The Forecasting Grand Prix: Self-Driving Neural Networks for Inflation Prediction

Sicco Kooiker[(a)*] *Janneke van Brummelen*[(a,b)] *Julia Schaumburg*[(a,b)]

[(a)] Vrije Universiteit Amsterdam [(b)] Tinbergen Institute

June 2026

Abstract

We propose a time-varying neural network framework for macroeconomic forecasting in which the weights and biases of a neural network evolve over time using a self-driving updating scheme based on past forecast errors. While maintaining the flexibility of neural networks, we introduce observation-driven adaptation through parameter dynamics, linking the machine learning and observation-driven parameter literatures. To address sensitivity to hyperparameter choices, a stacked recursive least squares method find the best performing hyperparameters on-the-fly. The method is fully online and does not require re-estimation over a rolling window. Using predictors from FRED-MD, we evaluate multi-horizon U.S. inflation forecasts over the most recent 15 years against a broad set of machine learning benchmarks. We show that the proposed method improves forecast accuracy relative to static neural network specifications and competing benchmarks. In-sample diagnostics illustrate how the optimal hyperparameter configuration and variable importance change over time.

Keywords: time-varying neural networks, observation-driven dynamics.

JEL Classification: C45, E43, C38.

*Email address corresponding author: s.h.kooiker@vu.nl (Sicco Kooiker). Schaumburg and Kooiker thank the Dutch Science Foundation (NWO, grant VI.VIDI.191.169) for financial support. We thank the Dutch National supercomputer Snellius at SURF and the Ada cluster (formerly Bazis) at the Vrije Universitiet Amsterdam for providing computational resources.

1 Introduction

The relationship between inflation and its predictors is well documented to change over time (Stock and Watson, 1996, 2007), making accurate inflation forecasting a persistent challenge in macroeconomics. A growing body of evidence demonstrates that machine learning methods can improve inflation and macroeconomic forecasts, particularly in data-rich environments (Medeiros et al., 2021; Naghi et al., 2024; Giannone et al., 2021; Babii et al., 2022; Goulet Coulombe et al., 2022; Goulet Coulombe, 2024, 2025). Yet most machine learning forecasting models are estimated as static mappings that adapt only through re-estimation on rolling or expanding windows.

At the same time, observation-driven time-varying parameter models have proven effective in macroeconomic prediction (Primiceri, 2005; D'Agostino et al., 2013; Koop and Korobilis, 2012). The score-driven framework of Creal et al. (2013) and Harvey (2013), surveyed in Artemova et al. (2022a,b), and the closely related approach in Creal et al. (2024), which allows parameter updates based on general moment conditions rather than only the score, provide observation-driven methods that can intrinsically adapt to time-varying conditions. The score-driven framework has also demonstrated its value in macroeconomic applications: Delle Monache and Petrella (2017) demonstrate improvements in inflation forecasting using score-driven time-varying parameter models; De Polis et al. (2024) use score-driven models to forecast macroeconomic downside risk; and Castro et al. (2023) confirm the forecasting value of the score-driven approach for inflation in Brazil. We aim to find a model that can flexibly adapt to time-varying conditions in high-dimensional settings.

This paper studies adaptive machine learning for macroeconomic forecasting through time-varying neural networks. Neural networks are natural candidates because they can approximate complex nonlinear functions and combine many predictors in a parsimonious way, avoiding the curse of dimensionality that affects conventional nonparametric methods (Cybenko, 1989;

Hornik et al., 1989; Barron, 1993). We propose a framework in which the weights and biases of a neural network evolve over time using a self-driving update based on Adam (Adaptive Moment Estimation) steps (Kingma and Ba, 2015). Unlike standard neural network training, which processes the full dataset in repeated passes, our approach operates in an online fashion: parameters are updated sequentially as each new observation arrives, without revisiting past data. Adam maintains exponential moving averages of the gradient and its squared values, providing per-parameter adaptive learning rates with built-in momentum. The update uses the gradient of a loss objective in a manner analogous to the observation-driven methodology of Creal et al. (2024). While gradient-based updates can still overshoot the optimum, the adaptive scaling of Adam automatically adjusts the effective step size for each parameter based on its historical gradient variance, which helps stabilize the updates.

While neural network performance in macroeconomic forecasting has been mixed (Medeiros et al., 2021; Naghi et al., 2024), related evidence in economics and finance suggests that their nonlinear structure can be valuable when tuned to the forecasting environment (Goulet Coulombe et al., 2022; Goulet Coulombe, 2025; Goulet Coulombe et al., 2026; Masini et al., 2023; Gu et al., 2020; Bianchi et al., 2021; Kelly et al., 2024). Neural networks are notoriously hard to optimize. They require careful selection of hyperparameters and initialization. As Kalfa et al. (2025) document, the out-of-sample forecasting performance of machine learning models, neural networks in particular, depends critically on hyperparameter choices, with certain configurations virtually guaranteeing good or poor performance. To reduce the dependence on the initialization of the weights and biases, the predictions are often averaged over separately trained models with fixed hyperparameters. Hyperparameters are often set ad hoc and sometimes selected based on a so-called validation set. Hyperparameters are assumed to be fixed within the estimation window. We treat the hyperparameters as part of the model rather than as fixed inputs chosen in advance, and embed their selection directly in the estimation. Instead of selecting a single hyperparameter configuration, as is common in hyperparameter tuning, we form an ensemble

over candidate configurations and weight them through a recursive regression framework. Committing to one configuration leads to overfitting and weaker out-of-sample performance, while the online ensemble is more robust to this choice. The ensemble method we propose is inspired by Stock and Watson (2004), and we compare it against the other methods proposed in this paper.

The challenge is then to learn the best-performing ensemble of candidates. We treat this combination as fixed and recover it through a recursive least squares algorithm. The ensemble weights in (2) still carry the time subscript t because we update them sequentially as new observations arrive. The estimate sharpens as the sample grows rather than tracking changes in the underlying combination. Since the seminal work of Bates and Granger (1969), a large literature has studied how to combine competing forecasts into a single prediction (Timmermann, 2006; Raftery et al., 2010; Koop and Korobilis, 2012; Billio et al., 2013; Del Negro et al., 2016), considering both static and adaptive weighting schemes. We contribute a framework where the candidate parameters update through observation-driven dynamics while the best ensemble weights are recovered simultaneously. We now formalize this framework, starting from the forecasting problem it addresses.

We state the forecasting problem as follows.

$$\pi_{t+h} = \mathcal{G}_{t,h}(x_t) + \varepsilon_{t+h}, \qquad t = 1, \dots, T, \quad h = 1, \dots, H, \tag{1}$$

where π_{t+h} is the endogenous variable at horizon h, x_t is a high-dimensional predictor vector, $\mathcal{G}_{t,h}(\cdot)$ is a potentially time-varying nonlinear function and ε_{t+h} the disturbances.

We model $\mathcal{G}_{t,h}(\cdot)$ as a time-varying mixture of time-varying neural network candidates:

$$\mathcal{G}_{t,h}(x_t) = \sum_{k=1}^{K} w_{k,t,h}\, f_k(x_t; \theta_{k,t,h}, \eta_k), \tag{2}$$

where $f_k(\cdot; \theta_{k,t,h}, \eta_k)$ denotes expert k with time-varying parameters $\theta_{k,t,h}$ and fixed hyperpa-

rameters η_k, and $w_{k,t,h}$ denotes the combination weight assigned to expert k. The dynamics of $w_{k,t,h}$ are specified in Section 2.3. We estimate multiple neural network models, each with different hyperparameters representing distinct candiates. Rather than selecting a single model, we form a weighted forecast combination where the weights evolve autonomously over time. We compare seven different constraint observation driven methods to evolve the arbiter over time. This approach links time-varying parameter learning and forecast combination in a single self-driving update architecture.

