.3 of [Chen et al. \(2025\)](#) for more details of the deseasonalization.

Ripple is a digital payment protocol and cryptocurrency platform created in 2012. Comparing to Bitcoin, Ripple is using the Ripple (XRP) Ledger Consensus Protocol for the Consensus mechanism while Bitcoin is using the Proof-of-Work which is also called mining. This distinguishes the features that we can use for Ripple. For example, Ripple does not have ‘hash-rate’, ‘difficulty’ and block size related features but it has ‘sharpe ratio’, ‘reddit subscribers’, etc apart from ‘transaction fee’, ‘transaction volume’, ‘active address’. Unlike Bitcoin that some features has significant weekly periodicity, Ripple features do not have such periodicity, therefore no deseasonalization is needed. www.blockchain.com and cryptoquant.com provide access to these two feature sets.

Figure 5 shows the correlation heatmap between features, returns and volatility for both Bitcoin and Ripple. We can see that the Parkinson volatility for both cryptocurrency have higher correlations with the features than the daily log return. Within features, they are not so correlated except ‘block size’, ‘cost per transaction’, ‘difficulty’ and ‘hash rate’ for Bitcoin.

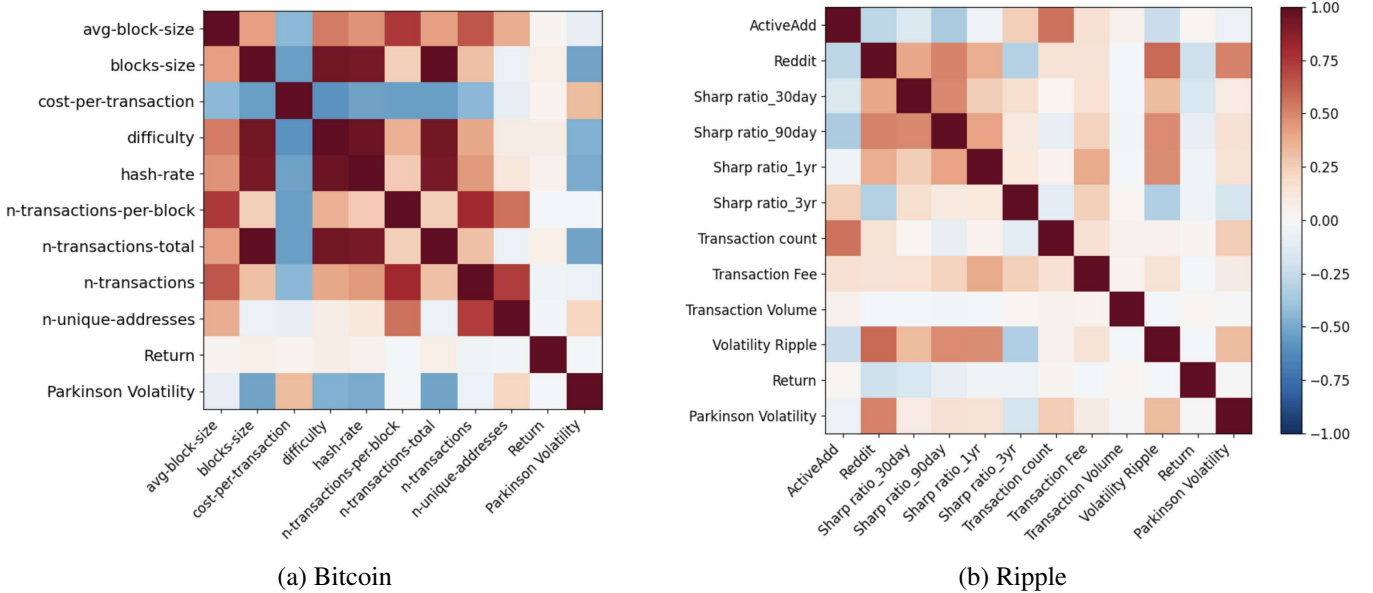


Figure 5: Input feature autocorrelation heatmap for Bitcoin and Ripple during the training period.

4 Experimental Result

This section evaluates the model performance of the first and second stage models including three kinds of VaR results. A comparison between LSTM variance model, GARCH model and Bayesian CARR model is conducted under Section 4.1.1 for Bitcoin application. For the quantile model result comparison between statistical CAViAR model and neural network models, see Section 4 of [Chen et al. \(2025\)](#).

4.1 First Stage Variance Model

This section discusses the first stage variance model forecast result for Parkinson variance measure using Bitcoin and Ripple daily data.

4.1.1 Bitcoin

The first stage is the variance model based on the Parkinson variance measure. For Bitcoin, two negative log likelihood (NLL) functions are considered for the density network using Weibull and GB2 distributions. Table 3 reports, for the optimal model out of 10 repeats (section 2.4.3), the searched hyperparameters as well as the searched and optimized model parameters for GB2 and Weibull distribution respectively, together with performance metrics during the test period. The bracket value next to the performance metrics indicates the standard deviation of certain metric across the result of 10 repeats. For Bitcoin results, it is obvious that the variance model with GB2 NLL gives lower Loss and MAE. Although the MSE of GB2 model is slightly larger than that of Weibull model, it may be more sensitive to noise as the MSE value is small. Moreover, the standard deviation of MSE and MAE across 10 repeats for GB2 model are also smaller than Weibull model showing that forecasts from GB2 model are consistent and well performed. On the other hand, the standard deviation of the loss for GB2 model is two times larger than Weibull. After checking the 10 losses of Weibull model, it just indicates that they are consistently worst. With these superior performances, GB2 model is selected to provide the first stage variance estimates λ_t which will be used as the scale parameter of the second stage return model. Hence, for the Ripple application, only GB2 model is applied to estimate the variance of returns.

To visualize the performance, the observed and predicted/forecast variances are displayed in Figure 6a and 6b during the whole time period (T^* observations) with red lines representing fitted volatility during the training period, green for validation period and yellow for testing period. From the plots, the predicted volatilities using Weibull NLL and GB2 NLL have similar performance. They follow closely the pattern of the Parkinson variance measures during the training period for both distributions. Predicted volatilities using Weibull distribution catch more extreme jumps than GB2 distribution. However, those from Weibull distribution are slightly lifted up resulting in positive biases. This phenomenon may be caused by the lack of flexibility of Weibull distribution making it sensitive to jumps and so, overestimating variances. Besides, the Parkinson variance measures look relatively stable during validation and test periods comparing to the training period, which results in stable forecast with minor jumps and also suggests potential distribution drift that may affect forecasting performance. The better performances of GB2 distribution agree with Table 6.

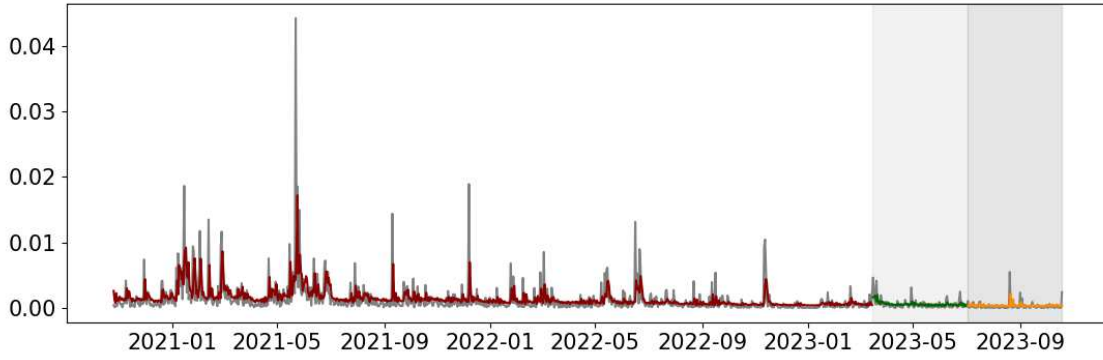
Table 3: Parameter estimates and performance metrics for the first stage volatility model for Bitcoin and Ripple data during the validation and testing period. note, $^\dagger = \times 10^{-4}$

