



# SERIE DOCUMENTOS DE TRABAJO

No. 323

5 de marzo de 2025

Stability focused end to end frameworks for risk  
budgeting portfolios

---

**Carlos Castro-Iragorri**

**Manuel Parra-Díaz**

# Stability-Focused End-to-End Frameworks for Risk Budgeting Portfolios

Manuel Parra-Diaz\* and Carlos Castro-Iragorri†  
Universidad del Rosario, Bogotá, Colombia.

March 5, 2025

## Abstract

Recent advances in deep learning have spurred the development of end-to-end frameworks for portfolio optimization that utilize implicit layers. However, many such implementations are highly sensitive to neural network initialization, undermining performance consistency. This research introduces a robust end-to-end framework tailored for risk budgeting portfolios that effectively reduces sensitivity to initialization. Importantly, this enhanced stability does not compromise portfolio performance, as our framework consistently outperforms the risk parity benchmark.

Keywords: end-to-end framework, neural networks, risk budgeting, stability

JEL: C45, C13, G11

## 1 Introduction

Portfolio optimization is a crucial financial task in which investors aim to maximize returns while effectively managing risk. The End-to-End framework offers a promising approach to portfolio optimization by integrating deep learning techniques with traditional financial models [Uysal et al., 2024]. This innovative architecture enables a comprehensive and efficient resolution of various steps in portfolio optimization, including parameter estimation and the determination of optimized weights. The decision-making process can be tailored to accommodate different investor objectives, making the approach highly flexible. Additionally, the framework is tractable, potentially scalable, and provides numerical sensibilities to all elements of the problem. Its features can be implemented using data-driven insights, model-based decisions, or a combination of both, significantly enhancing portfolio allocation efficiency for both institutional and individual investors.

Despite its advantages, implementing the End-to-End framework presents several challenges. A primary issue lies in ensuring smooth and coherent connections among

---

\*corresponding author: manuel.parra@urosario.edu.co

†carlos.castro@urosario.edu.co

its layers. Specifically, the relationships between inputs and outputs must be well-defined and exhibit stable behavior to produce reliable results. If these connections are compromised, the integrity of the optimization process may be affected, leading to numerical instability. For instance, instability in one layer’s output can propagate through the framework, ultimately undermining the optimization’s effectiveness.

Furthermore, the framework must be sufficiently adaptable to accommodate various investment objectives, such as maximizing returns, minimizing risk, or striking a balance between the two. This flexibility is essential for practitioners seeking to tailor strategies to specific investment goals while navigating complex market dynamics.

To address these challenges, we propose an extension to the End-to-End framework that enhances numerical stability within the optimization layer. Our approach introduces a bounded softmax layer, which ensures that the inputs to the optimization process remain stable and well-behaved. By constraining the layer’s outputs, we mitigate risks associated with extreme values that could disrupt the optimization process. This enhancement not only improves the reliability of the optimization results but also strengthens the framework’s overall performance, allowing it to accommodate a broader range of assets and investment scenarios.

To validate our proposed extension, we conduct a case study applying the bounded softmax layer to a well-established portfolio optimization methodology: risk budgeting. Risk budgeting is an increasingly popular method that allocates risk across assets based on their contributions to overall portfolio risk. By integrating our bounded softmax layer into this methodology, we demonstrate its effectiveness in enhancing the stability and reliability of the optimization process.

Our case study results indicate that the bounded softmax layer provides performance results that are 70% to 100% more reliable (implying a lower dispersion) than without this layer. Using performance metrics such as the Sharpe Ratio and the cumulative returns in the out-of-sample periods we obtain dispersion among results below 1% for different starting values of the neural network weight that are a fundamental part of the end-to-end framework; this is a significant gain over the results replicated from [?].

Notably, this extension enables the framework to accommodate a larger number of assets without encountering substantial difficulties in achieving stable weight allocations. This capability is particularly valuable in today’s financial markets, where investors often seek to diversify portfolios across a wide range of asset classes to mitigate risk. The bounded softmax layer provides a robust mechanism for managing the numerical complexities of larger portfolios, ensuring that the optimization process remains both effective and computationally efficient.

The paper is organized as follows. Section 2 provides an overview of End-to-End modelling for portfolio optimization. Section 3 presents the fundamentals of risk budgeting portfolios and the *plug-in* approach to portfolio optimization. Section 4 explains End-to-End frameworks emphasizing the shortcomings in terms of numerical stability and provides a detailed explanation of the proposed bounded softmax layer. Section 5 illustrates the main results with a case study based on real market data and a detailed explanation of the computational setup. Finally, Section 6 presents the conclusions and discuss possible extensions.

## 2 Literature review

End-to-end modeling in portfolio optimization represents a significant shift from traditional methods by integrating prediction and decision-making processes into an interconnected layered framework. While traditional portfolio optimization typically involves independent steps and employs the so-called *plug-in* approach to link estimation and optimization stages, the layered approach provides a more cohesive modeling structure. This approach aims to address the limitations of conventional two-step processes by directly optimizing portfolio weights based on raw data inputs and targeting a specific performance outcome, thereby enhancing efficiency and potentially improving financial performance. The integration is expected to mitigate errors associated with separate prediction and optimization stages, leading to more robust investment decisions [Anis and Kwon, 2025].

Recent studies propose initial implementations of the end-to-end modeling framework applied to mean-variance optimization [Butler and Kwon, 2022] or portfolio allocations defined by risk contributions [Uysal et al., 2024]. In the former, the authors design a neural network with a layer for parametric estimation in a factor model, connected to a differentiable quadratic programming layer that solves a regularized mean-variance problem. In [Uysal et al., 2024], the authors present two approaches for portfolio construction: model-free and model-based networks, with the latter incorporating a risk budgeting model as an implicit layer. Risk budgets are estimated directly from the data and integrated into the portfolio optimization process to obtain optimal asset weights targeting specific outcomes such as maximizing cumulative ex-post returns or the Sharpe Ratio.

Empirical applications vary from seven to fifty assets, demonstrating promising results in controlling portfolio volatility and outperforming approaches that do not employ the end-to-end methodology, such as nominal risk parity or mean-variance optimization without regularization. The authors also highlight the need for further research to validate the efficacy and stability of these solutions. Recent contributions provide notable extensions to these initial approaches. [Zhang et al., 2021] propose a score-based layer that circumvents the reliance on specific moments such as mean and variance. The scores are linked to portfolio weights, determining the optimal stock combination according to a pre-specified financial outcome, such as cumulative returns adjusted for transaction costs. This method accommodates a large asset space (over one hundred stocks) and incorporates constraints on weights and the number of stocks (cardinality constraints). [Anis and Kwon, 2025] evaluate the complexities of including cardinality constraints as a layer between estimation and optimization layers. Their focus is on the cardinality-constrained minimum-variance portfolio optimization problem, utilizing a linear factor model for returns. This approach enhances the realism of portfolio construction while introducing computational challenges, as the optimization process must navigate a larger solution space while adhering to constraints on stock selection and weight limits. Their study initially considers fifty assets with varying degrees of cardinality constraints, targeting portfolios containing between 10 and 20 assets. The primary challenge is ensuring optimization efficiency while achieving optimal asset selection, as these constraints complicate the feasible region of the problem. To address this, advanced optimization techniques such as heuristic algorithms or mixed-integer programming can streamline the selection process while maintaining adherence to car-

dinality limits. This framework generates in-sample and out-of-sample portfolios with lower volatility and higher returns than traditional decoupled approaches when applied to real-world financial data.

[Zhong et al., 2024] propose a Direct Sorted Portfolio Optimization (DSPO) end-to-end framework. This framework constructs characteristic-sorted portfolios directly from raw stock data without extensive manual feature engineering. Portfolio sorting is carried out using a Monotonic Logistic Regression loss function to rank assets optimally in relation to ex-post realized returns. The DSPO model achieved a rank information coefficient of 10.12 and an accumulated return of 121.94% on NYSE stocks. Overall, end-to-end portfolio optimization and its multiple extensions show promising results. However, further refinement is necessary to enhance result stability and ensure clear replicability, particularly as some extensions introduce greater complexity.

For portfolio optimization problems there are different interpretations of instability problems. The classical ill-posed problem is a recurrent one in the "predict then optimize" strategy where generally the optimal weights are unstable and non-unique, meaning that small variation in the inputs that are *plugged-in* to the optimization problem can generate important source of instability in the end-results: portfolio weights and performance. There is a large literature that proposed different correction to address the problems, for example the use of regularization techniques specially in the context of large portfolios (50-100 assets) [Brodie et al., 2009]. L1 type regularization imposes sparsity in the final results, that is the effective number of active positions (portfolio weights different from zero) is significantly reduced. Some extensions [Dai and Wen, 2018] add shrinkage to L1 regularization in order to improve performance.