Our empirical application considers U.S. inflation forecasting using predictors from FRED-MD. We evaluate multi-horizon forecasts over the most recent 15 years and assess performance with relative RMSE and MAE against a random walk benchmark, complemented by model confidence set procedures to evaluate statistical significance. We compare our approach against a selection of the best performing machine learning forecast methods from Naghi et al. (2024), including tree-based, kernel-based, and penalized methods. Per forecast horizon, a model with 10,000 candidates produces 700 forecasts within 30 seconds of GPU time, despite the candidates spanning close to 200 million parameters in total. Our method outperforms state-of-the-art benchmarks in out-of-sample forecasting and is the only method that enters every model confidence set.

The remainder of the paper is organized as follows. Section 2 presents the adaptive model and estimation procedure. Section 3 presents the empirical forecasting results and in-sample diagnostics. Section 4 concludes.

2 Methodology

We consider the direct multi-horizon forecasting problem in Equation (1) and the mixture of expert formulation in Equation (2) where y_{t+h} is the forecast target observed h periods ahead of t, $x_t \in \mathbb{R}^n$ is a vector of stationary predictors, potentially including lags of those, available at time t, and ε_{t+h} is the prediction error.

We model the forecast function $f_{h,t}$[1] as a feedforward neural network with time-varying parameters,

$$f_{h,t}(x_t) = \mathrm{NN}_\eta(x_t;\, \theta_t). \tag{3}$$

The network consists of d hidden layers. Layer outputs are defined recursively as

$$h_{\ell,t} = \sigma(W_{\ell,t}\, h_{\ell-1,t} + b_{\ell,t})\,, \quad \ell = 1, \ldots, d, \tag{4}$$

with $h_{0,t} = \tilde{x}_t$, where $W_{\ell,t}$ and $b_{\ell,t}$ are the weight matrix and bias vector of layer ℓ at time t, and $\sigma(\cdot)$ is a nonlinear activation function applied elementwise. The output layer produces a scalar forecast $\hat{y}_{t+h} = W_{d+1,t}h_{d,t} + b_{d+1,t}$. The full parameter vector $\theta_t = \{W_{\ell,t}, b_{\ell,t}\}_{\ell=1}^{d+1}$ collects all weights and biases at time t.

A standard neural network uses a fixed θ estimated on a training sample. The self-driving neural network (SDNN) instead updates θ_t at each observation using a self-driving method described in Section 2.1. The driving signal in the SDNN is the gradient of the loss function with respect to the weights and biases of the model, so no distributional assumption is needed.

2.1 Self-Driving dynamics through adaptive moments

Due to the time-varying nature of the inflation dynamics, we are in a scenario where the best model for forecasting inflation at time t may differ over time. We therefore frame the learning problem as an online optimization task. At time t, the most recently available training observation for forecast horizon h is the pair (x_{t-h}, y_t), where x_{t-h} denotes the predictor vector used to generate the forecast of y_t. Let $\ell_t : \mathbb{R}^p \to \mathbb{R}$ denote a loss function, mapping the parameter vector $\theta \in \mathbb{R}^p$ to a scalar measure of forecast error at time t. The goal is to track a time-varying

[1]For notational convenience, we drop the subscript k here and reintroduce it in Section 2.3.

parameter vector

$$\theta_t = \arg\min_{\theta} \mathbb{E}\left[\ell_t(\theta)\right], \tag{5}$$

where the expectation is taken with respect to the data-generating process at time t. Since the expected loss is unknown and observations arrive sequentially, parameters must be updated online using only the most recently available data.

A general online updating scheme takes the form

$$\theta_{t+1} = \theta_t - A\, S_t\, g_t, \tag{6}$$

where $g_t = \nabla_\theta \ell_t(\theta_t) \in \mathbb{R}^p$ is the gradient of the loss at θ_t, $A \in \mathbb{R}^{p \times p}$ is a learning rate matrix, and $S_t \in \mathbb{R}^{p \times p}$ is a matrix that scales the gradient. This general formulation nests a variety of online learning algorithms as special cases and draws an explicit parallel to score-driven models, in which time-varying parameters are updated by the scaled score of the log-likelihood. In the remainder of this section, we introduce specifications of Equation 6. These methods are common in the neural network optimization literature, but we use them in an online fashion. This means that data arrives sequentially, ordered in time, and passes through the filter only once as opposed to the more general usage of these algorithms, where data can be unordered and may be passed through the filter in batches over multiple epochs. Nevertheless, we still refer to the algorithms by the names familiar to audiences in neural network optimization, even though this slightly abuses the terminology, since we consider the algorithms exclusively in an online setting.

A first special case of (6) is stochastic gradient descent SGD, obtained by setting $A_t = \alpha I$ and $S_t = I$, yielding

$$\theta_{t+1} = \theta_t - \alpha\, g_t, \tag{7}$$

where $\alpha > 0$ is a fixed scalar step size. While computationally simple, SGD applies the same

learning rate to all parameters regardless of their historical gradient magnitudes, which can lead to slow adaptation when the optimal parameter vector shifts over time and poor convergence when gradients vary substantially across dimensions.

A second, more flexible special case is the Adam optimizer (Kingma and Ba, 2015), which constructs S_t adaptively from exponential moving averages of the gradient and its per-parameter squared values. Specifically, Adam maintains two moment estimates,

$$m_t = \beta_1 \, m_{t-1} + (1 - \beta_1) \, g_t, \quad v_t \qquad\qquad = \beta_2 \, v_{t-1} + (1 - \beta_2) \, g_t^2,$$

where $\beta_1, \beta_2 \in [0, 1)$ are exponential decay rates and $g_t^2 = g_t \odot g_t$ denotes the per-parameter squared gradient. Since both moments are initialized at zero, they are biased toward zero in early time steps. This bias is corrected via

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}.$$

The parameter update corresponds to (6) with $A_t = \alpha I$ and $S_t = \operatorname{diag}\big(1/(\sqrt{\hat{v}_t} + \epsilon)\big)$, giving

$$\theta_{t+1} = \theta_t - \alpha \, \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{8}$$

where $\alpha > 0$ is the step size, $\epsilon > 0$ is a small constant for numerical stability, and all operations apply parameter by parameter. The adaptive scaling by $\sqrt{\hat{v}_t}$ sets a separate learning rate for each parameter, yielding larger updates for parameters with historically small gradient variance and smaller updates for those with large gradient variance. Adam offers two advantages over SGD. First, $\hat{m}_t$ acts as a momentum term. By averaging past gradients, it smooths out noisy gradient estimates, when gradients oscillate across updates in SGD, they cancel in $\hat{m}_t$, resulting in more stable updates and accelerating convergence. Second, $\hat{v}_t$ provides a per-parameter estimate of curvature: parameters that have historically received large gradients accumulate a

large $\hat{v}_t$, which shrinks their effective step size via the $1/\sqrt{\hat{v}_t}$ scaling in (8). This is inspired by Newton's method, which rescales the gradient by the inverse Hessian. Adam approximates this by using $\hat{v}_t$ as a diagonal estimate of the Hessian, requiring only first-order information.

Although Adam has these advantages over SGD, it does not guarantee that each stochastic update reduces the contemporaneous loss, since the update can still overshoot the optimum. Hence we also considered a stochastic proximal point algorithm (Asi and Duchi, 2019), which guards against overshooting by solving an interior optimization problem at each update step rather than taking a single gradient step. While the proximal formulation guards against overshooting the optimum, it imposes a higher computational burden per iteration. In our application, this additional cost did not translate into improvements in forecasting performance. We therefore retain Adam as the parameter update scheme.

