BANCO CENTRAL DE RESERVA DEL PERÚ

FORECASTING INFLATION WITH A FRAMEWORK FOR MODEL AND NEURAL ARCHITECTURE SEARCH WITH TREE-STRUCTURED SEARCH SPACES

Andrew R. Garcia y Marco Vega*

* Banco Central de Reserva del Perú.

DT. N°. 2024-009
Serie de Documentos de Trabajo
Working Paper series
Mayo 2024

FORECASTING INFLATION WITH A FRAMEWORK FOR MODEL AND NEURAL ARCHITECTURE SEARCH WITH TREE-STRUCTURED SEARCH SPACES

Andrew R. Garcia
garcia.gtr@gmail.com

Marco Vega
Banco Central de Reserva del Perú
Pontificia Universidad Católica del Perú
marco.vega@bcrp.gob.pe

ABSTRACT

This study automates the design of machine learning models for economic forecasting, with an application focus on Peru's inflation. Such is achieved by employing an Automated Machine Learning (AutoML) framework that selects the best model configurations and data processing steps. This allows us to build models without manually trying out different options, saving time and potentially improving accuracy. The specific models explored are deep learning neural networks, which are machine learning models often used for complex forecasting tasks. We use two inflation forecasting schemes: one using a single model for headline inflation and another using two models one for food and energy inflation and another for inflation excluding food and energy, which are combined to predict inflation. By establishing this automated approach, we pave the way for further research on using machine learning to forecast economic data like inflation in Peru.

1 INTRODUCTION

Accurate inflation forecasting plays a vital role in guiding monetary policy decisions by central banks. These forecasts serve as crucial tools for navigating the complexities of economic policy-making. Traditionally, central banks have relied on various models to predict inflation, with a primary focus on structural macroeconomic models. These models capture the intricate relationships between key economic variables, such as economic activity, interest rates, and external factors like commodity prices and capital flows.

However, structural models typically used for forecasting are not suited to deal with nonlinearities that may arise during abnormal times. For example, after the post-COVID inflation surge, some studies (see for example Benigno and Eggertsson 2023; Benigno and Eggertsson 2024) showed evidence that Phillips curves had become slanted so that when there were labor shortages, the Phillips curve would steepen. This feature amplifies the effects of aggregate demand and supply shocks on inflation. Likewise, Bernanke and Blanchard forthcoming had shown that labor market overheating induces shocks to have more persistent effects on inflation than during normal times.

Machine learning models are becoming increasingly important at this junction due to their potential for unlocking highly nonlinear behavior in real-time. Nonetheless, just as typical macroeconomic forecasting models need a set of inputs and assumptions besides the usual parameterizations, machine learning models need the modeler to specify hyperparameters that define distinct forecasting models, such as the number of layers in a Multilayer Perceptron (MLP) model or their corresponding number of nodes.

To overcome the hassles stemming from hyperparameter selection, this work introduces a framework that leverages Tree Parzen Estimators (TPE) for hyperparameter optimization, targeting the automated holistic design of machine learning models (AutoML).

As shown in Figure 1, the proposed AutoML consists of an initialization stage in the outermost section of the flowchart, where user-defined Knowledge is sent to the hyperparameter optimization framework inside the boxed section, which finds the most optimized machine learning model and

hyperparameters thereof. For the case of employing TPE as the framework, such Knowledge consists of a search space, which is the chosen set of hyperparameters to change with their corresponding exploration ranges, which imposes a constraint on the variable space to be explored by the machine.

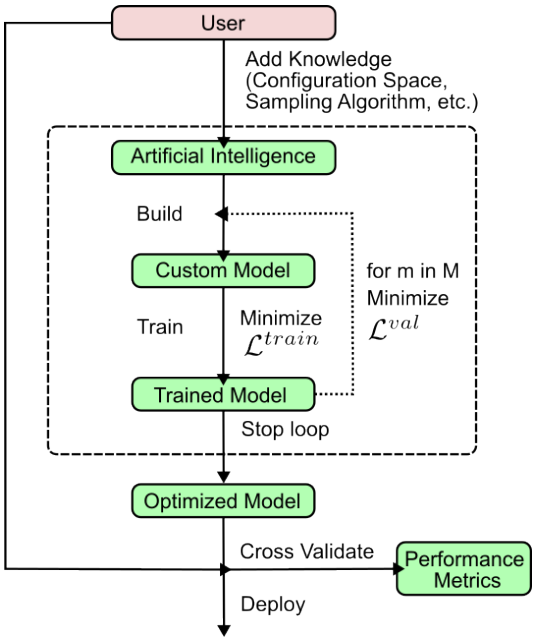


Figure 1: Proposed Automated Machine Learning (AutoML) framework for model-agnostic hyperparameter optimization. The outermost section depicts the initialization stage, where user-defined knowledge about the hyperparameter search space is provided to the optimization framework.

TPE employs Bayesian optimization to fit probability densities for hyperparameters within a defined search space (Bergstra, Bardenet, et al. 2011), creating one distribution for hyperparameters resulting in losses above a threshold and another for those below. This process allows for the iterative improvement of model performance. TPE's compatibility with tree-structured search spaces (Figure 2) enables a refined optimization process, where hyperparameters can be dependent on the values of others, allowing for a nuanced and efficient search for the optimal models.

The TPE's tree-structured approach facilitates a targeted search within the hyperparameter space and significantly reduces the computational complexity compared to regular search spaces. Taking $\mathbb{S}$ as the isolated search space of any individual hyperparameter, while an unfolded search space has a search complexity of $\mathcal{O}(\mathbb{S}^{|\lambda|})$ that scales exponentially with respect to the $|\lambda|$ number of hyperparameters, the search complexity on a tree-structured space is $\mathcal{O}(\mathbb{S}^{D})$, and scales exponentially with the D depth of the tree, often offering a more efficient optimization process as it is likely that D is significantly smaller than $|\lambda|$. Leveraging a tree-structured search space within the TPE method thus facilitates a more efficient selection of hyperparameters by exploiting contingent relationships within hyperparameters.

Though our application of Tree Parzen Estimators (TPE) broadly encompasses hyperparameter tuning for both machine learning and deep learning models, we also extend its use to neural architecture search (NAS) problems. For these NAS problems, we have devised a unique tree-structured search space that incorporates a comprehensive array of hyperparameters, which not only address learning aspects but also the network topology, as shown in Figure 3. Thus for the NAS part of the work we have developed tree structures for hyperparameters linked to the inclusion of architectural blocks, as well as distinctive tree structures designed for the layer topologies of our base neural networks, in a manner that simulates dynamic sampling of layers and internal hyperparameters thereof.

For the base network, we employ Multilayer Perceptron (MLP) blocks. Choosing an MLP-based approach strikes a balance between simplicity and forecasting efficacy, as MLPs are fundamental components of many modern neural network architectures, including N-BEATS and NHITS (Challu et al. 2023; Oreshkin et al. 2019). Notably, simpler neural networks like MLPs have been demonstrated

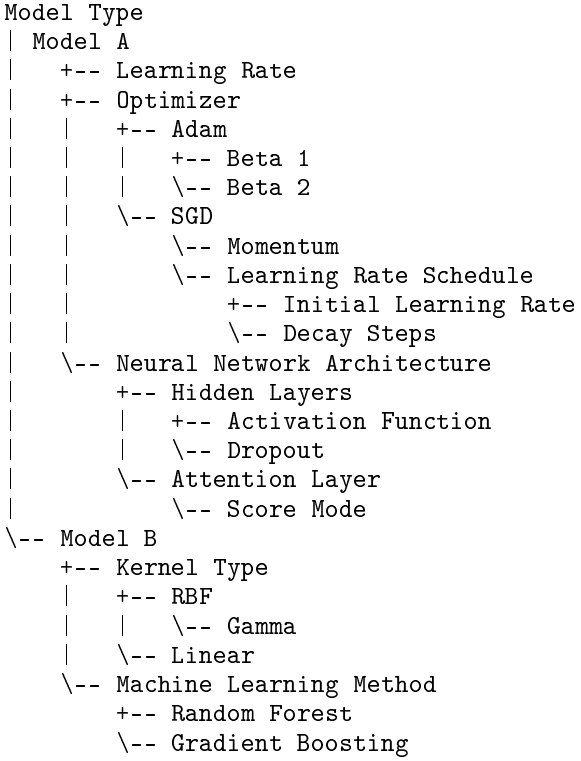


Figure 2: A tree-structured search space for a hypothetical hyperparameter optimization routine.

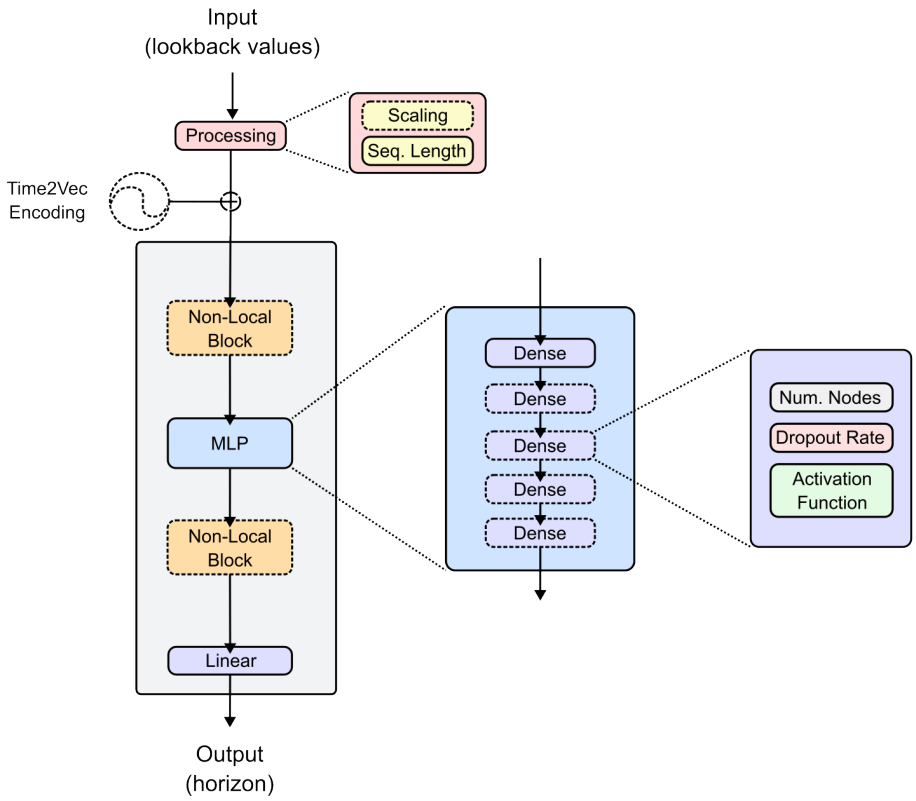


Figure 3: Template for dynamic assembly of custom neural networks. Optional blocks enclosed in dashed boxes, such as Time2Vec Encoding and Non-Local Blocks, can be dynamically included or excluded during the TPE search. Core features include input processing, modifiable dense layer stacks, and an output layer, all designed for efficient experimentation in neural architecture searches.

to achieve comparable or even superior forecasting accuracy relative to more complex models (Makridakis, Spiliotis, and Assimakopoulos 2020). The application of Tree Parzen Estimators (TPE) in this study represents a notable advancement in the field of inflation forecasting. By employing TPE and tree-structured search spaces for both hyperparameter optimization and neural architecture search, we present a sound methodology to tune machine learning models and configure neural network architectures dynamically.