Another approach is motivated by accounting for model risk and the intent is to obtain robust results, for example by explicitly accounting for uncertainty in the data. Robust portfolio optimization (RO) is a strategy in asset allocation that focuses on optimizing the performance of a portfolio under the worst-case scenarios of uncertain inputs. By embedding uncertainties into a deterministic framework, robust portfolio optimization provides a stable solution that tends to perform better out-of-sample compared to classical mean-variance portfolios [Fabozzi et al., 2010, Xidonas et al., 2020]. More recently [Georgantas et al., 2024] compares the performance of different RO versions of classical models: mean-variance with box (MVBU) or ellipsoid (MVEU) uncertainty, worst-case VaR and CVaR, worst-case Omega Ratio or a combination of some of them (multi-objective optimization). The performance of the models is evaluated considering the number of active assets and risk return trade-off. In general, RO performed better than their non-RO counterparts although their asset compositions are relatively similar in terms of diversification and turnover. Specifically, the MVBU model, while underperforming compared to the MV model during stable periods, showed consistently better results during stock market crises. Conversely, the MVEU model outperformed the MV model in five out of six indicators during the full test period but did not perform as well during crisis conditions.

### 3 Risk Budgeting Portfolios

Consider an investment universe with  $n$  assets. Let  $\mathbf{w} = (w_1, w_2, \dots, w_n)^\top$  be the vector of portfolio weights and  $\Sigma \in \mathbb{R}^{n \times n}$  be the covariance matrix of asset returns and the

portfolio risk  $\sigma$  is measured by the volatility, i.e.

$$\sigma(\mathbf{w}) = \sqrt{\mathbf{w}^\top \Sigma \mathbf{w}}. \quad (1)$$

The *marginal risk contribution* of asset  $i$  to the overall portfolio risk is given by the partial derivative of  $\sigma(\mathbf{w})$  with respect to  $w_i$ :

$$\frac{\partial \sigma(\mathbf{w})}{\partial w_i} = \frac{(\Sigma \mathbf{w})_i}{\sigma(\mathbf{w})},$$

so that the *total risk contribution* of asset  $i$  is

$$RC_i(\mathbf{w}) = \frac{\partial \sigma(\mathbf{w})}{\partial w_i} = \frac{w_i (\Sigma \mathbf{w})_i}{\sigma(\mathbf{w})}. \quad (2)$$

Assume a pre-specified *risk budget vector*  $\mathbf{b} = (b_1, b_2, \dots, b_n)^\top$  where each  $b_i \geq 0$  and  $\mathbf{1}^\top \mathbf{b} = 1$ . This vector represents the target proportions of total portfolio risk that each asset should contribute.

The *risk budgeting portfolio optimization problem* is then to determine the portfolio weights  $\mathbf{w}$  such that the risk contribution of each asset is proportional to its risk budget. Formally this require that

$$\frac{\partial RC_i(\mathbf{w})}{\sigma(\mathbf{w})} = b_i, \quad \text{for } i = 1, 2, \dots, n. \quad (3)$$

Substituting the equations (1) and (2), this condition becomes:

$$\frac{w_i (\Sigma \mathbf{w})_i}{\mathbf{w}^\top \Sigma \mathbf{w}} \Leftrightarrow w_i (\Sigma \mathbf{w})_i = b_i \mathbf{w}^\top \Sigma \mathbf{w}, \quad \text{for } i = 1, 2, \dots, n. \quad (4)$$

subject to the standard portfolio constraint  $\mathbf{1}^\top \mathbf{w} = 1$ .

This formulation ensures that each asset's contribution to the overall portfolio risk aligns with its predetermined risk budget. When all  $b_i = 1/n$ , the problem reduces to the well-known *risk parity* portfolio optimization problem.

In many practical portfolio optimization settings, a "predict then optimize" strategy—also known as the classical plug-in approach—is commonly employed. This methodology divides the decision-making process into two distinct stages. First, historical or auxiliary data are used to generate estimates of key model parameters, such as expected returns and covariances (or risk budgets for the risk budgeting portfolio optimization problem). These estimates are then directly substituted into an optimization framework in the second stage, where the optimal portfolio weights are determined under given constraints and objectives. Although this decoupled approach simplifies the process by treating prediction and optimization separately, it inherently links the quality of the final portfolio to the accuracy of the parameter estimates. The formal definition provided below captures this two-step process, detailing how parameter estimation feeds into the optimization of portfolio weights.

Let  $\theta$  denote the (unknown) parameters of the return distribution of  $n$  assets. For instance, in the risk budgeting framework, one have  $\theta = (\mathbf{b}, \Sigma)$ . Given historical or auxiliary data  $\mathbb{D}$ , the investor uses a statistical model to compute an estimate  $\hat{\theta} = \mathcal{P}(\mathbb{D})$ , where  $\mathcal{P}$  is a prediction procedure e.g., maximum likelihood estimation, that maps the

data  $\mathbb{D}$  to an estimate in the parameter space  $\Theta$ . Now, suppose the investor’s objective is to maximize a utility function or minimize a risk measure that depends on the portfolio weights  $\mathbf{w}$  and the parameters  $\theta$ . We denote this objective function by  $U(\mathbf{w}; \theta)$ . For example, in the risk budgeting setup the objectives might be written as

$$U(\mathbf{w}; \mathbf{b}, \Sigma) = \sum_i^n (w_i (\Sigma \mathbf{w})_i - b_i \mathbf{w}^\top \Sigma \mathbf{w})^2 \quad (5)$$

In the plug-in approach, the estimated parameters  $\hat{\theta}$  are substituted or *plugged in* into the objective function, and the optimal portfolio weights are chosen as if the estimated parameters were the true ones. For example, the risk budgeting portfolio optimization problem in the plug-in approach is defined as

$$\mathbf{w}^* = \arg \min_{w \in \mathcal{W}} U(\mathbf{w}; \mathbf{b}, \Sigma) \quad (6)$$

where  $\mathcal{W} = \left\{ (w_1, w_2, \dots, w_n) \in \mathbb{R}^n \mid \sum_{i=1}^n w_i = 1 \right\}$ .

## 4 End-to-End Frameworks

The end-to-end portfolio provides a unified framework for estimation, optimization allocation, and risk management in a classical static problem with two periods: the decision period  $t$  and the realization period  $t + 1$ .

Let  $(X_t, X_{t+1} \in \mathcal{X})$  represent the input and output variables in a predictive problem. We assume that the variables follow an unknown stationary distribution  $\mathcal{D}$  whose moments provide relevant information for the portfolio problem.

The general end-to-end portfolio problem will be described as follows:

$$\underset{\theta}{\text{minimize}} \quad R_\theta(Z) := E_t[f(X_t, X_{t+1}, z^*(X_t; \theta))] \quad (7a)$$

$$\text{subject to} \quad z^*(X_t, \theta) = \underset{z \in \mathcal{Z}}{\text{argmin}} \quad E_D[c(H(X_t), z)] \quad (7b)$$

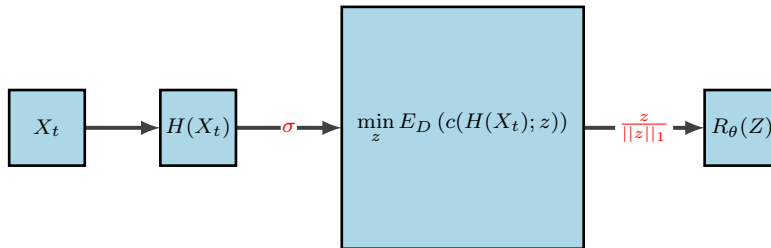


Figure 1: Neural network architecture for generic portfolio end-to-end problem

As explained in section 2, [Uysal et al., 2024] proposed end-to-end modelling framework for risk budgeting portfolios. The empirical illustration is based on a portfolio of

seven exchange-traded funds (ETFs). The framework employs a fully connected neural network that begins with a hidden layer of seven neurons activated by a Leaky ReLU function (with parameter  $\alpha = 0.1$ ). This is followed by a second hidden layer of seven neurons—representing the risk budget vector—utilizing a softmax activation function. Next, a CvxPyLayer is incorporated to solve the risk budgeting portfolio optimization problem, and a final layer computes a risk-reward function based on the normalized outcomes of this convex optimization. The training algorithm adopts a rolling window approach. For the market data experiments, approximately 35% of the dataset is used for training, 25% for hyperparameter optimization, and the remaining 40% for evaluation. The experiments report results for both the Sharpe ratio and cumulative return as risk-reward functions. In the following section we replicate [?] end-to-end framework and the main empirical results in order to illustrate the dependence of this particular setup on the initial conditions of the neural network weights used to implement the framework.

### 4.1 Stability to initial conditions

To show the stability to initial conditions that exists in standard end-to-end frameworks, we replicate the model-based approach proposed in [?], we ran the training algorithm with optimized parameters for each risk-reward function: Sharpe ratio and Cumulative return. Using 15 different random seeds, i.e., 15 different initializations of the neural network weights.