In this work, we define the loss function $\ell_t(\theta)$ with the Huber loss, defined as

$$\ell_t(\theta) = \begin{cases} \frac{1}{2}(y_t - \mathrm{NN}(x_{t-h};\, \theta))^2 & \text{if } |y_t - \mathrm{NN}(x_{t-h};\, \theta)| \leq \delta, \\ \delta |y_t - \mathrm{NN}(x_{t-h};\, \theta)| - \frac{1}{2}\delta^2 & \text{otherwise,} \end{cases} \tag{9}$$

where $\delta > 0$ controls the transition between the quadratic and linear regimes and $\mathrm{NN}(\,\cdot\,;\theta)$ denotes the neural network forecast function with parameter vector $\theta \in \mathbb{R}^p$. The Huber loss generalizes both the squared error loss and the absolute error loss: for $\delta \to \infty$ it converges to the squared error loss, while for $\delta \to 0$ it converges to the absolute error loss. We choose the Huber loss because inflation data periodically contains large shocks to which the pure squared error loss is very sensitive. The Huber loss downweights these large residuals linearly rather than quadratically.

2.2 Hyperparameter Configuration

The SDNN model introduced in Section 2.1 requires choosing a set of hyperparameters that governs both the network architecture and the self-driving dynamics. These include the layer

configuration, the activation function, the weight initialization standard deviation of the random normal weight initialization, the Huber loss threshold and the Adam step size, momentum parameters and parameter decay rates. Table 2 lists all hyperparameters together with their domains. We include the typical hyperparamters that are considered in the literature (Kalfa et al., 2025), except for dropout or batch-size hyperparameters as these methods commonly used in batch training violate the online setup.

The standard approach in forecasting with neural networks is to select hyperparameters by grid search over a held-out validation set, where the configuration that minimizes validation loss is selected. This is computationally demanding, and in practice the grid is often kept small to contain the cost. In a rolling-window forecast evaluation the problem grows. Hyperparameters are typically fixed at the first estimation window or re-estimated only every few windows to avoid repeated grid searches. Both strategies introduce estimation error by overfitting the validation data and either incapable or severely limited in tracking the dynamics of evolving hyperparameters.

TABLE 1: Hyperparameter grid for each SDNN candidate. The network architecture group governs the feedforward structure in Equations (3)–(4); the Adam group governs the update rule in Equation (8); the Huber group governs the loss in Equation (9); the regularization group governs the penalty terms added to the training objective.

Group	Symbol	Description	Domain
Architecture	$\boldsymbol{w} = (w_1, \ldots, w_d)$	Layer width vector; d is the depth and w_ℓ the width of hidden layer ℓ	$\mathbb{N}^d,\ d \in \mathbb{N}$
Architecture	σ	Activation function	{ReLU, Tanh, ...}*
Architecture	s_0	Weight initialization scale	$\mathbb{R}_{>0}$
Loss Function	δ	Huber loss threshold	$\mathbb{R}_{>0}$
Adam	α	Step size (learning rate)	$\mathbb{R}_{>0}$
Adam	β_1	First-moment decay rate	$[0, 1)$
Adam	β_2	Second-moment decay rate	$[0, 1)$
Adam	ϵ	Numerical stability constant	$\mathbb{R}_{>0}$
Regularization	λ_{wd}	Decoupled weight decay coefficient (AdamW)	$\mathbb{R}_{\geq 0}$
Regularization	λ_1	ℓ_1 penalty coefficient	$\mathbb{R}_{\geq 0}$

* Examples include ReLU and Tanh; the list is not exhaustive.

We take a different approach. Rather than selecting a single hyperparameter configuration, we instantiate each SDNN candidate with a random draw without replacement from the discrete hyperparameter grid, as described in Section 2.3.

2.3 Aggregation of candidates

We are in a "forecast then aggregate" scenario described by Rossi (2021). We are in a high-dimensional setting, much larger than most of the methods discussed by Rossi consider. To aggregate over large number of experts, we need a efficient method and we require the method to represented by a filter such that we do not have perform expensive reestimation on a rolling window. The methods we use are inspired by Stock and Watson (2004). Although they allow for dynamic aggregation methods, we limit ourselves to static methods. The underlying assumption is that we are looking for the best static ensemble of candidates, where one could potentially be interested in finding a dynamic ensemble of candidates. We model the time-variation within the neural network and restrict the ensemble to be static.

We generate K SDNN candidates $\{\mathrm{NN}(\cdot;\, \theta_t^k)\}_{k=1}^K$, each characterized by a distinct hyperparameter configuration drawn from the grid in Table 2, so that each candidate represents a different model architecture and adaptation speed. At each time t, candidate k produces a point forecast $\hat{y}_{t+h}^k$ for horizon h. The ensemble forecast is

$$\hat{y}_{t+h} = \sum_{k=1}^{K} w_{k,t}\, \hat{y}_{t+h}^k, \tag{10}$$

where $w_{k,t} \in \mathbb{R}$ is the weight for candidate k at time t. The key question is how to specify and update $w_{k,t}$. We discuss three categories of methods.

2.3.1 Simple Combinations: Mean and Median

The simplest specifications of (10) choose the weight independently of the historical performance of the individual experts. The *mean* combination assigns $w_{k,t} = 1/K$ uniformly, yielding

$$\hat{y}_{t+h} \;=\; \frac{1}{K}\sum_{k=1}^{K} \hat{y}^{k}_{t+h}. \tag{11}$$

The *median* combination returns the sample median of $\{\hat{y}^{k}_{t+h}\}_{k=1}^{K}$. Both methods require no parameter updates. Additionally, we will also apply this to the subsequent methods, *trimming* can be performed. Some of the candidates lead to unrealistically high forecasts. These should be omitted. Over a growing in-sample period we estimate the distribution of the target. If a candidate produces a prediction that lies outside 1.5 times the 98% interquantile range we remove it.

2.3.2 Error-Based Combination

A natural approach is to weight experts by the inverse of their accumulated forecast error. Let $\ell(\cdot)$ be any non-negative loss function of the forecast error; common choices include the squared error $\ell = (y_{s+h} - \hat{y}^{k}_{s+h})^2$ and the absolute error $\ell = |y_{s+h} - \hat{y}^{k}_{s+h}|$. Define the growing cumulative loss of expert k as

$$m_{k,t,h} \;=\; \sum_{s=1}^{t-h} \ell\big(y_{s+h},\, \hat{y}^{k}_{s+h}\big), \tag{12}$$

and set

$$w_{k,t} \;=\; \frac{m_{k,t,h}^{-1}}{\sum_{j=1}^{K} m_{j,t,h}^{-1}}. \tag{13}$$

Because the window grows with t, all past observations contribute equally and no forgetting is applied. A rigorous variant, the *recent best* (RB) rule, places all weight on the single expert with the lowest cumulative loss.