In the next section, we delve into the related work that has shaped and informed our approach to forecasting. This exploration is crucial not only to understand the current landscape of machine learning applications in macroeconomic forecasting but also to highlight the innovative contributions of our study in utilizing Tree Parzen Estimators (TPE) for both hyperparameter optimization and dynamic neural architecture search.

2 Method

In this work, we make use of different isolated tasks $\mathcal{T} = \{\mathcal{D}, \mathcal{L}\}_{i=1}^{N}$ to forecast inflation through a neural architecture search based on hyperparameter optimization. Each of these tasks consists of a dataset $\mathcal{D} = \{(y_i, \mathrm{X}_i)\}_{i=1}^{N}$ to develop and evaluate the performance of predictive models through the use of loss functions $\mathcal{L}(\mathcal{D}; \theta, \lambda)$ which measure the prediction performance on the data $\mathcal{D}$, given a vector of parameters θ for a specific neural architecture, and a vector of hyperparameters λ which design the architecture. In the above specification, N is the number of time series observations, y_i is

the output (horizon) series, and X_i is the autoregressive input series, as appreciated more graphically in Figure 4. Taken from the notation by Staněk 2023, the loss function $\mathcal{L}$ is expressed in the same way for all tasks:

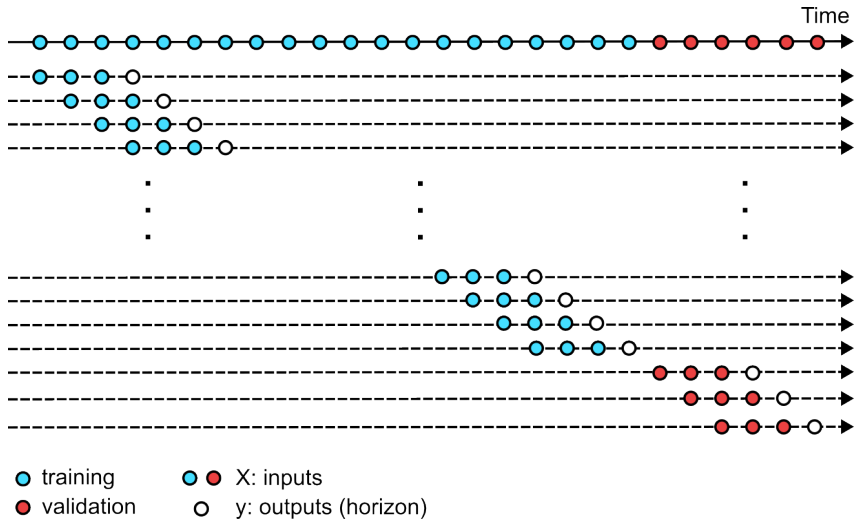


Figure 4: Data partitioning for training and validation sets. Each sample consists of a series of autoregressive inputs stored in an X vector and horizon outputs stored in a y vector.

$$\mathcal{L}(\mathcal{D}; \theta, \lambda) = \frac{1}{N} \sum_{i=1}^{N} \gamma(y_i, f_{\mathcal{M}}(\mathrm{X}_i; \theta)) \tag{1}$$

Where in our case, $f_{\mathcal{M}}$ is the forecasting function for a given $\mathcal{M}$ model with λ hyperparameters, and γ is a measure of error between the prediction made by $f_{\mathcal{M}}(\mathrm{X}_i; \theta)$ and its corresponding y_i realization or ground truth.

A typical neural architecture search (NAS) takes the form of a nested optimization problem (C. Zhang, Ren, and Urtasun 2020) where the inner loop finds the optimal parameters $\hat{\theta}$ for a given $\mathcal{M}$ architecture model with λ hyperparameters that design it such that the training loss $\mathcal{L}^{\text{train}}$ is minimized, while the outer loop searches the optimal architecture hyperparameters $\hat{\lambda}$ with respect to a validation loss $\mathcal{L}^{\text{val}}$:

$$\hat{\lambda} = \arg\min_{\lambda} \mathcal{L}^{\text{val}}(\mathcal{D}; \hat{\theta}, \lambda) \tag{2}$$

$$\text{s.t.:} \quad \hat{\theta} = \arg\min_{\theta} \mathcal{L}^{\text{train}}(\mathcal{D}; \theta, \lambda) \tag{3}$$

With this in mind, the proposed neural architecture search uses Tree Parzen Estimators (TPE) to optimize the expected improvement (EI) criterion for the best $\mathcal{M}$ neural model, and though this optimization problem also generalizes to Equations 2 and 3, it may be specified to the below logic:

$$\mathcal{M}_{n+1} = \arg\max \mathrm{EI}_n(\mathcal{M}_n(\hat{\theta}, \lambda)) \qquad \text{for } n = 1, \ldots, N_M \tag{4}$$

$$\text{s.t.:} \quad \hat{\theta}^{(\mathcal{T})} = \arg\min_{\theta} \mathcal{L}^{\text{train}}(\mathcal{D}; \theta, \lambda)|_{\mathcal{T}}, \qquad \text{for } \mathcal{T} \in \mathrm{Range}(N_{\mathcal{T}}) \tag{5}$$

Herein, the inner loop finds the optimal parameters $\hat{\theta}^{(\mathcal{T})} \in \mathbb{R}^{N_{p,\mathcal{T}}}$ for every $\mathcal{T}$ task (Equation 5), the collection of which makes an $\mathcal{M}$ model. Meanwhile, the outer loop (Equation 4) builds an

open-ended $\mathbb{M} \in \mathbb{R}^{N_M}$ pipeline of models as it searches for the optimal hyperparameters $\hat{\lambda} \in \mathbb{R}^{\Sigma N_{hp}}$, where $\Sigma N_k := \sum_{\mathcal{T}=1}^{N_\mathcal{T}} N_{k,\mathcal{T}}$. As Equation 4 defines it, the outer loop is iterative in TPE, evaluating a proposed $\mathcal{M}_n = \mathcal{M}_n(\hat{\theta}, \lambda)$ model with the previous best in $\mathbb{M}$ to generate the next $\mathcal{M}_{n+1}$ model for every n iteration. As defined by Bergstra, Bardenet, et al. 2011, the expected improvement is the expectation that our y loss metric will exceed negatively over a mutable loss threshold $y*$ when evaluated against a model with specific λ hyperparameters:

$$\mathrm{EI}_n(x) = \int_{-\infty}^{y*} (y^* - y) p(y|x)\, dy \tag{6}$$

$$y = \mathcal{L}^{\text{val}}(\mathcal{D}_n; \hat{\theta}_n, \lambda) \tag{7}$$

With the abovementioned optimization problem, two distinct optimization strategies are used for automating the design of machine learning models and conducting neural architecture searches to refine forecasting systems. These strategies are tailored to either model headline inflation from a single vector output or model it from two vector outputs through a nuanced component-based inflation optimization, leveraging the hyperparameter optimization framework to minimize validation loss (Equation 2) while concurrently minimizing the corresponding training loss (Equation 3).

2.1 Optimization strategies

Univariate headline inflation optimization

With this optimization strategy, our model architecture is streamlined to directly predict headline Consumer Price Index (CPI) inflation with a univariate model, without considering an explicit decomposition into its main two or more components and thus employing a single $|\mathcal{T}| = 1$ task. In this approach, the forecasting function $f_\mathcal{M}$ from Equation 1 for the losses of both the outer and inner loops (Equations 2 and 3) is expressed as

$$\Pi_t = \Pi_t(\mathrm{X}; \theta, \lambda) \tag{8}$$

where Π_t represents headline inflation prediction based on model parameters θ and hyperparameters λ. To reiterate, this strategy focuses on optimizing the forecasting accuracy for headline inflation Π_t by exploring hyperparameter configurations $\lambda \in \mathbb{R}^{N_{hp}}$ which best minimize the error of the validation set $\mathcal{L}^{val}$ obtained from a trained model with parameters $\hat{\theta} \in \mathbb{R}^{N_p}$ which best minimize the training set error $\mathcal{L}^{train}$.

It is worth noting that in the hyperparameter optimization routine, the outer loop (Equation 4) first sends a proposal for an $\mathcal{M}$ model with a specific hyperparameter configuration λ to the inner loop (Equation 5) to train it. After the inner loop is run, the learned model with adjusted parameters $\hat{\theta}$ is sent back to the outer loop to optimize from the $\mathbb{M}$ collection of models with different hyperparameter configurations λ.

Component-based inflation optimization

In contrast to the former, the component-based inflation optimization strategy employs $|\mathcal{T}| = 2$ tasks and is thus designed to forecast headline inflation from the weighted sum of two primary inflation components, the Π_t^{sae} inflation component which excludes food and energy and its Π_t^{ae} counterpart, which includes it. This objective is met by employing a single TPE hyperparameter search stream in the inner loop, which forks into two model design streams that build the two separate models from the chosen hyperparameter configuration list found in the initial search stream:

$$\Pi_t^{sae} = \Pi_t^{sae}(\mathrm{X}^{(1)}; \theta^{(1)}, \lambda^{(1)}) \tag{9}$$

$$\Pi_t^{ae} = \Pi_t^{ae}(\mathrm{X}^{(2)}; \theta^{(2)}, \lambda^{(2)}) \tag{10}$$

After building the two models in the inner loop, the forecasting function $f_{\mathcal{M}}$ of the aggregated headline inflation Π_t is obtained as the weighted sum of its components from Equations 9 and 10 to be evaluated in the outer loop:

$$\hat{\Pi}_t(\mathrm{X}; \hat{\theta}, \lambda) = W_c \Pi_t^{sae} + (1 - W_c)\Pi_t^{ae} \tag{11}$$

Where the weight W_c is known and the parameter and hyperparameter sets of both machine learning systems are lumped as $\hat{\theta} \in \mathbb{R}^{\Sigma N_p}$ and $\lambda \in \mathbb{R}^{\Sigma N_{hp}}$, respectively. As it was for the former univariate headline inflation optimization, it is worth noting that here the routine also begins by having the outer loop send the inner loop a hyperparameter configuration λ proposal, where in this case λ contains hyperparameters specific to each neural network, as well as global hyperparameters that may be shared for both.

2.2 Economic interpretation with gradients

To enhance our understanding of the nonlinearities of trained machine learning models, we explored the impact of variations in the last input on prediction through gradient analysis. This method quantifies the sensitivity of the predicted inflation rate to changes in the most recent input data. Namely, we calculate first-order autocorrelations, which are no longer constant values, as opposed to linear models.