Figure 2 illustrates the stability to initial conditions by showing the regions defined by the minimum ( $\min(t)$ ) and maximum ( $\max(t)$ ) cumulative returns achieved during the training period for each risk-reward function. Additionally, we plot the time series of midpoints of these regions, providing a better understanding for the measure of dispersion at each time step, that in our case is the range or dispersion estimator:

$$v(t) = \max(t) - \min(t).$$

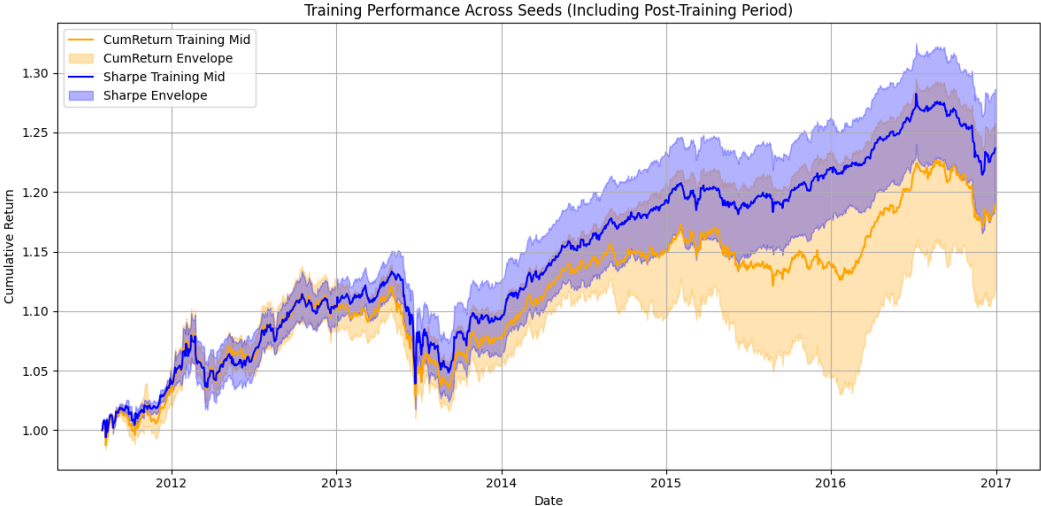


Figure 2: Dispersion of cumulative return during the training period for 15 seeds.

It is important to clarify that we do not expect dispersion to be zero, as this could indicate an overfitting issue. Some variation across seeds is natural due to the stochastic optimization algorithm and the inherent bias-variance tradeoff.

However, following the argument in [Zhong et al., 2024], we emphasize that in industry, portfolio allocation strategies should not exhibit excessive dispersion in results when using numerical optimization. This is why we aim to eliminate the dependence of the trained model’s performance on its weight initialization, which is a reasonable and practical objective.

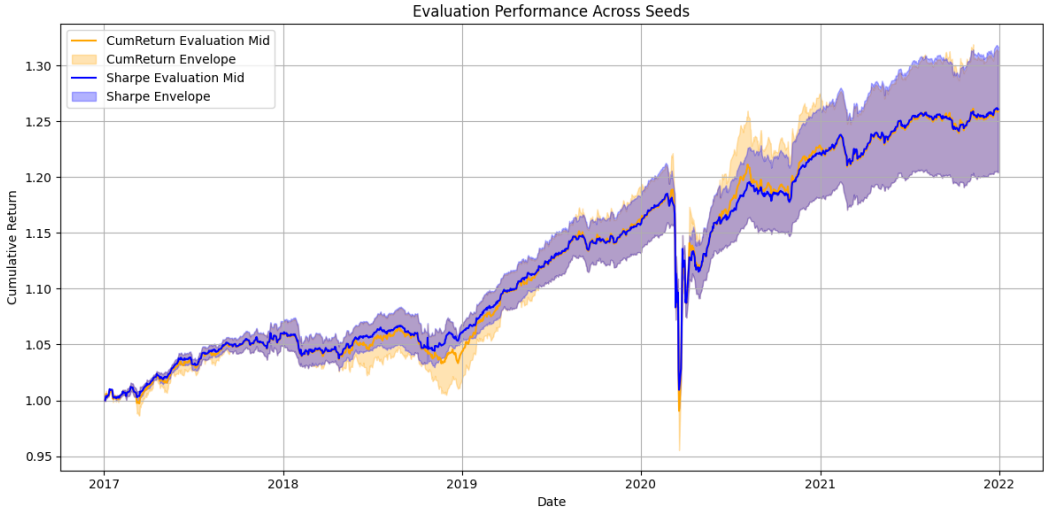


Figure 3: Dispersion of cumulative return during the evaluation period for 7 seeds.

Figure 3 presents the same setup for the evaluation period, further justifying our objective of improving the robustness of the risk-budgeting portfolio construction. Specifically, we aim for a significant reduction in the range  $v(t)$  across all  $t$  in the training period to ensure that decisions made during evaluation remain reliable. A summary of the dispersion results for this replication is shown in Table 1.

Table 1: Summary of dispersion results

	Cumulative Return		Sharpe Ratio	
	Training	Evaluation	Training	Evaluation
Maximum $v(t)$	20.07 %	11.52 %	10.19 %	11.32%
Average $v(t)$	7.78 %	4.87 %	5.83 %	4.49%
Last day $v(t)$	13.75 %	10.87 %	9.96 %	11.27%

## 4.2 Stability-Focused End-to-End Framework

Recent literature has been shifting towards developing robust end-to-end frameworks for portfolio allocation. In [Costa and Iyengar, 2023] the authors introduced a decision layer formulated to maximize an expression similar to a mean-variance utility function. However, instead of using portfolio variance as a risk measure, they define deviation

risk measure estimated over a distribution of data that admits ambiguity concerning the true distribution of the data. Under this set up the decision layer can be framed as a minmax problem that can be efficiently managed through convex duality

One common critique of classical frameworks, such as Markowitz’s theory, is that they often yield low-variance but highly biased results. However, we cannot move to the opposite extreme, where high-variance but low-bias results dominate.

Would an investor be willing to rely on a strategy that, due to high variability, might underperform the risk-parity benchmark? The dispersion of the Sharpe ratio in this setup suggests that such cases arise—not due to the fundamental risk-reward tradeoff of investing but rather as a consequence of the inherent limitations of early end-to-end frameworks.

To mitigate the high dispersion caused by instability to initial conditions, we propose several modifications to the previously studied framework. We optimize the number of neurons in the first hidden layer and we use a set of classical and aggressive learning rates to be tested. Next, we introduce a *lower-bounded softmax function* to prevent risk budgets from vanishing before reaching the risk-budgeting optimization layer. This ensures that allocations remain meaningful and avoids degenerate solutions where certain assets receive near-zero weight allocations.

### 4.2.1 Forward Pass

Figure 4 presents the computational graph of the proposed end-to-end framework. Given an initial dataset of asset features, the forward pass estimates the risk budget vector, utilizes this estimation to solve the risk budgeting optimization problem, normalizes the resulting allocation, and finally computes the loss function based on the obtained portfolio weights.

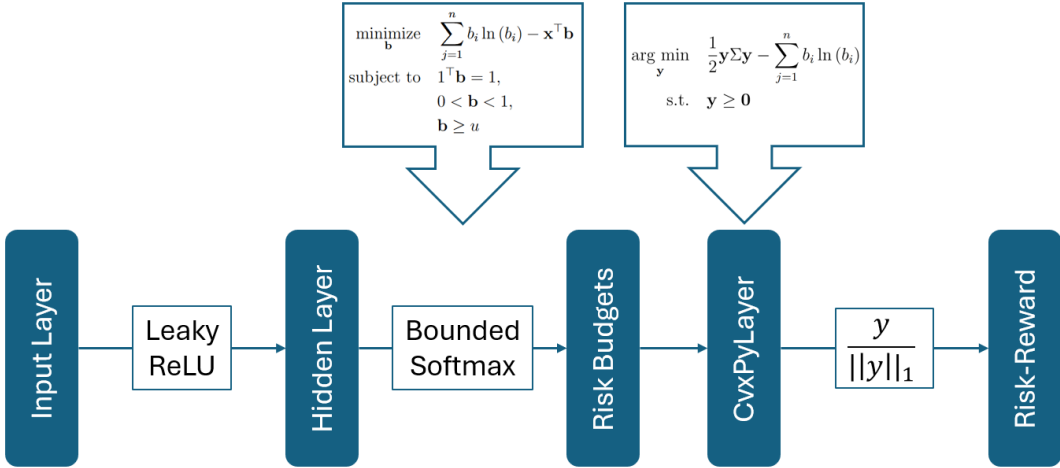


Figure 4: Scheme of the architecture.