2.3.3 Recursive Least Squares Combination

We model the realisation as a linear combination of the K expert forecasts plus white noise,

$$y_{t+h} \;=\; \hat{\mathbf{y}}_{t+h}^{\top}\mathbf{w}_t \;+\; \varepsilon_{t+h}, \qquad \varepsilon_{t+h} \overset{\text{iid}}{\sim} (0, \sigma^2), \tag{14}$$

$$\mathbf{w}_{t+1} \;=\; \mathbf{w}_t \;+\; \boldsymbol{\eta}_t, \qquad \boldsymbol{\eta}_t \sim (\mathbf{0}, \mathbf{Q}), \quad \mathbf{Q} = \mathbf{0}, \tag{15}$$

where $\hat{\mathbf{y}}_{t+h} \in \mathbb{R}^K$ stacks the K expert forecasts and $\mathbf{w}_t \in \mathbb{R}^K$ are the time-varying combination weights. The weights follow a random walk with state covariance matrix equal to $\mathbf{0}$, so the prediction step becomes trivial and the Kalman filter collapses to recursive least squares with updates

$$K_t = \frac{P_{t-1}\,\hat{\mathbf{y}}_{t+h}}{\sigma^2 + \hat{\mathbf{y}}_{t+h}^{\top} P_{t-1}\,\hat{\mathbf{y}}_{t+h}}, \qquad \text{(Kalman gain)} \tag{16}$$

$$\mathbf{w}_t = \mathbf{w}_{t-1} + K_t\big(y_{t+h} - \hat{\mathbf{y}}_{t+h}^{\top}\mathbf{w}_{t-1}\big), \tag{17}$$

$$P_t = \big(I_K - K_t\hat{\mathbf{y}}_{t+h}^{\top}\big)P_{t-1}. \tag{18}$$

We initialise $\mathbf{w}_0$ and P_0 with an in-sample ridge regression. Since the gain depends only on the ratio P_{t-1}/σ^2, jointly rescaling P_0 and σ^2 leaves the weight path unchanged. We may therefore set $\sigma^2 = 1$ without loss of generality and let P_0 determine the scale.

3 Empirical Study

We evaluate the forecasting performance of the Self-Driving Neural Network Ensemble (SDNNE) against a broad set of machine learning benchmarks for U.S. inflation forecasting. The evaluation follows the framework of Medeiros et al. (2021) and Naghi et al. (2024), allowing direct comparison with established results in the literature. We compare the root mean squared error (RMSE) and mean absolute error (MAE) relative to a random walk (RW) benchmark. Ratios below one indicate improvement over the benchmark. Statistical significance of differences

in forecast performance across methods is assessed using multi-horizon model confidence set (MCS) procedures (Quaedvlieg, 2021), which identify the set of models that cannot be statistically distinguished from the best performer at a given significance level. We report uniform and average MCS sets across forecast horizons 1–24.

3.1 Data

We consider U.S. headline inflation measured as the month-on-month log-difference of the Consumer Price Index (CPI). Predictors are drawn from the FRED-MD database (McCracken and Ng, 2016), a balanced panel of macroeconomic and financial indicators updated at monthly frequency. We apply the transformation codes recommended by McCracken and Ng (2016) except for the inflation variables where we take log differences.

The full sample runs from February 1960 through August 2025. We evaluate out-of-sample forecast performance between July 2011 and August 2025. This window captures the zero-lower-bound period, the recovery phase, the pandemic recession of 2020, and the high-inflation episode of 2021–2023. We additionally report subsample results for three subperiods: July 2011 – February 2016, March 2016 – October 2020, and November 2020 – August 2025.

3.2 Forecast Design

We produce direct multi-horizon forecasts at horizons $h \in \{1, \ldots, 24\}$ months. For each horizon h, a separate model is estimated. Benchmark models are estimated in an expanding window, re-estimating parameters as each new observation arrives. SDNNE are estimated in a single pass over the sample, with parameters subsequently updated observation-by-observation. Our framework naturally allows for expanding timeseries.

The candidates of the SDNNE are specified by their hyperparameter choices. We specify a grid of hyperparameters and randomly draw without replacement unique hyperparameters until K canidates are drawn. The hyperparameter grid is specified in Table 2.

TABLE 2: Hyperparameter search grid for each SDNNE candidate.

Symbol	Description	Values
$\boldsymbol{w}$	Hidden layer widths	$(16, 8)$, $(32, 16, 8)$, $(64, 32, 16, 8)$
σ	Activation function	ReLU, Leaky ReLU, Tanh, Sigmoid
s_0	Weight initialization scale	$\{0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0\}$
δ	Huber loss quantile	$\{0.5, 0.9, 0.98, 0.99, 1.0\}$
α	Step size (learning rate)	$\{10^{-5}, 5{\cdot}10^{-5}, 10^{-4}, 5{\cdot}10^{-4}, 10^{-3}, 5{\cdot}10^{-3}, 10^{-2}, 5{\cdot}10^{-2}\}$
α_b	Step size, terminal bias[a]	$\{10^{-5}, 5{\cdot}10^{-5}, 10^{-4}, 5{\cdot}10^{-4}, 10^{-3}, 5{\cdot}10^{-3}, 10^{-2}, 5{\cdot}10^{-2}\}$
β_1	First-moment decay rate	$\{0.0, 0.5, 0.9, 0.95\}$
β_2	Second-moment decay rate	$\{0.95, 0.98, 0.99, 0.995\}$
λ_{wd}	Weight decay (AdamW)	$\{0, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$
λ_1	ℓ_1 penalty coefficient	$\{0, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$

[a] Separate step size applied to the bias of the output layer; all other parameters use α.

3.3 Main Forecasting Results

Table 3 reports relative root mean squared error (RRMSE) and Table 4 relative mean absolute error (RMAE), each computed as the ratio of model loss to the random walk loss, over the out-of-sample period. Values below one indicate improvement over the random walk.

TABLE 3: Relative RMSE by forecast horizon for CPI inflation. Direct h-step block: RRMSE of direct month-over-month forecast. Cumulative-horizon block: RRMSE of cumulative forecasts. Right block: multi-horizon MCS p-values computed over per-horizon squared losses (subscripts denote the maximum horizon $h_{\max}$ included). First data row shows RW absolute values; stars mark Diebold–Mariano tests vs. RW. Evaluation: 2011-07-01 to 2025-08-01.

	Direct horizon					Cumulative horizon				Multi-horizon MCS			
Model	$h=1$	$h=3$	$h=6$	$h=12$	$h=24$	$\Sigma_{1\text{–}3}$	$\Sigma_{1\text{–}6}$	$\Sigma_{1\text{–}12}$	$\Sigma_{1\text{–}24}$	unif.$_{12}$	avg.$_{12}$	unif.$_{24}$	avg.$_{24}$
RW	0.0026	0.0034	0.0033	0.0035	0.0040	0.0079	0.0150	0.0298	0.0656	—	—	—	—
AR	0.88*	0.77*	0.80*	0.79*	0.75*	0.72*	0.64*	0.61*	0.58*	0.23	0.00	0.26	0.00
SDNNE RLS.	**0.82***	**0.73***	**0.75***	0.75*	0.69*	**0.66***	**0.57***	0.54*	0.49*	**1.00**	**1.00**	**1.00**	0.35
SDNNE Mean	0.94	0.76*	0.78*	0.75*	**0.68***	0.72*	0.64*	0.59*	0.50*	0.65	0.00	0.98	0.07
ERT	1.30*	0.90	0.92	0.81	0.69*	1.00	0.94	0.84	0.65*	0.07	0.00	0.26	0.00
LGBoost	1.33*	0.94	0.98	0.82	0.70*	1.04	1.00	0.92	0.71*	0.06	0.00	0.20	0.00
Neural Network	0.85	0.83*	0.76*	**0.70***	0.70*	0.72*	0.63*	**0.53***	0.47*	0.56	0.02	0.82	0.14
Random Forest	0.83*	0.79*	0.78*	0.78*	0.70*	0.72*	0.66*	0.59*	**0.41***	0.13	0.00	0.98	**1.00**
RLasso	1.43*	0.95	0.99	0.88	1.00	1.07	1.01	0.93	0.81	0.05	0.00	0.02	0.00

Bold = best competitor per column, underlined = second-best (RW excluded from competition). Stars denote Diebold–
accuracy tests against the random-walk benchmark (two-sided, squared-loss): * $p < 0.05$.