We define the gradient of the predicted value y^{pred} with respect to the last input value x_N from the input vector $X \in \mathbb{R}^N$ as follows:

$$\frac{\partial y^{pred}}{\partial x_N} := \frac{y^{pred}(X + \varepsilon K) - y^{pred}(X)}{\varepsilon} \tag{12}$$

$$K = \delta_{iN}\mathbf{e}_i = [0, 0, \cdots, 0, 1] \tag{13}$$

where $K \in \mathbb{R}^N$ is a sparse vector expressed in Einstein notation, δ_{ij} is the Kronecker delta. The gradient measures the change in predicted inflation resulting from a small perturbation $\varepsilon \in [-2\sigma, -\sigma, \sigma, 2\sigma]$ in the latest input data, where σ is the standard deviation of the complete time series.

In linear models, $\frac{\partial y^{pred}}{\partial x_N}$ is a constant coefficient, irrespective of the value of x_N. This exercise will assess how the specified gradient changes under varying values of x_N observed in time.

2.3 Hyperparameter Search Space for NAS Models

The hyperparameter search space shown in Table 1 encompasses a comprehensive range of parameters used to automate the whole neural network development process, from its construction through a TPE-powered neural architecture search (NAS) to its out-of-sample (OOS) testing and deployment. These hyperparameters thus include network design elements, learning algorithms, and data-processing parameters.

For practical purposes, we have divided these hyperparameters into Global and Network-specific sections, where global hyperparameters are applied to all neural networks in our model, while network-specific hyperparameters are exclusive to each of the neural networks selected in the optimization problem. For instance, in the component-based CPI strategy, since its outer-loop forecasting function (Equation 1) is dependent on outputs from two different neural networks (Equation 11), two exclusive sets of network-specific hyperparameters are defined per each initialization of the hyper-optimization outer loop (Equation 4).

Within the Global section on Table 1 there are four sub-sections containing hyperparameter sets, where all hyperparameters are separate in the selection process and do not bear any children hyperparameters. The first of these, the random seed, is a hyperparameter itself, and was tuned to escape underperforming local minima, as was done in the hyperparameter tuning of other neural network articles (Olivares et al. 2023; Challu et al. 2023). The second sub-classification is a set containing three hyperparameters for the Adam optimizer, a Keras method for the Adam algorithm of

Table 1: Considered hyperparameters for neural architecture design.

Hyperparameter	Detail	Configuration Space
	Global	
Random Seed for Initialization		DiscreteRange(1,100)
Adam Optimizer	Learning Rate	LogRange(-4, -2)
	β_1	Range(0.85, 0.95)
	β_2	Range(0.995, 0.999)
EarlyStopping Callback	Patience	DiscreteRange(20, 100, 5)
	Restore Best Weights	True
ReduceLROnPlateau Callback	Factor	0.5
	Patience	DiscreteRange(5, 20, 5)
	Min LR	0.0001
	Network-Specific	
Batch Size		DiscreteRange(1,10)
Input Size		DiscreteRange(3,40)
Activation Function		{None, ReLU, LeakyReLU, Swish}
L2 Layer Weight Regularization*		{False, True}
Neural Network Topology*		$\{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5\}$
Non-Local Blocks, Pre-MLP*		{False, True}
Non-Local Blocks, Post-MLP*		{False, True}
Time Encoding*		{False, True}
Scalers		{False, True}

Note: Hyperparameters tagged with an asterisk (*) have children hyperparameters (shown in Table 2) efficiently sampled with the TPE method due to its tree structure.
* Hyperparameters with an asterisk have additional hyperparameters, as detailed in subsequent tables.

gradient-based stochastic optimization (see Kingma and Ba 2017). Such hyperparameters are the optimizer's learning rate and its two exponential decay rates, which are critical in designing how the neural network learns. The remaining two sub-sections from the Global section are Keras callbacks, which are basically objects that can perform actions during the learning process itself, both of which monitor the training at every single epoch. The first of these, the `EarlyStopping` callback, stops the training after a certain threshold is met, while the second, the `ReduceLROnPlateau`, is a learning rate scheduler that adjusts the Adam learning rate according to some pre-determined settings. Both of these callback sub-sections have some hyperparameters listed in Table 1 as single values rather than ranges and are thus fixed to those values. The only hyperparameters with ranges within these callbacks are the Patience hyperparameters, which are both the number of epochs after which the callback action described for both callbacks above will be enforced if there is no improvement in the validation loss.

The Network-Specific section on Table 1 encompasses a more complex variety of hyperparameters. In contrast to the hyperparameters found in the former section, some hyperparameters found here have children hyperparameters. As we are dealing with tree-structured search spaces, here we use terminology from tree data structures, specifically parent and child nodes. In this context, *parent hyperparameters* are those whose selected values initiate the definition of *children hyperparameters*. These dependent hyperparameters may be chosen from predefined search spaces or assigned fixed values, to simplify the configuration process in our model. To clarify this relationship visually, we have marked parent hyperparameters with single asterisks and their corresponding children directly below them with double asterisks.

To enable the NAS algorithm to design more complex architectures, we have incorporated options for non-local blocks and Time2Vec encoding into the hyperparameter search space, as detailed in Table 1. These options are briefly explained below.

Non-local blocks (NLBs)

These neural components are considered because they capture long-range dependencies in data similar to the self-attention mechanism used in machine translation Xiaolong Wang et al. 2018; Vaswani et al. 2023, and are thus incorporated as a design option in our neural network architectures. The inclusion of NLBs, either before or after the base MLP network, depends on their selection by the TPE algorithm. If selected, the compression rate of the NLB is determined. As can be seen in Figure 5, these custom NLBs use `MaxPooling1D` operations to reduce memory usage and potentially enhance noise reduction and generalization capabilities. The pooling size, L_{pool}, is variable and determined by the compression rate R_{NLB}, calculated as $L_{\text{pool}} = \text{int}(R_{\text{NLB}} \times L_{\text{in}})$, where L_{in} is the input length to the `MaxPooling1D` operation.

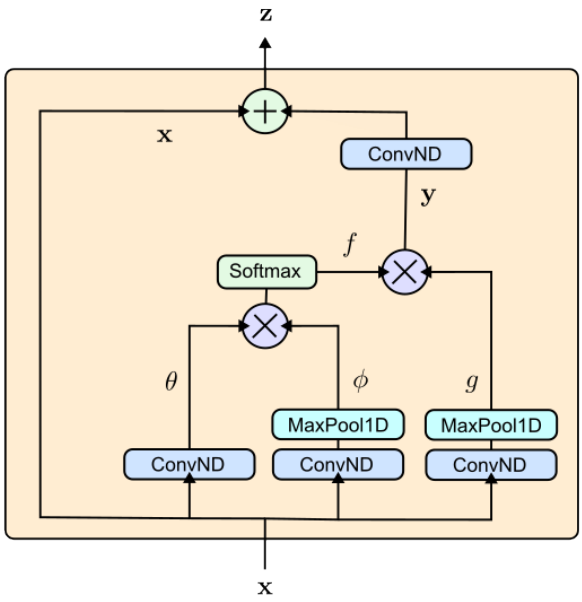


Figure 5: Custom Non-local Block architecture with `MaxPooling1D` pooling operations.

Time2Vec encoding

Time2Vec encoding (see Kazemi et al. 2019) is an innovative method for embedding temporal information, which enhances the model's ability to capture time-based patterns and dependencies. The inclusion of Time2Vec as an option in our hyperparameter search space potentially allows the NAS algorithm to explore architectures that leverage temporal dynamics more effectively, improving forecasting accuracy for time-sensitive data. Figure 6 illustrates the conceptual design of the Time2Vec encoding used in our models. The flexibility to select specific harmonics within the Time2Vec block highlights the depth and adaptability of our optimization process.

Building on these foundational enhancements, we next delve into the specifics of network topology hyperparameters.

MLP neural network topology

Table 2 consolidates the set of MLP hyperparameters from Table 1 under an MLP Features category. This category includes an Activation Function hyperparameter, applied uniformly across all neural layers of the chosen topology. It also includes an L2 Regularization option which, when selected, imposes an L2 regularization penalty on all layers with a specific strength determined by sampling

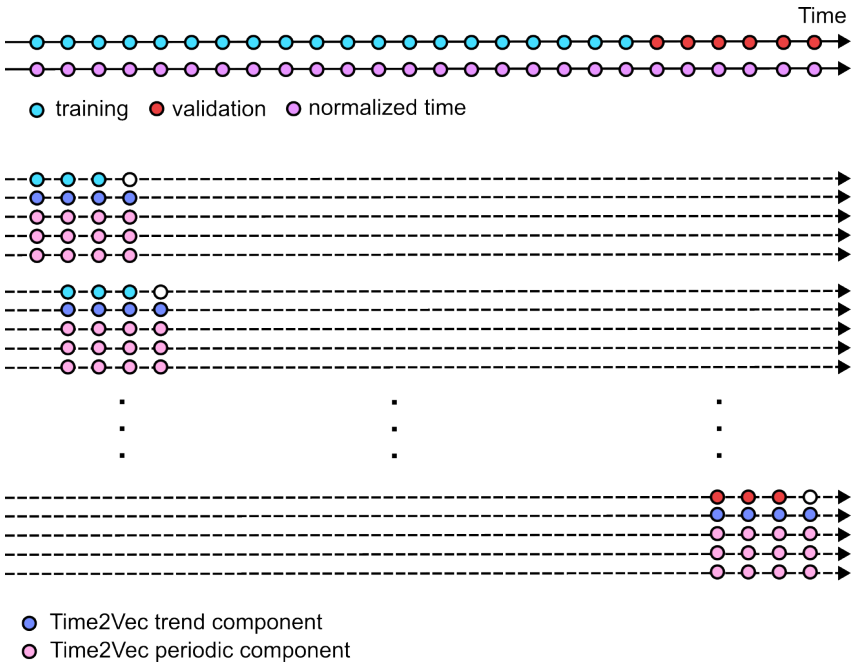


Figure 6: Data partitioning with Time2Vec encoding. Unlike conventional systems (Figure 4), normalized time values are integrated with time series data into the neural network as a complete, uninterrupted tensor, indirectly labeling each time series value with a notion of time. This tensor is then processed into batches, and the Time2Vec block extracts trend and periodic component vectors from the normalized time component.

its child hyperparameter. L2 regularization is employed due to its efficacy in preventing overfitting and ensuring model stability, particularly in configurations with numerous kernels, thus enhancing robustness and generalizability.

Lastly, there is the Neural Network Topology hyperparameter, which comprises of five distinct network topology options per each evaluation. By looking closely at the loop in this particular setting we can see that each τ_j neural network topology has one additional layer added to it. So, the number of layers in each topology matches its position (j) and ergo, generating a $\mathbb{T}$ set of topologies as the one shown in Figure 7 to be selected for every TPE evaluation.