### Bounded Softmax

The softmax activation function transforms an n-dimensional input vector ( $\mathbf{x}$ ) into a probability-like distribution ( $\mathbf{b}$ ), which, in this case, represents the investor’s risk

budgets. The standard softmax function is defined as:

$$\mathbf{b}_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}} \quad (8)$$

The standard softmax allows risk budget values that are arbitrarily close to zero. However, as demonstrated in [Roncalli, 2013], the risk budgeting optimization problem has a unique solution only when all risk budgets ( $\mathbf{b}$ ) are strictly positive. If any risk budget is exactly zero, the problem loses uniqueness, introducing instability in the optimization process.

We find that using an unbounded softmax activation to generate risk budgets leads to instability in both problem formulation and solutions. Specifically, certain training periods exhibited near-zero risk budgets (e.g.,  $10^{-6}$ ), which effectively behaved as computationally vanishing risk budgets, leading to unreliable allocations. This phenomenon was confirmed by solving the optimization problem outside the neural network framework for comparison. From a financial perspective, enforcing a strictly positive lower bound on risk budgets is also meaningful. Portfolios with vanishing risk budgets imply complete exclusion of assets, contradicting diversification principles. Additionally, even minimal exposure to all assets ensures a better-conditioned optimization problem and reduces overfitting to specific asset groups.

To address this issue, we introduce a modified softmax activation function that incorporates a lower bound constraint. This is achieved by solving the following convex optimization problem:

$$\begin{aligned} & \underset{\mathbf{b}}{\text{minimize}} && \sum_{j=1}^n b_j \ln(b_j) - \mathbf{x}^\top \mathbf{b} \\ & \text{subject to} && \mathbf{1}^\top \mathbf{b} = 1, \\ & && 0 < \mathbf{b} < 1, \\ & && \mathbf{b} \geq u \end{aligned} \quad (9)$$

The solution to this problem results in a softmax-like function with a lower bound  $u$ . This result is formalized in Theorem 1, with the proof provided in Appendix A.

**Theorem 1** (Bounded Softmax). Let be  $u$  the desired lower bound to the softmax function and  $k = \#(A)$ , with

$$A = \left\{ i \in \{1, 2, \dots, n\} : \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}} \geq u \right\}$$

The function

$$\mathbf{b}_i(\mathbf{x}) = \begin{cases} \frac{e^{x_i}}{\sum_{j \in A} e^{x_j}} \cdot (1 - (n - k)u) & \text{if } i \in A \\ u & \text{if } i \notin A \end{cases} \quad (10)$$

is the unique solution to problem 9.

### Risk Budgeting Optimization Layer

[Agrawal et al., 2019] introduced the concept of implicit layers in neural networks that solve convex optimization problems formulated as *Disciplined Convex Programs*. The bounded softmax function presented above is an example of such an implicit layer. Similarly, we aim to integrate the risk budgeting optimization problem within the neural network framework by formulating it as a convex optimization problem.

[Feng and Palomar, 2015] showed that classical risk budgeting problem is non-convex, making direct optimization challenging, especially for large asset universes. However, [Richard and Roncalli, 2019] demonstrated an equivalent convex formulation of the problem, allowing for computationally efficient optimization:

$$\begin{aligned} \mathbf{y}^* = \arg \min_{\mathbf{y}} \quad & \frac{1}{2} \mathbf{y}^\top \Sigma \mathbf{y} - \sum_{j=1}^n b_j \ln(y_j) \\ \text{s.t.} \quad & \mathbf{y} \geq \mathbf{0} \end{aligned} \tag{11}$$

The optimal portfolio weights are then obtained as:

$$\mathbf{w}^* = \frac{y_i^*}{\sum_{j=1}^n y_j^*}$$

Notably, we use the quadratic form  $\frac{1}{2} \mathbf{y}^\top \Sigma \mathbf{y}$  instead of the original square root formulation  $\sqrt{\mathbf{y}^\top \Sigma \mathbf{y}}$  for computational efficiency. Since both functions are monotonic, they yield the same optimal solutions, while the quadratic form is more amenable to numerical optimization.

By embedding this convex optimization layer into the neural network, we enable efficient learning of risk budget allocations while ensuring that the obtained solutions remain well-conditioned and financially meaningful. The next section describes the backward pass, detailing how gradients propagate through the framework to facilitate end-to-end learning.

#### 4.2.2 Backward Pass

The backward pass ensures that the gradients of all parameters within the end-to-end framework are correctly computed and propagated through the layers. This enables efficient optimization of the model by leveraging automatic differentiation and convex optimization layers.

Once the forward pass completes, the gradients of the loss function with respect to the model parameters must be computed. Let  $\mathcal{R}$  denote the loss function. Using the chain rule of differentiation, the gradients propagate backward from the loss function to the earlier layers of the network. Mathematically, for any parameter  $\theta$  in the network, we compute:

$$\frac{\partial \mathcal{R}}{\partial \theta} = \frac{\partial \mathcal{R}}{\partial \mathbf{w}} \cdot \frac{\partial \mathbf{w}}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \mathbf{b}} \cdot \frac{\partial \mathbf{b}}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{x}}{\partial \theta}$$

Each term in this expression represents a gradient computed at a specific stage of the computational pipeline.

## Differentiability of the Risk Budgeting Optimization Layer

Since the risk budgeting allocation is obtained through a convex optimization layer, special care must be taken to ensure differentiability. The implicit function theorem [Agrawal et al., 2019] provides a way to differentiate the optimal solution  $\mathbf{y}^*$  with respect to the risk budget vector  $\mathbf{b}$ . Given the convex program:

$$\mathbf{y}^* = \arg \min_{\mathbf{y}} \frac{1}{2} \mathbf{y} \Sigma \mathbf{y} - \sum_{j=1}^n b_j \ln(b_j)$$

the gradient  $\frac{\partial \mathbf{y}^*}{\partial \mathbf{b}}$  can be obtained by differentiating the KKT conditions. The resulting jacobian provides the necessary gradients for backpropagation.

## Differentiability of the Risk Budgeting Optimization Layer

The bounded softmax layer introduces additional constraints by ensuring a lower bound  $u$  on the risk budgets. Its gradient propagation follows from differentiating the optimality conditions of the constrained optimization problem (9). The Jacobian of the bounded softmax transformation is computed as:

$$\frac{\partial \mathbf{b}}{\partial \mathbf{x}} = J(\mathbf{x})$$

where  $J(\mathbf{x})$  is derived in Appendix A. This Jacobian is then used to propagate gradients to earlier layers.

## Backpropagation

Using the computed Jacobians, the gradients of the loss function are propagated through the model. The backward pass consists of:

1.  $\frac{\partial \mathcal{R}}{\partial \mathbf{w}}$ : Loss function gradient with respect to the normalized allocation.
2.  $\frac{\partial \mathbf{w}}{\partial \mathbf{y}}$ : Allocation gradient with respect to the risk budgeting layer solution.
3.  $\frac{\partial \mathbf{y}}{\partial \mathbf{b}}$ : Propagating gradients through the risk budgeting optimization layer.
4.  $\frac{\partial \mathbf{b}}{\partial \mathbf{x}}$ : Propagating gradients through the bounded softmax layer.
5.  $\frac{\partial \mathbf{x}}{\partial \theta}$ : Propagating gradients to earlier neural network layers for parameter updates.

### 4.2.3 Loss Function

There are numerous ways to define a loss function for an end-to-end framework. For example, [Uysal et al., 2024] and [Anis and Kwon, 2025] employed decision-based measures, specifically the Sharpe Ratio, to guide the learning process. In their approach, the

model is trained by maximizing the Sharpe Ratio, effectively making the loss function its negative counterpart:

$$\mathcal{R}_{SR}(\mathbf{w}) = -\frac{\mathbb{E}[r^w]}{\sigma^w}$$

where  $r_w$  represents the portfolio return for allocation  $\mathbf{w}$  and  $\sigma_w$  its standard deviation. We found this loss function gives a dual formulation aligned with the nature of risk budgeting, where the convex optimization layer minimizes portfolio variance, while the neural network seeks to maximize expected returns. By optimizing these complementary objectives, the model achieves a balance between risk and return, leading to robust portfolio allocations.

For the analysis of stability we will also use the Sharpe Ratio and the cumulative return to define the loss function to train two different architectures, having a duality in the end-to-end problem: the risk budgeting minimize the variance and the neural network maximize the return, using the cumulative return as our standard performance metric for results.

$$\mathcal{R}_{CR}(\mathbf{w}) = -\prod_{t=1}^T (1 + r_t^w)$$

## 5 Computational Experiments

### 5.1 Computational Setup

All computations were conducted using the Python programming language. The bounded softmax and risk budgeting optimization problems were implemented as implicit layers within a neural network, following the architecture presented in Figure 4. These layers were formulated using disciplined convex programming principles and solved using the ECOS 2.0.14 solver, ensuring numerical stability and efficient optimization.