The RLS combination of neural networks is on average the only method in the obtains lowest relative RMSE and MAE for all horizons except horizon 1.

Additionally to the loss comparisons relative to the random walk, we compare models with the multi-horizon model confidence set (MCS) p-values, using both uniform and average loss

TABLE 4: Relative MAE by forecast horizon for CPI inflation. Direct h-step block: RMAE of direct month-over-month forecast. Cumulative-horizon block: RMAE of cumulative forecasts. Right block: multi-horizon MCS p-values computed over per-horizon absolute losses (subscripts denote the maximum horizon $h_{\max}$ included). First data row shows RW absolute values; stars mark Diebold–Mariano tests vs. RW. Evaluation: 2011-07-01 to 2025-08-01.

	Direct horizon					Cumulative horizon				Multi-horizon MCS			
Model	$h=1$	$h=3$	$h=6$	$h=12$	$h=24$	$\Sigma_{1\text{–}3}$	$\Sigma_{1\text{–}6}$	$\Sigma_{1\text{–}12}$	$\Sigma_{1\text{–}24}$	unif.$_{12}$	avg.$_{12}$	unif.$_{24}$	avg.$_{24}$
RW	0.0019	0.0026	0.0025	0.0027	0.0029	0.0060	0.0112	0.0221	0.0469	—	—	—	—
AR	0.89*	0.77*	0.78*	0.76*	0.73*	0.72*	0.66*	0.65*	0.64*	0.03	0.00	0.23	0.00
SDNNE RLS.	**0.83***	0.70*	0.72*	0.72*	0.66*	**0.64***	**0.56***	**0.54***	0.53*	**1.00**	**1.00**	**1.00**	**1.00**
SDNNE Mean	0.93	**0.70***	0.73*	0.71*	**0.65***	0.69*	0.62*	0.61*	0.54*	0.99	0.13	**1.00**	0.43
ERT	1.27*	0.83	0.89	0.80	0.66*	0.93	0.90	0.84	0.70*	0.02	0.00	0.68	0.00
LGBoost	1.31*	0.86	0.95	0.82	0.68*	0.98	0.96	0.90	0.77*	0.02	0.00	0.35	0.00
Neural Network	0.85	0.74*	**0.71***	**0.68***	0.71*	0.64*	0.58*	0.55*	0.50*	0.91	0.21	0.99	0.21
Random Forest	0.83*	0.76*	0.76*	0.74*	0.67*	0.68*	0.63*	0.55*	**0.43***	0.02	0.01	0.99	0.43
RLasso	1.44*	0.90	0.95	0.88	1.01	1.04	0.98	0.90	0.90	0.02	0.00	0.03	0.00

Bold = best competitor per column, underlined = second-best (RW excluded from competition). Stars denote Diebold–
accuracy tests against the random-walk benchmark (two-sided, squared-loss): * $p < 0.05$.

criteria for the horizons 1 to 12 and 1 to 24. Models included in the MCS at the 10% level are indicated with an asterisk.

Based on the average version of the MCS for the squared error till 12 months ahead, all models except for the SDNNE-RLS are excluded from the model confidence set. For the other MCSs SDNNE-RLS is always among the best performing models.

Figure 1 plots the cumulative squared forecast error of SDNNE-RLS (denoted as TVNN Ensemble), AR, Random Forest and Neural Network relative to the random walk over the out-of-sample period. A declining curve indicates that the model accumulates less error than the benchmark over time.

3.4 Ensemble Methods

We aggregate forecasts from 10,000 self-driving neural network models with different hyperparameter configurations using seven ensemble methods described in Section 2.

Table 5 reports in-sample performance for forecast horizons $h \in \{1, 6, 12, 24\}$. Table 6 reports corresponding out-of-sample results over the evaluation period 2010–2024. Performance is measured by relative RMSE and MAE against the random walk benchmark. Values below one indicate improvement over the benchmark.

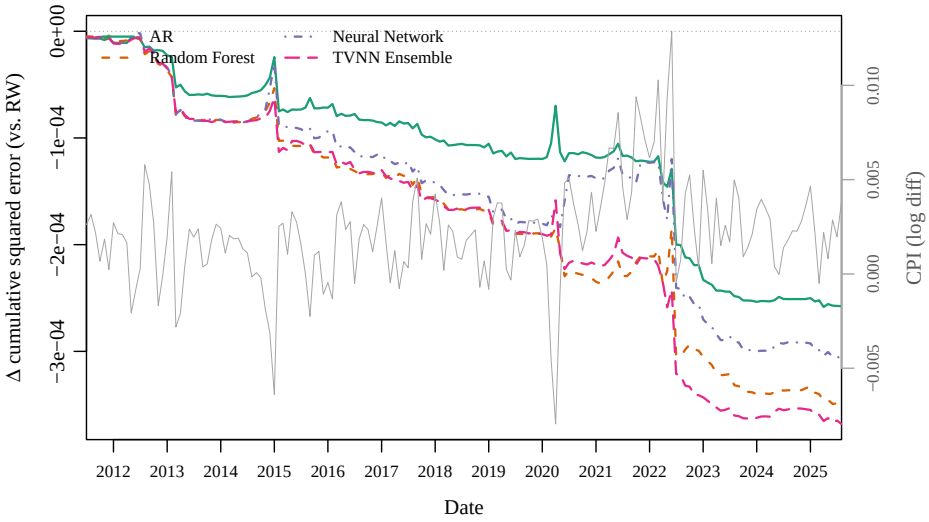

FIGURE 1: Cumulative squared forecast error of RW minus cumulative error of machine learning method , 2010–2024, for horizons $h \in \{1\}$.

TABLE 5: CPI forecast accuracy across horizons — In-sample (last 50%) period. Metrics are relative to the random-walk benchmark.

Method	$h = 1$		$h = 6$		$h = 12$		$h = 24$		Average	
	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE
Random Walk	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Mean	0.93	0.85	0.71	0.71	0.66	0.66	<u>0.71</u>	0.70	0.75	0.73
Median	0.94	0.86	0.71	0.71	0.67	0.66	0.71	**0.70**	0.75	0.73
RB-SE	<u>0.90</u>	<u>0.84</u>	0.72	0.73	0.68	0.70	0.71	0.70	0.75	0.74
RB-AE	0.94	0.84	0.71	0.71	0.67	0.67	0.71	0.70	0.76	0.73
InvMSE	0.92	0.84	<u>0.71</u>	<u>0.71</u>	0.66	0.66	0.71	0.70	<u>0.75</u>	<u>0.73</u>
InvMAE	0.92	0.84	0.71	0.71	<u>0.66</u>	<u>0.66</u>	0.71	0.70	0.75	0.73
RLS	**0.83**	**0.77**	**0.70**	**0.69**	**0.64**	**0.62**	**0.71**	<u>0.70</u>	**0.72**	**0.70**

Sample: 1985:12 – 2011:06 (in-sample (last 50%), $N = 307$). All RMSE and MAE values are normalised by the Random Walk benchmark (lower is better). The Average columns report simple means across horizons. Bold entries indicate the best-performing aggregation method and underlined entries the second-best in each column.

TABLE 6: CPI forecast accuracy across horizons — Out-of-sample period. Metrics are relative to the random-walk benchmark.