Model	ϑ_T					ϑ_M		Loss (sd)	MSE (sd)	MAE (sd)
	$\mathcal{B}$	$\mathcal{R}$	$\mathcal{H}_1$	$\mathcal{H}_d$	$\mathcal{L}$	κ	ρ_1, ρ_2			
BTC-GB2	52	0.3817	3	87	0.0405	2.2472	1.5654, 0.8384	-0.0651 (0.0343)	0.0047 [†] (0.0001 [†])	0.0003 (0.0813 [†])
BTC-Wei	54	0.2254	1	49	0.0213	1.0943	-	0.9497 (0.0179)	0.0046 [†] (0.0148 [†])	0.0004 (2.1318 [†])
RIP-GB2	54	0.3778	1	47	0.0437	11.2091	0.2919, 0.1029	0.6929 (0.0446)	0.0001 (0.0002)	0.0031 (2.1318 [†])

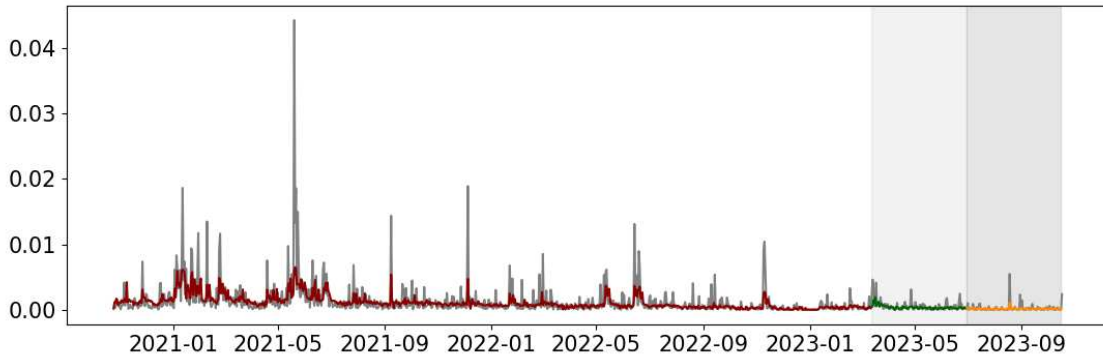
Predicted variances of Bitcoin using GB2 density network is further compared with the Bayesian CARR(1,1) and GARCH outputs. They are chosen not only because GARCH and CARR models are popular; we also aim to compare models with different estimation methods. However, under the parametric modeling approach, we are unable to add those features in NN models as predictors in the CARR model due to limitations in computational capacity of the Bayesian MCMC process. We also note the sensitivity of CARR model parameters to the prior choices. In the search for priors, We start with noninformative priors such as $\Gamma(10^{-4}, 10^{-4})$ for positive parameters and $N(0, 10)$ for real-valued parameters, and then gradually sharpen the prior distribution. We adopt the Bayesian modeling framework of [Chen et al. \(2025\)](#), where full technical details are available.

For GARCH models, they are fitted using the rugarch package in R while the order and type are searched using the SBAGM package ([Kumar et al., 2025](#)) in R. ARMA model of order (0,0) (based on Akaike Information Criteria AIC), with GJR-GARCH of order (1,1) and skew normal distribution (based on RMSE and MAE) is selected. Since the package allows feature input, the same set of Bitcoin features are inserted

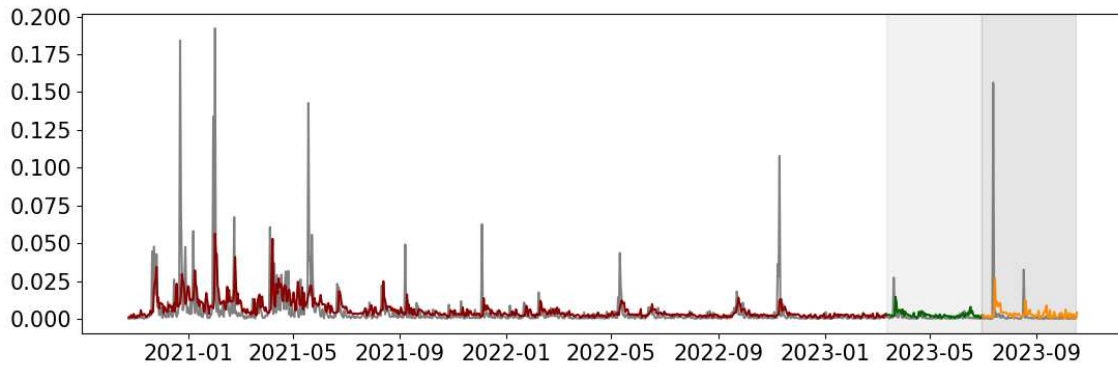
(a) Weibull-BTC



(b) GB2-BTC



(c) GB2-XRP



— True Variance — Training Period — Validation period — Testing Period

Figure 6: Observed Parkinson variance and fitted variance by using (a) Weibull NLL; (b) GB2 NLL based on Bitcoin data; and (c) GB2 NLL based on Ripple

as external regressors. To facilitate comparability, forecasts from both GARCH and Bayesian CARR(1,1) models adopt the common practice as LSTM, that is, applying one best model to the whole test period.

Figure 7 compares the output from LSTM, GARCH and Bayesian CARR models using GB2 NLL with Bitcoin data. The shaded area in the plot refers to the test period. We note that statistical models such as Bayesian CARR and GARCH models do not require validation period to check overfit. During the combined training and validation period, Bayesian CARR model (dark blue) tends to be better in capturing the jumps than other models. However, Bayesian CARR model has slower decay after jumps than other models indicating higher persistence. The output from GARCH model (orange) is relatively flat comparing to LSTM (orange) and Bayesian CARR models. During the testing period, GARCH model seems to overestimate the variance whereas Bayesian CARR model is flat and close to zero. The forecast performance metrics (MSE, MAE) for LSTM, GARCH and Bayesian CARR model are $(4.17 \times 10^{-7}, 2.53 \times 10^{-4})$, $(5.03 \times 10^{-7}, 5.47 \times 10^{-4})$ and $(4.49 \times 10^{-7}, 2.54 \times 10^{-4})$ respectively during the test period. In comparison, the LSTM model displays the lowest MSE and MAE. On the other hand, the plot shows that GARCH model might have better MSE/MAE during the combined training and validation period but its long term memory overestimates the forecasts during the test period which results in higher MSE/MAE comparing to LSTM.

