Erasmus University Rotterdam
Erasmus School of Economics
Bachelor Thesis Econometrics & Operations Research

Integrating Neural Networks and Metaheuristics for Volatility Forecasting: A Hybrid Approach

Finn van der Knaap (573834)

Abstract

Volatility forecasting is crucial to any participant in the financial market, as precise forecasts are essential for financial decision-making and risk management. This paper investigates the application of machine learning (ML) and metaheuristic approaches in order to improve forecasting accuracy. We consider combining either particle swarm optimization or a genetic algorithm with neural networks (NNs), as metaheuristic algorithms offer powerful optimization opportunities in the context of parameter optimization. To be precise, we construct hybrid approaches which implement metaheuristics to optimize the weights of a NN further and compare these models to several NN algorithms and various heterogeneous autoregressive models. Our dataset consists of 25 of the 30 Dow Jones Industrial Average constituents from January 29, 2001, to December 31, 2021. We evaluate the forecasting performance using both the mean squared error and the quasi-likelihood loss function. The findings highlight the possibilities of hybrid models for the task at hand, as the hybrid models beat their counterparts by 5-10%. However, performance is highly dependent on the data period, as performance gain is only sometimes present. In addition, we discern that the hybrid model struggle with high-dimensional input, indicating that perhaps feature selection is needed to overcome this limitation.

Supervisor:	Dr. Onno Kleen
Second assessor:	Max Welz
Date final version:	July 2, 2023

1 Introduction

Financial markets exhibit the phenomenon of volatility clustering (Bollerslev, 1986), implying that large changes in returns are often followed by similar fluctuations and small changes in returns by small fluctuations. Accurately predicting these fluctuations is of utmost importance to investors, as volatility forecasts play a pivotal role in many applications, such as asset pricing and risk management, thus highlighting the importance of obtaining precise forecasts. Most statistical models, such as the heterogeneous autoregressive (HAR) model proposed by Corsi (2009), only use past information to predict volatility, even though Ederington and Lee (1993) show that news and volatility are strongly correlated. However, statistical models often rely upon linear regressions, which are more prone to produce spurious results when extra forward-looking variables are added. Another direction one could explore is the use of machine learning (ML) approaches, which achieve promising results in a variety of tasks (Rahimikia & Poon, 2020; Petrozziello et al., 2022).

Christensen, Siggaard and Veliyev (2022) conduct an extensive out-of-sample analysis where the authors compare numerous methods for volatility forecasting, making it a paper to build upon. The authors investigate the performance of many baseline ML methods, leaving room to delve into the performance of more sophisticated models. In addition, Christensen et al. (2022) do not extensively optimize the parameters of the considered models due to capacity limitations, which motivates the use of hybrid models in this paper. Hybrid models combine different approaches, such as metaheuristics and ML models, to improve performance. Metaheuristics are strategies or algorithms that guide a solution to a near-optimal solution in the search space and are among others used for stock price prediction (Göçken, Özçalıcı, Boru & Dosdoğru, 2016) and portfolio optimization (Doering, Kizys, Juan, Fito & Polat, 2019). The research question is formulated as follows:

To what extent does the accuracy of volatility forecasts improve from utilizing metaheuristics for parameter optimization of neural networks?

In this paper, inspired by the work of Christensen et al. (2022), we implement some of the most well-known methods in the field of volatility forecasting and compare them against several models which utilize hybrid models to optimize the weights of neural networks (NN). In more detail, we compare the HAR model and its extensions against several variants of NN approaches. Moreover, to mitigate a potential weight optimization problem, we propose using metaheuristics, namely a genetic algorithm (GA) and particle swarm optimization (PSO), to optimize the weights of the NNs further. To examine the effect of hybrid models, we use the mean squared error (MSE) and quasi-likelihood (QLIKE) measures to evaluate predictions and compare the performance of hybrid models against the aforementioned models. We base our main analysis on the intraday returns of 25 of the 30 Dow Jones Industrial Average (DIJA) constituents from January 29, 2001, to December 31, 2017, and further extend this analysis to December 31, 2021. Extending the data period allows us to analyze the performance of the above-mentioned models during the covid crisis.

The main contributions of this paper are as follows. First, we investigate the performance of two different metaheuristics for NN optimization in the framework of volatility forecasting. The first approach combines NNs and PSO to optimize the weights of NNs. The optimized weights

by means of backpropagation are inputted into the PSO algorithm, enabling us to implement an off-the-shelf (re-estimating the weights by incorporating new information) approach of a NN whilst keeping computational costs relatively cheap. The second approach mimics the first approach but implements a GA instead of PSO, allowing us also to compare the two hybrid approaches.

Second, we investigate the performance of ML approaches compared to various HAR models. We build upon Christensen et al. (2022) and extend their dataset to 2021, thus structuring the in-sample and out-of-sample datasets differently, enabling us to qualitatively compare the impact of using different data periods.

The rest of the paper is structured as follows. First, Section 2 presents an overview of previous literature regarding volatility forecasting and the application of metaheuristics. Then, Section 3 presents the proposed methods in this thesis, followed by, in Section 4, an overview of the used data. Next, Section 5 discusses the forecasting results. Last, Section 6 presents limitations and concluding remarks.

2 Literature

This section discusses previous literature regarding volatility forecasting, ML approaches, and the use of metaheuristics. First, Section 2.1 gives an overview of volatility forecasting and especially ML approaches in this framework. Then, in Section 2.2, we discuss metaheuristics and their applications in the financial domain.

2.1 Volatility Forecasting and Machine Learning

Corsi (2009) laid the foundation for the concept of the by now well-known HAR model, which, in contrast to previously proposed ARCH-type models, is able to capture the main empirical features of financial returns. As the name suggests, it is an autoregressive model, using the realized variance measured at numerous frequencies to predict volatility, and is able to partially capture the heterogeneity of the data. Due to the promising performance of the HAR model, it is seen as a benchmark, taking over the role of the by Bollerslev (1986) introduced GARCH(1,1) model. However, due to its parsimonious structure and the complicated structure of financial data, it is inadequate in certain situations. Therefore, many extensions of the baseline HAR model try to tackle its shortcomings. Examples of such extensions use a leverage effect to improve predictions (Corsi & Renò, 2012), allow for variation in negative and positive returns (Patton & Sheppard, 2015), let the parameters vary explicitly with the degree of measurement error, (Bollerslev, Patton & Quaedvlieg, 2016), or combine ARCH-type models and HAR models to characterize time-varying volatility in realized variance (Qu, Duan & Niu, 2018).

Yet, all of the aforementioned models utilize information regarding past returns, which might not be optimal, as previous literature has shown that news and volatility are often heavily correlated (Ederington & Lee, 1993; Jiang, Konstantinidi & Skiadopoulos, 2012; Bollerslev, Li & Xue, 2018), indicating that perhaps just using past information is inadequate for volatility forecasting. A supposedly straightforward solution is to add additional covariates to a model to capture realized variance to a greater extent. However, the previously mentioned models

rely on linear regressions, which are prone to break down or produce spurious results when the explanatory variables are strongly correlated, have a low signal-to-noise ratio, or have a nonlinear relationship. Nonetheless, with the growing amount of available data, it is tempting to seek models which can handle the limitations of these more statistical methods.

As Varian (2014) shows, ML approaches are a solution to the above problem, and at the time of writing this paper, also thoroughly investigated in previous literature (Zhou, Pan, Wang & Vasilakos, 2017; Kolisetty & Rajput, 2020). In the framework of volatility forecasting, various ML techniques have been investigated and compared against the more conventional models (e.g., ARCH-type models). For example, Liu (2019) examines the performance of Support Vector Machines (SVM) and NNs, showing that these approaches outperform the GARCH model. Gavrishchaka and Banerjee (2006) propose to solely use SVMs to predict the S&P500 index, finding that this technique captures long-memory volatility and is often superior to conventional methods. Furthermore, Audrino and Colangelo (2010), Mittnik, Robinzonov and Spindler (2015), and Döpke, Fritsche and Pierdzioch (2017) make use of several regression trees to forecast volatility.

In contrast to the literature mentioned above, which mainly uses a single approach, Christensen et al. (2022) conduct an extensive out-of-sample analysis, comparing many of the above-mentioned models, such as HAR-type models, regression trees, and NNs. They aim not only to investigate the performance of the considered models but also to understand why ML approaches improve the accuracy of predictions. Christensen et al. (2022) find that ML approaches improve out-of-sample forecasts and work better with the nonlinear structure of financial markets, especially NNs and regression trees, where NNs perform better at shorter horizons and regression trees at longer horizons. In more detail, even though nonlinearity, a feature NNs incorporate easily, remains essential for longer horizons, functionality and interaction effects, something regression trees trump NNs at, become vital for longer horizon predictions.

To this end, we take inspiration from Christensen et al. (2022) and compare numerous HAR models and ML approaches for volatility forecasting. However, we propose the use of more sophisticated hybrid models, instead of relatively simplistic ML methods. In Christensen et al. (2022), the weights of the NNs are not extensively tuned due to computational capacity problems, which motivates the use of metaheuristics in this paper. Metaheuristics are, as Blum and Roli (2003) show, likely to find a feasible solution in less computation time when dealing with limited capacity. Furthermore, compared to backpropagation, metaheuristics, such as PSO (Kennedy & Eberhart, 1995) and GAs (Holland, 1992), have a fast convergence rate and often provide near-global optimal solutions, therefore overcoming possible limitations of backpropagation (Pradeepkumar & Ravi, 2017).

2.2 Metaheuristics in the Financial Domain

Metaheuristics are strategies or algorithms that try to efficiently guide the input to a near-optimal solution, using a trade-off of local and global exploration (Gandomi, Yang, Talatahari & Alavi, n.d.). Many of such approaches use nature as the backbone of their algorithm (Yang, 2010a). For example, Dorigo and Stützle (2003) introduce ant colony optimization, an algorithm based on the behavior of ants and their colony. Moreover, Yang (2010b) introduces a new

algorithm based on the echolocation behavior of bats. In this thesis, we implement adaptations of two of the most well-known metaheuristics, namely PSO (Kennedy & Eberhart, 1995) and a GA (Holland, 1992), as they often achieve state-of-the-art results (Zhu, Wang, Wang & Chen, 2011; Chung & Shin, 2018). Both metaheuristics are also based on natural phenomena, as PSO is based on the social behavior of birds in a flock, and GAs are based on natural selection. Metaheuristics are used for a wide range of applications. For example, Luo et al. (2018) apply metaheuristics for financial stress prediction, whereas Zivkovic et al. (2021) utilize metaheuristics to predict covid-19 cases.

The implementation of metaheuristics in the domain of finance and ML is not a new concept. For example, Göçken et al. (2016) implement metaheuristics to determine the optimal structure and input variables of a NN for stock price prediction. Ghasemiyeh, Moghdani and Sana (2017) focus on optimizing the weights of a NN through metaheuristic optimization in the framework of stock price prediction. Combining both of the above-mentioned literature, Shahvaroughi Farahani and Razavi Hajiagha (2021) use metaheuristics to select technical indicators and to optimize a NN its weights. Yet, all of the aforementioned works focus on stock price prediction. In the framework of volatility forecasting, research regarding the fusion of NNs and metaheuristics is scarce, and mostly focused on optimizing the corresponding hyperparameters (Ribeiro, Santos, Mariani & dos Santos Coelho, 2021; Ji, Liew & Yang, 2021). Therefore, this thesis focuses on the implementation of metaheuristics for the optimization of the weights of a NN, overcoming possible limitations of a backpropagation approach.

3 Methodology

This section contains a description of the models we use. First, Section 3.1 presents the setting for forecasting volatility. Next, Section 3.2 discusses the HAR model and its considered extension. Then, Section 3.3 goes in depth about ML approaches, followed by, in Section 3.4, a detailed overview of the metaheuristics for parameter optimization. Last, in Section 3.5, we discuss the forecast evaluation measures.