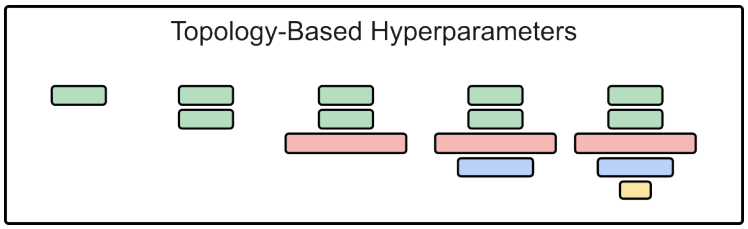


Figure 7: Static generation of a $\mathbb{T}$ set of neural network topologies for non-LayerNAS methodology Table 2

This clever static design was developed to simulate a dynamic TPE hyperparameter selection process because TPE does not allow its hyperparameter space to be designed dynamically. Perhaps a downside to this static design in the $\mathbb{T}$ set of neural network topologies would be an increase in memory, which may need to be observed for the case of many layers and/or multiple children hyperparameters for each parent Layer. Another challenge in our approach concerns the theoretical search complexity. Despite the practical reductions achieved through the Tree Parzen Estimator (TPE), theoretically, the complexity of the search space could still scale exponentially with the L number of layers, as represented by $\mathcal{O}(\mathbb{S}^L)$.

Table 2: Inner hyperparameters within the network-specific categories from Table 1.

Detail	Configuration Space
MLP Layer Features	
Activation Function	{None, ReLU, LeakyReLU, Swish}
L2 Layer Weight Regularization*	{False, True}
L2 Strength**	LogRange(-5, -1)
Neural Network Topology*	$\{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5\}$
For each Layer $i \in \{1, \ldots, j\}$, for each τ_j Neural Network Topology	
Number of Nodes in Layer i**	DiscreteRange(1, 50)
Non-Local Blocks	
Non-Local Blocks, Pre-MLP*	{False, True}
Compression Rate**	Range(0, 1)
Non-Local Blocks, Post-MLP*	{False, True}
Compression Rate**	Range(0, 1)
Time Encoding with Time2Vec (t2v)	
Time Encoding*	{False, True}
Number of Trend Terms**	1
Number of Harmonics**	DiscreteRange(4, 100, 2)

* Hyperparameters with an asterisk have additional hyperparameters.
** Hyperparameters or sets with double asterisks are the children of said hyperparameters or children sets thereof.

While TPE effectively reduces practical search complexity by focusing exploration on promising regions of the search space, the theoretical complexity under an extensive search scenario may remain comparably high. This discrepancy indicates that while TPE is efficient, it might cover a smaller portion of the total search space compared to a more layer-conscious approach like LayerNAS, recently developed by Fan et al. 2023. To address this potential limitation future work will explore the integration of LayerNAS with TPE, potentially enhancing the breadth of the search space coverage while maintaining the efficiency gains of TPE.

2.4 Optimization and out-of-sample testing stages

The optimization of hyperparameters was fundamentally aimed at minimizing the validation loss, $\mathcal{L}^{val}$, as delineated by Equation 2. More specifically, within the TPE framework, this optimization sought to maximize the expected improvement (EI) in validation loss (Equation 6). This approach thus involved a comprehensive exploration of $\mathcal{M}$ model candidates built by various hyperparameter configurations and was facilitated by a systematic selection process governed by Equations 4 to 7, where the hyperparameter search space is optimized by progressively minimizing the $\mathcal{L}^{val}$ validation loss of these candidates. For consistency, 300 model candidate trials with the same data sets for $\mathcal{L}^{train}$ and $\mathcal{L}^{val}$ were run in the hyperparameter optimization step of these models.

For the hyperparameter optimization stage, as shown in Figure 8, we divided the data into training and validation sets, while reserving a separate set for testing in the out-of-sample stage. Model training involved establishing relationships within several sets of x-y samples, specifically training the relationships between X autoregressive inputs and y horizon outputs, as depicted in Figure 4. The validation of model performance, $\mathcal{L}^{val}$, entailed comparing actual y inflation figures with the y_{pred} model forecasts derived from the validation dataset. This comparison was conducted using a forecasting function, in accordance with the validation loss minimization strategy outlined in Equation 1.

For the testing stage, a pseudo-out-of-sample (POOS) “walk-forward" cross-validation technique was used (Figure 8), designed to closely mimic a real-world scenario of making predictions with a limited historical dataset. To counter concerns about potential overfitting, this method uses the information held out during the hyperparameter optimization stage to test the optimized models in an unexplored

space. In the POOS method, the time series data is partitioned in a way that simulates standing at a specific point in the past, with all subsequent data considered unknown.

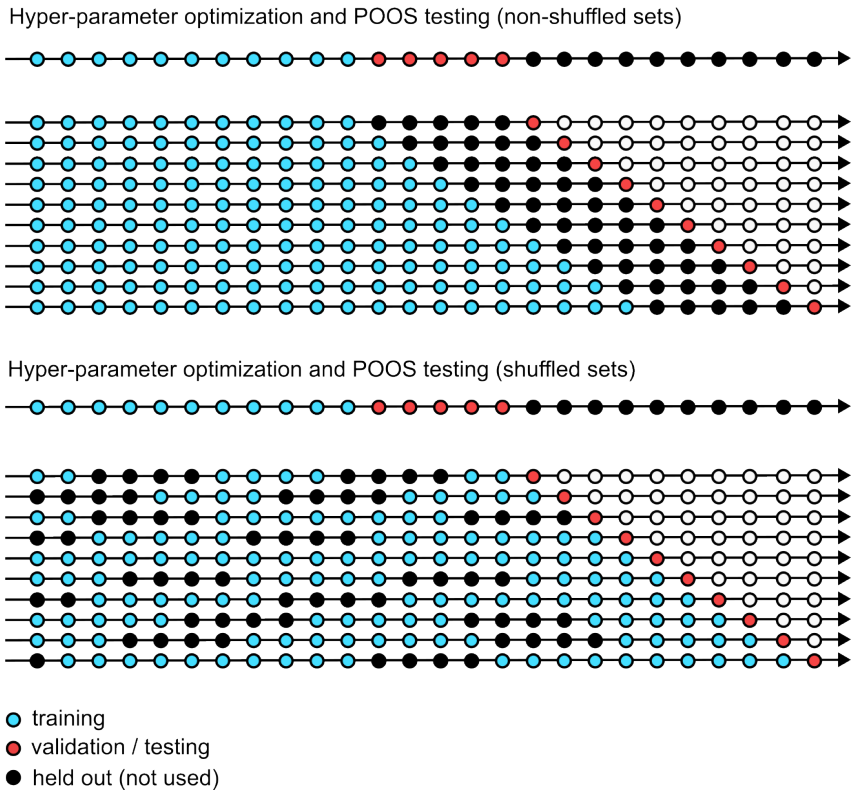


Figure 8: Data division in hyperparameter and pseudo-out-of-sample (POOS) testing stages across different neural architecture search (NAS) protocols. The top panel shows non-shuffled samples, while the bottom panel shows shuffled samples in the OOS testing stages. For all other machine learning methods, including NHITS and N-BEATS, full uninterrupted training sets without held-out samples are used in their OOS cross-validation stages.

For each segment, data preceding the cut-off year is used to train and test the model, which then forecasts the value for the next month. This forecast is compared with the actual observed value for that month. The process iterates by progressively incorporating an additional month into the training set, thereby generating a sequence of forecasts that are compared against their actual counterparts. This iterative procedure resulted in predictions for two out-of-sample forecasting windows: one spanning January 2022 to December 2023 (24 months), and another from January 2019 to December 2023 (60 months). The Root Mean Square Error (RMSE), calculated over these two periods, served as the primary metric for evaluating the forecasting performance.

Lastly, although these are conventional neural network models, we have specifically assessed the effect of shuffling the train-validation sets in the out-of-sample testing stage (as shown in Figure 8). The exploration of shuffling is innovative, as its impact is rarely examined in time series forecasting, a field where machine learning applications are emerging and often limited by traditional assumptions.

While shuffling was explored during the testing stage, the hyperparameter optimization stage was initially set to a train-validation split without shuffling to stress the optimization process, ensuring it performs optimally in resource-scarce conditions. A not-shuffled setup maintains a longer divide between the training data and the points to be forecasted. Although this investigation has focused on such a setup as depicted in Figure 8, future work will also evaluate the use of K-folds during the hyperparameter optimization stage. Recent findings suggest that K-fold cross-validation is the best practice for hyperparameter optimization in machine learning algorithms applied to macroeconomic forecasting (see Goulet Coulombe et al. 2022).

3 Related work

It has been emphasized that for the neural architecture search (NAS) case, our approach incorporates a hyperparameter optimization strategy that addresses parameters both internal and external to the neural network's topology. This broadens the scope of the optimization challenge, a facet that has been seldom explored in existing literature. Consequently, we will start with the more specific exploration of literature work on NAS as achieved through methods similar to the ones explored herein.

In the survey by Elsken, Metzen, and Hutter 2019, the process of neural architecture search (NAS) is categorized into three distinct dimensions: search space, search strategy, and performance estimation strategy. Concerning the first category, our approach adopts a tree-structured search space, which is typical of a Tree Parzen Estimator (TPE) method. Relevant to this kind of search space is the work by Qian et al. 2022, who employs a tree-structured search space for NAS with a binary tree search strategy. Similarly, graph data structures are used as search spaces for NAS in L. Wang et al. 2019 and Sasaki 2023, both of which implement Monte Carlo Tree search as their NAS search strategy.

Regarding the second category, the search strategy, the Tree-Parzen Estimator falls within the category of Bayesian optimization algorithms. Direct applications of TPE for NAS, such as those by Bergstra, Yamins, and Cox 2013, have led to state-of-the-art results in various computer vision problems. Furthermore, in the broader realm of Bayesian optimization, significant advancements have been reported by Domhan, Springenberg, and Hutter 2015 and Mendoza et al. 2016. These authors have used it as a search strategy, achieving faster optimization of hyperparameters and being the first to report winning competition datasets against human experts, respectively.

Lastly, for the third category, performance estimation, we employ a walk-forward, pseudo-out-of-sample (POOS) testing set held out during the optimization process. This method is extensively detailed in Section 2 and is a common practice in economic and financial forecasting. On that note, we now shift our focus to related work on the forecasting side of our study. This includes an exploration into two main areas within the forecasting literature: the application of Tree Parzen Estimators (TPE) in general forecasting models, and the specific use of neural network models for inflation forecasting, highlighting how our work integrates and builds upon these existing methodologies.

The application of TPE for hyperparameter optimization in forecasting is a recent development. Several studies have demonstrated its effectiveness in various domains. For instance, Massaoudi et al. 2021 employed TPE to probabilistically approximate the search space and identify optimal hyperparameters for short-term solar cell power generation forecasts. Similarly, Nguyen, Liu, and Zio 2020 utilized TPE for automatic hyperparameter optimization in an LSTM-based multi-step prediction model for Prognostics and Health Management (PHM). TPE's versatility is further highlighted in J. Zhang et al. 2021 where it optimized hyperparameters within a hybrid deep learning model for sugar price forecasting. Xu et al. 2021 leveraged TPE to optimize an attention-based LSTM network for forecasting Heating, Ventilation, and Air Conditioning (HVAC) energy demand. Finally, Shen et al. 2022 applied TPE to optimize a water runoff prediction model based on natural gradient boosting.

So far, to our best knowledge, TPE has been applied for hyperparameter optimization of machine learning models in macroeconomic times series forecasting only in Moreira, Rodrigues Moreira, and Oliveira Silva 2024 in the context of forecasting the Brazilian policy interest rate with a deep neural network. However, this paper does not perform a neural architecture search but only uses TPE to optimize a handful of hyperparameters.