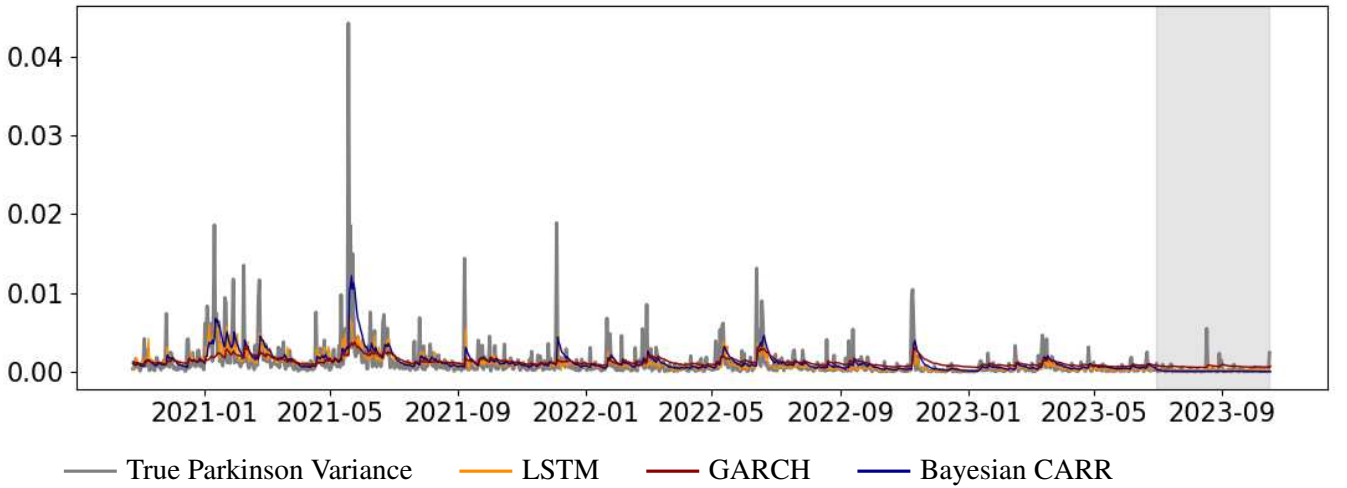


Figure 7: Observed Parkinson variance and predicted/forecast variance using LSTM, GARCH and Bayesian CARR(1,1) model GB2 distribution for Bitcoin.

4.1.2 Ripple

We adopt the best model of Bitcoin to Ripple without going through the model selection process since GB2 distribution offers the necessary flexibility to model the complex and leptokurtic nature of cryptocurrencies. Figure 6c shows the predicted and forecast variances of Ripple using the density network with GB2 distribution. The predicted variances trace the jumps of Parkinson variance closely during the training period and model the trend near perfectly during the calm validation period. Last row of Table 3 reports the tuning and model parameters for the variance model with Ripple along with the loss and performance metrics. Comparing to Bitcoin data, Ripple has higher loss, MSE and MAE. By referring back to the summary statistics of Parkinson Variance in Table 2, Ripple V_{PK} tend to have higher kurtosis and skewness which may present challenge the model. On the other hand, as κ , the tail/curvature parameter of GB2 distribution, is estimated to be higher, the fitted GB2 distribution has a sharper peak and thinner tail. However, the overall kurtosis of GB2 distribution is also determined by the other two shape parameters, ρ_1 and ρ_2 .

4.2 Second Stage Return Model

By setting the fitted volatility result from the first stage based on GB2 NLL to be the scale parameter (with adjustment factor ψ) of the normal and t return distribution, the modeling of the mean of returns can be more focused. Similar to the first stage model, we do 10 repeats with different seeds for the second stage selected model (section 2.4.3) and the best model is chosen based on the validation and testing period performance. The features used in the second stage are the same as the first stage which is described under section 3.2.

Results of return model for Bitcoin, including the tuning parameter ϑ_T , model parameter ϑ_M and performance metrics during the validation and testing period with normal and t distributions are reported in the first two rows of table 4. Results show that t distribution gives a lower NLL during the validation and testing periods. The MSE and MAE results for both models are quite similar whereas t model has slightly lower MAE. The standard deviation for each metrics across 10 repeats is reported in the bracket. Student t model has lower standard deviation for the NLL loss; however it has higher MSE and MAE standard deviation, indicating higher variability across model estimates, possibly due to effect from the extra degrees of freedom parameter ν . This possibly demonstrates how t distribution adjusts ν to capture outliers but it may not provide outstanding predictability compared with normal distribution. Based on the much lower NLL and comparable MSE/MAE, t distribution is therefore adopted for the Ripple return modeling.

Return models for Bitcoin is also visualized under Figure 8a and Figure 8b with normal and t distribution respectively. Again, red line shows the training period fitted mean, and green and yellow line represents the fitted mean during the validation and testing period. Comparing between the fitted lines of the normal and t models, we see that fitted line of t model traces the observed trend better despite it also displays some under estimations near the end of the training period. On the other hand, the normal model fails to capture the trend in general and has some overestimation during the training period. Moreover, normal model tends to underestimate the trend during the validation and testing periods. This phenomenon may due to distribution drift in which the validation and testing periods are more calm with much less volatility clustering than the training period. Nevertheless, t model lessens the effects of outliers and so can possibly capture the real trend better.

Table 4: Parameter estimates and performance metrics of the second stage return model for Bitcoin and Ripple data during the validation and testing period

Model	ϑ_T					ϑ_M	Loss (sd)	MSE (sd)	MAE (sd)
	$\mathcal{B}$	$\mathcal{R}$	$\mathcal{H}_1$	$\mathcal{H}_d$	$\mathcal{L}$	ν			
BTC Normal	54	0.4515	2	96	0.0001	-	-1.6138 (0.8504)	0.0005 (0.0002)	0.0158 (0.0044)
BTC t	54	0.2020	3	34	0.0004	6.1947	-2.8583 (0.2739)	0.0005 (0.0011)	0.0157 (0.0121)
RIP t	52	0.2608	1	32	0.0227	4.7211	-2.4331 (0.0003)	0.0025 (<0.0001)	0.0245 (0.0004)

For Ripple, the parameter estimates and performance metrics are recorded under the last row of Table 4. The degrees of freedom parameter ν for T distribution is slightly lower than Bitcoin showing heavier tails. This may stem from the higher kurtosis and mild negative skewness of Ripple returns, indicating heavier tails that complicate the modeling process. Hence, Ripple's t model also has higher loss, MSE and MAE. However, the 10 repeats display lower variability and hence higher confidence in forecasts than Bitcoin.

The predicted returns and forecasts for Ripple is graphed in Figure 8c. Although Ripple returns have higher volatility (Figure 3b), they also display less trends, agreed with the milder skewness and lower Hurst Exponent in Table 2. Hence, the fitted mean looks more straight than the Bitcoin case but it is unbiased throughout the whole period. During the training period, the model tend to capture some negative spikes. The return of Ripple tends to be more volatile in the first half of the training period and more stable in the second half of the training period, demonstrating some distribution drift. During the testing period, the return forecast fails to capture the instantaneous extreme downward spike but it does capture the next

day upward spike. In summary, the t return model can capture the mean process well during the testing period.

4.3 Quantile Models

There are two ways of estimating the quantiles of return distribution. One calculates quantiles based on the two stage mean regression model whereas the other approach models quantiles directly based on asymmetric loss function via proxy skewed distribution. In this section, we compare their results with Bitcoin and Ripple data. In quantile regression, the MSE and MAE metrics are no longer suitable to assess the performance of quantile estimates, as these metrics fail to assess the tail behavior targeted by specific quantiles. In this case, Hit Rate (HR) (Christoffersen, 1998), also called Empirical violation rate, given by

$$\text{HR} = \frac{1}{N} \sum_{t=1}^N \mathbf{1}(y_t < q_{t;\alpha}^H) \quad \text{if } \alpha < 0.5,$$

is introduced as a performance metric. The inequality sign changes to ‘>’ if the quantile level $\alpha > 0.5$. HR directly verifies whether the proportion of returns that fall below (or above if $\alpha > 0.5$) the forecasted quantiles aligns with the nominal quantile level α . HR is widely used in financial risk management and it also forms the basis for the Kupiec unconditional coverage test. HR Error defined as $\text{HR} - \alpha$ further reports the discrepancy between fitted proportion from its nominal probability.