3.1 Realized Volatility

Let the log-price $X = (X_t)_{t\geq 0}$ be supported by a filtered probability space $(\Omega, (\mathcal{F}_t)_{t\geq 0}, \mathcal{F}, \mathbb{P})$. Then, X is a semimartingale process if the price is determined in an arbitrage-free frictionless market, and X_t is defined as follows:

$$X_t = X_0 + \int_0^t \mu_s ds + \int_0^t \sigma_s dW_s + \sum_{s=1}^{N_t} J_s, t \geq 0, \tag{1}$$

where X_0 is $\mathcal{F}_0$-measurable, $u = (u_t)_{t\geq 0}$ is a drift term, $\sigma = (\sigma_t)_{t\geq 0}$ denotes the stochastic volatility process, $W = (W_t)_{t\geq 0}$ is a standard Brownian motion, $N = (N_t)_{t\geq 0}$ denotes a counting process, which represents the number of jumps in X, and $J = (J_s)_{s=1,\dots,N_t}$ is a series of nonzero random variables of jump sizes with jump times $\tau = (\tau_s)_{s=1,\dots,N_t}$. In this paper, we aim to

predict the daily quadratic variation, which is defined as follows:

$$\mathrm{QV}_t = \int_{t-1}^{t} \sigma_s^2 ds + \sum_{t-1 \leq \tau_s \leq t} J_s^2, \text{ for } t = 1, ..., T, \tag{2}$$

where t is the predicted observation, and T is the total number of days in the sample. However, in practice, the quadratic variance is not observable, which motivates the use of the realized variance as an estimator of the quadratic variance, which is defined as follows:

$$\mathrm{RV}_t = \sum_{j=1}^{n} |\Delta_{t-1,j}^n X|^2, \tag{3}$$

where n is the number of intraday returns, and $\Delta_{t-1,j}^n X = X_{t-1+\frac{j}{n}} - X_{t-1+\frac{j-1}{n}}$. We opt to use the realized variance as an estimator, as $\mathrm{RV}_t \xrightarrow{\mathbb{P}} \mathrm{QV}_t$ when $n \to \infty$ (Barndorff-Nielsen & Shephard, 2002). It is thus a consistent estimator of the quadratic variance when n increases.

3.2 HAR Models

As a benchmark model, we consider the baseline HAR model proposed by Corsi (2009), which uses past realized variance proxies computed at different frequencies to predict the present volatility. The baseline HAR model is defined as follows:

$$\mathrm{RV}_t = \beta_0 + \beta_1 \mathrm{RV}_{t-1} + \beta_2 \mathrm{RV}_{t-1|t-5} + \beta_3 \mathrm{RV}_{t-1|t-22} + \epsilon_t, \tag{4}$$

where $\mathrm{RV}_{t-1|t-h} = \frac{1}{h} \sum_{i=1}^{h} RV_{t-h}$ and ϵ_t is an error term, implying that we predict volatility at time t using proxies of the average daily, weekly, and monthly lagged realized variance.

In the same paper, Corsi (2009) also proposed a logarithmic version of HAR, denoted by logHAR, allowing for a nonlinear relationship between the dependent and explanatory variables. The logHAR model is defined as follows:

$$\log(\mathrm{RV}_t) = \beta_0 + \beta_1 \log(\mathrm{RV}_{t-1}) + \beta_2 \log(\mathrm{RV}_{t-1|t-5}) + \beta_3 \log(\mathrm{RV}_{t-1|t-22}) + \epsilon_t. \tag{5}$$

The logHAR model, however, produces forecasts of log-realized variance. To obtain forecasts of the realized variance, we need to apply a nonlinear transformation, which implies that the realized variance forecasts are biased by Jensen's inequality (Jensen, 1906). We, therefore, bias the predictions as follows:

$$\mathbb{E}[\widehat{\mathrm{RV}}_t] = \exp\left(\mathbb{E}[\log(\widehat{\mathrm{RV}}_t)] + 0.5\mathrm{var}[\log(\widehat{\mathrm{RV}}_t)]\right), \tag{6}$$

where $\mathrm{var}[\log(\widehat{\mathrm{RV}}_t)]$ is the variance of the residuals in the training and validation set. This bias is applicable when the distribution of log-realized variance is Gaussian, which is approximately true in practice (see Andersen, Bollerslev, Diebold and Ebens (2001)).

Besides the above three lagged realized variance explanatory variables, we consider a broader selection of variables, as explained in Section 4. To ensure compatibility with the NNs, we construct distributed lag-type versions of both HAR models. HAR-X is denoted as the distributed

lag-type version for the baseline HAR model, and logHAR-X is denoted as the distributed lag-type version of the logHAR model. We estimate all HAR models by means of Ordinary Least Squares, thus minimizing the sum of squared errors.

3.3 Neural Networks

Moving onto ML approaches, NNs are a subset of ML algorithms that mimic the way the human brain operates. Due to their nonlinear and flexible structure, NNs are extensively investigated in literature and show promising results, including in the area of volatility forecasting (Christensen et al., 2022).

NNs are compromised of several node layers, containing an input layer, hidden layers, and an output layer. First, the NN receives an input Z_t in the input layer. Then, the data is transformed through numerous hidden layers through an activation function g, which eventually produces the desired output at the output layer. In general, the lth layer in a NN is defined as follows:

$$a_t^{\theta_{l+1},b_{l+1}} = g_l\left(\sum_{j=1}^{N_l} \theta_j^{(l)} a_t^{\theta_l,b_l} + b^{(l)}\right), \quad 1 \leq l \leq L, \tag{7}$$

where L is the total number of layers, g_l is the activation function, $\theta^{(l)}$ is the weight matrix, $b^{(l)}$ is the bias, N_l is the number of hidden neurons, and $a_t^{\theta_{l+1},b_{l+1}}$ is the predicted value.

The downside of the flexibility of a NN is the number of options for its structure, which depends on the problem at hand and is determined through hypertuning. In line with Christensen et al. (2022), we construct four models which are inspired by the geometric pyramid. NN_1 has a single hidden layer accompanied by two neurons. Then, NN_2 is two-layered with four and two neurons, respectively, and NN_3 has three hidden layers with eight, four, and two neurons, respectively. Last, NN_4 is four-layered with sixteen, eight, four, and two neurons, respectively. We illustrate the NN_2 structure in Figure 1.

As an activation function, which adds non-linearity to the model, we opt to use the Leaky Rectified Linear Unit (L-ReLU) from Maas, Hannun, Ng et al. (2013), which is defined as follows:

$$\text{L-ReLU}(x) = \begin{cases} cx, & \text{if } x < 0, \\ x, & \text{otherwise,} \end{cases} \tag{8}$$

where $c \geq 0$[1]. Its base version (ReLU) can result in dead neurons (Lu, Shin, Su & Karniadakis, 2019), something L-ReLU attempts to fix by having a small negative slope (c). We employ Adaptive Moment Estimation (ADAM) (Kingma & Ba, 2014) as optimizer with default hyperparameters and initialize the weights of a NN using the glorot normal distribution from Glorot and Bengio (2010), remaining in line with Christensen et al. (2022). We train the NNs through backpropagation using 500 epochs and use the MSE as the corresponding loss function.

A rather often occurring problem with NNs is overfitting, which implies that the model fits the training data exactly but performs modestly on the test data. Besides employing a validation set, regularization is performed to overcome the issue of overfitting. In this work,

[1] We set $c = 0.1$.

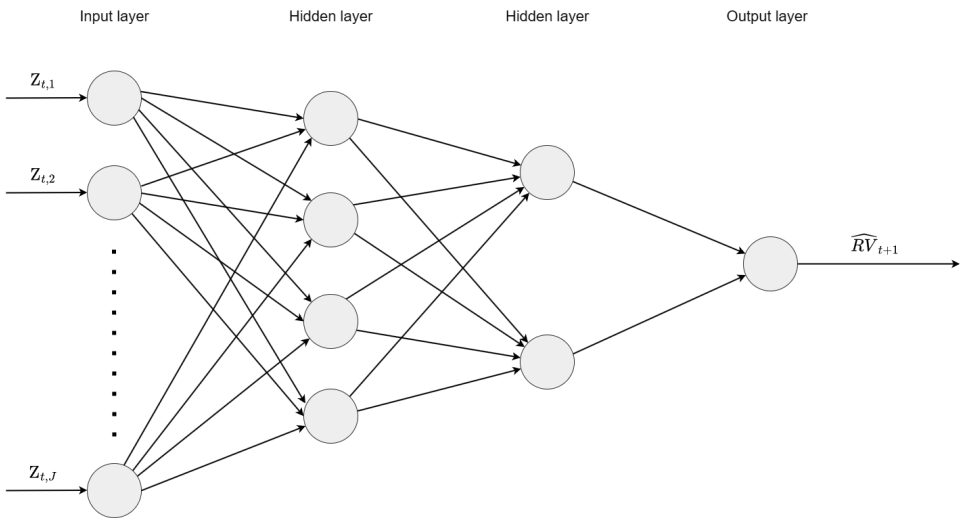


Figure 1: A two-layered feed-forward NN.

Notes: We illustrate the NN_2 structure described in this section. The input layer receives data Z_t with J explanatory variables. Each hidden layer transforms the input through the corresponding activation function. The output layer returns the prediction.

we employ four of the most well-known regularization techniques: dropout, ensemble, learning rate decay, and early stopping. Ensemble refers to combining predictions from different models, which in this case comes down to implementing various NNs with different random seeds. We combine the predictions based on the results of the validation set. We explore the performance of ensembles of 1, corresponding to no averaging, and 10 out of 100, which is denoted as NN_1^1 and NN_1^{10} for NN_1, respectively. Next, dropout is a reduction technique that randomly drops out nodes during training and is set to 0.9. Then, early stopping implies we stop the training of a NN if the validation set MSE does not improve over a given number of epochs; in this case, early stopping its patience is set to 100. Learning rate decay decreases the learning rate after a specified number of iterations and is used to improve convergence. We employ step-wise learning rate decay, implying that the learning rate is reduced by a fixed factor, 0.5, every 50 epochs.

The hyperparameter choice is partially in line with existing literature and partially set by trial and error. In this work, we opt to use two different datasets ($\mathcal{M}_{\text{HAR}}$ and $\mathcal{M}_{\text{ALL}}$), one with a substantially higher dimension, thus leading to a higher dimensional NN. As employing learning rate decay only seemed beneficial for the NNs with the input from $\mathcal{M}_{\text{ALL}}$, we do not employ learning rate decay for the $\mathcal{M}_{\text{HAR}}$ dataset. Without learning rate decay, loss values were exploding in some instances, thus motivating the use of a learning rate decay for $\mathcal{M}_{\text{ALL}}$. All other regularization techniques are employed for all frameworks. An overview of hyperparameter choice can be found in Table 5 in the Appendix.

3.4 Metaheuristics

To further improve performance and avoid entrapment in local minima, we apply metaheuristics to optimize the weights and biases of a NN, which efficiently explore the search space in order to find a near-optimal solution. First, Section 3.4.1 gives an overview of PSO, followed by, in Section 3.4.2, the explanation of the GA. Last, Section 3.4.3 discusses the implementation of

the considered metaheuristics in the framework of volatility forecasting.

3.4.1 Particle Swarm Optimization

PSO, introduced by Kennedy and Eberhart (1995), is an algorithm based on the behavior of birds in their flock. Particles in a swarm move around in a search space and determine their best position based on their current position and velocity. In the task at hand, a current position is a list containing the weights and biases of a NN. The best positions of the swarm move it to an optimal solution.

The algorithm starts with initializing the particles in a swarm, each particle receiving a current position and velocity. In order to transfer information from the backpropagation algorithm, we initialize the current position of all the particles uniformly near the optimal weights found by the NN. Then, in each iteration, the fitness of the particles is calculated, which in this framework is the in-sample MSE. With in-sample, we refer to the MSE over the specified training set. If a particle's current position is better than its best position, we update the best position to its current position. In addition, if the current position is better than the global best-known position, we also update the global best-known position.

The current position and velocity are defined as follows:

$$\mathrm{CP}_{i,t} = \mathrm{CP}_{i,t-1} + \mathrm{V}_{i,t}, \tag{9}$$

$$\mathrm{V}_{i,t} = w\mathrm{V}_{i,t-1} + c_1 r_1(\mathrm{PBP}_i - \mathrm{CP}_{i,t-1}) + c_2 r_2(\mathrm{GBP} - \mathrm{CP}_{i,t-1}), \tag{10}$$

where $\mathrm{CP}_{i,t}$ is the current position and $\mathrm{V}_{i,t}$ is the velocity of the ith particle at iteration t, w is the inertia weight, which affects the local and global search, PBP_i is the personal best position of particle i, GBP is the global best position of the swarm, c_1 and c_2 influence the effect of the personal best and global best position, and r_1 and r_2 are randomly drawn variables between 0 and 1. To allow for a dynamic search, we use an adaptive inertia weight, local influence, and global influence. Furthermore, to avoid non-convergence, we apply a maximum volatility value, which sets any volatility value larger than the specified maximum to the maximum. A detailed overview of hyperparameter choice is displayed in Table 5 in the Appendix, and further information about the implementation can be found in Section 3.4.3. In addition, Algorithm 1 shows the pseudocode of PSO.