The convex optimization layers were implemented using the `cvxpylayers` package within the PyTorch framework, allowing for seamless integration of differentiable convex programs into the end-to-end learning process. This approach enables efficient gradient propagation through the optimization layers, facilitating stable training and robust convergence.

The neural network training process was executed on a machine equipped with a 13th Gen Intel(R) Core(TM) i7-13620H 2.40GHz processor and 16.0 GB RAM.

### 5.2 Addressing stability concerns

This first set of results demonstrates the improvements in stability achieved by our proposed framework. To assess this improvement, we use the framework introduced in [Uysal et al., 2024] as a benchmark for comparison.

#### 5.2.1 Data

For this stability analysis, we use daily returns from seven exchange-traded funds (ETFs) to represent broad stock, bond, and commodity market conditions. The selected ETFs include: VTI (Vanguard Total Stock Market ETF), IWM (iShares Russell

2000 ETF), AGG (iShares Core U.S. Aggregate Bond ETF), LQD (iShares iBoxx Investment Grade Corporate Bond ETF), MUB (iShares National Muni Bond ETF), DBC (Invesco DB Commodity Index Tracking Fund), and GLD (SPDR Gold Shares). The dataset spans from 2011 to 2021, with daily frequency, and was sourced from Yahoo Finance.

To construct the dataset, we compute daily log-returns based on adjusted closing prices to account for dividends and stock splits. Any missing values due to market closures were forward-filled to maintain data continuity. The dataset is split into two periods:

- In-Sample Period (2011-2016/12): Used for training and hyperparameter tuning.
- Out-of-Sample Period (2017-2021/12): Used for evaluating model performance.

For hyperparameter selection, we further subdivide the in-sample data, using 2011-2014/12 for training and 2015-2016/12 for validation. The ETFs were selected to ensure diverse market representation, allowing the framework to adapt to various economic conditions. We assume a frictionless market and a static rebalancing strategy.

### 5.2.2 In-Sample Results

To determine the optimal hyperparameters, we perform a random grid search with 100 iterations during the validation period (2015-01-01 to 2016-12-31). The hyperparameter search space is defined as follows:

- Hidden layer neurons:  $\{7, 16, 32\}$ ,
- Learning rate:  $\{0.05, 0.1, 0.5, 1.0, 5.0, 10, 50, 100, 150, 200, 250, 300, 350, 400\}$ ,
- Training steps:  $\{5, 10, 15, 20, 25, 30\}$

The learning rate is a hyperparameter that determines how much a model’s parameters (weights and biases) are adjusted in response to the loss at each training step. Specifically, during backpropagation, the network computes the gradient of the loss function with respect to each parameter, and the learning rate scales these gradients before updating the parameters. A high learning rate can speed up the initial learning process by taking larger steps in the parameter space; however, it may also cause instability and overshooting of the minimum, leading to divergent or oscillatory training. Conversely, a low learning rate results in smaller, more controlled updates that often promote more reliable convergence—but it can slow down training significantly and may cause the model to become trapped in suboptimal local minima.

Optimizing the number of neurons in a neural network is equally critical because it directly affects the model’s capacity to learn and generalize. Too few neurons may fail to capture the complexity of the data, resulting in underfitting where the model misses important patterns. On the other hand, too many neurons can lead to overfitting, where the model memorizes the training data (including noise) and performs poorly on unseen data. The objective is to strike a balance that provides enough complexity to model the data effectively without overcomplicating the network, ensuring robust performance on both training and validation datasets.

Finally, training steps refer to the number of iterations during which the model updates its parameters. At each step, a batch of data is processed, the loss is computed, and gradients are used to adjust the model’s weights. The number of training steps determines how long the model has to learn: too few steps may result in underfitting, while too many can cause overfitting. To maintain a simple architecture, we decided to optimize this parameter as a fixed hyperparameter. However, another alternative is to run the training algorithm for the number of steps required to reach a desired loss level.

We include “aggressive” learning rates (i.e., values  $\geq 1.0$ ) to compare with those used by [Uysal et al., 2024]. Figure 5 illustrates the results of our hyperparameter optimization. Each line in the figure corresponds to one model configuration, showing how the chosen hyperparameter values—such as learning rate, number of neurons in the hidden layer, and number of training steps—lead to a particular Sharpe ratio at the end of the validation period. The red line represents the best model, which we identify by its highest Sharpe ratio during validation. Meanwhile, the four blue lines indicate other model configurations that also performed well, completing the top five models (i.e. highest Sharpe ratio) evaluated during the hyperparameter optimization.

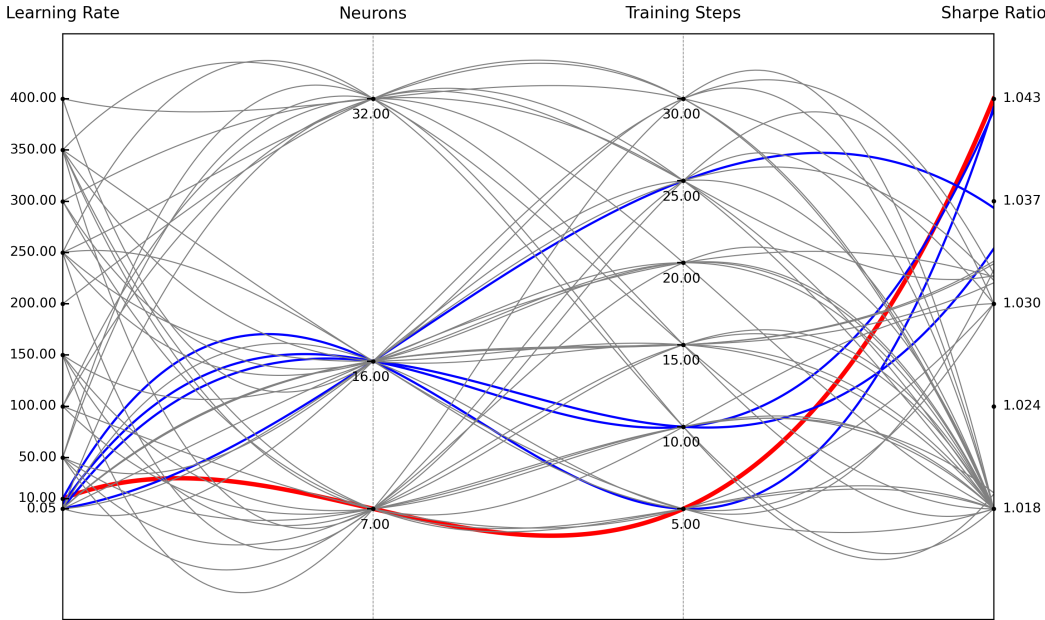


Figure 5: Hyperparameter Tuning results for Sharpe Ratio

The best model (red line) is an end-to-end framework featuring a hidden layer with 7 neurons, a learning rate of 10, and trained over 5 steps. In contrast, the other top-performing models (blue lines) utilize a hidden layer with 16 neurons and require a greater number of training steps, resulting in a more computationally complex architecture. A similar process is carried out using cumulative return as the loss function, yielding the results shown in Figure 6. In this scenario, the best model employs 16 neurons and requires more training steps. In contrast, the model trained with the Sharpe Ratio converges faster, which is expected given its lower neuron count and reduced computational complexity.

To assess the stability of the optimized models, we repeat the training process across 15 different random seeds and measure the dispersion of cumulative returns during the training period. The results for this stability analysis are presented in Figure 7.