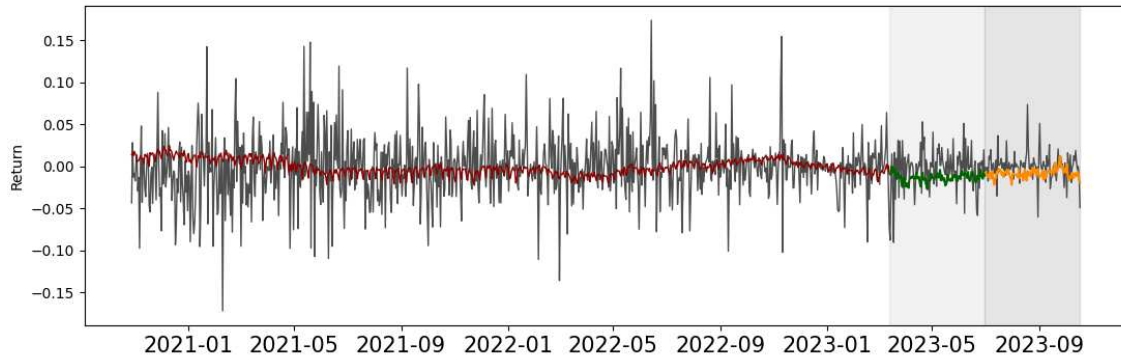
4.3.1 Two stage mean regression (parametric quantiles)

The two stage mean regression method is a parametric way based on distribution assumption for returns. We can calculate parametric quantiles from return model based on Eq. (12). Figures 10a and 10b depicts the quantile estimates with normal and T distributions for Bitcoin. The quantile level we compute including $\alpha = [0.025, 0.05, 0.1, 0.25, 0.75, 0.9, 0.95, 0.975]$. The quantile levels are shaded in pairs, with the darkest line in the middle represents the middle 50% of the return. Red shaded areas represent to quantiles for the training period, green for validation period and yellow for testing period. With the symmetric normal or t distribution assumption, the quantiles are evenly, and symmetrically spread along the estimated return mean.

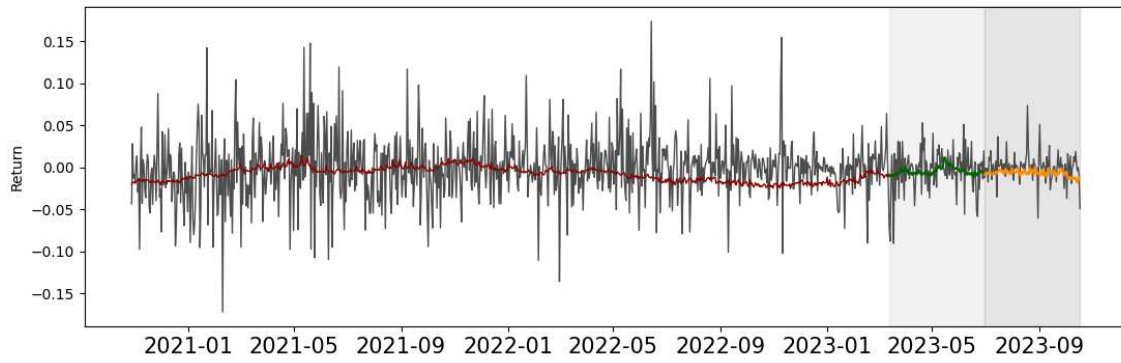
For *Bitcoin*, quantiles using normal or t distributions have similar quantile estimates. However, quantiles using t model have slightly negative bias near the end of training period due to its underestimated returns as shown in Figure 8b. Both models seem to provide reasonable coverage, but t model seems to have more precise middle 50% confidence interval during the validation and testing period. Table 5 reports HR Errors during the validation and testing period. Most HR Errors for both distributions are fairly low, meaning that the fitted coverages are close to the nominated α with majority of the α s.

For *Ripple*, we again use the two stage mean regression with t distribution to evaluate the parametric quantiles and they are plotted under Figure 11c. The quantiles behave similar to those for Bitcoin data, that is, the quantiles are evenly, symmetrically spread along the return mean estimates. Compare with quantiles using Bitcoin and t distribution, quantiles for Ripple tend to capture more extreme spikes, possibly attributable to the smaller degrees of freedom with heavier tails to capture the spikes.

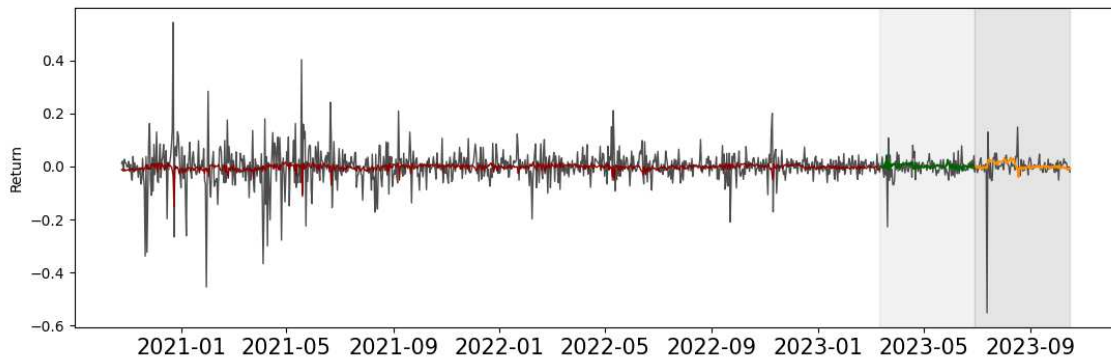
The HR Errors in Table 5 for Ripple are slightly larger than Bitcoin. These gaps of differences are particularly noticeable for $\alpha = 0.25$ and 0.75 , and they imply that the middle 50% of the estimated quantile covers more than 50% of the observed returns which may be driven by the over estimated variance $\hat{\lambda}$ from the first stage during the testing period in Figure 6c. This overestimation may stem from the larger variance across the training period leading also to enlarged variance during the testing period.



(a) Normal-BTC



(b) T-BTC



(c) T-XRP

— True Variance — Training Period — Validation period — Testing Period

Figure 8: True return and fitted return by using (a) Normal NLL; (b) T NLL based on Bitcoin data; and (c) T NLL based on Ripple

4.3.2 Direct quantile regression

It is less straightforward to estimate quantiles based on patterns of the mean process. Direct quantile regression captures dynamics of the quantiles of return distribution using the asymmetric loss. In the context, there are two direct quantile regressions considered in this paper. The first one follows the two-stage model, replacing the second stage mean regression with direct quantile regression, where the scale parameter $\hat{\sigma}_t$ is still time-varying but the model avoids the use of Eq. (12). The second one is called single stage direct quantile regression in which the scale parameter σ is fixed throughout the whole time series.

For *Bitcoin*, Figures 10d and 10c plot quantile forecasts from two-stage and single-stage direct quantile regression models using the same set of quantile levels. Comparing with parametric quantiles, direct non-parametric quantiles shows less volatility clustering and more crossover of (hence less regular) quantiles because of the flexibility of modeling individual quantiles. Hence, nonparametric quantiles model capture less the extreme spikes but instead, tend to provide regular and stable quantiles. Comparing the two stage model with single stage model during the training period, two stage model have wider middle 50% confidence interval and less crossovers between $\alpha \in \{0.05, 0.1\}$ extreme quantiles than the single stage model. It also has less volatility clustering possibly due to weaker long memory terms than the single stage model with fixed scale parameter. During the validation and testing period, quantiles from the two stage model, however, tend to be more volatile and behave similar to parametric quantiles.

Table 7 reports the parameter estimates, and performance metrics including NLL losses and HR Errors for the best quantile regression models among the ten repeats. The standard deviations of the metrics across repeats are reported in brackets. The upper half of the table relate to the application of Bitcoin data. The results for single stage quantile regression are highlighted in gray color. The model parameter ϑ_M contains σ for single stage model and the adjustment parameter ϕ for two stage model. The σ have larger estimates when α is close to 0.5. Although extreme quantiles are expected to have higher uncertainty, part of it can be explained by the skewness of the distribution. The σ estimates may also interact with other factors. For ϕ , when α is close to 0.5, the ϕ is larger and closer to 1. This is aligned with the experiences that $\phi \simeq 1$ for the mean regressions as in [Savage and Chan \(2025\)](#). Since the λ_t is estimated from the mean regressions, the adjustment factor ϕ contains the uncertainty of the quantiles due to the skewness of the distribution and other factors.