3.4.2 Genetic Algorithm

Holland (1992) first introduces GAs, which are heuristic search algorithms based on the idea of natural selection and genetics. The evolutionary process consists of mutation, crossover, and selection. In the algorithm, the population and its individuals (chromosomes) consist of the weights of the corresponding NN, where each individual represents one variation of these weights.

We initialize the algorithm similarly to PSO, such that the population is uniformly initialized around the optimal solution found by backpropagation. Then, in each iteration, the population is updated, partially with information from the old population and partially with new chromosomes. First, the best n (based on an elitism rate) performing individuals based on the fitness

Algorithm 1: Particle Swarm Optimization

Initiate: w, w_p, c_1, c_2, c_{1p}, c_{2p}, r_1, $r_2 \leftarrow$ set hyperparameters
n_iterations, *n_particles* $\leftarrow$ set hyperparameters

Input: A list containing the weights and biases
X_{in}, $\mathrm{y}_{in} \leftarrow$ Input data to calculate performance

Output: The optimized list with weights and biases

InitializeParticles(X_{in},y_{in}) `// Initialize particles around optimal position`
InitializeVelocity(X_{in},y_{in}) `// Initialize velocity around 0`
for t **in** range(*n_iterations*) **do**
  $w = w - w_p$ `// Update inertia weight each iteration`
  $c_1 = c_1 + c_{1p}$ `// Update local influence each iteration`
  $c_2 = c_2 + c_{2p}$ `// Update global influence each iteration`
  for i **in** range(*n_particles*) **do**
    `// Update velocity`
    $\mathrm{V}_{i,t} = w\mathrm{V}_{i,t-1} + c_1 r_1(\mathrm{PBP}_i - \mathrm{CP}_{i,t-1}) + c_2 r_2(\mathrm{GBP} - \mathrm{CP}_{i,t-1})$
    `// Check maximum velocity rule`
    CheckVolatility($\mathrm{V}_{i,t}$)
    `// Update current position`
    $\mathrm{CP}_{i,t} = \mathrm{CP}_{i,t-1} + \mathrm{V}_{i,t}$
    `// Check whether the current position is better than the global best position or its current best position`
    `// Update these positions if that is true`
    CheckBest($\mathrm{CP}_{i,t}$, i, X_{in},y_{in})
  `// Update weight of the NN to the global best-known position`
  SetWeight(GBP)

score are transferred over to the new population, where we again examine the training set MSE. Then, we perform crossover on these parent chromosomes, or best n performing individuals, which implies that we select two individuals, randomly select one layer of weights or biases from the first individual, and switch this with the latter individual, resulting in a new chromosome. Whilst one could also opt to perform crossover using a single weight or bias, swapping a whole layer could increase diversity. As swapping weight matrices has a more considerable effect than swapping just a bias vector, we pick weight matrices with 80%. Last, a mutation operator selects part of the new population from the crossover operator, which is exposed to mutation using a mutation probability. The mutation operator works in a similar fashion compared to the crossover operator in the sense that we expose a whole weight matrix or bias vector (a whole layer of weights or biases) to mutation. A detailed overview of hyperparameter choice can be found in Table 5 in the Appendix. Algorithm 2 shows the pseudocode of the GA.

3.4.3 Implementation and Hyperparameter Choice

Due to the results from Christensen et al. (2022), we only consider the NN_2 and NN_3 formulation for both metaheuristics. A two-layered NN without ensemble (NN_2^1) in combination with PSO is

Algorithm 2: Genetic Algorithm

Initiate: *n_highest, mutation_probability* ← set hyperparameters
n_iterations, n_chromosomes ← set hyperparameters
Input: A list containing the weights and biases
X_{in}, y_{in} ← Input data to calculate performance
Output: The optimized list with weights and biases

InitializeChromosomes(X_{in},y_{in}) `// Initialize chromosomes around optimal position`
for t **in** range(*n_iterations*) **do**
    mse_ranked ← {} `// Create empty dictionary for loss values`
    for i **in** range(*n_chromosomes*) **do**
        `// Calculate fitness score and add to dictionary`
        mse_i ← **FitnessScore**(i,X_{in},y_{in})
        mse_ranked[i] = mse_i
    `// Sort` *mse_ranked* `based on loss function and choose` *n_highest* `for next population, and add these to the new population`
    SortOnLoss(*mse_ranked*)
    new_population_n_highest ← **HighestN**(*mse_ranked*,*n_highest*)
    `// Fill remaining population using the crossover operator and the parent individuals, which later are exposed to a mutation operator`
    new_population_rest ← **Crossover**(*new_population_n_highest*)
    Mutation(*new_population_rest*)

referred to as PSO_2^1. After obtaining the optimal weights from backpropagation, we implement either PSO or GA to optimize the NN further. To be precise, the input of either metaheuristic is the obtained weights through backpropagation. Then, we randomly initialize the particles or chromosomes near the input and start iterating with the objective to minimize the training set MSE. Similarly to the NNs, we employ an ensemble, implying that we again explore the performance of 1 and 10 ensembles out of 100, where each ensemble is evaluated based on the validation set MSE.

Whilst having a lower computational time compared to NNs, running the algorithms for all considered stocks still takes time. One could perform a grid search for each stock to find the optimal hyperparameters, but this would exceed the available resources in this study. We opt for hyperparameters partially in line with existing literature and partially found by trial and error. This is, however, only in favor of the other considered methods. Table 5 in the Appendix shows a detailed overview of hyperparameter choice.

Starting with PSO, We use an adaptive inertia weight, decreasing exploration and increasing exploitation throughout the iterations. We decrease w by 0.02 every iteration. Then, local influence decreases, whereas we increase global influence after an iteration, implying that we support exploration in the first iterations and slowly move all particles towards the best global position throughout the iterations. We set *init_vel* to 3, implying that we initialize a particle's velocity uniformly between 1 and 3. To avoid the non-convergence and explosion of a particle's velocity, we set the maximum velocity to 15. For both metaheuristics, we set *init_particles* and

init_chromosomes to 3, implying that we add an uniformly drawn random number between -3 to 3 to every weight or bias.

For both algorithms, we let the same number of individuals explore the search space, whereas we increase the number of iterations for GA by 10 as the convergence rate of the GA is often slower than PSO (Panda & Padhy, 2008; Li, Liu, Duan & Huang, 2010). We specifically choose to use a modest number of iterations in order to reduce overfitting. We select the 30% best-performing individuals for the upcoming population for the GA. Then, *crossover_choice* = 0.8 implies that we select a layer of weights with 80% probability for the crossover operator, whereas we select a layer of biases with 20% probability. Moving onto the mutation operator, we expose an individual to mutation with a probability of 70%, allowing for more exploration in the population. Like the crossover operator, we select a layer of weights with 80% probability, which is subject to mutation. Each weight or bias in the layer is randomly mutated between -1 and 1.

3.5 Forecast Evaluation

To evaluate the obtained prediction, we opt to use the MSE and QLIKE as the out-of-sample [2] measures, which are defined as follows:

$$\text{MSE}_i = \frac{1}{T_o} \sum_{t \in \text{oos}} (\text{RV}_{t+1} - \widehat{\text{RV}}_{i,t+1})^2, \tag{11}$$

$$\text{QLIKE}_i = \frac{1}{T_o} \sum_{t \in \text{oos}} (\log(\widehat{\text{RV}}_{i,t+1}) + \frac{\text{RV}_{t+1}}{\widehat{\text{RV}}_{i,t+1}}), \tag{12}$$

where T_o is the number of observations in the out-of-sample dataset, RV_{t+1} the actual realized variance at time $t+1$, and $\widehat{\text{RV}}_{i,t+1}$ is the predicted realized variance of model i. To compare the results for different models across numerous stocks, we report the relative MSE, which is a cross-sectional average of the relative MSEs per stock: $\frac{1}{N} \sum_{j=1}^{N} \frac{\text{MSE}_{i,j}}{\text{MSE}_{b,j}}$, where N is the number of considered stocks, $\text{MSE}_{i,j}$ is the MSE of model i for stock j, and $\text{MSE}_{b,j}$ is the benchmark model its MSE for stock j.

If a prediction is negative, we replace it with the minimum in-sample realized variance. In addition, we construct the Model Confidence Set (MCS) of Hansen, Lunde and Nason (2011), which constructs a collection of models containing the best performing one given a pre-specified confidence level of either 75% or 90%.

4 Data

In our research, we consider high-frequency data from 25 of the 30 DIJA constituents, disregarding Visa, JPMorgan Chase, UnitedHealth, Procter & Gamble, and Dow Chemical due to limited data availability. To be precise, the following tickers are considered in this study: AAPL, AXP, BA, CAT, CSCO, CVX, DIS, GE, GS, HD, IBM, INTC, JNJ, KO, MCD, MMM, MRK, MSFT, NKE, PFE, RTX, TRV, VZ, WMT, and XOM. Partially in line with Christensen et al. (2022),

[2] oos is used as an abbreviation for out-of-sample in Equation (11) and Equation (12).

we consider a sample period from January 29, 2001, to December 31, 2017, and extend this analysis until December 31, 2021, which includes highly-volatile periods, such as the covid crisis. This results in a total of $T = 5264$ observations. The high-frequency data is a subset of the data in Kleen and Tetereva (2022), and the thesis supervisor provided it. Alike Bollerslev, Li and Zhao (2020), we merge daily Center for Research in Security Prices (CRSP) data with New York Stock Exchange (NYSE) Trade and Quote (TAQ) intraday data. We obtain daily open and close prices from the daily CRSP data files and all other intraday transaction data from the NYSE TAQ. We merge the two data sets via the Wharton Research Data Services (WRDS) linking tables. The intraday data is cleaned according to Barndorff-Nielsen and Shephard (2002), and we include only trades from the exchange that are referenced in the daily CRSP data.

We split the data up into a training set, validation set, and test set, containing 70% (3670 days), 10% (524 days), and 20% (1048 days) of the original data, respectively. Note that 22 observations are lost due to lagged variables. Moreover, the test set consists partially of the covid crisis, making it intriguing to investigate the performance of the aforementioned models during a generally highly-volatile period.

In the HAR models, we merge the training and validation set and allow for time-varying parameters by constructing a rolling window approach consisting of 4194 observations, thus resulting in an off-the-self approach for these models. On the contrary, we employ a fixed estimation window for the NNs (and hybrid models), as the NNs are computationally heavier compared to the HAR models, and we do not have the computational capacity to re-estimate the weights of a NN for every observation. An important note is that this is only beneficial for the HAR models because they incorporate new and potentially valuable information into their framework.

To analyze the effect of adding extra covariates to the model, we construct two datasets, namely $\mathcal{M}_{\text{HAR}}$ and $\mathcal{M}_{\text{ALL}}$. $\mathcal{M}_{\text{HAR}}$ consists of the three explanatory variables in the baseline HAR model: the average daily, weekly, and monthly lagged realized variance. $\mathcal{M}_{\text{ALL}}$ includes and extents $\mathcal{M}_{\text{HAR}}$ by including nine other variables that have been extensively researched by other literature, and arguably been proven to be powerful predictors of volatility. We include five macroeconomic variables: The CBOE volatility (VIX) index, the Hang Seng stock index daily squared log-return (HSI)[3], the Aruoba, Diebold and Scotti (2009) business index conditions (ADS), the first-differenced US 3-month T-bill rate (US3M), and the economic policy uncertainty (EPU) index[4] from Baker, Bloom and Davis (2016). We also include four firm-specific variables: Model-free implied volatility (IV)[5], an indicator for earnings announcements (EAs)[6], 1-week momentum (M1W), and the first-differenced logarithm of dollar trading volume (VOL)[7]. In addition, we log-transform the VIX and IV for the logarithmic HAR model discussed in Section 3.2. Missing observations for all return variables (i.e., HSI) are set to 0, whereas other missing variables are set to the most recent available observation. Table 4 in the Appendix shows the explanatory variables' descriptive statistics.

[3]HSI originates from Yahoo Finance.

[4]US3M and EPU are from the federal reserve bank (FRED) of St. Louis, and ADS from FRED of Philadelphia.

[5]The implied volatility originates from OptionMetrics and is calculated as the average implied volatility of all options with a 30-day expiration date.

[6]EAs are from the U.S. securities and exchange commissions.

[7]Both M1W and VOL originate from CRSP from the WRDS.

5 Results

Table 1 shows the obtained relative out-of-sample MSEs for the dataset $\mathcal{M}_{\text{HAR}}$. Each column reports MSEs for the forecast model relative to the benchmark model in the corresponding row, where the MSE is a cross-sectional average across all considered stocks. Note that the HAR-X is identical to the baseline HAR model, as $\mathcal{M}_{\text{HAR}}$ only consists of lagged realized variances.