The use of machine learning models to forecast macroeconomic variables like GDP or inflation is growing fast (see for example Richardson, Florenstein Mulder, and Vehbi 2021; Tenorio and Perez 2024). Our brief review here concentrates on inflation forecasting.

Earlier applications of neural networks showed some promise. Moshiri and Cameron 2000 which demonstrated that simple back-propagation networks could achieve performance comparable to traditional ARIMA and VAR models for Canadian inflation. However, hyperparameter selection relied on manual exploration, limiting replicability of the results. Similarly, Nakamura 2005 trained a feed-forward network for US CPI inflation, focusing on preventing overfitting. This study concluded that neural networks performed competitively with AR models.

Recent studies have further emphasized the potential of machine learning for economic forecasting. Medeiros et al. 2021 demonstrated that machine learning models, particularly Random Forests with a large number of covariates, can outperform traditional benchmarks in forecasting US inflation. However, their study did not address the question of optimal hyperparameter tuning, relying solely on expert judgment. In contrast, Goulet Coulombe et al. 2022 investigated the advantages of machine learning over standard macroeconometric methods. Their work highlights the ability of machine learning models to capture nonlinearities, a feature particularly relevant to our research.

In the context of long-term inflation forecasting, Paranhos 2023 investigated the suitability of LSTM neural networks for predicting US inflation. The study revealed the effectiveness of LSTM models in capturing long-term inflation trends, demonstrating reduced sensitivity to short-term fluctuations. However, it found no significant improvement in overall forecasting performance compared to Random Forest models. Almosova and Andresen 2023 further explored LSTM models for US CPI inflation forecasting, comparing them to feed-forward networks and standard linear benchmarks. Their findings concurred with the effectiveness of LSTMs for long-term horizons and emphasized the importance of network architecture in achieving optimal results.

On the other hand, Barkan et al. 2023 demonstrated the effectiveness of a hierarchical recurrent neural network for forecasting headline US CPI inflation by leveraging the disaggregation of CPI data into its lower-level components. This finding aligns with our own research, where even a simple disaggregation into two main CPI components improves forecast accuracy across all model types.

Two studies hold particular relevance. Xuanzheng Wang et al. 2022 demonstrates improved accuracy using state-of-the-art deep learning models like N-BEATS for forecasting Chinese economic indicators. In contrast, our study applies N-BEATS and NHITS models specifically to Peruvian inflation forecasting. Additionally, Moreira, Rodrigues Moreira, and Oliveira Silva 2024 utilized the TPE algorithm to optimize hyperparameters for deep neural networks forecasting Brazilian interest rates. However, their work did not explore higher-level neural architecture search.

4 Results and Discussion

Herein, the discussion is organized into three sub-sections. In the first, the optimal models found by the AutoML are presented. In the second, a comparative analysis of forecast precision among the top-performing models is provided. Finally, in the third sub-section, the gradients of the estimated optimal models are explored, offering insights into their performance dynamics.

4.1 Optimal models

This study evaluated forecasting models under two paradigms: univariate and component-based, as explained in subsection 2.1. Within each paradigm, the following three prominent model approaches have been explored: neural networks, machine learning regressions, and the NeuralForecast suite of models. For future reference all optimal models found through this AutoML process have been listed in Tables 6 to 11.

NAS neural networks

In Table 6 in Appendix C, we show the hyperparameters for the optimal univariate neural network model obtained through Neural Architecture Search (NAS-UV) and its corresponding topology in Appendix A.1. In addition, the component-based model (NAS-CB), being a function of two separate univariate neural network models each predicting an inflation component, was solved as two isolated tasks, the hyperparameters thereof which are shown in Table 7 in Appendix C, while the topologies for the two networks are shown in Appendix A.2. For simplicity, neural architecture blocks or algorithms not selected by the TPE optimizer were not put in these tables. From this information, we can see that no L2 kernel regularizers were chosen to be advantageous on any of the models. This could be because all neural models explored in this preliminary work were in fact univariate and at such level, L2 regularization might not give a significant improvement on generalization. It can also be found that time encoding was only necessary for the univariate, single-task model (Table 6) and not for the component-based model (Table 7), suggesting the inflation decomposition on the NAS-CB

model works as a good substitute for the decomposition into several frequencies that the time2vec does for the NAS-UV model.

Machine learning regressors

As it was for the former case, our examination of machine learning models for inflation forecasting (Table 8 in Appendix C) begins with the optimal univariate model of headline inflation, which was found to be a Gradient Boosting regressor with an impressive ensemble of 910 tree estimators. The substantial number of trees employed suggests the considerable complexity involved in autoregressing headline inflation, which integrates a diverse array of both stable and volatile elements within a single variable. Shifting focus to the component-based inflation models, which analyze inflation in segmented categories, we find a divergence in the optimal models for AE inflation (including food and energy) and SAE inflation (excluding food and energy). For AE inflation, where the data exhibits greater variability due to the inclusion of volatile items like food and energy, the Random Forest model proved most effective, requiring more than twice the number of trees compared to its counterpart used in SAE inflation (see Table 9 in Appendix C). This stark contrast highlights the need for a robust model capable of managing broader data fluctuations. In contrast, the SAE model, which excludes these volatile components, optimally utilizes the Gradient Boosting model but with significantly fewer trees, reflecting the reduced complexity and greater predictability of the data. The use of Gradient Boosting in both the univariate headline inflation and the more stable SAE inflation scenarios, albeit with different configurations, underlines the model's adaptability and effectiveness across varying levels of data complexity. The marked increase in tree count for the univariate model highlights the complex challenge of integrating a full spectrum of inflationary influences. This contrasts with the more targeted approach required for SAE inflation, and the detailed yet comprehensive handling necessary for AE inflation, demonstrates the critical importance of model selection tailored to the specific characteristics of each inflation measure and therefore, the significant practicality of a well-structured AutoML methodology.

NeuralForecast

From this suite of models, it was notably observed that the N-BEATS model emerged as the optimal choice in all cases, both as a univariate model to forecast headline inflation (Table 10 in Appendix C), as well as the component models for the component-based paradigm in Table 11 in Appendix C. From an economist's perspective, N-BEATS's use of polynomial basis functions to model trend components, alongside harmonic basis functions for seasonality, provides a comprehensive framework for forecasting inflation. The polynomial functions are particularly effective in capturing both linear and non-linear economic trends–such as fiscal policies, economic shocks, and gradual market shifts–which are critical for accurate long-term inflation predictions. The addition of harmonic functions allows N-BEATS to also effectively model the cyclical nature of economic variables, such as seasonal price fluctuations. This dual capability makes N-BEATS well-suited to handling the complex dynamics of inflation forecasting, where both trend and periodicity significantly influence overall accuracy. In contrast, NHITS focuses primarily on harmonic signals, potentially underestimating non-periodic, trend-driven changes that are crucial in economic forecasting. Nonetheless, we have also analyzed the optimal NHITS model from this approach (Table 10), which ranks seventh in performance among the NeuralForecast models. Comparison between the optimal N-BEATS and NHITS models reveals that both have almost identical input sizes and the same number of maximum steps. This similarity suggests that a comparable approach to data treatment and training is applied to both models, aiming to achieve optimal performance. In the component-based inflation models, N-BEATS continues to demonstrate its adaptability, being the preferred model for both SAE and AE inflation categories (Table 11). However, in contrast to the earlier point made, the hyperparameters for the N-BEATS models of SAE and AE inflation component models differ significantly, indicating that specialized approaches are required to address the distinct forecasting challenges posed by each component variable. The larger input size of 165 in the SAE model compared to 91 in the AE model suggests a more complex data handling requirement, potentially due to the inclusion of a broader range of economic indicators when food and energy are excluded. Both models employ four polynomials, emphasizing their capacity to accurately model the underlying trends in inflation across different components. Additionally, the variation in the number of harmonic functions–two for AE and one for SAE–may reflect the increased variability and cyclical patterns introduced by including food and energy in the AE model.

4.2 FORECAST PRECISION COMPARISON

This optimization project yielded models evaluated through the aforementioned pseudo-out-of-sample (POOS) testing method, simulating two forecasting out-of-sample windows, the results of which are shown in Table 3. As can be seen, the first window was 24 months in length, covering the span between January 2022 and December 2023, while the second window considered January 2019 to December 2023, with 60 months in length. The Expected Values from first and second-order autoregression (AR) models as well as those from the random walk hypothesis are used as benchmarks for the machine learning models that have been optimized herein.

Table 3: Comparison of Hyperparameter-Tuned Models for One-Step-Ahead Inflation Forecasting

ML Methods	Total CPI Annual Var Dec-23	Out-of-Sample RMSE	
		Jan-22 to Dec-23	Jan-19 to Dec-23
Univariate Models - Π_t			
N-BEATS	2.999615	0.394971	0.371733
Gradient Boosting Regression	2.915297	0.439374	0.395667
NAS-UV (shuffled sets)	3.126959	0.450079	0.390514
NAS-UV (not shuffled)	3.187929	0.482760	0.399762
Component-Based Models - Π_t^{sae} and Π_t^{ae}			
NeuralForecast	2.922576	0.380683	0.377425
ML Regressors	2.980602	0.442415	0.378559
NAS-CB (shuffled sets)	3.025956	0.401964	0.393179
NAS-CB (not shuffled)	3.096297	0.467090	0.381727
Expected Value (Benchmark)			
Ground-Truth	3.237383	-	-
AR(2)	3.404827	0.482822	0.450001
AR(1)	3.638480	0.559239	0.469780
Random Walk	3.638228	0.559123	0.463400

NAS = "neural architecture search", UV = "univariate model", CB = "component-based model".

It is found that both the Univariate (UV) models, which predict the CPI as a non-linear autoregressive exercise, as well as the Component-Based (CB) models, predicting it from its main two components, outperform all Expected Value benchmarks in both forecasting windows.

As shown in Table 3, root mean forecast errors (RMSE) are consistently lower for the 2019-2023 window compared to the 2022-2023 window. This observation holds true for all forecasts evaluated in this study. This finding corroborates the notion that inflation forecasting has become particularly challenging in the post-COVID period (2022-2023).

In Table 3, we compare the out-of-sample performance of our models, as described in the Methods section, against three well-established benchmarks: the random walk hypothesis and first- and second-order autoregression (AR) models. It is evident that the NAS models outperform all established benchmarks. Notably, the component-based models demonstrate superior performance compared to the univariate models, suggesting that decomposing the data enhances effectiveness even at the deep-learning level.

Interestingly, shuffling the sets during out-of-sample testing appears to improve forecasting performance, implying that with an optimized architecture, a shuffled set can enhance the estimation of future values. While this might be misconstrued as a fallacy, we argue otherwise for two reasons:

1. The neural networks are designed to identify non-linear correlations between X autoregressive inputs and y horizon output pairs, where each $X - y$ pair is independent. Although it is often argued that shuffling "breaks the sequential nature of time series," the integrity of each X sequence within its pair is preserved. Thus, even without shuffling, the training aims

to correlate X values to predict the subsequent y, and a conceptual 'break' exists between different $X - y$ samples. This means 'not-shuffled' sets might actually be disadvantageous as they distance the training from the test values, potentially affecting learning accuracy (as shown in Figure 8 where the comparison between shuffled and non-shuffled strategies is depicted).