Comparing NLL losses, both two stage and single stage models are smaller when α is close to 0.5 and have a wider discrepancy at the two ends for extreme quantiles because extreme quantiles with less returns information are harder to estimate. Compared to single state model, two stage model tends to have lower loss but larger standard deviations than the single stage model. This better fit but higher variability across models stem from the higher flexibility of the two stage model structure. Moreover, quantiles from two stage model are closer to α for lower quantiles, slightly worse for middle quantiles ($\alpha = 0.25, 0.75$) and similar for upper quantiles. It is reasonable that the standard deviation of HR is relatively small for tail α s and larger for mid α s because the range of coverage is larger for middle α .

To compare with parametric quantiles, results of HR Errors for direct quantile regression models are also reported in Table 5. Results show that quantiles from direct quantile regression provide smaller errors than parametric quantiles and two stage model provides better lower quantiles whereas single stage model provides better upper quantiles. Figure 9 visualizes the absolute HR Error for different quantile models with red lines representing Bitcoin models. It shows that the absolute error trend is lower for extreme quantiles and higher for middle quantiles and that the two-stage mean regression model displays higher errors compared with the two quantile regression models.

For *Ripple*, Figures 11b and 11c show the quantile forecasts for two-stage and single-stage quantile regression models. Similar to Bitcoin, the single-stage model displays wider intervals and more volatility clustering. However, when $\alpha = 0.75$, quantiles from single stage model stay nearly constant with very mild jump, differ greatly from the two-stage model. Second half of Table 7 summaries the parameters and the performance metrics for the best of the ten repeated models. Both single- and two-stage models have

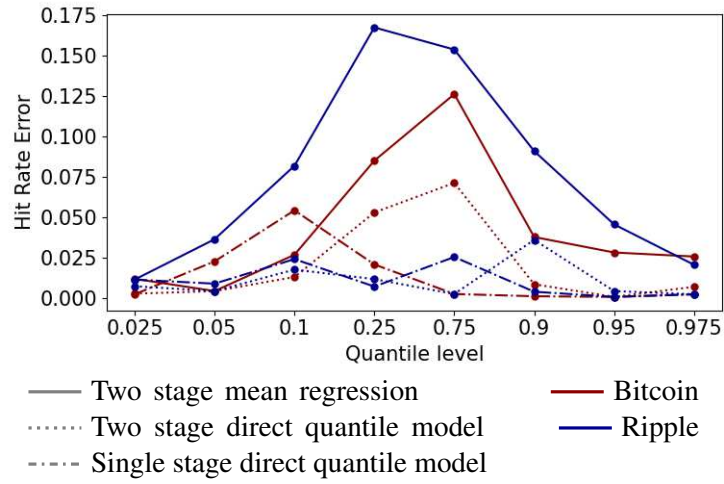


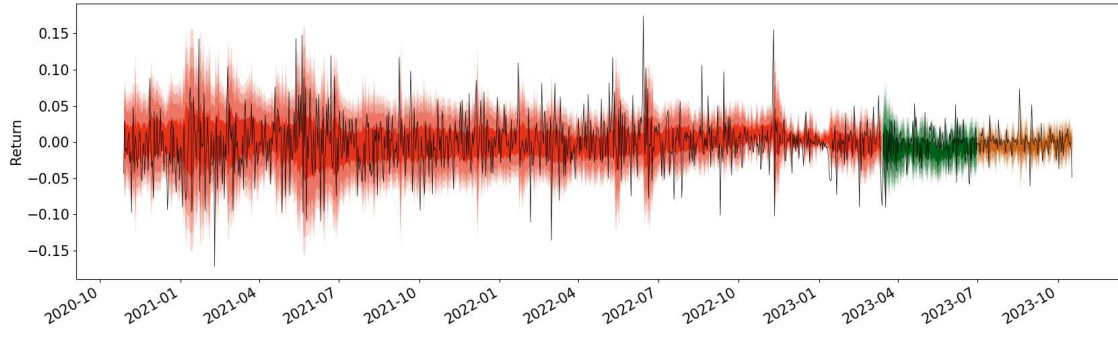
Figure 9: Hit rate error value during the validation and test period for different quantile models with Bitcoin and Ripple

reasonable HR with majority of α s but relatively, two-stage models have closer-to- α HR. For the same reasons as Bitcoin, standard deviation of HR and model parameter ϑ_M are higher when α is close to 0.5. And the loss from two-stage model is lower than single-stage model but the standard deviation of losses across ten repeats is higher for two-stage model. Comparing with Bitcoin, NLL losses of Ripple models are, in general, higher due to the higher kurtosis of Ripple data which consequently increases the difficulty of accurate modeling.

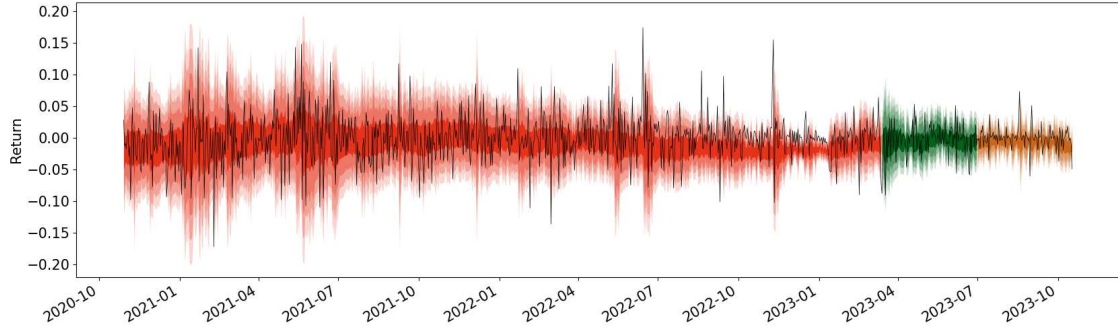
HR Errors in Table 5 again show that all direct quantile regression models have better HR than parametric mean regression models. Between the two quantile regression models, two-stage model provides slightly better lower quantile estimates. Figure 9 visualizes the absolute HR Error for different quantile models with blue color lines for Ripple. It is obvious that middle quantiles and parametric mean regression quantiles have higher errors. These differences are higher than those of Bitcoin demonstrating the clear advantages of adopting quantile regression for Ripple.

Table 5: Hit Rate Error ($|HR - \alpha|$) during the validation and test period for different α levels with Bitcoin and Ripple data.