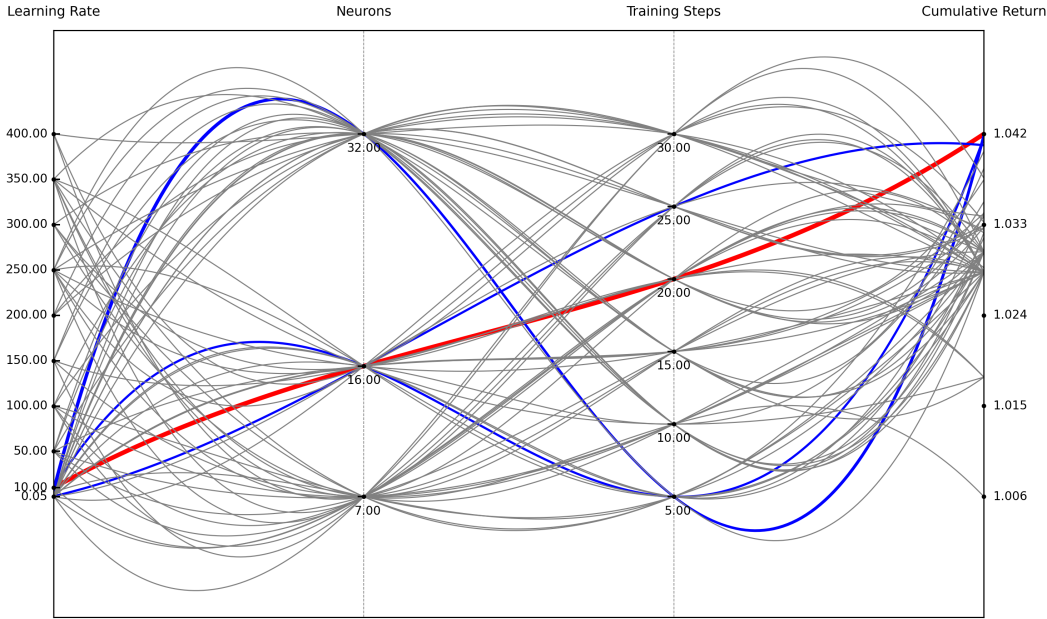


Figure 6: Hyperparameter Tuning results for Cumulative Return

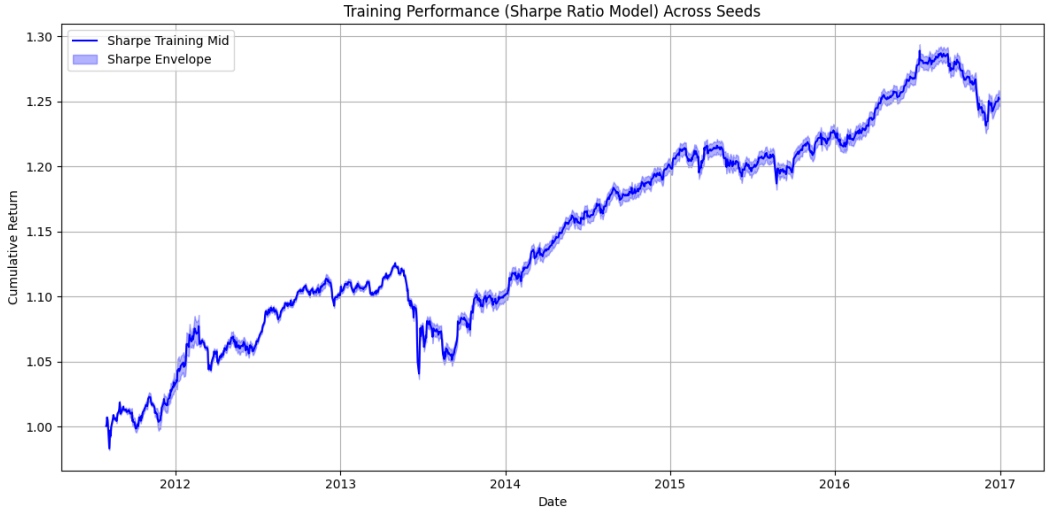


Figure 7: Dispersion of cumulative return during training period for sharpe-ratio.

A comparison of the results of the dispersion (measured as the range statistic) between our framework and the replication of the existing approach in [Uysal et al., 2024] is summarized in Table 2. The results demonstrate a significant reduction in dispersion during both the training period and the evaluation period, highlighting the stability improvements introduced by our framework when using the Sharpe Ratio as the risk-reward function.

We repeat this process for the model trained using the cumulative return, we show the graph that represent the results for this case in Figure 8. Although for this risk-reward function the Uysal model was more stable (lower dispersion than sharpe ratio’s model) we still are getting better results.

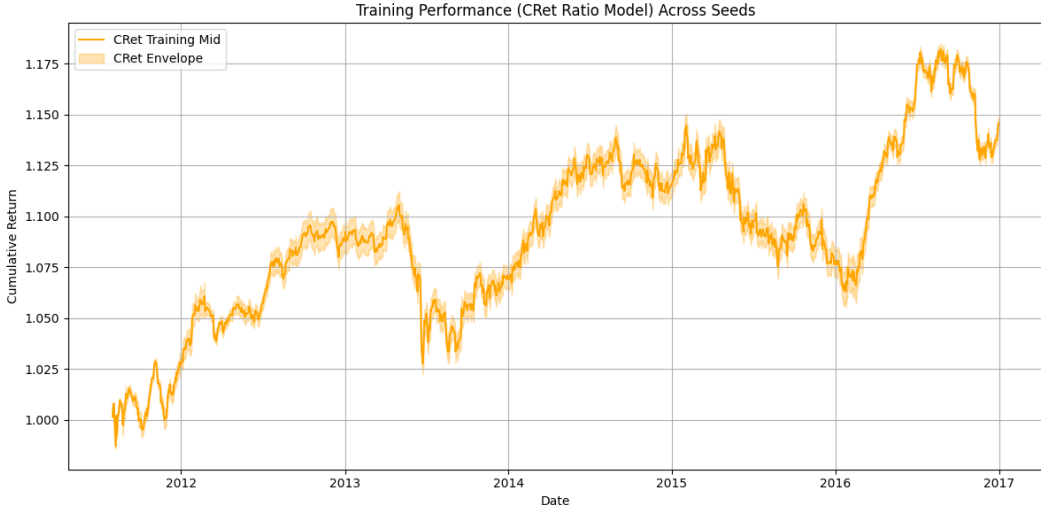


Figure 8: Dispersion of cumulative return during training period for Cumulative Return.

Table 2: Summary of dispersion statistics for In-Sample period.

	Cumulative Return		Sharpe Ratio	
	Uysal	Proposed	Uysal	Proposed
Maximum $v(t)$	20.07 %	<b>1.71%</b>	10.19 %	<b>1.91%</b>
Average $v(t)$	7.78 %	<b>1.02%</b>	5.83 %	<b>0.81%</b>
Last day $v(t)$	13.75 %	<b>0.56%</b>	9.96 %	<b>1.13%</b>

Finally, we assess whether the observed improvements in stability come at the expense of performance, measured in terms of cumulative return. Figure 9 has the time series of average cumulative return for both best models across 15 seeds and confirms that, despite the stabilization of results, we expect an end-to-end framework trained with Sharpe-Ratio that outperforms during the whole In-Sample period the risk-parity benchmark. Moreover, our architecture exhibits convergence independent to initial conditions, ensuring a global optimum solution.

By other hand, the end-to-end framework trained with cumulative returns seems to be more sensitive to market conditions and giving similar outcomes that the risk parity strategy. This result is not surprising since the training algorithm of neural network parameters are not taking into account the return-variance trade off.

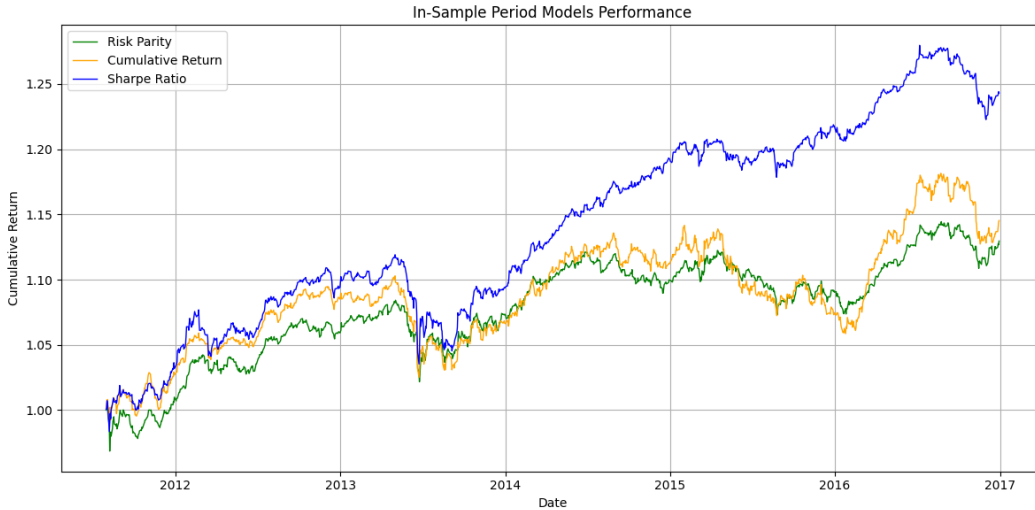


Figure 9: Mean Cumulative return of optimized end-to-end models.

### 5.2.3 Out-of-Sample Results

To assess the robustness and generalization capability of our proposed framework, we evaluate its performance during the out-of-sample period (2017-2021/12). Specifically, we analyze the dispersion of cumulative returns across multiple random seeds to examine the stability of the model to different initializations.

Figure 10 presents the dispersion of cumulative return trajectories obtained from 15 different random seeds during the evaluation period. The results indicate that our framework maintains the performance in out-of-sample conditions, mitigating the instability to initial conditions observed in the benchmark model. Moreover, it shows that we do not have an overfitting problem using this data split.