2. The performance evaluations, as presented in Table 3, use a reserved set that the neural networks have not previously encountered. If this were not the case, concerns about overfitting might be valid. However, since this is fresh data, the observed improvements in performance are not due to overfitting but are likely a result of a more effective structure of these neural models for making forecasts.

In contrast to conventional MLP and CNN topologies that utilize traditional training and validation setups, other models like N-BEATS Olivares et al. 2023 and NHITS Challu et al. 2023 feature built-in sequential linkages in their architectures, which fundamentally alter how they utilize data sets.

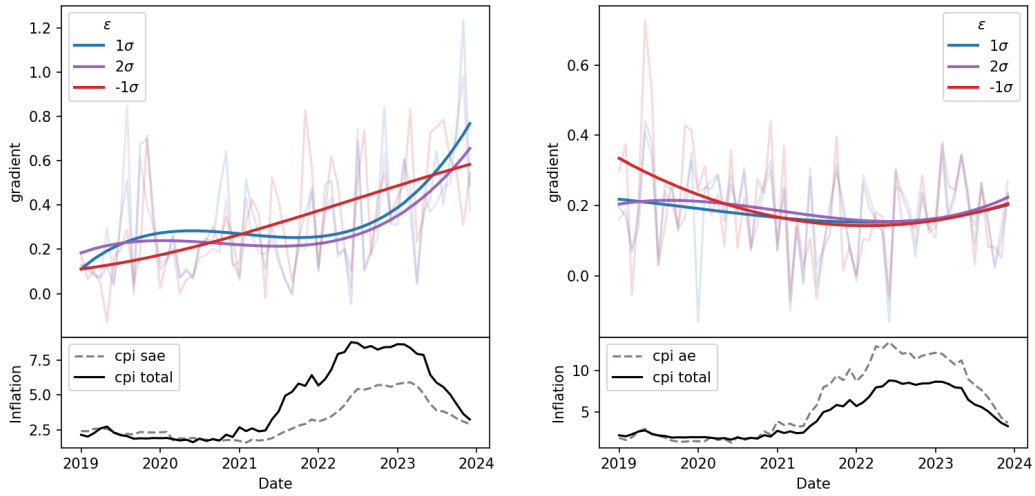
The results show how effective our preliminary NAS models are in handling complex time series data, especially when it comes to out-of-sample testing. The component-based approach, coupled with strategic data shuffling, has been shown to enhance predictive accuracy, effectively countering traditional concerns about sequence disruption in time series analysis. This not only validates our methodological innovations but also demonstrates their practical utility in overcoming the inherent challenges of using machine learning methods in macroeconomic forecasting. The success of these techniques highlights the potential for further innovations in neural network architectures, suggesting a promising direction for future research in economic data analysis.

Our evaluation confirms the superiority of NeuralForecast models in achieving lower forecasting errors. The N-BEATS model emerged as the most effective univariate model from the suite, exhibiting consistently lower RMSE values compared to those of other models (refer to Table 10). Notably, the best-performing NHITS model ranked significantly lower, at seventh place. While the variation in RMSE values across the top models was minimal, the N-BEATS model still achieved a slight edge, suggesting a potential performance advantage in this optimal region.

Unlike the sample division done within training and validation sets of NAS and ML Regressor models (Figure 4), NeuralForecast models, due to their peculiar model mechanics, predict the entire 60-month horizon as a single validation set in one shot. Due to this fact, the Input Size was not chosen as a tuned hyperparameter for the NeuralForecast models but was rather controlled by an Input-to-Output Size hyperparameter during the hyperparameter optimization stage (Table 5). As the horizon (i.e. the output) becomes $h = 1$ during the POOS testing, the Input Size obtained from the optimization stage must be the same as the one used for POOS testing because the information in the horizon is a response rather than a variable. This reasoning was likely a highly influential factor for the notable out-of-sample performance of this model.

For the univariate case, it was also found that the best-performing machine learning regression was a Gradient Boosting model, outperforming the univariate NAS-UV model with shuffled sets in the shorter-range POOS testing 24-month timeframe while keeping a comparable performance with the NAS-UV in the longer range, 60-month timeframe.

In contrast, for the component-based case, it was found the NAS-CB model with shuffled sets (Table 7) outperformed the ML regression model (Table 9) notably in the shorter-range POOS timeframe while losing some predictive power in the longer-range timeframe. This may be attributed to the increased sensitivity of the SAE component model of the NAS-CB explained by its gradient (Figure 9) within the shorter-range timeframe, starting in January 2022, while for that of the SAE component of the ML Regressors model (Figure 10) such sensitivity is actually significantly decreased within such time period. The consistent confirmation of predictive power with these gradient observations should not be overlooked, as it applies to all models, including the NeuralForecast CB models, which have SAE gradients of the highest magnitude relative to the other models across its whole 60-month timeframe, and outperform the other CB models in lowest RMSE in both forecasting POOS windows (Table 3).

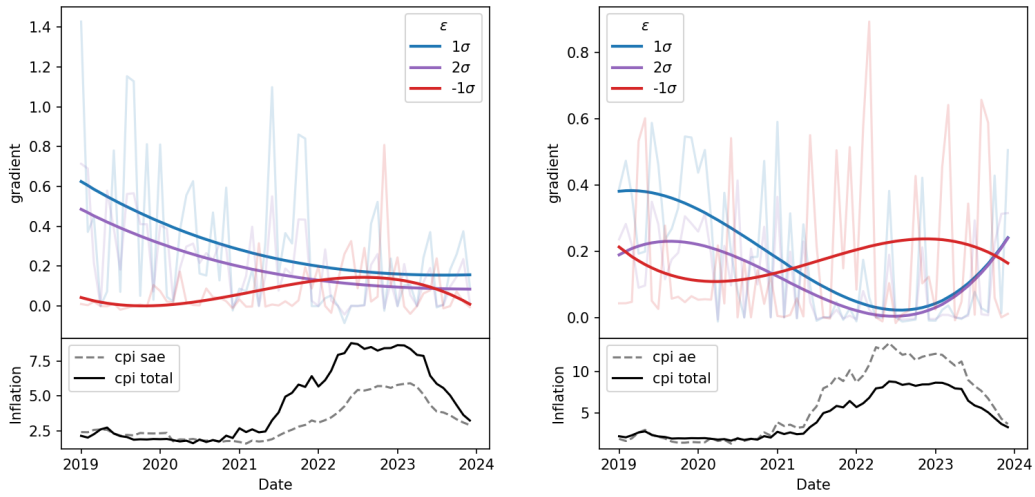


(a) Gradients for CPI without food and energy (SAE) (b) Gradients for CPI with food and energy (AE)

Figure 9: Gradient comparison for NAS neural networks: the left image shows the gradient magnitudes and directions for the optimal model excluding food and energy (SAE), and the right image for the optimal model including food and energy (AE). Year-on-year inflation series are shown for clarity.

4.3 Economic interpretation with gradients

Figures 9 to 10 visualize the autoregressive gradients obtained during training with the optimal models. Notably, in linear models, these gradients would remain constant regardless of the input size or sign. However, due to the non-linear nature of neural networks, gradients can exhibit variability. Our observations based on these figures are as follows:



(a) Gradients for CPI without food and energy (SAE) (b) Gradients for CPI with food and energy (AE)

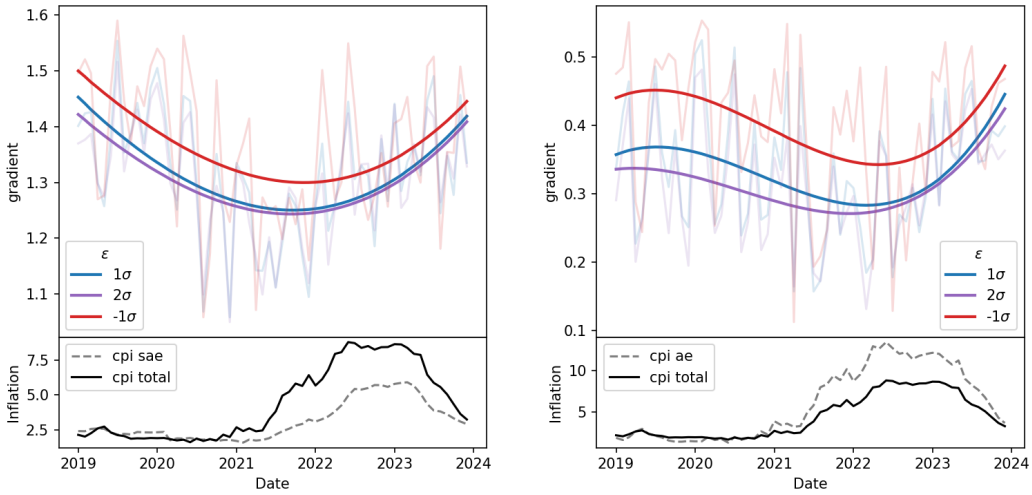
Figure 10: Gradient comparison for ML regressors: the left image displays the gradient magnitudes and directions for the optimal model excluding food and energy (SAE), and the right image for the optimal model including food and energy (AE). Year-on-year inflation series are shown for clarity.

First, as opposed to the constancy of gradients in linear models, neural network gradients exhibit fluctuations throughout training for all optimized models. These fluctuations, while irregular in their raw calculations, reveal interpretable patterns when smoothed.

Second, across all models, the autoregressive coefficients for models with AE inflation tend to be lower than those for inflation models excluding food and energy. This result may indicate that food and energy inflation has more complex dynamics than SAE inflation.

Third, the autoregressive gradient of SAE model for the N-BEATS component-based case (the best forecasting performer) exceeds unity. This suggests the presence of negative gradients at higher-order lags within the SAE dynamics. Interestingly, the gradient exhibits a U-shaped pattern across the sample period. The gradient initially decreased throughout 2019 and 2020, followed by an increase starting in mid-2021. Notably, this coincides with a rise in inflation. This behavior suggests that SAE inflation might gain momentum as headline inflation reaches historically high levels, potentially reflecting a structural change beyond the scope of this paper. Importantly, this heightened sensitivity to recent inflation is observed in the neural network model as well, but not in the machine learning regression model.

Fourth, our analysis of gradient variations due to input sign and scale revealed an interesting asymmetry in the N-BEATS model (Figure 11). While scale variations had minimal impact, sign variations exhibited a strong effect. Specifically, the autoregressive gradient associated with inflation was consistently higher when inflation was negative compared to positive values. This observation suggests a stronger influence of negative inflation on the model's learning process, potentially indicating an underlying asymmetry in the dynamics of both the SAE and AE components.



(a) Gradients for CPI without food and energy (SAE)

(b) Gradients for CPI with food and energy (AE)

Figure 11: Gradient comparison for N-BEATS models: the left image shows the gradient magnitudes and directions for the optimal model excluding food and energy (SAE), and the right image for the optimal model including food and energy (AE). Year-on-year inflation series are shown for clarity.