Method	$h = 1$		$h = 6$		$h = 12$		$h = 24$		Average	
	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE
Random Walk	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Mean	0.94	0.93	0.78	0.73	0.75	0.71	**0.68**	0.65	0.79	0.76
Median	0.96	0.94	0.78	0.73	0.75	<u>0.71</u>	0.69	0.65	0.79	0.76
RB-SE	0.86	0.88	<u>0.78</u>	0.77	0.76	0.71	0.69	**0.65**	0.77	<u>0.75</u>
RB-AE	<u>0.84</u>	<u>0.85</u>	0.83	0.81	0.78	0.74	0.69	0.65	0.78	0.76
InvMSE	0.93	0.92	0.78	<u>0.73</u>	<u>0.75</u>	0.71	0.68	0.65	0.78	0.75
InvMAE	0.93	0.93	0.78	0.73	0.75	0.71	0.68	0.65	0.79	0.75
RLS	**0.82**	**0.82**	**0.76**	**0.72**	**0.74**	**0.71**	<u>0.68</u>	<u>0.65</u>	**0.75**	**0.73**

Sample: 2011:07 – 2025:08 (out-of-sample, $N = 170$). All RMSE and MAE values are normalised by the Random Walk benchmark (lower is better). The Average columns report simple means across horizons. Bold entries indicate the best-performing aggregation method and underlined entries the second-best in each column.

3.5 Hyperparameter Importance

We now examine how the arbiter uses the candidate pool and what this reveals about the value of the underlying hyperparameters. Four questions organize the analysis. First, how does the arbiter allocate weight across candidates over the out-of-sample period, and is the resulting ensemble sparse or dense? Second, do the hyperparameters matter, in the sense that good and bad configurations produce a meaningful spread in forecast performance? Third, how close does the arbiter come to the best attainable combination of candidates? Fourth, do we need the full pool of 10,000 candidates? Because the candidates differ only through their hyperparameter choices, the weights assigned by the RLS arbiter provide a direct summary of which configurations contribute most to forecast performance.

We first measure how concentrated the arbiter weights are. Figure 2 summarizes this concentration in two ways. The left panel reports the effective number of candidates over time, computed as $(\sum_k |w_{k,t,h}|)^2 \,/\, \sum_k w_{k,t,h}^2$, which equals the number of candidates when weight is spread evenly and falls toward one as weight concentrates on a single candidate. The effective number is stable throughout the evaluation period and rises with the horizon, from about 2,800 candidates at $h = 1$ to roughly 3,300 at $h = 6$ and $h = 12$ and about 6,100 at $h = 24$. The

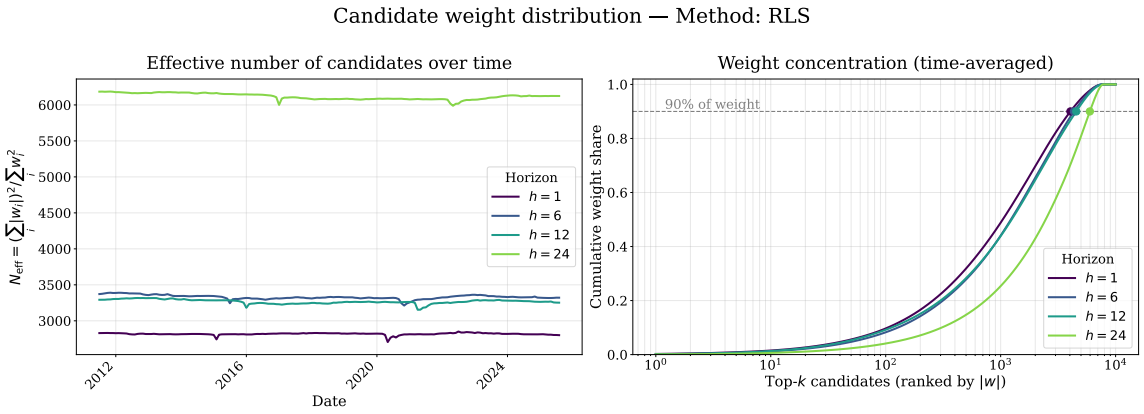

FIGURE 2: Concentration of the RLS arbiter weights, for forecast horizons $h = 1, 6, 12$ and 24. Left: effective number of candidates over the out-of-sample period, computed as $(\sum_k |w_{k,t,h}|)^2 / \sum_k w_{k,t,h}^2$. Right: time-averaged cumulative weight share against candidates ranked by $|w|$, with the 90% level marked.

one-step-ahead ensemble is therefore the sparsest, yet it still spreads weight over thousands of candidates. The right panel ranks candidates by the absolute value of their weights and plots the cumulative weight share. Reaching 90% of the total weight requires close to 4,000 candidates at horizons $h = 1$ through $h = 12$ and about 6,000 at $h = 24$. The arbiter thus builds a dense combination rather than selecting a handful of dominant configurations.

Sparsity alone does not tell us whether the candidates are worth combining. Figure 3 plots the distribution of candidate RMSE relative to the random walk for the same four horizons, with the random walk and the RLS ensemble marked. At $h = 1$ the candidates are widely dispersed and the median candidate is slightly worse than the random walk, at a ratio of 1.04, with more than 1,000 candidates falling beyond 1.5 times the random walk RMSE. At horizons $h = 6$, 12 and 24 the distribution shifts left and most candidates beat the random walk, with median ratios of 0.83, 0.80 and 0.72, though a tail of several hundred configurations remains far worse than the benchmark. The wide and horizon-dependent spread shows that the choice of hyperparameters matters for forecast accuracy. The RLS ensemble attains ratios of 0.82, 0.76, 0.74 and 0.68 across the four horizons and sits in the left tail of the candidate distribution throughout. At $h = 1$ only 25 of the 10,000 candidates beat the ensemble, while between 390 and 570 beat it at the longer horizons. Many of these superior candidates also outperform every

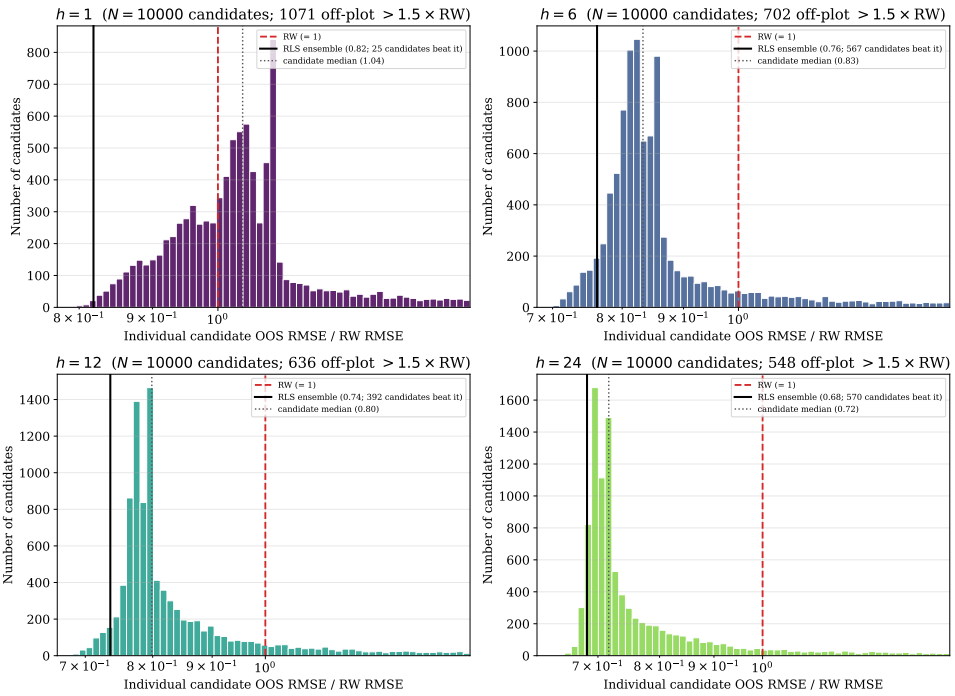

FIGURE 3: Distribution of individual candidate out-of-sample RMSE relative to the random walk benchmark, by forecast horizon. Vertical lines mark the random walk, the candidate median, and the RLS ensemble. Candidates beyond 1.5 times the random walk RMSE are omitted and counted in each panel title.

competing static machine learning method in our comparison, which shows that the self-driving neural network is very effective once it is paired with the right hyperparameters. Identifying those hyperparameters in advance is difficult.