Contrary to Christensen et al. (2022), we observe that neither the logHAR nor all NNs outperform the baseline HAR model. To be precise, the cross-sectional MSE of the logHAR is 0.5% above one, whereas all NNs perform even worse. Amongst the NNs, the one-layered NN obtains the worst MSE by far, whereas the difference between the other NN frameworks is minuscule, observing a difference of at most 3%. However, compared to the baseline HAR model, these NNs still obtain a cross-sectional MSE of at least 6.5% above one. Ensembles marginally improve performance for NN2-NN3, but the difference is insignificant. An explanation behind the performance could be the structure of the data split. While Christensen et al. (2022) evaluate their models until 2017, we consider an extended dataset until 2021, thus including the covid crisis, which is a rather highly-volatile period. As the validation set does not necessarily include such an event, overfitting is a likely explanation behind the performance of the NNs. Although we perform regularization, a configuration might fit great on the validation set, a relatively low volatile period, but still perform weak on the test set.

Shifting focus on the metaheuristics, both PSO and GA perform significantly better than their NN counterparts. The GA models beat their counterparts, on average, by 5-10%, whilst this is marginally smaller for the PSO models. This highlights the limitations of backpropagation, as the normal NNs seem find it more challenging to converge. Comparing the hybrid models amongst themselves, performance is alike between different structured NNs incorporated with a metaheuristic, while, in turn, the GA outperforms PSO by 1 to 2 percent. In addition, in line with the NNs, we observe an insignificant difference between ensembles and a single model.

All hybrid models do not seem to outperform the baseline HAR model in this setting, observing an increase of at most 3.8%. Additionally, we perform a robustness check to ascertain whether the results are consistent across different datasets. Therefore, we consider the same data period as in Christensen et al. (2022), from January 29, 2001, to December 31, 2017. Table 9 in the Appendix shows the results of the robustness check.

Table 9 displays an entirely different story. In line with Christensen et al. (2022), all NNs outperform the baseline HAR model, highlighting that nonlinearity does seem to be essential for predicting volatility. However, whilst all hybrid models beat the HAR models, they do not seem to improve the performance of regular NNs. Moreover, performance slightly deteriorates, indicating that the metaheuristic approaches possibly overfit the training set. Furthermore, it indicates that if NNs that use backpropagation quickly converge, the effect of using hybrid models diminishes.

Table 7 in the Appendix displays the results of using the QLIKE results for the $\mathcal{M}_{\text{HAR}}$ dataset. An intriguing contradiction with the MSE results is that all models are highly similar in terms of relative QLIKE value, observing a difference of at most 1.4% compared to the HAR model. For the QLIKE, both the logHAR model and the GA model are able to beat the HAR model, although the difference is a mere 0.1%. An explanation behind the contrasting

Table 1: One-day-ahead relative MSE for dataset $\mathcal{M}_{\text{HAR}}$ over the test set from the entire sample period January 29, 2001, to December 31, 2021

	HAR	HAR-X	logHAR	NN_1^1	NN_1^{10}	NN_2^1	NN_2^{10}	NN_3^1	NN_3^{10}	NN_4^1	NN_4^{10}	PSO_2^1	PSO_2^{10}	PSO_3^1	PSO_3^{10}	GA_2^1	GA_2^{10}	GA_3^1	GA_3^{10}
HAR	-	1.000	1.005	1.172	1.146	1.097	1.087	1.081	1.077	1.065	1.070	1.017	1.038	1.020	1.023	1.007	1.027	1.003	1.009
HAR-X	1.000	-	1.005	1.172	1.146	1.097	1.087	1.081	1.077	1.065	1.070	1.017	1.038	1.020	1.023	1.007	1.027	1.003	1.009
logHAR	1.006	1.006	-	1.183	1.154	1.097	1.088	1.082	1.078	1.065	1.071	1.019	1.041	1.023	1.024	1.008	1.029	1.005	1.010
NN_1^1	0.874	0.874	0.881	-	0.981	0.958	0.950	0.947	0.943	0.932	0.936	0.891	0.908	0.893	0.896	0.883	0.899	0.877	0.883
NN_1^{10}	0.890	0.890	0.896	1.021	-	0.975	0.967	0.963	0.959	0.948	0.952	0.906	0.923	0.909	0.911	0.898	0.915	0.893	0.898
NN_2^1	0.923	0.923	0.923	1.084	1.058	-	0.995	0.991	0.987	0.975	0.982	0.934	0.952	0.937	0.939	0.924	0.941	0.921	0.926
NN_2^{10}	0.927	0.927	0.929	1.091	1.064	1.009	-	0.996	0.992	0.982	0.987	0.939	0.957	0.941	0.944	0.930	0.933	0.926	0.931
NN_3^1	0.933	0.933	0.933	1.098	1.071	1.015	1.007	-	0.998	0.986	0.992	0.944	0.963	0.948	0.949	0.934	0.953	0.931	0.936
NN_3^{10}	0.935	0.935	0.935	1.101	1.073	1.018	1.009	1.004	-	0.989	0.994	0.946	0.965	0.949	0.951	0.936	0.955	0.933	0.938
NN_4^1	0.948	0.948	0.948	1.114	1.087	1.031	1.024	1.018	1.014	-	1.008	0.960	0.979	0.964	0.965	0.949	0.968	0.946	0.952
NN_4^{10}	0.941	0.941	0.941	1.105	1.078	1.025	1.017	1.011	1.007	0.995	-	0.952	0.972	0.956	0.957	0.942	0.961	0.939	0.944
PSO_2^1	0.990	0.990	0.991	1.165	1.136	1.080	1.071	1.065	1.060	1.049	1.053	-	1.023	1.006	1.008	0.992	1.012	0.989	0.994
PSO_2^{10}	0.968	0.968	0.970	1.137	1.109	1.055	1.046	1.041	1.037	1.026	1.031	0.980	-	0.984	0.986	0.971	0.989	0.967	0.972
PSO_3^1	0.985	0.985	0.987	1.159	1.130	1.075	1.065	1.060	1.056	1.045	1.050	0.998	1.018	-	1.003	0.988	1.007	0.984	0.989
PSO_3^{10}	0.982	0.982	0.983	1.156	1.127	1.071	1.062	1.057	1.052	1.041	1.046	0.994	1.015	0.998	-	0.985	1.004	0.981	0.986
GA_2^1	1.000	1.000	1.001	1.178	1.149	1.090	1.082	1.075	1.071	1.059	1.064	1.012	1.034	1.016	1.018	-	1.022	1.000	1.004
GA_2^{10}	0.980	0.980	0.980	1.153	1.125	1.066	1.058	1.053	1.049	1.037	1.042	0.991	1.012	0.995	0.997	0.981	-	0.977	0.983
GA_3^1	1.004	1.004	1.005	1.179	1.151	1.095	1.085	1.080	1.076	1.064	1.069	1.017	1.037	1.019	1.021	1.007	1.026	-	1.008
GA_3^{10}	0.996	0.996	0.997	1.171	1.143	1.086	1.077	1.072	1.067	1.056	1.061	1.009	1.030	1.012	1.014	0.998	1.018	0.995	-

Notes: We report the out-of-sample realized variance forecast MSE of each model in the selected column relative to the benchmark in the selected row. Each number is a cross-sectional average of such pairwise relative MSEs for each stock. The HAR-X is identical to HAR in this setting.

results for the considered forecast evaluation measures is the difference in penalizing over- and under-predictions. The MSE is symmetrical, implying that both over- and under-predictions are penalized in the same way. However, the QLIKE function penalizes under-predictions more than over-predictions (Patton, 2011). Therefore, the HAR model seems to produce relatively more under-predictions, whereas the nonlinear models seem to produce more over-predictions.

Moving onto the $\mathcal{M}_{\text{ALL}}$ dataset results displayed in Table 2, we first consider the inclusion of the extra variables into the models. Performance increases by 8.1% for the HAR model if additional covariates are included, indicating that these variables do contain helpful information when predicting volatility. However, it seems that all nonlinear models perform significantly worse compared to their counterparts in Table 1. First, even though the HAR-X model performs better than its offset, the performance of the logHAR model deteriorates when including extra variables, indicating that the logarithmic transformation does not improve performance. We hypothesize that some important dynamics or patterns in the data are lost when applying this logarithmic transformation, making it difficult to accurately fit the logHAR model.

Turning attention to the NNs and hybrid models, we observe a similar pattern. Whilst performance is slightly better than the logHAR-X model, all models suffer from overfitting. It seems that the NNs often fit exquisitely on the validation set, whereas these configurations obtain modest test set MSEs. Although Table 1 indicates that hybrid models increase performance, the effect evaporates in this framework. A possible explanation behind the performance of the hybrid models might be that the input structure is too comprehensive (high-dimensional) for the algorithms, making it difficult to accurately explore the search space and find an optimal solution. Another explanation behind the weak performance of the hybrid models is that the input itself is inadequate, implying that the used hyperparameter framework does not work well in this situation. As the NNs are more prone to overfitting, the starting point of the algorithms might be inappropriate. Herefore, the algorithms do not converge to a local optimum, indicating that perhaps more hyperparameter optimization is needed. We perform the same robustness check as for the $\mathcal{M}_{\text{HAR}}$ dataset, of which the results are shown in Table 10 in the Appendix. The results for the period from January 29, 2001, to December 31, 2017, display completely different results, indicating that the construction and split of the data seem to be the driving factor behind the poor performance of the nonlinear models. The logHAR-X model performs remarkably well, followed by the NNs. However, the hybrid models perform worse than the NNs, a pattern we observed earlier.

An interesting observation is that, in contrast to Christensen et al. (2022), the inclusion of extra variables seems to only marginally improve performance for some models. Moreover, the performance of the HAR model actually worsens when adding extra variables. An explanation behind these contrasting results could be the setup of the data. Christensen et al. (2022) shine little light upon the extraction of the data, making it difficult to obtain the same results. They find that the IV is one of the most influential variables when predicting volatility. We follow a simplistic approach (averaging) when constructing the IV, whereas they do not explain in detail how the construct the IV variable. It might be that IV loses some of its predictive power as a result.

Table 8 in the Appendix shows the results using the QLIKE evaluation measure for the $\mathcal{M}_{\text{ALL}}$

Table 2: One-day-ahead relative MSE for dataset $\mathcal{M}_{\text{ALL}}$ over the test set from the entire sample period January 29, 2001, to December 31, 2021

	HAR	HAR-X	logHAR-X	NN_1^1	NN_1^{10}	NN_2^1	NN_2^{10}	NN_3^1	NN_3^{10}	NN_4^1	NN_4^{10}	PSO_2^1	PSO_2^{10}	PSO_3^1	PSO_3^{10}	GA_2^1	GA_2^{10}	GA_3^1	GA_3^{10}
HAR	-	0.919	2.455	1.426	1.829	1.824	1.673	1.405	1.552	1.322	1.568	1.946	1.745	2.059	1.738	1.878	2.219	1.834	2.012
HAR-X	1.096	-	2.632	1.543	1.945	1.941	1.781	1.517	1.652	1.430	1.658	2.061	1.848	2.186	1.839	1.960	2.283	1.966	2.090
logHAR-X	0.539	0.492	-	0.740	0.828	0.835	0.767	0.706	0.732	0.676	0.725	0.863	0.772	0.843	0.760	0.826	0.861	0.853	0.837
NN_1^1	0.836	0.760	1.836	-	1.339	1.400	1.256	1.104	1.188	1.049	1.193	1.394	1.281	1.462	1.286	1.391	1.580	1.317	1.439
NN_1^{10}	0.705	0.640	1.465	0.882	-	1.044	0.952	0.872	0.909	0.864	0.907	1.024	0.960	1.115	0.967	0.975	1.040	1.042	1.010
NN_2^1	0.699	0.638	1.470	0.957	1.061	-	0.977	0.894	0.927	0.854	0.906	1.075	0.989	1.183	0.993	0.998	1.060	1.108	1.039
NN_2^{10}	0.728	0.661	1.522	0.949	1.069	1.089	-	0.914	0.954	0.895	0.945	1.086	1.012	1.188	1.020	1.036	1.104	1.126	1.074
NN_3^1	0.787	0.717	1.775	1.044	1.217	1.241	1.136	-	1.072	0.986	1.070	1.263	1.164	1.393	1.170	1.186	1.313	1.277	1.255
NN_3^{10}	0.759	0.688	1.626	1.006	1.139	1.152	1.062	0.963	-	0.934	0.989	1.183	1.078	1.279	1.081	1.094	1.173	1.190	1.135
NN_4^1	0.824	0.750	1.836	1.152	1.369	1.343	1.251	1.095	1.172	-	1.164	1.411	1.280	1.488	1.274	1.379	1.545	1.383	1.427
NN_4^{10}	0.776	0.703	1.661	1.035	1.185	1.169	1.094	0.993	1.026	0.952	-	1.241	1.107	1.327	1.107	1.125	1.195	1.238	1.158
PSO_2^1	0.739	0.669	1.502	0.948	1.045	1.073	0.983	0.912	0.952	0.902	0.948	-	0.992	1.162	1.010	1.021	1.082	1.127	1.062
PSO_2^{10}	0.733	0.665	1.501	0.938	1.059	1.081	0.994	0.915	0.949	0.894	0.939	1.077	-	1.171	1.008	1.028	1.083	1.113	1.056
PSO_3^1	0.713	0.646	1.409	0.976	1.040	1.060	0.974	0.899	0.930	0.860	0.925	1.074	0.982	-	0.953	1.021	1.067	1.040	1.035
PSO_3^{10}	0.735	0.666	1.491	0.966	1.073	1.088	1.005	0.922	0.955	0.893	0.942	1.115	1.012	1.135	-	1.035	1.087	1.105	1.059
GA_2^1	0.751	0.680	1.602	0.987	1.085	1.104	1.025	0.935	0.971	0.930	0.964	1.117	1.040	1.225	1.041	-	1.088	1.132	1.068
GA_2^{10}	0.724	0.656	1.452	0.945	1.021	1.035	0.962	0.896	0.918	0.880	0.907	1.044	0.966	1.108	0.966	0.967	-	1.069	0.995
GA_3^1	0.728	0.660	1.481	0.886	1.048	1.093	1.000	0.914	0.952	0.882	0.952	1.107	1.007	1.105	0.999	1.038	1.122	-	1.060
GA_3^{10}	0.728	0.658	1.486	0.919	1.029	1.054	0.973	0.901	0.924	0.877	0.911	1.068	0.976	1.126	0.975	0.978	1.025	1.056	-