5 Conclusions

This study addresses the complex challenge of refining Consumer Price Index (CPI) inflation forecasts, which are crucial for the policy-making processes of central banks. By employing an automated machine learning (AutoML) approach utilizing Tree Parzen Estimators (TPE) for rigorous hyperparameter optimization, we have developed and assessed a variety of machine learning and novel neural network architectures. This approach has not only streamlined model design but also significantly enhanced the predictive accuracy of these models.

Our findings demonstrate that all models optimized through TPE surpassed traditional forecasting benchmarks, underscoring the method's effectiveness across various model types. Notably, the

component-based models provided superior performance compared to traditional univariate models, highlighting the benefits of decomposing macroeconomic data for more precise predictions. Additionally, the exploration of data shuffling in out-of-sample testing has challenged conventional approaches to time series analysis, suggesting that such techniques may substantially improve forecasting accuracy.

The N-BEATS model, optimized via AutoML, emerged as a standout performer in both univariate and component-based paradigms. Its integration of polynomial basis functions for trend analysis, coupled with harmonic functions for capturing seasonality, enables it to adeptly navigate the complex dynamics of inflation data. This dual functionality allows N-BEATS to address both the cyclical patterns and unexpected shifts that are typical in economic data influenced by broader policy and market conditions.

Furthermore, the study confirms that machine learning models faced increased forecasting errors in the post-COVID period, indicating that recent economic instability has made inflation more difficult to predict globally. The component-based approach to forecasting, which predicts headline inflation by first forecasting its individual components, has markedly improved model performance by capturing the unique dynamics of each segment.

Our analysis also reveals insights into the adaptability of machine learning models to dynamic economic conditions, crucial for maintaining forecast accuracy. We observed significant changes in the behavior of inflation excluding food and energy, suggesting potential structural shifts in the underlying economic relationships as headline inflation rates climb to historically high levels.

Despite these advances, the study acknowledges the inherent complexities and ever-evolving nature of economic forecasting, which pose continuous challenges in data availability and model applicability. Future research should consider incorporating additional economic indicators, applying other advanced machine learning techniques, and examining model performance across diverse economic cycles.

In conclusion, this study significantly advances economic forecasting by applying sophisticated machine learning techniques to CPI forecasting, as well as explaining them. As economic relationships and computational methods continue to evolve, ongoing innovation remains essential. Future efforts should focus on evaluating the scalability of these models in different economic contexts and their practical efficacy in shaping real-world economic policies.

REFERENCES

Almosova, Anna and Niek Andresen (2023). "Nonlinear inflation forecasting with recurrent neural networks". In: *Journal of Forecasting* 42.2, pp. 240–259.

Barkan, Oren et al. (2023). "Forecasting CPI inflation components with hierarchical recurrent neural networks". In: *International Journal of Forecasting* 39.3, pp. 1145–1162.

Benigno, Pierpaolo and Gauti B Eggertsson (2023). *It's baaack: The surge in inflation in the 2020s and the return of the non-linear phillips curve*. working paper 31197. National Bureau of Economic Research. DOI: 10.3386/w31197. URL: https://www.nber.org/papers/w31197.

– (2024). *The Slanted-L Phillips Curve*. working paper 32172. National Bureau of Economic Research. DOI: 10.3386/w32172. URL: https://www.nber.org/papers/w32172.

Bergstra, James, Rémi Bardenet, et al. (2011). "Algorithms for Hyper-Parameter Optimization". In: 24. Ed. by J. Shawe-Taylor et al. URL: https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

Bergstra, James, Daniel Yamins, and David Cox (17–19 Jun 2013). "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 1. Atlanta, Georgia, USA: PMLR, pp. 115–123. URL: https://proceedings.mlr.press/v28/bergstra13.html.

Bernanke, Ben and Olivier Blanchard (forthcoming). "What Caused the U.S. Pandemic-Era Inflation?" In: *American Economic Journal: Macroeconomics*.

Challu, Cristian et al. (2023). "Nhits: Neural hierarchical interpolation for time series forecasting". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 37. 6, pp. 6989–6997.

Domhan, Tobias, Jost Tobias Springenberg, and Frank Hutter (2015). "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves". In: *Proceedings of the 24th International Conference on Artificial Intelligence*. IJCAI'15. Buenos Aires, Argentina: AAAI Press, pp. 3460–3468. ISBN: 9781577357384.

Elsken, Thomas, Jan Hendrik Metzen, and Frank Hutter (2019). *Neural Architecture Search: A Survey*. arXiv: 1808.05377 [stat.ML].

Fan, Yicheng et al. (2023). *LayerNAS: Neural Architecture Search in Polynomial Complexity*. arXiv: 2304.11517 [cs.LG].

Goulet Coulombe, Philippe et al. (2022). "How is machine learning useful for macroeconomic forecasting?" In: *Journal of Applied Econometrics* 37.5, pp. 920–964.

Kazemi, Seyed Mehran et al. (2019). "Time2vec: Learning a vector representation of time". In: *arXiv preprint arXiv:1907.05321*.

Kingma, Diederik P. and Jimmy Ba (2017). *Adam: A Method for Stochastic Optimization*. arXiv: 1412.6980 [cs.LG].

Makridakis, Spyros, Evangelos Spiliotis, and Vassilios Assimakopoulos (2020). "The M4 Competition: 100,000 time series and 61 forecasting methods". In: *International Journal of Forecasting* 36.1, pp. 54–74.

Massaoudi, Mohamed et al. (2021). "Enhanced deep belief network based on ensemble learning and tree-structured of Parzen estimators: An optimal photovoltaic power forecasting method". In: *IEEE Access* 9, pp. 150330–150344.

Medeiros, Marcelo C et al. (2021). "Forecasting inflation in a data-rich environment: the benefits of machine learning methods". In: *Journal of Business & Economic Statistics* 39.1, pp. 98–119.

Mendoza, Hector et al. (24 Jun 2016). "Towards Automatically-Tuned Neural Networks". In: *Proceedings of the Workshop on Automatic Machine Learning*. Ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Vol. 64. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, pp. 58–65. URL: https://proceedings.mlr.press/v64/mendoza_towards_2016.html.

Moreira, Rodrigo, Larissa Rodrigues Moreira, and Flávio de Oliveira Silva (2024). "Brazilian Selic Rate Forecasting with Deep Neural Networks". In: *Computational Economics*, pp. 1–21.

Moshiri, Saeed and Norman Cameron (2000). "Neural network versus econometric models in forecasting inflation". In: *Journal of forecasting* 19.3, pp. 201–217.

Nakamura, Emi (2005). "Inflation forecasting using a neural network". In: *Economics Letters* 86.3, pp. 373–378.

Nguyen, Hoang-Phuong, Jie Liu, and Enrico Zio (2020). "A long-term prediction approach based on long short-term memory neural networks with automatic parameter optimization by Tree-structured Parzen Estimator and applied to time-series data of NPP steam generators". In: *Applied Soft Computing* 89, p. 106116.

Olivares, Kin et al. (2023). "Neural basis expansion analysis with exogenous variables: Forecasting electricity prices with NBEATSx". In: *International Journal of Forecasting* 39.2, pp. 884–900.

Oreshkin, Boris N et al. (2019). "N-BEATS: Neural basis expansion analysis for interpretable time series forecasting". In: *International Conference on Learning Representations*.

Paranhos, Livia (2023). "Predicting Inflation with Recurrent Neural Networks". In: *arXiv preprint arXiv:2104.03757*.

Qian, Guocheng et al. (2022). *When NAS Meets Trees: An Efficient Algorithm for Neural Architecture Search*. arXiv: 2204.04918 [cs.AI].

Richardson, Adam, Thomas van Florenstein Mulder, and Tuğrul Vehbi (2021). "Nowcasting GDP using machine-learning algorithms: A real-time assessment". In: *International Journal of Forecasting* 37.2, pp. 941–948.

Sasaki, Yuya (2023). *Efficient and Explainable Graph Neural Architecture Search via Monte-Carlo Tree Search*. arXiv: 2308.15734 [cs.LG].

Shen, Keyan et al. (2022). "Runoff probability prediction model based on natural Gradient boosting with tree-structured parzen estimator optimization". In: *Water* 14.4, p. 545.

Staněk, Filip (2023). "A Note on the M6 Forecasting Competition: Designing Parametric Models with Hypernetworks". In: *Available at SSRN 4355794*.

Tenorio, Juan and Wilder Perez (2024). *GDP nowcasting with Machine Learning and Unstructured Data*. working paper 003-2024. Banco Central de Reserva del Peru. URL: https://www.bcrp.gob.pe/docs/Publicaciones/Documentos-de-Trabajo/2024/documento-de-trabajo-003-2024.pdf.

Vaswani, Ashish et al. (2023). *Attention Is All You Need*. arXiv: 1706.03762 [cs.CL].

Wang, Linnan et al. (2019). *Neural Architecture Search using Deep Neural Networks and Monte Carlo Tree Search*. arXiv: 1805.07440 [cs.LG].
Wang, Xiaolong et al. (2018). *Non-local Neural Networks*. arXiv: 1711.07971 [cs.CV].
Wang, Xuanzheng et al. (2022). "EcoForecast: An interpretable data-driven approach for short-term macroeconomic forecasting using N-BEATS neural network". In: *Engineering Applications of Artificial Intelligence* 114, p. 105072.
Xu, Yang et al. (2021). "Potential analysis of the attention-based LSTM model in ultra-short-term forecasting of building HVAC energy consumption". In: *Frontiers in Energy Research* 9, p. 730640.
Zhang, Chris, Mengye Ren, and Raquel Urtasun (2020). *Graph HyperNetworks for Neural Architecture Search*. arXiv: 1810.05749 [cs.LG].
Zhang, Jinlai et al. (2021). "A novel hybrid deep learning model for sugar price forecasting based on time series decomposition". In: *Mathematical Problems in Engineering* 2021, pp. 1–9.

A Neural Network Architectures

A.1 Topology for NAS-UV Model

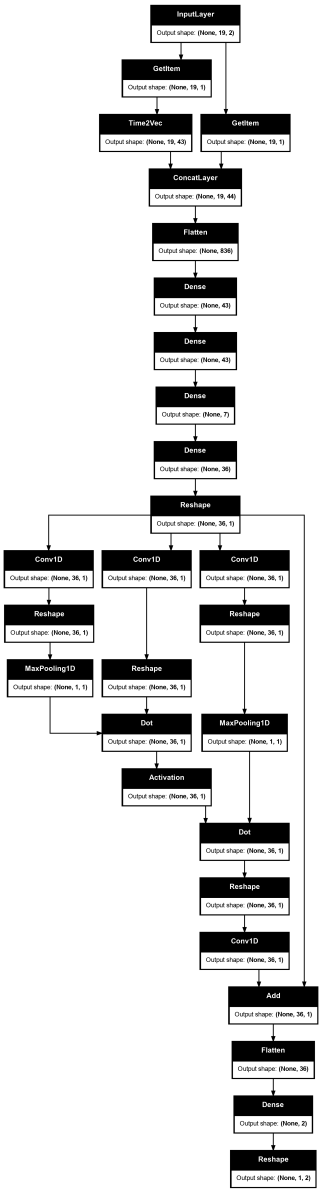


Figure 12: Computational graph of the optimal Univariate Neural Network model. This neural network, designed for univariate forecasting of the total inflation time series, was built automatically via Neural Architecture Search.