To gauge how close the arbiter comes to the best feasible combination, we compare it against oracle ensembles. We rank candidates by their realized out-of-sample MSE and form equally weighted ensembles of the top k, for $k \in \{1, 10, 100, 1{,}000, 10{,}000\}$. These oracle ensembles use future information and are infeasible, but they bound what selection on realized accuracy can achieve. Figure 4 plots them against the RLS ensemble at each horizon. The oracle is minimized around $k = 10$, reaching ratios of about 0.79, 0.70, 0.68 and 0.62 across the four horizons, so even with perfect foresight a moderately sized combination is preferred to the single best candidate. Beyond $k = 10$ the oracle deteriorates as weaker candidates enter the equal-weight average. The RLS ensemble performs worse than the top-10 oracle at every horizon. The gap widens with

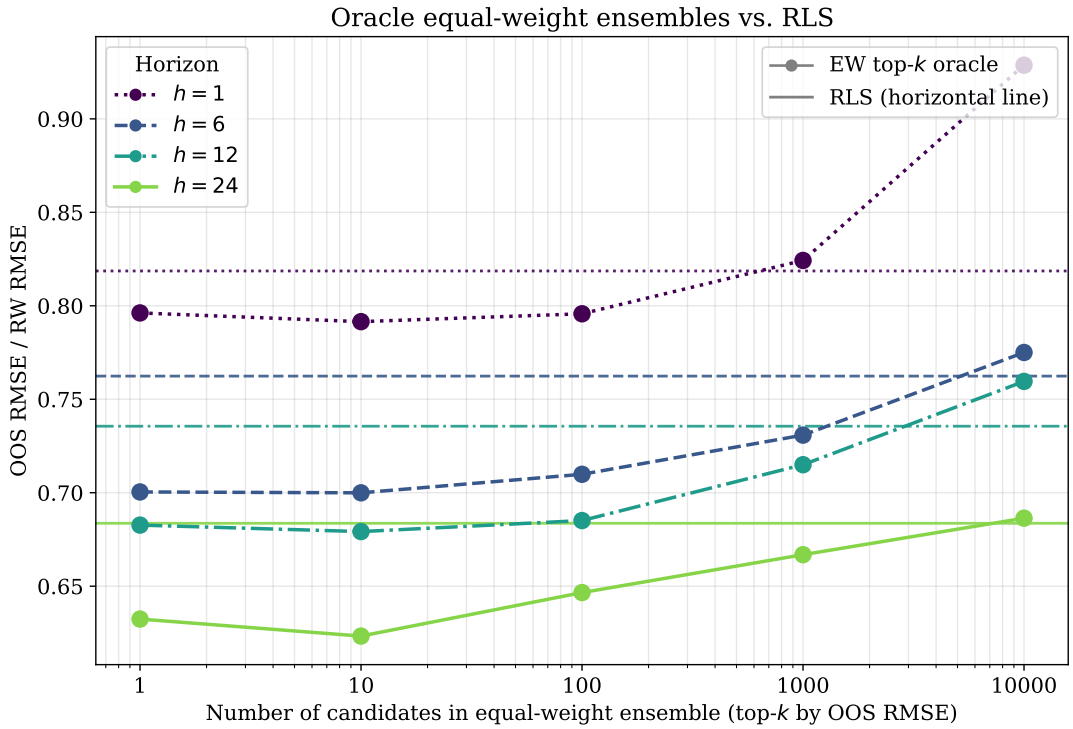

FIGURE 4: Out-of-sample performance of equally weighted oracle ensembles of the top 1, 10, 100 and 1,000 candidates, ranked by out-of-sample MSE, compared with the RLS ensemble across forecast horizons.

the horizon in a specific sense. At $h = 1$ the RLS ratio reaches the oracle that averages the top 1,000 candidates, whereas at $h = 24$ it matches the oracle that averages all 10,000. The arbiter therefore behaves more like a broad average and discriminates less sharply among candidates as the horizon grows.

Finally, we ask whether the full pool of 10,000 candidates is necessary to achieve good forecast performance. Figure 5 reports the RLS performance when the arbiter is restricted to a random subset of k candidates, with shaded bands across draws of the subset. Performance improves monotonically with the number of candidates at every horizon. The gains are largest at $h = 1$, where the ratio falls from 0.90 at $k = 10$ to 0.82 at $k = 10{,}000$, and smaller but still present at the longer horizons. The curve does not flatten into a plateau or turn upward, so adding candidates never hurts and the full pool of 10,000 delivers the best forecasts. This is a useful property in itself, because it means the arbiter can absorb an arbitrarily rich set of configurations without overfitting to the larger pool. The spread across random subsets also

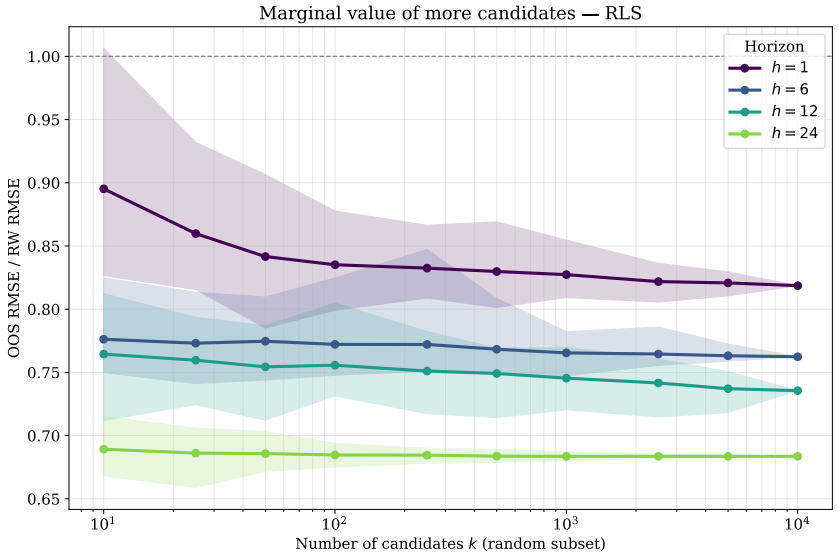

FIGURE 5: Out-of-sample RMSE of the RLS ensemble relative to the random walk benchmark when the arbiter is restricted to a random subset of k candidates, by forecast horizon. Shaded bands show the spread across random draws of the subset.

narrows as k grows, so a large pool additionally makes the ensemble less sensitive to which candidates happen to be available.

4 Conclusion

This paper develops a self-driving neural network framework for macroeconomic forecasting. The weights and biases of each network evolve over time through an observation-driven update based on Adam, so the model adapts to incoming data without re-estimation over a rolling window. We treat the hyperparameters as part of the model rather than as fixed inputs chosen in advance, and form an ensemble over 10,000 candidate configurations whose combination weights are tracked online by a recursive least squares arbiter. The framework links the machine learning and observation-driven parameter literatures, and it turns the usual problem of hyperparameter selection into a forecast combination problem.