Notes: We report the out-of-sample realized variance forecast MSE of each model in the selected column relative to the benchmark in the selected row. Each number is a cross-sectional average of such pairwise relative MSEs for each stock.

dataset. Analogously to the results for $\mathcal{M}_{\text{ALL}}$, we observe that the difference in relative QLIKE value to the HAR model shrinks compared to the relative MSE value, again indicating that the HAR model is more prone to under-predictions compared to the other models. However, all NNs are unable to beat the HAR model, a task only the logHAR model is able to complete. Out of the NNs, the more sophisticated structures perform better than more simplistic frameworks, which is an observation present in all results for the data period until 2021.

Turning the attention to the hybrid models, we again observe that performance deteriorates in the $\mathcal{M}_{\text{ALL}}$ setting, which again suggests that the metaheuristics struggle with high-dimensional input, implying that either proper hyperparameter tuning or feature selection is needed.

To shed more light on the performance of the considered models, we construct boxplots of a model's performance compared to the HAR model for both the MSE and QLIKE, allowing us to understand whether the performance is driven by a single stock, or consistent across the cross-section of stocks. Figure 2 displays the obtained boxplots.

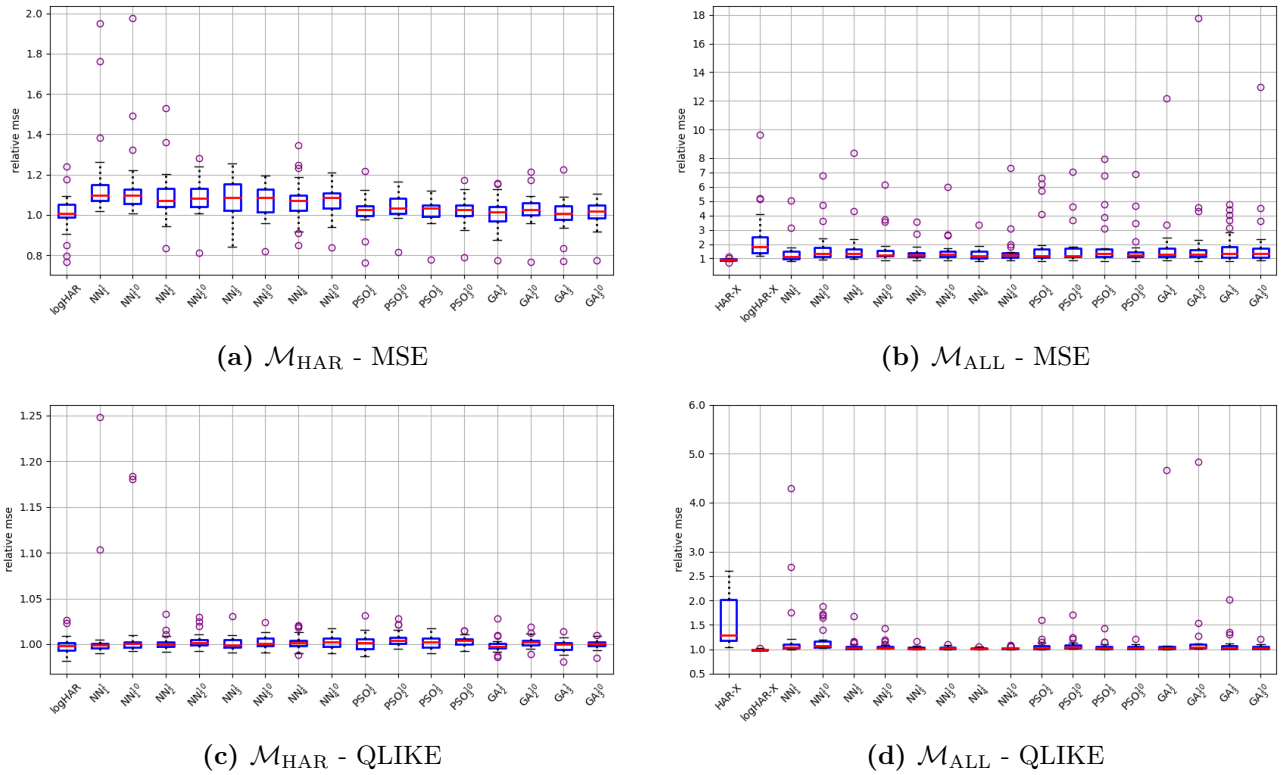
We first consider the MSE boxplots. For the $\mathcal{M}_{\text{HAR}}$ dataset, it seems that the one-layered NN produces many outliers, which is reflected in the relative MSE. All models seem to perform, on average, just worse than the HAR model, even though many models are able to beat the HAR model for several stocks. In the $\mathcal{M}_{\text{ALL}}$ dataset, the nonlinear models seem to suffer more from outliers. All models produce steady MSEs for the majority of stock but seemingly overfit much more for a few stocks. This shows that, even though all nonlinear models still perform worse than the HAR model, the difference is not all that great for the majority of stocks.

The QLIKE boxplots display different results. The performance across stocks is more concentrated around the median, indicating that the performance seems robust across stocks. This is especially the case for the $\mathcal{M}_{\text{HAR}}$ dataset, where the largest outlier is a relative QLIKE value of around 1.25. In contrast, the QLIKE values deviate more using $\mathcal{M}_{\text{ALL}}$, which is in line with the results when using MSE as the evaluation measure.

In addition to the per-stock performance, we construct MCSs from Hansen et al. (2011), which are sets of models that contain the best-performing model with a given confidence level. We compute the MCS at both a 75% and a 90% confidence level. The results are shown in Table 6 in the Appendix.

We first consider the $\mathcal{M}_{\text{HAR}}$ dataset. Using the MSE, it seems that most models are almost always included, with the NN1 having the lowest inclusion rate of 52%. Considering the QLIKE evaluation measure, we observe that the logHAR model and GA_2^1 have the highest inclusion rate, which aligns with the results obtained from Table 7 in the Appendix. An explanation behind the reason why all models are almost always included when using the MSE as the loss function could be the variance of the losses. Models with larger variances may require larger loss differences to establish statistical significance. Furthermore, the variance of the losses influences the size of the MCS. A larger variance often leads to a wider MCS, allowing for more models to be included in the final set. In addition, it implies more uncertainty in the estimates, resulting in a broader range of models that cannot be distinguished as significantly worse.

For the $\mathcal{M}_{\text{ALL}}$ dataset, we generally observe more considerable differences in relative MSE compared to the $\mathcal{M}_{\text{HAR}}$ dataset. This is also supported by the inclusion rates, which seem more



(a) $\mathcal{M}_{\text{HAR}}$ - MSE

(b) $\mathcal{M}_{\text{ALL}}$ - MSE

(c) $\mathcal{M}_{\text{HAR}}$ - QLIKE

(d) $\mathcal{M}_{\text{ALL}}$ - QLIKE

Figure 2: Boxplots of relative MSEs and QLIKE values compared to baseline HAR model

Notes: The data consists of 25 samples. The central mark is the median MSE (QLIKE), while the bottom and top edge of the box indicate the interquartile range. The whiskers are the outermost observations not flagged as outliers (the latter are marked with a circle). Outliers are points that fall below Q1 $-$ 1.5IQR or above Q3 + 1.5IQR, where Q1 is the $25th$ percentile, Q3 $75th$ percentile, and IQR is the interquartile range (Q3 -Q1). HAR-X is omitted from Panel (a) and Panel (c), as it is identical to HAR in $\mathcal{M}_{\text{HAR}}$.

conclusive compared to the inclusion rates for $\mathcal{M}_{\text{HAR}}$. For the MSE, all HAR models have an inclusion rate of at least 60%, whereas the NNs and hybrid models have an inclusion rate of at most 60%. The difference in inclusion rate becomes even more prominent when using the QLIKE loss function. As observed by Table 8 in the Appendix, the logHAR-X model seems to be the best, having an inclusion rate of 80%-90%. Besides the HAR model, all other models obtain an inclusion rate of less than 25%, often near 0%. Figure 2 shows that all models, besides the logHAR-X, consistently perform worse than the HAR model, which could explain the low inclusion rates.

To get a better grasp of the poor performance of the NNs and hybrid models, we construct extra models to find the main factor behind their weak performance. We turn our focus to the two-layered hybrid models, as it appears that the metaheuristics struggle with high-dimensional input. To be precise, we replace backpropagation with either metaheuristic when training the model to investigate whether backpropagation is the problem. These models are referred to as PSO-BP$_2^1$ for a two-layered structure using PSO. In addition, we construct an off-the-self approach of the hybrid models, where we re-estimate the weights of the NN with either meta-

heuristic every half a trading year (i.e., 126 days). These models are referred to as PSO+BP-T_2^1 and PSO-BP-T_2^1 for the hybrid models that first use backpropagation and for the hybrid models without backpropagation, respectively. The off-the-shelf approaches of these models are calculated for their 10 best seeds, thus honoring the same ranking as their reported ensemble variants.

Table 3: One-day-ahead relative MSE and QLIKE over test set from entire sample period January 29, 2001, to December 31, 2021, for models that either replace backpropagation or implement time-varying weights

Panel A: $\mathcal{M}_{\text{HAR}}$												
	PSO-BP$_2^1$	PSO-BP$_2^{10}$	GA-BP$_2^1$	GA-BP$_2^{10}$	PSO+BP-T$_2^1$	PSO+BP-T$_2^{10}$	GA+BP-T$_2^1$	GA+BP-T$_2^{10}$	PSO-BP-T$_2^1$	PSO-BP-T$_2^{10}$	GA-BP-T$_2^1$	GA-BP$_{T}$-T$_2^{10}$
HAR (MSE)	1.158	1.125	1.110	1.126	1.044	1.033	1.046	1.027	1.114	1.075	1.059	1.077
HAR (QLIKE)	1.014	1.045	1.005	1.019	1.006	1.010	1.006	1.008	1.017	1.038	1.016	1.021
Panel B: $\mathcal{M}_{\text{ALL}}$												
	PSO-BP$_2^1$	PSO-BP$_2^{10}$	GA-BP$_2^1$	GA-BP$_2^{10}$	PSO+BP-T$_2^1$	PSO+BP-T$_2^{10}$	GA+BP-T$_2^1$	GA+BP-T$_2^{10}$	PSO-BP-T$_2^1$	PSO-BP-T$_2^{10}$	GA-BP-T$_2^1$	GA-BP-T$_2^{10}$
HAR (MSE)	2.222	2.585	2.473	2.468	2.003	2.055	3.769	2.931	1.769	2.050	3.250	3.115
HAR (QLIKE)	1.327	1.871	1.423	1.643	1.084	1.136	1.085	1.087	1.521	1.454	1.440	1.434

Notes: We report the out-of-sample realized variance forecast MSE (QLIKE) of each model in the selected column relative to the benchmark in the selected row. Each number is a cross-sectional average of such pairwise relative MSEs (QLIKE) for each stock. Panel A shows the results for the $\mathcal{M}_{\text{HAR}}$ dataset, whereas panel B reports the results for the $\mathcal{M}_{\text{ALL}}$ dataset.