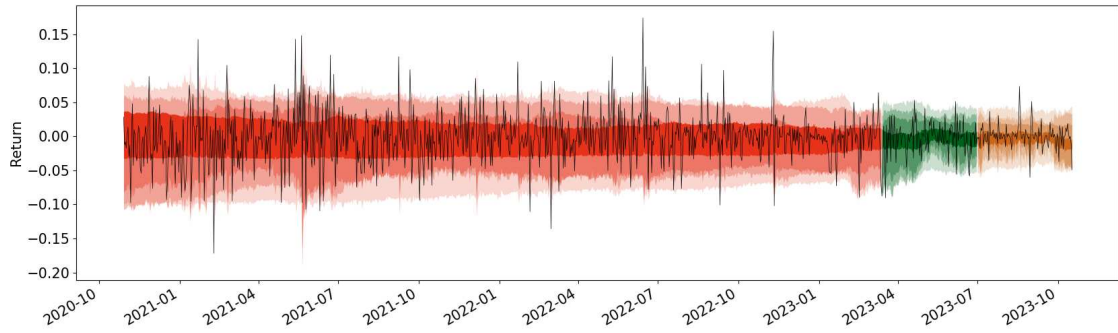
Model	Dist.	α							
		0.025	0.05	0.1	0.25	0.75	0.9	0.95	0.975
Bitcoin									
Two stage mean reg.	T	0.0117	0.0041	0.0266	0.0849	0.1261	0.0376	0.0280	0.0255
Two stage quantile reg.	AL	0.0025	0.0041	0.0128	0.0528	0.0711	0.0083	0.0005	0.0068
Single stage quantile reg.	AL	0.0021	0.0225	0.0541	0.0206	0.0023	0.0009	0.0005	0.0021
Ripple									
Two stage mean reg.	T	0.0112	0.0362	0.0817	0.1674	0.1537	0.0908	0.0454	0.0204
Two stage quantile reg.	AL	0.0071	0.0041	0.0174	0.0115	0.0023	0.0358	0.0041	0.0021
Single stage quantile reg.	AL	0.0112	0.0087	0.0239	0.0069	0.0252	0.0037	0.0005	0.0021



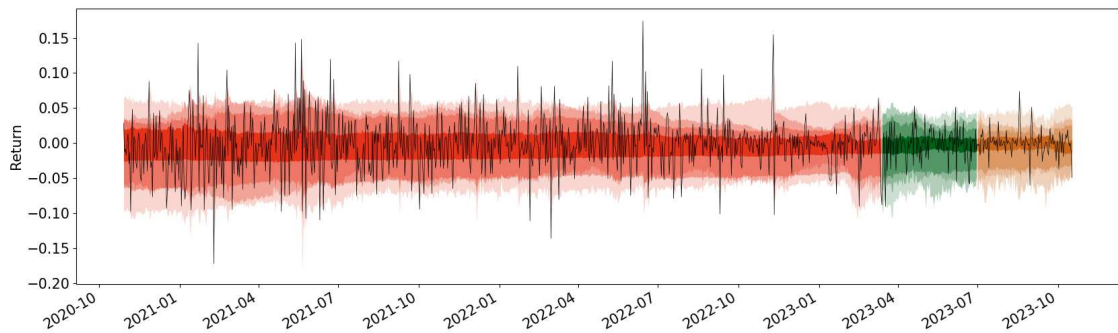
(a) Two stage mean regression model with normal distribution



(b) Two stage mean regression model with t distribution

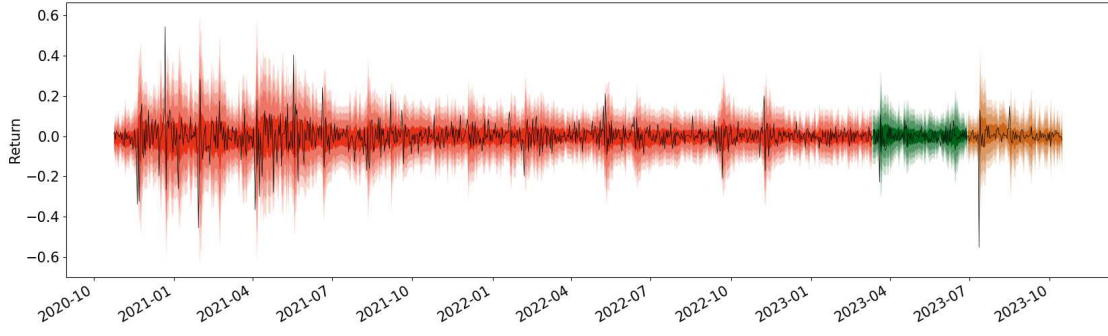


(c) Two stage direct quantile regression model

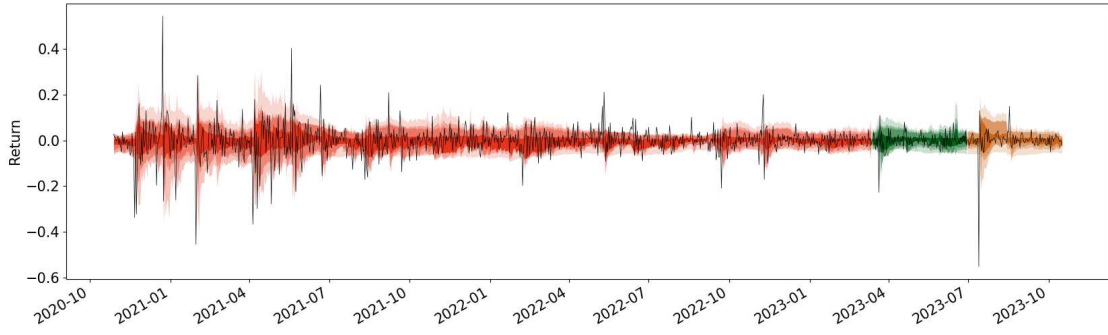


(d) Single stage direct quantile regression model

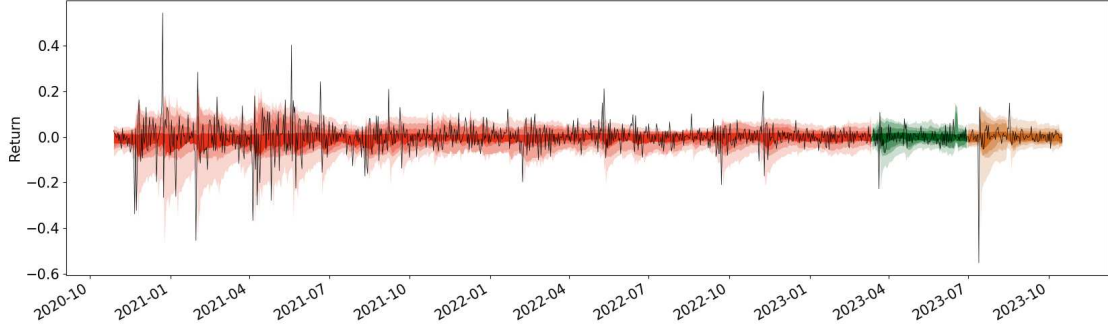
Figure 10: True return and fitted quantile with (a) Two stage mean regression with Normal; (b) Two stage mean regression with t ; (c) Two stage direct quantile regression; and (d) Single stage direct quantile regression with $\alpha = \{0.025, 0.05, 0.1, 0.25, 0.75, 0.95, 0.975\}$ for Bitcoin.



(a) Two stage mean regression model with t distribution



(b) Two stage direct quantile regression model



(c) Single stage direct quantile regression model

Figure 11: Observed returns and fitted quantiles with (a) Two stage mean regression model; (b) Two stage direct quantile regression model; and (c) Single stage direct quantile regression model with $\alpha = \{0.025, 0.05, 0.1, 0.25, 0.75, 0.95, 0.975\}$ for Ripple.

5 Conclusion

This study presents a comprehensive investigation into volatility, return and quantile forecast with application to Bitcoin and Ripple, since the highly volatile cryptocurrency market needs more advanced models for risk forecasting. In risk management, attention was directed to modeling returns directly with ARMA and GARCH models. However, these models are often conducted for the mean of return distribution. Even though GARCH models accommodate the time-varying volatility effect for the returns, they fail to

differentiate effects affecting specific quantiles of the distribution. Therefore, we propose some novel two stage volatility-return and volatility-quantile LSTM models to accommodate these issues.

This paper makes a number of pioneer contributions to extend the two-stage volatility-return model to the NN framework with LSTM and FNN. Firstly, we propose a two-stage LSTM model framework where the first stage estimates mean of the variance process using the Parkinson variance measures as input. These variance estimates are fitted into the second stage model as a time-varying variance parameter after allowing an adjustment factor. Two types of two stage model are introduced: two-stage volatility-return model and the two-stage volatility-quantile model. Both models implement the fitted Parkinson volatility λ_t from the first stage volatility model with GB2 distribution to the second stage.