We evaluate the framework on U.S. inflation using predictors from FRED-MD, over the 15 years from 2010 to 2024 and forecast horizons of 1 to 24 months. The RLS ensemble achieves the lowest relative RMSE and MAE at every horizon beyond the first, and it is the only method that

enters every model confidence set we consider. Against the average squared-error confidence set up to 12 months it is the single method that is included. The hyperparameter importance analysis shows why the approach works. The arbiter spreads weight over thousands of candidates with improved forecast performance compared to a single candidate choice. The method allows for rich set of configurations without overfitting and improving forecast performance as the candidate pool grows.

We have studied a single target, CPI inflation, and it would be natural to apply the framework to other macroeconomic and financial series. The self-driving update and the recursive arbiter are generic, so the approach extends beyond inflation forecasting to any setting where models must adapt online to changing conditions.

References

Artemova, M., Blasques, F., Van Brummelen, J., and Koopman, S. J. (2022a). Score-Driven Models: Methodology and Theory. In *Oxford Research Encyclopedia of Economics and Finance*. Oxford University Press.

Artemova, M., Blasques, F., Van Brummelen, J., and Koopman, S. J. (2022b). Score-Driven models: Methods and applications. In *Oxford Research Encyclopedia of Economics and Finance*. Oxford University Press.

Asi, H. and Duchi, J. C. (2019). Stochastic (Approximate) Proximal Point Methods: Convergence, Optimality, and Adaptivity. *SIAM Journal on Optimization*, 29(3):2257–2290.

Babii, A., Ghysels, E., and Striaukas, J. (2022). Machine learning time series regressions with an application to nowcasting. *Journal of Business & Economic Statistics*, 40(3):1094–1106.

Barron, A. R. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945.

Bates, J. M. and Granger, C. W. J. (1969). The combination of forecasts. *Operational Research Quarterly*, 20(4):451–468.

Bianchi, D., Büchner, M., and Tamoni, A. (2021). Bond Risk Premiums with Machine Learning. *The Review of Financial Studies*, 34(2):1046–1089.

Billio, M., Casarin, R., Ravazzolo, F., and van Dijk, H. K. (2013). Time-varying combinations of predictive densities using nonlinear filtering. *Journal of Econometrics*, 177(2):213–232.

Castro, L. d., Gonçalves, S., Medeiros, M. C., and Perron, B. (2023). Forecasting inflation time series using score-driven dynamic models and combination methods: The case of Brazil. *Journal of Forecasting*, 42(2):369–401. Verify authors – the lead author may differ; the paper is published in J. of Forecasting 2023, Vol. 42(2).

Creal, D., Koopman, S. J., and Lucas, A. (2013). Generalized autoregressive score models with applications. *Journal of Applied Econometrics*, 28(5):777–795.

Creal, D., Koopman, S. J., Lucas, A., and Zamojski, M. (2024). Observation-driven filtering of time-varying parameters using moment conditions. *Journal of Econometrics*, 238(2):105635.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

D'Agostino, A., Gambetti, L., and Giannone, D. (2013). Macroeconomic forecasting and structural change. *Journal of Applied Econometrics*, 28(1):82–101.

De Polis, A., Delle Monache, D., and Petrella, I. (2024). Modeling and forecasting macroeconomic downside risk. *Journal of Business & Economic Statistics*, 42(3):1010–1025.

Del Negro, M., Hasegawa, R. B., and Schorfheide, F. (2016). Dynamic prediction pools: An investigation of financial contagion. *Journal of Applied Econometrics*, 31(3):551–574.

Delle Monache, D. and Petrella, I. (2017). Adaptive models and heavy tails with an application to inflation forecasting. *International Journal of Forecasting*, 33(2):482–501.

Giannone, D., Lenza, M., and Primiceri, G. E. (2021). Economic predictions with big data: The illusion of sparsity. *Econometrica*, 89(5):2409–2437.

Goulet Coulombe, P. (2024). The macroeconomy as a random forest. *Journal of Applied Econometrics*, 39(3):401–421.

Goulet Coulombe, P. (2025). A neural Phillips curve and a deep output gap. *Journal of Business & Economic Statistics*, 43(3):669–683.

Goulet Coulombe, P., Frenette, M., and Klieber, K. (2026). From reactive to proactive volatility modeling with hemisphere neural networks. *Journal of Applied Econometrics*. Forthcoming. Published online January 2026.

Goulet Coulombe, P., Leroux, M., Stevanovic, D., and Surprenant, S. (2022). How is machine learning useful for macroeconomic forecasting? *Journal of Applied Econometrics*, 37(5):920–964.

Gu, S., Kelly, B., and Xiu, D. (2020). Empirical Asset Pricing via Machine Learning. *The Review of Financial Studies*, 33(5):2223–2273.

Harvey, A. C. (2013). *Dynamic Models for Volatility and Heavy Tails: With Applications to Financial and Economic Time Series*. Number 52 in Econometric Society Monographs. Cambridge University Press.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

Kalfa, S. Y., Timmermann, A., and van der Zwan, T. (2025). Overhyped? Can ML models reliably predict stock returns? Working Paper, University of California San Diego and Erasmus University Rotterdam.

Kelly, B., Malamud, S., and Zhou, K. (2024). The Virtue of Complexity in Return Prediction. *The Journal of Finance*, 79(1):459–503.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*.

Koop, G. and Korobilis, D. (2012). Forecasting inflation using dynamic model averaging. *International Economic Review*, 53(3):867–886.

Masini, R. P., Medeiros, M. C., and Mendes, E. F. (2023). Machine learning advances for time series forecasting. *Journal of Economic Surveys*, 37(1):76–111.

McCracken, M. W. and Ng, S. (2016). FRED-MD: A Monthly Database for Macroeconomic Research. *Journal of Business & Economic Statistics*, 34(4):574–589.

Medeiros, M. C., Vasconcelos, G. F. R., Veiga, , and Zilberman, E. (2021). Forecasting Inflation in a Data-Rich Environment: The Benefits of Machine Learning Methods. *Journal of Business & Economic Statistics*, 39(1):98–119.

Naghi, A. A., O'Neill, E., and Danielova Zaharieva, M. (2024). The benefits of forecasting inflation with machine learning: New evidence. *Journal of Applied Econometrics*, 39(7):1321–1331.

Primiceri, G. E. (2005). Time varying structural vector autoregressions and monetary policy. *The Review of Economic Studies*, 72(3):821–852.

Quaedvlieg, R. (2021). Multi-Horizon Forecast Comparison. *Journal of Business & Economic Statistics*, 39(1):40–53.

Raftery, A. E., Kárný, M., and Ettler, P. (2010). Online prediction under model uncertainty via dynamic model averaging: Application to a cold rolling mill. *Technometrics*, 52(1):52–66.

Rossi, B. (2021). Forecasting in the Presence of Instabilities: How We Know Whether Models Predict Well and How to Improve Them. *Journal of Economic Literature*, 59(4):1135–1190.

Stock, J. H. and Watson, M. W. (1996). Evidence on structural instability in macroeconomic time series relations. *Journal of Business & Economic Statistics*, 14(1):11–30.

Stock, J. H. and Watson, M. W. (2004). Combination forecasts of output growth in a seven-country data set. *Journal of Forecasting*, 23(6):405–430.

Stock, J. H. and Watson, M. W. (2007). Why has U.S. inflation become harder to forecast? *Journal of Money, Credit and Banking*, 39(s1):3–33.

Timmermann, A. (2006). Forecast combinations. In Elliott, G., Granger, C. W. J., and Timmermann, A., editors, *Handbook of Economic Forecasting*, volume 1, pages 135–196. Elsevier.