The results of the extended models are displayed in Table 3. The first observation is that using backpropagation does not seem to be the main problem, as training the NNs with the metaheuristics does not improve performance for either dataset. However, the modest performance of the metaheuristics could also be attributed to no hyperparameter optimization. These results are supported by the time-varying models, as using backpropagation first to train the model does seem to help the convergence of the weights. In short, these results highlight the fact that hyperparameter optimization seems essential for performance. The opted set of hyperparameters improves performance in one scenario, but even then, not all stocks benefit from it, as is displayed in the boxplots in Figure 2.

A supposedly surprising result is that the off-the-shelf approaches do not necessarily improve performance. Although off-the-shelf metaheuristics and backpropagation approaches are able to beat a simplistic NN, they are unable to improve upon the hybrid models in Table 1 and Table 2. Furthermore, in the $\mathcal{M}_{\text{ALL}}$ dataset, especially the GA produces poor results, whereas this is not the case for the $\mathcal{M}_{\text{HAR}}$ dataset. This again indicates that the algorithms struggle with the dimension of the input variables (which are lists containing the weights of a NN), thus highlighting a limitation of the metaheuristics.

6 Conclusion

In this paper, we build upon Christensen et al. (2022) and investigate the performance of different ML approaches and metaheuristics in the framework of volatility forecasting. We develop two hybrid models to enhance forecasting accuracy through metaheuristics. The first approach combines particle swarm optimization (PSO) and a feed-forward neural network (NN) in order to optimize the weights of a NN further. The second approach mimics the first approach with one central difference: We replace the PSO algorithm with a genetic algorithm (GA). Both metaheuristics are adapted to perform the task at hand. We compare the hybrid models against HAR models and feed-forward NNs. We evaluate the performance of the considered models over a period from January 29, 2001, to December 31, 2021, using the mean squared error (MSE) and quasi-likelihood (QLIKE) evaluation measures.

The results of our forecast evaluation show that our proposed hybrid models are able to beat the NNs considering both MSE and QLIKE when solely using lagged realized variances as input variables. Despite these promising results, the effect diminishes when adding extra forward-looking variables to the model, indicating that the metaheuristics struggle with high-dimensional input. However, the modest performance could also be attributed to little to no hyperparameter optimization. Nevertheless, no models beat the HAR model considering the MSE, while only the GA is able to marginally beat the HAR model in view of QLIKE. The robustness checks show that the dataset plays a pivotal role when performing volatility forecasts, as using a smaller data period improves the performance of many nonlinear models, such that the NNs are able to beat the HAR model. However, the importance of metaheuristics vanishes in this setting, indicating that they are a great tool when NNs are more prone to overfit but struggle to improve the performance of a well-fit NN.

This underlines a notable limitation of our research. We observe that the performance is highly dependent on the used data period. The weak performance of the nonlinear models could partially be explained by the fact that no hyperparameter optimization was conducted, which could significantly hinder the optimization process of the NNs and metaheuristic algorithms. Combined with no hyperparameter optimization, we observe that the performance of the metaheuristics is contingent on the input, which could explain the alternating performance of the hybrid models.

Further research regarding the implementation of metaheuristics in the framework of volatility forecasting is therefore needed. An exciting avenue could be analyzing the performance of different dataset configurations (i.e., using different training and test sets) in order to improve the convergence of NNs and metaheuristics. In addition, future research could explore per-stock hyperparameter optimization. A hyperparameter setting is not guaranteed to work well for every stock and dataset, therefore discrediting possible performance. However, this might prove challenging as much computational power is needed for thorough hyperparameter optimization. Alternatively, one might investigate the effect of feature selection on the performance of hybrid models, as not all variables are necessarily helpful when predicting volatility, and especially not helpful for the optimization process of the metaheuristics. Last, using different metaheuristics, or a combination of metaheuristics, could enhance forecasting accuracy. One could develop metaheuristics that work well with high-dimensional input or include volatility-specific attributes.

References

Andersen, T. G., Bollerslev, T., Diebold, F. X. & Ebens, H. (2001). The distribution of realized stock return volatility. *Journal of Financial Economics*, *61*(1), 43–76.

Aruoba, S. B., Diebold, F. X. & Scotti, C. (2009). Real-time measurement of business conditions. *Journal of Business & Economic Statistics*, *27*(4), 417–427.

Audrino, F. & Colangelo, D. (2010). Semi-parametric forecasts of the implied volatility surface using regression trees. *Statistics and Computing*, *20*(4), 421–434.

Baker, S. R., Bloom, N. & Davis, S. J. (2016). Measuring economic policy uncertainty. *The Quarterly Journal of Economics*, *131*(4), 1593–1636.

Barndorff-Nielsen, O. E. & Shephard, N. (2002). Econometric analysis of realized volatility and its use in estimating stochastic volatility models. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *64*(2), 253–280.

Blum, C. & Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM computing surveys (CSUR)*, *35*(3), 268–308.

Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, *31*(3), 307–327.

Bollerslev, T., Li, J. & Xue, Y. (2018). Volume, volatility, and public news announcements. *The Review of Economic Studies*, *85*(4), 2005–2041.

Bollerslev, T., Li, S. Z. & Zhao, B. (2020). Good volatility, bad volatility, and the cross section of stock returns. *Journal of Financial and Quantitative Analysis*, *55*(3), 751–781.

Bollerslev, T., Patton, A. J. & Quaedvlieg, R. (2016). Exploiting the errors: A simple approach for improved volatility forecasting. *Journal of Econometrics*, *192*(1), 1–18.

Christensen, K., Siggaard, M. & Veliyev, B. (2022, 06). A machine learning approach to volatility forecasting. *Journal of Financial Econometrics*.

Chung, H. & Shin, K.-s. (2018). Genetic algorithm-optimized long short-term memory network for stock market prediction. *Sustainability*, *10*(10), 3765.

Corsi, F. (2009). A simple approximate long-memory model of realized volatility. *Journal of Financial Econometrics*, *7*(2), 174–196.

Corsi, F. & Renò, R. (2012). Discrete-time volatility forecasting with persistent leverage effect and the link with continuous-time volatility modeling. *Journal of Business & Economic Statistics*, *30*(3), 368–380.

Doering, J., Kizys, R., Juan, A. A., Fito, A. & Polat, O. (2019). Metaheuristics for rich portfolio optimisation and risk management: Current state and future trends. *Operations Research Perspectives*, *6*, 100121.

Döpke, J., Fritsche, U. & Pierdzioch, C. (2017). Predicting recessions with boosted regression trees. *International Journal of Forecasting*, *33*(4), 745–759.

Dorigo, M. & Stützle, T. (2003). The ant colony optimization metaheuristic: Algorithms, applications, and advances. *Handbook of Metaheuristics*, 250–285.

Ederington, L. H. & Lee, J. H. (1993). How markets process information: News releases and volatility. *The Journal of Finance*, *48*(4), 1161–1191.

Gandomi, A. H., Yang, X.-S., Talatahari, S. & Alavi, A. H. (n.d.). Metaheuristic algorithms in modeling and optimization. *Metaheuristic Applications in Structures and Infrastructures*.

Gavrishchaka, V. V. & Banerjee, S. (2006). Support vector machine as an efficient framework for stock market volatility forecasting. *Computational Management Science*, *3*(2), 147–160.

Ghasemiyeh, R., Moghdani, R. & Sana, S. S. (2017). A hybrid artificial neural network with metaheuristic algorithms for predicting stock price. *Cybernetics and Systems*, *48*(4), 365–392.

Glorot, X. & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics* (pp. 249–256).

Göçken, M., Özçalıcı, M., Boru, A. & Dosdoğru, A. T. (2016). Integrating metaheuristics and artificial neural networks for improved stock price prediction. *Expert Systems with Applications*, *44*, 320–331.

Hansen, P. R., Lunde, A. & Nason, J. M. (2011). The model confidence set. *Econometrica*, *79*(2), 453–497.

Holland, J. H. (1992). Genetic algorithms. *Scientific American*, *267*(1), 66–73.

Jensen, J. L. W. V. (1906). Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica*, *30*(1), 175–193.

Ji, Y., Liew, A. W.-C. & Yang, L. (2021). A novel improved particle swarm optimization with long-short term memory hybrid model for stock indices forecast. *IEEE Access*, *9*, 23660–23671.

Jiang, G. J., Konstantinidi, E. & Skiadopoulos, G. (2012). Volatility spillovers and the effect of news announcements. *Journal of Banking & Finance*, *36*(8), 2260–2273.

Kennedy, J. & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks* (Vol. 4, pp. 1942–1948).

Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kleen, O. & Tetereva, A. (2022). A forest full of risk forecasts for managing volatility. *Available at SSRN: 4161957*.

Kolisetty, V. V. & Rajput, D. S. (2020). A review on the significance of machine learning for data analysis in big data. *Jordanian Journal of Computers and Information Technology (JJCIT)*, *6*(01), 155–171.

Li, Z., Liu, X., Duan, X. & Huang, F. (2010). Comparative research on particle swarm optimization and genetic algorithm. *Computer and Information Science*, *3*(1), 120–127.

Liu, Y. (2019). Novel volatility forecasting using deep learning–long short term memory recurrent neural networks. *Expert Systems with Applications*, *132*, 99–109.

Lu, L., Shin, Y., Su, Y. & Karniadakis, G. E. (2019). Dying relu and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*.

Luo, J., Chen, H., Xu, Y., Huang, H., Zhao, X. et al. (2018). An improved grasshopper optimization algorithm with application to financial stress prediction. *Applied Mathematical Modelling*, *64*, 654–668.

Maas, A. L., Hannun, A. Y., Ng, A. Y. et al. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the 30th international conference on machine learning* (Vol. 30, p. 3).

Mittnik, S., Robinzonov, N. & Spindler, M. (2015). Stock market volatility: Identifying major drivers and the nature of their impact. *Journal of Banking & Finance*, *58*, 1–14.

Panda, S. & Padhy, N. P. (2008). Comparison of particle swarm optimization and genetic algorithm for facts-based controller design. *Applied soft computing*, *8*(4), 1418–1427.

Patton, A. J. (2011). Volatility forecast comparison using imperfect volatility proxies. *Journal of Econometrics*, *160*(1), 246–256.

Patton, A. J. & Sheppard, K. (2015). Good volatility, bad volatility: Signed jumps and the persistence of volatility. *Review of Economics and Statistics*, *97*(3), 683–697.

Petrozziello, A., Troiano, L., Serra, A., Jordanov, I., Storti, G., Tagliaferri, R. & La Rocca, M. (2022). Deep learning for volatility forecasting in asset management. *Soft Computing*, *26*(17), 8553–8574.

Pradeepkumar, D. & Ravi, V. (2017). Forecasting financial time series volatility using particle swarm optimization trained quantile regression neural network. *Applied Soft Computing*, *58*, 35–52.

Qu, H., Duan, Q. & Niu, M. (2018). Modeling the volatility of realized volatility to improve volatility forecasts in electricity markets. *Energy Economics*, *74*, 767–776.

Rahimikia, E. & Poon, S.-H. (2020). Machine learning for realised volatility forecasting. *Available at SSRN*, *3707796*.

Ribeiro, G. T., Santos, A. A. P., Mariani, V. C. & dos Santos Coelho, L. (2021). Novel hybrid model based on echo state neural network applied to the prediction of stock price return volatility. *Expert Systems with Applications*, *184*, 115490.

Shahvaroughi Farahani, M. & Razavi Hajiagha, S. H. (2021). Forecasting stock price using integrated artificial neural network and metaheuristic algorithms compared to time series models. *Soft Computing*, *25*(13), 8483–8513.

Varian, H. R. (2014). Big data: New tricks for econometrics. *Journal of Economic Perspectives*, *28*(2), 3–28.

Yang, X.-S. (2010a). *Nature-inspired metaheuristic algorithms*. Luniver press.

Yang, X.-S. (2010b). A new metaheuristic bat-inspired algorithm. *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, 65–74.

Zhou, L., Pan, S., Wang, J. & Vasilakos, A. V. (2017). Machine learning on big data: Opportunities and challenges. *Neurocomputing*, *237*, 350–361.