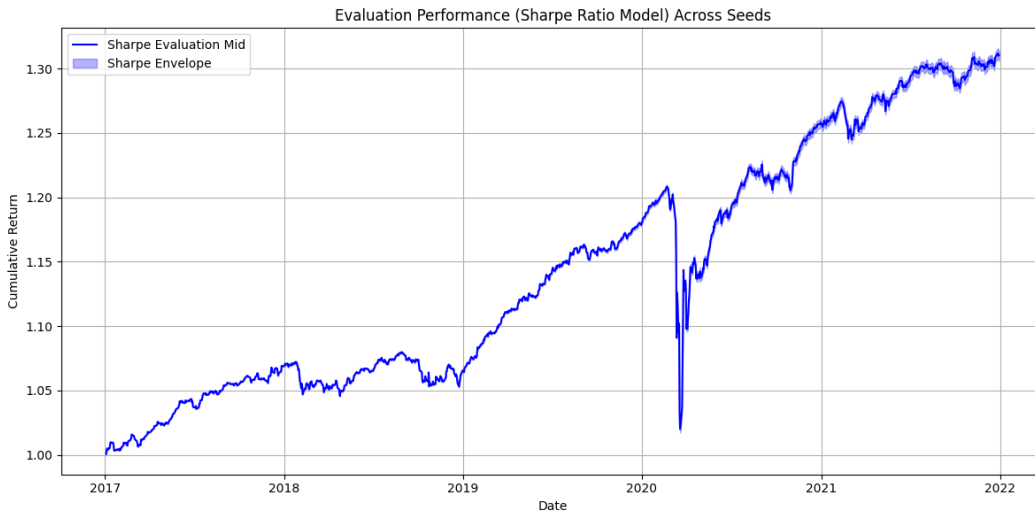


Figure 10: Dispersion of cumulative returns during evaluation period for sharpe-ratio.

To further quantify the stability improvements, Table 3 compares the dispersion results of our framework against those of the benchmark model from [?]. The pro-

posed model exhibits significantly lower dispersion throughout the evaluation period, reinforcing its capacity to produce consistent portfolio allocations.

Again, this process is also done for the model trained with cumulative return, obtaining the results in Figure 11 and Table 3.

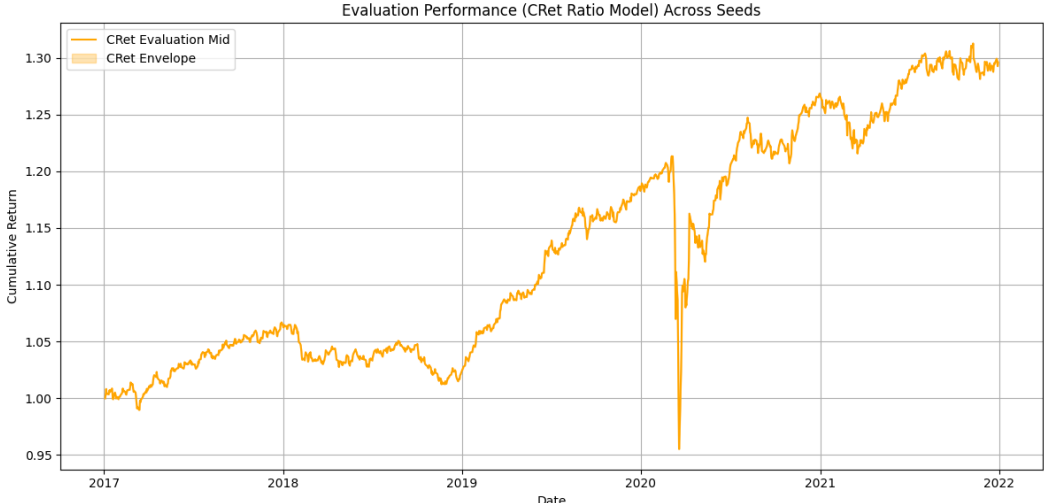


Figure 11: Dispersion of cumulative returns during evaluation period for sharpe-ratio.

Table 3: Summary of dispersion statistics for Out-of-Sample period.

	Cumulative Return		Sharpe Ratio	
	Uysal	Proposed	Uysal	Proposed
Maximum $v(t)$	11.52 %	<b>0.0183%</b>	11.32 %	<b>0.87%</b>
Average $v(t)$	4.87 %	<b>0.0073%</b>	4.49 %	<b>0.29%</b>
Last day $v(t)$	10.87 %	<b>0.0181%</b>	11.27 %	<b>0.78%</b>

These findings confirm that our end-to-end framework not only stabilizes results in the training phase but also preserves its advantages in an out-of-sample setting. The reduced dispersion in cumulative returns suggests that our approach enhances consistency and robustness, addressing the instability observed in the benchmark framework. Moreover, the sensitivity to market conditions of the model trained with cumulative returns is again an observable feature in Figure 12: After 2019, the model gains momentum and reaches the performance of the Sharpe Ratio–trained model starting in the third quarter of the year; from that point on, both models recover similarly from the shock caused by the COVID-19 pandemic. Most important, both end-to-end frameworks still improve the performance of risk parity strategies, supporting the results in the existing literature.

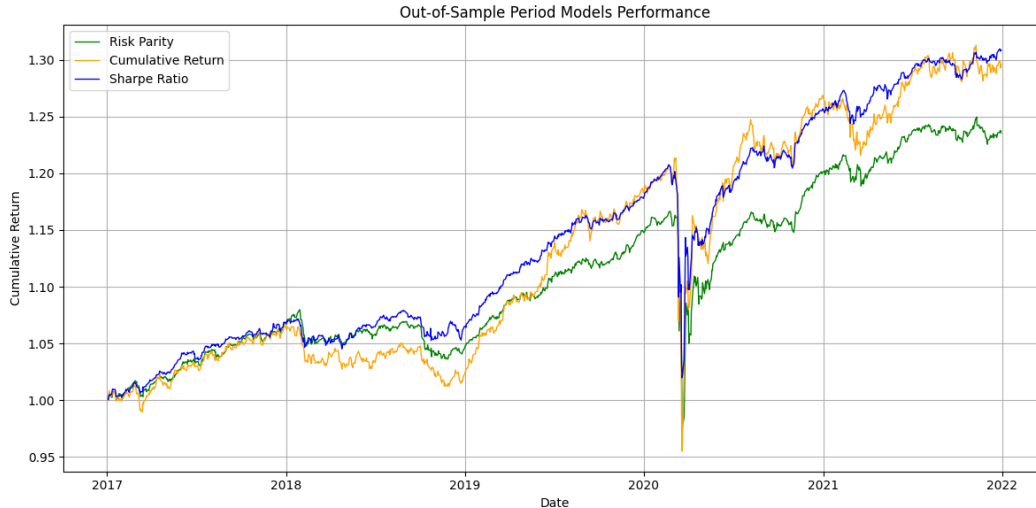


Figure 12: Mean Cumulative return of optimized end-to-end models.

## 6 Conclusion

In this study, we explored the existing numerical instability of end-to-end models with respect to the initial weight conditions in neural networks, aiming to assess the feasibility of constructing an environment capable of yielding reliable investment strategies. Our focus on risk-budgeting portfolios is particularly relevant given the current debate surrounding risk-parity strategies in financial markets.

Our findings reveal that the proposed model significantly reduces the dispersion of the “optimal” solution when trained under different initial weight conditions. This reduction in dispersion indicates that the architecture converges toward a global solution, effectively avoiding the pitfalls of local optima. We attribute this behavior to our model’s treatment of risk factors: by constraining these factors from reaching extreme values, we maintain the uniqueness of the risk-budgeting solution.

These results underscore the importance of accounting for the specific dynamics of different portfolio optimization models when designing end-to-end architectures. They also pave the way for the broader implementation of deep learning techniques in the financial industry, as a globally convergent model increases confidence in the portfolio management process.

Despite these promising outcomes, our study identifies limitations in the current optimization solvers. Future work should focus on integrating more robust solvers, such as those available through CvxPyLayer, to enhance performance and extend the applicability of our approach. Additionally, exploring the scalability of this methodology to larger and more diverse asset classes, as well as its performance in live market environments, would be valuable. A comparative analysis with traditional portfolio optimization methods could further highlight the advantages of our deep learning-based strategy.

## 6.1 Work In Progress

We plan to extend the use of risk-reward functions to a broader range of satisfaction measures by following the framework presented in [Meucci, 2007]. Specifically, we will define the loss function in an unsupervised setting and evaluate our architecture using representative measures such as the certainty equivalent (expected utility indices), Value at Risk (quantile-based indices), and Expected Shortfall (coherent indices). To assess robustness, we will test the model’s performance using real market conditions from recent years, and using several metrics in literature not only cumulative return e.g. maximum drawdown, Information ratio, etc.

By conducting these experiments, our goal is to develop a robust framework that remains stable across various market conditions and serves as an attractive portfolio management tool in today’s data-rich environment. Additionally, we plan to explore the interpretability of the risk preferences learned by the model and extend the approach to other asset classes, further validating its adaptability and practical utility.