A.2 TOPOLOGIES FOR NAS-CB MODELS

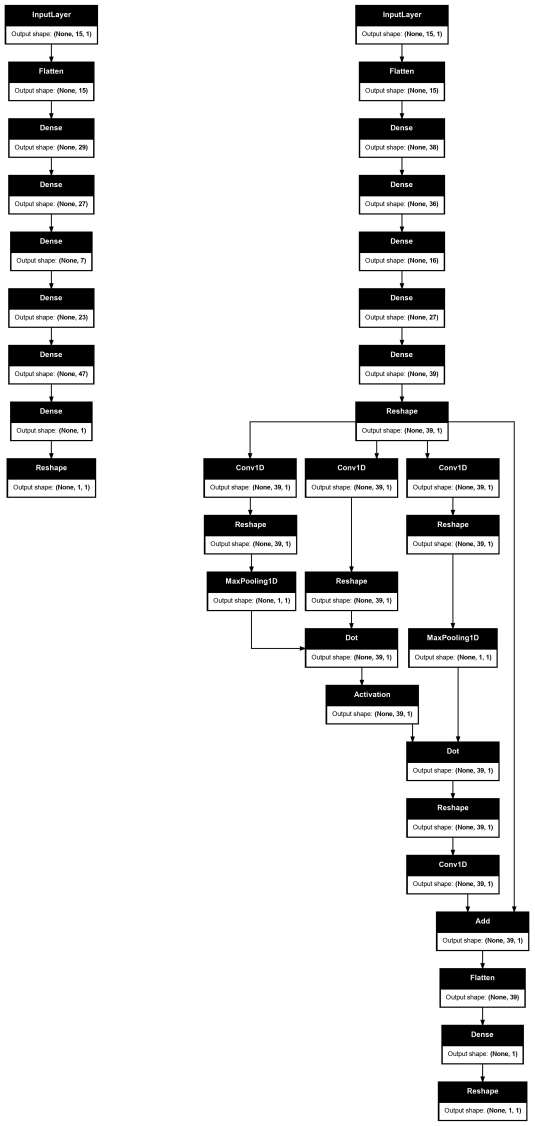


Figure 13: Computational graph of the optimal Component-Based CPI Neural Network models. These models, built automatically via Neural Architecture Search, include two distinct neural networks: the left model (SAE) forecasts the CPI component excluding food and energy, and the right model (AE) forecasts the CPI component including food and energy. The outputs are combined to calculate the aggregated inflation index. Both models operate on a univariate basis.

B HYPERPARAMETER OPTIMIZATION HISTORY

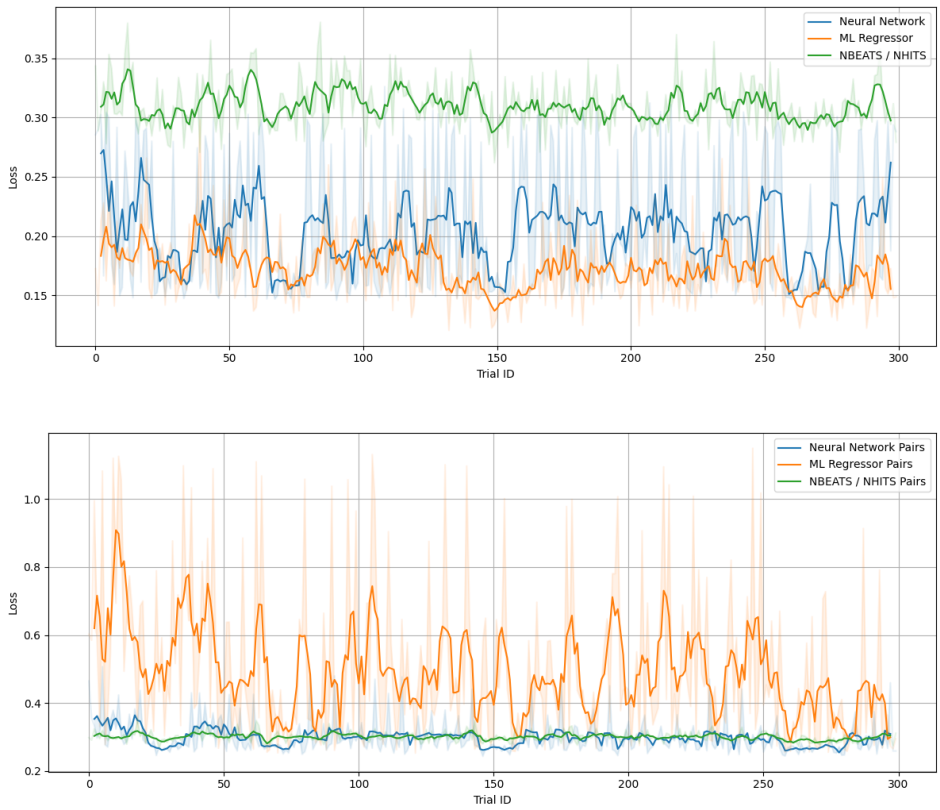


Figure 14: Hyperparameter optimization history for neural architecture search across models. The top image shows the optimization process for the univariate neural network model, and the bottom image depicts the optimization for the component-based CPI neural networks. These visualizations highlight the exploration of hyperparameter spaces and the iterative refinement of model architectures to enhance performance.

C Hyperparameter Search Spaces and Optimal Models

In this appendix, we provide detailed tables of the hyperparameters considered for the various models and the optimal configurations found through the Tree Parzen Estimator (TPE) optimization process. These tables are referenced throughout the results section to support the discussion on model performance and selection.

C.1 Hyperparameter Search Spaces

Table 4: Considered hyperparameters for machine learning regression models.

Hyperparameter	Configuration Space
Input Size	DiscreteRange (3, 108, 3)
Model*	{Random Forest, Gradient Boosting, Lasso}
Random Forest**	
Number of Estimators	DiscreteRange (10, 1000, 10)
Max Depth	{None, 10, 20, 30, 40, 50}
Min Samples Split	DiscreteRange (2, 20, 1)
Min Samples Leaf	DiscreteRange (1, 10, 1)
Max Features	{None, sqrt, log2}
Bootstrap	{True, False}
Gradient Boosting**	
Number of Estimators	DiscreteRange (10, 1000, 10)
Learning Rate	LogRange (-5, 0)
Max Depth	{None, 3, 5, 7, 9}
Min Samples Split	DiscreteRange (2, 10, 1)
Min Samples Leaf	DiscreteRange (1, 5, 1)
Subsample	Range (0.5, 1.0)
Lasso**	
Alpha	LogRange (-7, 2)

Note: The global parameters are chosen at the highest level in the Tree Parzen, while specific parameters are exclusive to each model type that may be chosen by the TPE.
* Hyperparameters with an asterisk have additional hyperparameters, while hyperparameters or sets with double asterisks are the children of said hyperparameters or children sets thereof.

Table 5: Considered hyperparameters for NeuralForecast models.

Hyperparameter	Configuration Space
Input-to-Output Size	Range (1.0, 3.5)
Input Size (Input-to-Output Size)†	DiscreteRange (60, 210)
Max Steps	DiscreteRange (50, 200, 10)
Model*	{NHITS, N-BEATS}
N-BEATS**	
Number of Polynomials	{2, 3, 4}
Number of Harmonics	{1, 2}

† Input Size is a function of the Input-to-Output Size hyperparameter chosen by the TPE.
* Hyperparameters with an asterisk have additional hyperparameters, while hyperparameters or sets with double asterisks are the children of said hyperparameters or children sets thereof.

C.2 Optimal Models

Table 6: Optimal Univariate Neural Architecture Search (NAS-UV) model

Hyperparameter	Optimal Values
Adam Learning Rate	0.00048
Adam β_1	0.94886
Adam β_2	0.99675
ES Patience	90
LR Patience	5
Batch Size	3
Input Size	19
Number of Dense Layers	4
Number of Layer Nodes	[43,43,7,36]
Activation Function	ReLU
Post-MLP NLB Compression L_{pool}	11
Scalers, Use	True
Number of Harmonics (t2v)	42
Performance	
$\mathcal{L}^{val}(\mathcal{D}; \Theta, \Lambda)$	0.14574

Table 7: Optimal Component-Based Neural Architecture Search (NAS-CB) model

Hyperparameter	Optimal Values	
Adam Learning Rate	0.00057	
Adam β_1	0.92726	
Adam β_2	0.99603	
ES Patience	25	
LR Patience	20	
Batch Size	7	8
Input Size	15	15
Number of Dense Layers	5	5
Number of Layer Nodes	[29,27,7,23,47]	[38,36,16,27,39]
Activation Function	LeakyReLU	LeakyReLU
Post-MLP NLB Compression L_{pool}	-	21
Scalers, Use	False	False
Performance		
$\mathcal{L}^{val}(\mathcal{D}; \Theta, \Lambda)$	0.24349	
$\mathcal{L}^{val,sae}(\mathcal{D}^{(1)}; \Theta^{(1)}, \Lambda^{(1)})$	0.15677	
$\mathcal{L}^{val,ae}(\mathcal{D}^{(2)}; \Theta^{(2)}, \Lambda^{(2)})$	0.47038	

Table 8: Optimal Univariate Machine Learning Regressor

Hyperparameter	Optimal Values
Model Type	Gradient Boosting
Input Size	99
Max Depth	None
Number of Estimators	910
Minimum Samples Split	4
Minimum Samples Leaf	3
Learning Rate	0.04270
Subsample	0.90294
Performance	
$\mathcal{L}^{val}(\mathcal{D};\Theta,\Lambda)$	0.12030

Table 9: Optimal Component-Based Machine Learning Regression model

Hyperparameter	Optimal Values	
Model Type	Gradient Boosting	RandomForest
Input Size	54	63
Max Depth	None	10
Number of Estimators	380	870
Minimum Samples Split	2	10
Minimum Samples Leaf	3	4
Learning Rate	0.06719	-
Subsample	0.84175	-
Bootstrap	-	True
Max Features	-	None
Performance		
$\mathcal{L}^{val}(\mathcal{D};\Theta,\Lambda)$	0.25319	

Table 10: Optimal Univariate NeuralForecast model (N-BEATS) and optimal Univariate NHITS model (7th best from NeuralForecast models)

Hyperparameter	Optimal Values	
	N-BEATS	NHITS
Model Rank	1	7
Input Size	76	75
Max Steps	160	160
Number of Polynomials	2	-
Number of Harmonics	2	-
Performance		
$\mathcal{L}^{val}(\mathcal{D};\Theta,\Lambda)$	0.26190	0.27325

Table 11: Optimal Component-Based NeuralForecast model

Hyperparameter	Optimal Values	
	SAE Model	AE Model
Model Type	N-BEATS	N-BEATS
Input Size	165	91
Max Steps	160	100
Number of Polynomials	4	4
Number of Harmonics	1	2
Performance		
$\mathcal{L}^{val}(\mathcal{D}; \Theta, \Lambda)$	0.26633	