Zhu, H., Wang, Y., Wang, K. & Chen, Y. (2011). Particle swarm optimization (pso) for the constrained portfolio optimization problem. *Expert Systems with Applications*, *38*(8), 10161–10169.

Zivkovic, M., Bacanin, N., Venkatachalam, K., Nayyar, A., Djordjevic, A., Strumberger, I. & Al-Turjman, F. (2021). Covid-19 cases prediction by using hybrid machine learning and beetle antennae search approach. *Sustainable Cities and Society*, *66*, 102669.

Appendices

A Tables

Table 4: Descriptive statistics of explanatory variables

Acronym	Mean		Median		Maximum		Minimum		Standard Deviation		Skewness		Kurtosis	
RVD	2.351	[1.212,3.793]	1.177	[0.622,1.912]	160.497	[73.815,423.074]	0.115	[0.065,0.173]	5.048	[2.667,11.255]	13.262	[6.561,26.357]	328.476	[74.626,1132.807]
RVW	2.348	[1.212,3.787]	1.247	[0.676,2.139]	71.471	[36.866,204.511]	0.203	[0.114,0.326]	4.198	[2.166,8.850]	7.747	[4.339,14.531]	90.809	[28.579,286.074]
RVM	2.351	[1.212, 3.764]	1.293	[0.622,2.266]	37.254	[18.625,96.988]	0.319	[0.191,0.561]	3.566	[1.774,7.123]	5.321	[3.397,9.103]	37.677	[13.279,107.095]
IV	0.255	[0.177,0.357]	0.224	[0.156,0.329]	1.088	[0.665,2.175]	0.114	[0.086,0.147]	0.107	[0.067,0.170]	2.157	[1.232,3.887]	7.587	[2.237,28.788]
EA														
VIX	19.764		17.345		82.690		9.140		8.922		2.259		7.781	
EPU	110.950		89.030		807.66		3.320		82.490		2.478		9.561	
US3M	-0.096		0.000		76.000		-81.000		4.505		-1.569		83.923	
HSI	0.000		0.000		0.015		0.000		0.000		23.541		793.363	
M1W	0.189	[0.042,0.635]	0.240	[-0.055,0.774]	55.71	[14.801,707.104]	-37.893	[-85.861,-16.542]	4.636	[2.730,21.672]	-0.049	[-5.037,30.180]	60.484	[2.780,953.701]
VOL	0.016	[-0.039,0.066]	-1.126	[-1.945,-0.079]	217.857	[157.448,307.820]	-177.395	[-206.678,-155.299]	34.607	[28.662,39.269]	0.272	[0.080,0.579]	2.098	[1.354,4.121]
ADS	-0.305		-0.140		9.139		-26.488		2.060		-7.533		89.571	

Notes: We report the descriptive statistics of each explanatory variable. The reported values are cross-sectional averages of every stock, and the brackets display the minimum and maximum value for that column across the stocks. As EA is an indicator variable, we do not report its descriptive statistics.

Table 5: Hyperparamaters of the NN and both algorithms

Neural Network	Particle Swarm Optimization	Genetic Algorithm
Learning rate = 0.01	$n\_iterations = 20$	$n\_iterations = 30$
Step-wise decay = 0.5 every 50 epochs	$n\_particles = 40$	$n\_chromosomes = 40$
Epochs = 500	$init\_particles = 3$	$init\_chromosomes = 3$
Patience = 100	$w = 0.9$	$n\_highest = 0.3$
Drop-out = 0.1	$w_p = 0.02$	$mutation\_probability = 0.7$
Initializer: Glorot normal	$c_1 = 3$	$mutation\_choice = 0.8$
ADAM = default	$c_{1p} = -0.1$	$mutation\_amount = 1$
Ensebmle $\in \{1, 10\}$	$c_2 = 1$	$crossover\_choice = 0.8$
	$c_{2p} = 0.1$	
	$init\_vel = 3$	
	$max\_vel = 15$	

Notes: Step-wise learning rate decay is only employed for $\mathcal{M}_{\text{ALL}}$ for the considered data period.

Table 6: Inclusion rate in the MCS using the test set

	Panel A: $\mathcal{M}_{\text{HAR}}$				Panel B: $\mathcal{M}_{\text{ALL}}$			
	MSE		QLIKE		MSE		QLIKE	
	$\widehat{\mathcal{M}}^*_{75\%}$	$\widehat{\mathcal{M}}^*_{90\%}$	$\widehat{\mathcal{M}}^*_{75\%}$	$\widehat{\mathcal{M}}^*_{90\%}$	$\widehat{\mathcal{M}}^*_{75\%}$	$\widehat{\mathcal{M}}^*_{90\%}$	$\widehat{\mathcal{M}}^*_{75\%}$	$\widehat{\mathcal{M}}^*_{90\%}$
HAR	96%	100%	44%	60%	92%	100%	36%	48%
HAR-X	96%	100%	44%	60%	100%	100%	4%	12%
logHAR	92%	92%	64%	68%	60%	96%	84%	88%
NN_1^1	52%	72%	48%	60%	40%	52%	0%	4%
NN_1^{10}	52%	84%	28%	44%	8%	8%	0%	0%
NN_2^1	92%	92%	52%	64%	12%	20%	0%	4%
NN_2^{10}	80%	96%	12%	20%	0%	4%	0%	0%
NN_3^1	96%	100%	40%	60%	24%	40%	4%	4%
NN_3^{10}	100%	100%	24%	32%	12%	16%	0%	4%
NN_4^1	100%	100%	52%	72%	36%	60%	8%	24%
NN_4^{10}	100%	100%	12%	24%	12%	28%	0%	0%
PSO_2^1	68%	92%	56%	60%	16%	28%	0%	0%
PSO_2^{10}	64%	88%	0%	16%	4%	8%	0%	0%
PSO_3^1	80%	92%	36%	48%	24%	36%	12%	12%
PSO_3^{10}	56%	88%	8%	8%	4%	12%	0%	0%
GA_2^1	84%	96%	76%	84%	16%	28%	0%	0%
GA_2^{10}	56%	92%	8%	12%	4%	8%	0%	8%
GA_3^1	88%	96%	52%	76%	28%	40%	4%	16%
GA_3^{10}	80%	84%	24%	44%	0%	0%	0%	0%

Notes: We report the inclusion rate for both $\mathcal{M}_{\text{HAR}}$ and $\mathcal{M}_{\text{ALL}}$. The inclusion rate is the percentage a model was retained in the final set of the MCS (Hansen et al., 2011) across all stocks. We compute the MCS at both a 75% and 90% confidence level using a moving block bootstrap, a block size of $l = 2$, and 1000 bootstrap repetitions. For $\mathcal{M}_{\text{ALL}}$, logHAR presents the values of its distributed lag-type version logHAR-X.

Table 7: One-day-ahead relative QLIKE for dataset $\mathcal{M}_{\text{HAR}}$ over test set from the entire sample period January 29, 2001, to December 31, 2021

	HAR	HAR-X	logHAR	NN_1^1	NN_1^{10}	NN_2^1	NN_2^{10}	NN_3^1	NN_3^{10}	NN_4^1	NN_4^{10}	PSO_2^1	PSO_2^{10}	PSO_3^1	PSO_3^{10}	GA_2^1	GA_2^{10}	GA_3^1	GA_3^{10}
HAR	-	1.000	0.999	1.012	1.014	1.002	1.004	1.002	1.003	1.002	1.003	1.002	1.005	1.002	1.004	0.999	1.003	0.999	1.000
HAR-X	1.000	-	0.999	1.012	1.014	1.002	1.004	1.002	1.003	1.002	1.003	1.002	1.005	1.002	1.004	0.999	1.003	0.999	1.000
logHAR	1.001	1.001	-	1.013	1.015	1.003	1.005	1.003	1.004	1.003	1.003	1.003	1.006	1.003	1.005	1.000	1.004	1.000	1.001
NN_1^1	0.990	0.990	0.989	-	1.002	0.992	0.994	0.992	0.993	0.992	0.993	0.992	0.995	0.992	0.994	0.989	0.993	0.989	0.990
NN_1^{10}	0.988	0.988	0.987	0.998	-	0.990	0.992	0.990	0.991	0.990	0.990	0.990	0.993	0.990	0.992	0.987	0.991	0.987	0.988
NN_2^1	0.998	0.998	0.997	1.010	1.012	-	1.002	1.000	1.001	1.000	1.001	1.000	1.003	1.001	1.002	0.997	1.001	0.997	0.999
NN_2^{10}	0.996	0.996	0.995	1.008	1.010	0.998	-	0.998	0.999	0.998	0.998	0.998	1.001	0.998	1.000	0.995	0.999	0.995	0.996
NN_3^1	0.998	0.998	0.998	1.011	1.012	1.000	1.002	-	1.001	1.001	1.001	1.000	1.004	1.001	1.002	0.997	1.001	0.997	0.999
NN_3^{10}	0.997	0.997	0.997	1.010	1.011	0.999	1.001	0.999	-	1.000	1.000	0.999	1.003	1.000	1.001	0.996	1.000	0.996	0.998
NN_4^1	0.998	0.998	0.997	1.010	1.012	1.000	1.002	0.999	1.000	-	1.000	0.999	1.003	1.000	1.001	0.997	1.000	0.996	0.998
NN_4^{10}	0.998	0.998	0.997	1.010	1.012	0.999	1.002	0.999	1.000	1.000	-	0.999	1.003	1.000	1.001	0.997	1.000	0.996	0.998
PSO_2^1	0.998	0.998	0.998	1.011	1.012	1.000	1.002	1.000	1.001	1.001	1.001	-	1.004	1.001	1.002	0.997	1.001	0.997	0.999
PSO_2^{10}	0.995	0.995	0.994	1.007	1.009	0.997	0.999	0.996	0.998	0.997	0.997	0.996	-	0.997	0.999	0.994	0.998	0.994	0.995
PSO_3^1	0.998	0.998	0.997	1.010	1.012	0.999	1.002	0.999	1.000	1.000	1.000	0.999	1.003	-	1.001	0.997	1.000	0.996	0.998
PSO_3^{10}	0.996	0.996	0.996	1.008	1.010	0.998	1.000	0.998	0.999	0.999	0.999	0.998	1.001	0.999	-	0.995	0.999	0.995	0.997
GA_2^1	1.001	1.001	1.000	1.013	1.015	1.003	1.005	1.003	1.004	1.003	1.003	1.003	1.006	1.003	1.005	-	1.004	1.000	1.001
GA_2^{10}	0.997	0.997	0.997	1.009	1.011	0.999	1.001	0.999	1.000	1.000	1.000	0.999	1.003	1.000	1.001	0.996	-	0.996	0.998
GA_3^1	1.001	1.001	1.001	1.014	1.015	1.003	1.005	1.003	1.004	1.004	1.004	1.003	1.007	1.004	1.005	1.000	1.004	-	1.002
GA_3^{10}	1.000	1.000	0.999	1.012	1.014	1.001	1.004	1.001	1.002	1.002	1.002	1.001	1.005	1.002	1.003	0.999	1.002	0.998	-

Notes: We report the out-of-sample realized variance forecast QLIKE of each model in the selected column relative to the benchmark in the selected row. Each number is a cross-sectional average of such pairwise relative QLIKEs for each stock. HAR-X is identical to HAR in this setting.