Secondly, we extend the second stage return model with mean regression to normal and t distributions to LSTM framework and apply the mean and variance forecasts to calculate parametric quantile forecasts. Thirdly, the second stage mean regression is further extended to nonparametric quantile regression via the proxy AL distribution in which minimize the asymmetric L_1 loss is performed by minimizing the NLL of AL model. As a comparison, we also consider a simplified single-stage AL model in which the scale parameter is fixed. We demonstrate the model selection process for the best quantile regression model based on the HR Errors. Empirical results show that our proposed two stage volatility-quantile regression models enhance the HR accuracy of quantile predictions and forecasts, compared to parametric quantiles for Bitcoin and Ripple, .

There are several ways to further improve the proposed models. The current stage two quantile regression models focus on quantile/VaR which lacks the coherent risk measures property and does not always satisfy the sub-additivity property (Artzner et al., 1997,1999). As a first extension, one could explore the stage two expectile regression as expectile satisfies sub-additivity property. Alternately, one may consider a mix of both quantile and expectile regression to combine the interpretability of quantiles with sub-additivity of expectiles. The second research direction can add as efficient input features the lagged CARR output apart from GARCH output for both volatility, return and quantile models with single or two states. Besides, we can explore more impactful features and use distinct and more focused feature sets to model separately the volatility, return and quantile/expectile.

Apart from features, the third extension could consider a hybrid model (Chen et al., 2025) by including the CARR, CAViaR and CARE functions apart from NN function into the variance mean, return quantile and return expectile functions, respectively. To improve the modeling capacity, one can upgrade the LSTM to more complex architectures such as attention LSTM and transformer. Lastly, for parameter estimation, the current two-stage models use some less efficient parameter search or MM for models with several non-regression parameters (such as κ, ρ_1, ρ_2 of GB2) or complicated structures (such as Student- t). More efficient optimizers and methods should be explored to improve the estimation efficiency.

Appendices

A FNN features

(a) Bitcoin

- **Avg Block Size:** The average size of Bitcoin blocks in bytes.
- **Blocks Size:** The total cumulative size of all blocks in the Bitcoin blockchain.
- **Cost Per Transaction:** The average cost in USD to process a single Bitcoin transaction, calculated by dividing total mining costs by number of transactions.
- **Difficulty:** A measure of how hard it is to find a valid block hash.
- **Hash Rate:** The total computational power of the Bitcoin network measured in hashes per second, indicating mining strength and security.
- **N Transactions Per Block:** The average number of individual transactions included in each Bitcoin block.
- **N Transactions Total:** The cumulative count of all transactions ever processed on the Bitcoin network since genesis.
- **N Transactions:** The daily of Bitcoin transactions processed.
- **N Unique Addresses:** The number of distinct Bitcoin addresses that have been active .

(b) Ripple

- **Transaction fee:** sum of USD value of all fees paid to miners that day. Fees do not include new issuance.
- **Reddit Active Users:** The number of active users on the asset's primary subreddit.
- **Active Addresses Count :** The sum count of unique addresses that were active in the network (either as a recipient or originator of a ledger change) that interval. All parties in a ledger change action (recipients and originators) are counted. Individual addresses are not double-counted.
- **Sharp ratio:** Performance of the asset returns allowing comparing to the risk free assets over a window of time (30 days, 90 days, 1 year and 3 years).
- **Transaction counts:** The sum count of transactions that interval. Transactions represent a bundle of intended actions to alter the ledger initiated by a user (human or machine). Transactions are counted whether they execute or not and whether they result in the transfer of native units.
- **Transaction volume:** The sum of USD value in all native transfer unit (the aggregate size in USD of all transfers).
- **Volatility:** The annualized standard deviation of the daily return over 3 months period.

Table 6: Features used for FNN

B Quantile model summary

Table 7: Parameter estimates and performance metrics across single stage and two stage quantile regression for different quantile levels with Bitcoin and Ripple data during the validation and test period. Single stage direct quantile regression result is highlighted in gray

α	Stage	ϑ_{T}					ϑ_{M}	Validation and Test loss	Hit Rate
		$\mathcal{B}$	$\mathcal{R}$	$\mathcal{H}_1$	$\mathcal{H}_{\text{d}}$	$\mathcal{L}$	σ / φ		
Bitcoin									
0.025	1	36	0.3786	2	101	0.0006	0.0026	-1.7768 (0.0447)	0.0229 (0.0069)
	2	36	0.4512	2	92	0.0023	17.6180	-1.8878 (0.0367)	0.0275 (0.0062)
0.05	1	54	0.3850	2	94	0.0004	0.0042	-1.7927 (0.0465)	0.0275 (0.0070)
	2	54	0.4542	3	41	0.0002	12.0048	-2.1347 (0.4207)	0.0459 (0.0131)
0.1	1	36	0.3267	2	63	0.0076	0.0072	-1.7336 (0.0480)	0.0459 (0.0111)
	2	36	0.3753	1	63	0.0363	10.3340	-2.1410 (0.1143)	0.0872 (0.0258)
0.25	1	36	0.2559	2	74	0.0125	0.0103	-2.1329 (0.0212)	0.2294 (0.0604)
	2	54	0.2282	2	78	0.0133	6.8382	-2.1113 (0.0726)	0.1972 (0.0422)
0.75	1	36	0.3781	1	65	0.0184	0.0112	-2.2351 (0.0115)	0.7477 (0.0723)
	2	54	0.3093	2	43	0.0378	2.736	-2.3057 (0.0189)	0.6789 (0.0604)
0.9	1	52	0.2958	1	86	0.0068	0.0067	-2.0964 (0.0141)	0.8991 (0.0347)
	2	52	0.3489	1	46	0.0201	7.2331	-2.2589 (0.0345)	0.9083 (0.0348)
0.95	1	26	0.4556	1	56	0.0286	0.0045	-1.8528 (0.0114)	0.9495 (0.0168)
	2	51	0.4184	2	92	0.0078	13.3448	-2.0052 (0.0247)	0.9495 (0.0244)
0.975	1	52	0.3571	1	41	0.0476	0.0029	-1.6519 (0.0182)	0.9771 (0.0058)
	2	52	0.4006	3	74	0.0006	19.7621	-1.8117 (0.1005)	0.9817 (0.0151)
Ripple									
0.025	1	54	0.2034	1	125	0.0001	0.0067	-0.7733 (0.0137)	0.0138 (0.0065)
	2	54	0.3705	1	110	0.0000	15.5654	-0.8882 (0.0357)	0.0321 (0.0137)
0.05	1	54	0.2339	1	118	0.0000	0.0088	-1.2119 (0.0261)	0.0413 (0.0076)
	2	54	0.3145	3	103	0.0097	6.4797	-1.3603 (0.0169)	0.0459 (0.0293)
0.1	1	27	0.2490	2	110	0.0000	0.01263	-1.4815 (0.0245)	0.1239 (0.0326)
	2	54	0.2296	1	66	0.0001	3.2800	-1.7110 (0.0324)	0.0826 (0.0159)
0.25	1	54	0.2309	1	65	0.0001	0.0172	-1.9663 (0.0148)	0.2569 (0.0330)
	2	54	0.4156	1	37	0.0002	1.274	-2.0847 (0.0230)	0.2615 (0.0376)
0.75	1	48	0.2945	1	60	0.0827	0.0162	-1.9190 (0.0052)	0.7752 (0.0352)
	2	49	0.3465	1	32	0.0008	1.1191	-2.0327 (0.0108)	0.7523 (0.0252)
0.9	1	49	0.2163	3	67	0.0482	0.0110	-1.6284 (0.0042)	0.9037 (0.0166)
	2	53	0.3124	3	38	0.0050	2.6798	-1.7692 (0.0207)	0.9358 (0.0098)
0.95	1	35	0.2069	2	117	0.0061	0.0079	-1.3625 (0.0054)	0.9495 (0.0079)
	2	51	0.3659	3	118	0.0999	6.5001	-1.1517 (2.3847)	0.9541 (0.0187)
0.975	1	52	0.3126	2	48	0.0274	0.0061	-1.0562 (0.0081)	0.9771 (0.0084)
	2	35	0.4796	2	94	0.0462	10.9236	-1.3593 (0.0209)	0.9771 (0.0108)