Another research avenue involves incorporating a module into the framework to estimate the covariance matrix using shrinkage techniques for factor models. This work is in its early stages, primarily due to the dimensionality challenges encountered when constructing portfolios with thousands of assets. By doing so, we aim to develop an optimal alternative to relying on the sample covariance matrix, which is notoriously unstable in high-dimensional settings. To our knowledge, the only attempt in the deep learning literature to improve covariance estimation for static portfolio settings is presented in [Huynh and Lenhard, 2022], which employs an asymmetric autoencoder model.

## References

- [Agrawal et al., 2019] Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, J. Z. (2019). Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, pages 9558–9570.
- [Anis and Kwon, 2025] Anis, H. T. and Kwon, R. H. (2025). End-to-end, decision-based, cardinality-constrained portfolio optimization. 320:739–753.
- [Brodie et al., 2009] Brodie, J., Daubechies, I., Mol, C. D., Giannone, D., and Loris, I. (2009). Sparse and stable markowitz portfolios. *PNAS Proceedings of the National Academy of Sciences of the United States of America*, 106(30):12267–12272.
- [Butler and Kwon, 2022] Butler, A. and Kwon, R. H. (2022). Data-driven integration of norm-penalized mean-variance portfolios.
- [Costa and Iyengar, 2023] Costa, G. and Iyengar, G. N. (2023). Distributionally robust end-to-end portfolio construction. 23:1465–1482.
- [Dai and Wen, 2018] Dai, Z. and Wen, F. (2018). Some improved sparse and stable portfolio optimization problems. 27:46–52.
- [Fabozzi et al., 2010] Fabozzi, F. J., Huang, D., and Zhou, G. (2010). Robust portfolios: contributions from operations research and finance. 176:191–220.

- [Feng and Palomar, 2015] Feng, Y. and Palomar, D. P. (2015). Scrip: Successive convex optimization methods for risk parity portfolio design. *IEEE Transactions on Signal Processing*, 63(19):5285–5300.
- [Georgantas et al., 2024] Georgantas, A., Doumpos, M., and Zopounidis, C. (2024). Robust optimization approaches for portfolio selection: a comparative analysis. 339:1205–1221.
- [Huynh and Lenhard, 2022] Huynh, K. and Lenhard, G. (2022). Asymmetric autoencoders for factor-based covariance matrix estimation. *In Proceedings of the Third ACM International Conference on AI in Finance*, pages 403–410.
- [Meucci, 2007] Meucci, A. (2007). *Risk and Asset Allocation*. Springer.
- [Richard and Roncalli, 2019] Richard, J.-C. and Roncalli, T. (2019). Constrained risk budgeting portfolios: Theory, algorithms, applications & puzzles. available at ssrn: <https://ssrn.com/abstract=3331184>.
- [Roncalli, 2013] Roncalli, T. (2013). *Introduction to risk parity and budgeting*. CRC Press.
- [Uysal et al., 2024] Uysal, A. S., Li, X., and Mulvey, J. M. (2024). End-to-end risk budgeting portfolio optimization with neural networks. 339:397–426.
- [Xidonas et al., 2020] Xidonas, P., Steuer, R., and Hassapis, C. (2020). Robust portfolio optimization: a categorized bibliographic review. 292:533–552.
- [Zhang et al., 2021] Zhang, C., Zhang, Z., Cucuringu, M., and Zohren, S. (2021). A universal end-to-end approach to portfolio optimization via deep learning.
- [Zhong et al., 2024] Zhong, J., Xu, Z., Wang, S., Wen, X., Guo, J., and Xu, Q. (2024). Dspo: An end-to-end framework for direct sorted portfolio construction.

## A Bounded Softmax Function

**Theorem 2** (Bounded Softmax). Let be  $u$  the desired lower bound to the softmax function and  $k = \#(A)$ , with

$$A = \left\{ i \in \{1, 2, \dots, n\} : \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}} \geq u \right\}$$

The function

$$\mathbf{b}_i(\mathbf{x}) = \begin{cases} \frac{e^{x_i}}{\sum_{j \in A} e^{x_j}} \cdot (1 - (k - n)u) & \text{if } i \in A \\ u & \text{if } i \notin A \end{cases} \quad (12)$$

is the unique solution to problem 9.

*Proof.* The KKT conditions for this problem are

- **Stationary:**

$$\mathbf{0} = \frac{\partial}{\partial \mathbf{b}} \left( \sum_{j=1}^n b_j \ln(b_j) - \mathbf{x}^\top \mathbf{b} - \sum_{j=1}^n \gamma_j b_j + \sum_{j=1}^n \eta_j (b_j - 1) + \dots \right. \\ \left. \dots + \sum_{j=1}^n \mu_j (u - b_j) + \lambda \left[ 1 - \sum_{j=1}^n b_j \right] \right)$$

- **Complementary slackness:**

$$-b_j \gamma_j = 0, \quad \eta_j (b_j - 1) = 0, \quad \mu_j (u - b_j) = 0$$

- **Primal Feasibility:**

$$\mathbf{1}^\top \mathbf{b} = 1, \quad 0 < \mathbf{b} < 1, \quad \mathbf{b} \geq u$$

Then, from the stationary condition we get

$$\frac{\partial}{\partial \mathbf{b}} (\mathbf{b}^\top \ln(\mathbf{b}) - \mathbf{x}^\top \mathbf{b} - \gamma^\top \mathbf{b} + \eta^\top (\mathbf{b} - \mathbf{1}) + \mu^\top (\mathbf{u} - \mathbf{b}) + \lambda [1 - \mathbf{1}^\top \mathbf{b}]) = \mathbf{0} \\ \ln(\mathbf{b}) + \mathbf{1} - \mathbf{x} - \gamma + \eta - \mu - \lambda = \mathbf{0} \\ -\mathbf{1} + \mathbf{x} + \gamma - \eta + \mu + \lambda = \ln(\mathbf{b})$$

Therefore, for each  $j = 1, \dots, n$ ,

$$b_j = e^{-1+x_j+\gamma_j-\eta_j+\mu_j+\lambda}$$

Now, using the primal feasibility conditions, since  $0 < \mathbf{b} < 1$ ,  $\gamma_j = \eta_j = 0$ , thus for each  $j = 1, \dots, n$ ,

$$b_j = e^{-1+x_j+\mu_j+\lambda}$$

Hence,

1. If  $b_j > u$ ,  $\mu_j = 0$  and for each  $j = 1, \dots, n$ , since  $\mathbf{b}^\top \mathbf{1} = 1$

$$\sum_{j=1}^n b_j = \sum_{j=1}^n e^{-1+x_j+\lambda} = 1 \implies \lambda = 1 - \ln \left( \sum_{j=1}^n e^{x_j} \right)$$

Then,

$$b_j = e^{-1+x_j+\lambda} \\ b_j = e^{-1+x_j+(1-\ln(\sum_{j=1}^n e^{x_j}))} \\ b_j = e^{x_j - \ln(\sum_{j=1}^n e^{x_j})} \\ b_j = \frac{e^{x_j}}{\sum_{j=1}^n e^{x_j}}$$

which is the classical softmax function (12). Moreover,

$$A = \left\{ i \in \{1, 2, \dots, n\} : \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}} \geq u \right\} = \{1, 2, \dots, n\} \implies k = \#(A) = n$$

and

$$b_j = \frac{e^{x_j}}{\sum_{j=1}^n e^{x_j}} = \frac{e^{x_j}}{\sum_{j=1}^n e^{x_j}} \cdot (1 - (k - n)u)$$

2. If  $b_j > u$  for  $j = 1, \dots, k$  and  $b_j = u$  for  $j = k + 1, \dots, n$ , then

$$\sum_{j=1}^k b_j = \sum_{j=1}^k e^{-1+x_j+\lambda} = 1 - (n-k)u \implies \lambda = 1 + \ln(1 - (n-k)u) - \ln\left(\sum_{j=1}^k e^{x_j}\right)$$

Then, for  $j = 1, \dots, k$

$$\begin{aligned} b_j &= e^{-1+x_j+\lambda} \\ b_j &= e^{-1+x_j+(1+\ln(1-(n-k)u)-\ln(\sum_{j=1}^k e^{x_j}))} \\ b_j &= e^{x_j+\ln(1-(n-k)u)-\ln(\sum_{j=1}^k e^{x_j})} \\ b_j &= \frac{e^{x_j}}{\sum_{j=1}^k e^{x_j}} \cdot (1 - (n-k)u) \end{aligned}$$

which is the classical softmax function (12) applied over the  $k$  coordinates left and multiplied by the remained weight. Thus,

$$b_j = \begin{cases} \frac{e^{x_j}}{\sum_{j=1}^k e^{x_j}} \cdot (1 - (n-k)u) & \text{if } j \in A \\ u & \text{if } j \notin A \end{cases}$$

□