Table 8: One-day-ahead relative QLIKE for dataset $\mathcal{M}_{\text{ALL}}$ over test set from the entire sample period January 29, 2001, to December 31, 2021

	HAR	HAR-X	logHAR-X	NN_1^1	NN_1^{10}	NN_2^1	NN_2^{10}	NN_3^1	NN_3^{10}	NN_4^1	NN_4^{10}	PSO_2^1	PSO_2^{10}	PSO_3^1	PSO_3^{10}	GA_2^1	GA_2^{10}	GA_3^1	GA_3^{10}
HAR	-	1.534	0.993	1.276	1.185	1.070	1.064	1.031	1.030	1.016	1.024	1.072	1.088	1.054	1.044	1.177	1.227	1.094	1.043
HAR-X	0.710	-	0.705	0.872	0.844	0.765	0.756	0.730	0.730	0.721	0.726	0.764	0.775	0.749	0.740	0.866	0.907	0.779	0.740
logHAR-X	1.007	1.545	-	1.279	1.192	1.077	1.071	1.037	1.037	1.023	1.031	1.080	1.095	1.062	1.051	1.187	1.231	1.102	1.050
NN_1^1	0.884	1.334	0.877	-	1.016	0.944	0.934	0.906	0.908	0.897	0.904	0.941	0.954	0.927	0.918	1.041	1.089	0.955	0.919
NN_1^{10}	0.873	1.339	0.866	1.052	-	0.926	0.920	0.896	0.897	0.886	0.893	0.927	0.937	0.919	0.909	0.989	1.010	0.939	0.906
NN_2^1	0.944	1.458	0.938	1.205	1.111	-	0.998	0.972	0.972	0.959	0.967	1.004	1.017	0.996	0.985	1.061	1.151	1.020	0.982
NN_2^{10}	0.945	1.452	0.939	1.192	1.109	1.004	-	0.973	0.972	0.960	0.968	1.007	1.020	0.996	0.986	1.077	1.147	1.023	0.983
NN_3^1	0.972	1.488	0.965	1.223	1.146	1.038	1.032	-	1.000	0.987	0.995	1.040	1.055	1.024	1.014	1.137	1.188	1.059	1.012
NN_3^{10}	0.972	1.489	0.965	1.233	1.148	1.038	1.032	1.000	-	0.986	0.995	1.040	1.054	1.024	1.013	1.137	1.188	1.060	1.012
NN_4^1	0.985	1.512	0.978	1.255	1.166	1.053	1.047	1.015	1.014	-	1.008	1.055	1.070	1.038	1.028	1.158	1.208	1.077	1.027
NN_4^{10}	0.977	1.498	0.970	1.243	1.156	1.044	1.038	1.006	1.006	0.992	-	1.047	1.062	1.030	1.019	1.150	1.196	1.068	1.018
PSO_2^1	0.941	1.448	0.935	1.184	1.105	0.997	0.995	0.968	0.968	0.955	0.963	-	1.014	0.992	0.981	1.062	1.152	1.016	0.979
PSO_2^{10}	0.930	1.430	0.923	1.170	1.088	0.985	0.982	0.956	0.956	0.944	0.952	0.988	-	0.979	0.969	1.044	1.128	1.001	0.966
PSO_3^1	0.954	1.463	0.947	1.200	1.127	1.020	1.014	0.982	0.982	0.968	0.976	1.022	1.036	-	0.994	1.125	1.173	1.042	0.994
PSO_3^{10}	0.959	1.472	0.953	1.213	1.133	1.026	1.019	0.988	0.988	0.974	0.982	1.028	1.042	1.010	-	1.130	1.176	1.049	1.000
GA_2^1	0.939	1.455	0.933	1.202	1.099	0.986	0.987	0.966	0.966	0.953	0.962	0.991	1.002	0.991	0.980	-	1.142	0.999	0.974
GA_2^{10}	0.908	1.409	0.901	1.150	1.044	0.963	0.959	0.934	0.934	0.922	0.929	0.967	0.978	0.958	0.947	1.031	-	0.983	0.944
GA_3^1	0.934	1.432	0.928	1.164	1.092	0.988	0.985	0.960	0.960	0.948	0.956	0.991	1.002	0.984	0.974	1.036	1.145	-	0.970
GA_3^{10}	0.960	1.474	0.954	1.218	1.132	1.024	1.019	0.989	0.988	0.975	0.983	1.026	1.040	1.012	1.002	1.111	1.173	1.044	-

Notes: We report the out-of-sample realized variance forecast QLIKE of each model in the selected column relative to the benchmark in the selected row. Each number is a cross-sectional average of such pairwise relative QLIKEs for each stock.

Table 9: One-day-ahead relative MSE for dataset $\mathcal{M}_{\text{HAR}}$ over test set from the entire sample period January 29, 2001, to December 31, 2017

	HAR	HAR-X	logHAR	NN_1^1	NN_1^{10}	NN_2^1	NN_2^{10}	NN_3^1	NN_3^{10}	NN_4^1	NN_4^{10}	PSO_2^1	PSO_2^{10}	PSO_3^1	PSO_3^{10}	GA_2^1	GA_2^{10}	GA_3^1	GA_3^{10}
HAR	-	1.000	1.004	0.967	0.981	0.963	0.979	0.973	0.985	0.971	0.977	0.976	0.995	0.984	0.999	0.984	0.995	0.975	0.989
HAR-X	1.000	-	1.004	0.967	0.981	0.963	0.979	0.973	0.985	0.971	0.977	0.976	0.995	0.984	0.999	0.984	0.995	0.975	0.989
logHAR	1.001	1.001	-	0.968	0.983	0.964	0.980	0.973	0.985	0.971	0.978	0.977	0.996	0.985	1.000	0.985	0.996	0.976	0.990
NN_1^1	1.035	1.035	1.039	-	1.015	0.996	1.013	1.007	1.020	1.005	1.012	1.010	1.029	1.018	1.034	1.019	1.030	1.009	1.024
NN_1^{10}	1.020	1.020	1.024	0.985	-	0.982	0.998	0.992	1.005	0.990	0.997	0.995	1.014	1.003	1.019	1.004	1.015	0.994	1.009
NN_2^1	1.039	1.039	1.043	1.004	1.019	-	1.017	1.010	1.023	1.009	1.015	1.014	1.033	1.022	1.038	1.023	1.034	1.013	1.028
NN_2^{10}	1.022	1.022	1.026	0.988	1.003	0.984	-	0.994	1.006	0.992	0.999	0.998	1.016	1.005	1.021	1.006	1.017	0.996	1.011
NN_3^1	1.029	1.029	1.032	0.995	1.010	0.991	1.007	-	1.013	0.999	1.005	1.005	1.024	1.012	1.028	1.013	1.024	1.004	1.018
NN_3^{10}	1.016	1.016	1.019	0.982	0.997	0.978	0.994	0.987	-	0.986	0.992	0.992	1.010	0.999	1.015	1.000	1.011	0.991	1.005
NN_4^1	1.031	1.031	1.033	0.997	1.012	0.993	1.009	1.002	1.015	-	1.007	1.006	1.025	1.014	1.030	1.014	1.025	1.005	1.019
NN_4^{10}	1.024	1.024	1.027	0.990	1.005	0.986	1.002	0.995	1.008	0.994	-	1.000	1.018	1.007	1.023	1.008	1.019	0.998	1.013
PSO_2^1	1.025	1.025	1.029	0.990	1.005	0.987	1.003	0.997	1.010	0.995	1.002	-	1.019	1.009	1.024	1.009	1.020	0.999	1.014
PSO_2^{10}	1.006	1.006	1.009	0.972	0.987	0.968	0.984	0.978	0.990	0.976	0.982	0.981	-	0.989	1.005	0.990	1.000	0.980	0.995
PSO_3^1	1.017	1.017	1.020	0.983	0.998	0.979	0.995	0.989	1.002	0.987	0.994	0.993	1.011	-	1.016	1.001	1.012	0.991	1.006
PSO_3^{10}	1.001	1.001	1.004	0.968	0.982	0.964	0.980	0.973	0.986	0.972	0.978	0.977	0.996	0.985	-	0.985	0.996	0.976	0.990
GA_2^1	1.017	1.017	1.020	0.983	0.998	0.979	0.995	0.989	1.001	0.987	0.993	0.992	1.011	1.000	1.016	-	1.011	0.991	1.006
GA_2^{10}	1.006	1.006	1.009	0.972	0.987	0.968	0.984	0.978	0.990	0.976	0.982	0.981	1.000	0.989	1.004	0.989	-	0.980	0.994
GA_3^1	1.027	1.027	1.030	0.992	1.007	0.988	1.004	0.998	1.011	0.997	1.003	1.002	1.021	1.010	1.025	1.010	1.021	-	1.015
GA_3^{10}	1.011	1.011	1.014	0.977	0.992	0.974	0.990	0.983	0.996	0.982	0.988	0.987	1.006	0.995	1.010	0.995	1.006	0.986	-

Notes: We report the out-of-sample realized variance forecast MSE of each model in the selected column relative to the benchmark in the selected row. Each number is a cross-sectional average of such pairwise relative MSEs for each stock. HAR-X is identical to HAR in this setting.

Table 10: One-day-ahead relative MSE for dataset $\mathcal{M}_{\text{ALL}}$ over test set from the entire sample period January 29, 2001, to December 31, 2017

	HAR	HAR-X	logHAR-X	NN_1^1	NN_1^{10}	NN_2^1	NN_2^{10}	NN_3^1	NN_3^{10}	NN_4^1	NN_4^{10}	PSO_2^1	PSO_2^{10}	PSO_3^1	PSO_3^{10}	GA_2^1	GA_2^{10}	GA_3^1	GA_3^{10}
HAR	-	1.042	0.925	0.967	0.964	0.945	0.977	0.971	0.991	0.966	0.984	0.961	0.998	0.984	1.004	0.967	0.987	1.003	1.006
HAR-X	0.988	-	0.908	0.951	0.946	0.925	0.956	0.950	0.970	0.948	0.966	0.940	0.975	0.964	0.983	0.945	0.964	0.986	0.985
logHAR-X	1.086	1.124	-	1.047	1.044	1.023	1.058	1.051	1.073	1.047	1.066	1.041	1.080	1.066	1.087	1.046	1.067	1.087	1.089
NN_1^1	1.053	1.089	0.972	-	1.011	0.990	1.023	1.018	1.039	1.013	1.033	1.008	1.045	1.027	1.051	1.013	1.033	1.055	1.054
NN_1^{10}	1.042	1.076	0.962	1.004	-	0.980	1.013	1.006	1.027	1.003	1.022	0.997	1.034	1.021	1.041	1.003	1.023	1.042	1.043
NN_2^1	1.066	1.098	0.983	1.025	1.023	-	1.034	1.029	1.050	1.024	1.044	1.018	1.056	1.043	1.063	1.023	1.043	1.064	1.065
NN_2^{10}	1.031	1.061	0.951	0.990	0.989	0.967	-	0.995	1.015	0.990	1.009	0.985	1.020	1.008	1.028	0.990	1.009	1.029	1.030
NN_3^1	1.037	1.068	0.957	0.999	0.994	0.974	1.007	-	1.021	0.996	1.016	0.991	1.027	1.015	1.034	0.996	1.016	1.035	1.036
NN_3^{10}	1.017	1.047	0.938	0.979	0.975	0.955	0.987	0.980	-	0.976	0.995	0.971	1.007	0.995	1.013	0.977	0.995	1.015	1.016
NN_4^1	1.045	1.080	0.965	1.006	1.003	0.982	1.015	1.009	1.029	-	1.021	1.000	1.036	1.024	1.043	1.005	1.024	1.040	1.044
NN_4^{10}	1.024	1.058	0.945	0.987	0.983	0.962	0.995	0.989	1.008	0.981	-	0.980	1.015	1.004	1.022	0.985	1.003	1.020	1.023
PSO_2^1	1.048	1.078	0.966	1.010	1.005	0.984	1.017	1.011	1.032	1.007	1.027	-	1.038	1.026	1.046	1.006	1.026	1.046	1.048
PSO_2^{10}	1.012	1.039	0.932	0.973	0.970	0.949	0.980	0.975	0.994	0.971	0.989	0.965	-	0.989	1.008	0.970	0.989	1.009	1.009
PSO_3^1	1.025	1.056	0.945	0.980	0.983	0.962	0.995	0.989	1.009	0.986	1.005	0.980	1.016	-	1.022	0.984	1.004	1.025	1.025
PSO_3^{10}	1.004	1.033	0.925	0.965	0.962	0.942	0.974	0.968	0.987	0.964	0.982	0.959	0.994	0.981	-	0.964	0.982	1.001	1.002
GA_2^1	1.043	1.072	0.961	1.003	1.001	0.978	1.012	1.006	1.027	1.002	1.021	0.996	1.032	1.020	1.040	-	1.020	1.040	1.042
GA_2^{10}	1.023	1.052	0.943	0.983	0.981	0.959	0.992	0.986	1.006	0.981	1.000	0.977	1.012	1.000	1.019	0.981	-	1.020	1.021
GA_3^1	1.010	1.046	0.932	0.975	0.970	0.949	0.981	0.976	0.995	0.968	0.987	0.966	1.002	0.991	1.008	0.971	0.990	-	1.009
GA_3^{10}	1.003	1.033	0.924	0.964	0.961	0.940	0.972	0.966	0.986	0.961	0.980	0.957	0.992	0.980	0.999	0.962	0.980	0.998	-

Notes: We report the out-of-sample realized variance forecast MSE of each model in the selected column relative to the benchmark in the selected row. Each number is a cross-sectional average of such pairwise relative MSEs for each stock.

B Programming Code

The code is written in Python 3.9.12. In addition, Excel is used to obtain the relative MSE and QLIKE values for all models. A detailed explanation can be found in the README file.