Ethics declaration: not applicable

References

- [1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.
- [2] Sassan Alizadeh, Michael W Brandt, and Francis X Diebold. Range-based estimation of stochastic volatility models. *Journal of Finance*, 57(3):1047–1091, 2002.
- [3] Alexander Amini, Wilko Schwarting, Ava Soleimany, and Daniela Rus. Deep evidential regression. *Advances in Neural Information Processing Systems*, 33:14927–14937, 2020.
- [4] Philippe Artzner, Freddy Delbaen, Jean-Marc Eber, and David Heath. Coherent measures of risk. *Mathematical finance*, 9(3):203–228, 1997,1999.
- [5] Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of econometrics*, 31(3): 307–327, 1986.
- [6] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.
- [7] Michael W Brandt and Christopher S Jones. Volatility forecasting with range-based EGARCH models. *Journal of Business & Economic Statistics*, 24(4):470–486, 2006.
- [8] John Y Campbell, Andrew W Lo, A Craig MacKinlay, and Robert F Whitelaw. The econometrics of financial markets. *Macroeconomic Dynamics*, 2(4):559–562, 1998.
- [9] Jennifer So Kuen Chan, Kok Haur Ng, and R Ragell. Bayesian return forecasts using realised range and asymmetric CARR model with various distribution assumptions. *International Review of Economics and Finance*, 61:188–212, 2019.
- [10] Nia Chen, Jennifer Chan, and Linh Nghiem. Expectile and value at risk modeling for cryptocurrency using bayesian approach. *Available at SSRN 5392611*, 2025.
- [11] Nia Chen, Jennifer SK Chan, and Linh Nghiem. A neural-statistical hybrid model for value at risk and expectile forecast with an application of cryptocurrencies. *Linh, A Neural-Statistical Hybrid Model for Value at Risk and Expectile Forecast with an Application of Cryptocurrencies (December 12, 2025)*, 2025.
- [12] Ray Yeutien Chou. Forecasting financial volatilities with extreme values: the conditional autoregressive range (carr) model. *Journal of Money, Credit and Banking*, pages 561–582, 2005.
- [13] Peter F Christoffersen. Evaluating interval forecasts. *International economic review*, pages 841–862, 1998.
- [14] Ajay K Dhamija and Vinod K Bhalla. Financial time series forecasting: comparison of neural networks and arch models. *International Research Journal of Finance and Economics*, 49:185–202, 2010.
- [15] R Glen Donaldson and Mark Kamstra. An artificial neural network-GARCH model for international stock return volatility. *Journal of Empirical Finance*, 4(1):17–46, 1997.
- [16] Robert F Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation. *Econometrica: Journal of the econometric society*, pages 987–1007, 1982.
- [17] Robert F Engle and Simone Manganelli. Caviar: Conditional autoregressive value at risk by regression quantiles. *Journal of business & economic statistics*, 22(4):367–381, 2004.
- [18] Richard H Gerlach, Cathy WS Chen, and Nancy YC Chan. Bayesian time-varying quantile forecasting for value-at-risk in financial markets. *Journal of business & economic statistics*, 29(4):481–492, 2011.
- [19] Trevor Hastie, Robert Tibshirani, Jerome Friedman, et al. *The elements of statistical learning*, 2009.

- [20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- [21] Harold Edwin Hurst. Long-term storage capacity of reservoirs. *Transactions of the American society of civil engineers*, 116(1):770–799, 1951.
- [22] Matteo Iacopini, Francesco Ravazzolo, and Luca Rossini. Bayesian multivariate quantile regression with alternative time-varying volatility specifications. *arXiv preprint arXiv:2211.16121*, 2022.
- [23] Suhwan Ji, Jongmin Kim, and Hyeonseung Im. A comparative study of Bitcoin price prediction using deep learning. *Mathematics*, 7(10):898, 2019.
- [24] Cuixia Jiang, Ming Jiang, Qifa Xu, and Xue Huang. Expectile regression neural network model with applications. *Neurocomputing*, 247:73–86, 2017.
- [25] Philippe Jorion. Value at risk: the new benchmark for controlling market risk. (*No Title*), 1997.
- [26] Angelos Kanas. Neural network linear forecasts for stock returns. *International Journal of Finance & Economics*, 6(3):245–254, 2001.
- [27] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015. URL <https://arxiv.org/abs/1412.6980>.
- [28] Roger Koenker and Gilbert Bassett Jr. Regression quantiles. *Econometrica: journal of the Econometric Society*, pages 33–50, 1978.
- [29] Rajeev Ranjan Kumar, Girish Kumar Jha, Dwijesh C Mishra, and Neeraj Budhlakoti. Package ‘sbagm’. 2025.
- [30] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
- [31] James B McDonald and Yexiao J Xu. A generalization of the Beta distribution with applications. *Journal of Econometrics*, 66(1-2):133–152, 1995.
- [32] I Nasirtafreshi. Forecasting cryptocurrency prices using recurrent neural network and long short-term memory. *Data & Knowledge Engineering*, 139:102009, 2022.
- [33] Michael Parkinson. The extreme value method for estimating the variance of the rate of return. *Journal of business*, pages 61–65, 1980.
- [34] Andrew Phillip, Jennifer SK Chan, and Shelton Peiris. A new look at cryptocurrencies. *Economics letters*, 163:6–9, 2018.
- [35] 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):1794, 2021.
- [36] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- [37] Hayden Savage and Jennifer Chan. Bayesian analyses of the two-stage carr-return models with applications to covid-19 impact on the cryptocurrency market. *Available at SSRN*, 2025.
- [38] Christian Schittenkopf, Georg Dorffner, and Engelbert J Dockner. Forecasting time-dependent conditional densities: a semi non-parametric neural network approach. *Journal of Forecasting*, 19(4):355–374, 2000.
- [39] Tamer Shahwan and Raed Said. A comparison of Bayesian methods and artificial neural networks for forecasting chaotic financial time series. *Journal of Statistics Applications & Probability*, 1(2):89, 2012.
- [40] Donald F Specht et al. A general regression neural network. *IEEE transactions on neural networks*, 2(6):568–576, 1991.

- [41] Shay-Kee Tan, Jennifer So-Kuen Chan, and Kok-Haur Ng. On the speculative nature of cryptocurrencies: A study on garman and klass volatility measure. *Finance Research Letters*, 32:101075, 2020.
- [42] Shay Kee Tan, Jennifer So Kuen Chan, and Kok Haur Ng. Modelling and forecasting stock volatility and return: A new approach based on quantile rogers–satchell volatility measure with asymmetric bilinear carr model. *Studies in Nonlinear Dynamics & Econometrics*, 26(3):437–474, 2022.
- [43] James W Taylor. Forecasting value at risk and expected shortfall using a semiparametric approach based on the asymmetric laplace distribution. *Journal of Business & Economic Statistics*, 37(1):121–133, 2019.
- [44] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.
- [45] Steven YK Wong, Jennifer SK Chan, and Lamiae Azizi. Quantifying neural network uncertainty under volatility clustering. *Neurocomputing*, 614:128816, 2025.
- [46] Qifa Xu, Xi Liu, Cuixia Jiang, and Keming Yu. Quantile autoregression neural network model with applications to evaluating value at risk. *Applied Soft Computing*, 49:1–12, 2016.
- [47] G Peter Zhang. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50: 159–175, 2003.