A Numerical example of the EF problem

$$
\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \boldsymbol{B} = \begin{bmatrix} 0.591 \\ 0.749 \\ 0.412 \\ 0.545 \\ 0. \\ 0. \\ 0. \\ 0. \\ -0.81 \\ 1. \\ 0.74 \\ 0.58 \end{bmatrix} \tag{12}
$$

$$
\boldsymbol{C} = [0, 0, 1, 1]; \boldsymbol{\zeta}_{\text{MAX}} = [0.74, 0.58]
$$

$$
\boldsymbol{X}_{\text{MAX}} = [0.5910.749, 0.412, 0.545]
$$

$$
\boldsymbol{X}_{\text{MIN}} = [0, 0, 0, 0]
$$

$$
\alpha_{\text{MIN}} = 0.81, \alpha_{\text{MAX}} = 1
$$

We provide a numerical example of the EF problem in the case of a 4-asset problem with 2 classes and $\alpha_{\text{MIN}} < 1$ (see Eq.12). Only the constraints are presented here. It is important to note that the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ representing all w constraints grow in size by $(2n+2+m)$ as n increases, where m is the number of distinct asset classes. If $\boldsymbol{X}_{\text{MAX}}$ is not defined, then all entries of $\boldsymbol{X}_{\text{MAX}}$ can be set to α_{MAX}. Solving Eq.1 is achieved by directly considering $\boldsymbol{A}$ and $\boldsymbol{B}$ and the covariance matrix $\boldsymbol{Q} = \text{diag}(\boldsymbol{V})\boldsymbol{P}\text{diag}(\boldsymbol{V})$ obtained from the other inputs. If $\mathcal{V}_{\text{min}} = \boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x}$, then we solve a second-order cone program (SOCP) to increase the portfolio's volatility without exceeding $\mathcal{V}_{\text{target}}$ and get better returns.

We obtain the volatility constraint through the Cholesky decomposition of the covariance matrix $\boldsymbol{Q}' = L(\boldsymbol{Q})L(\boldsymbol{Q})^T$ where L is the lower-triangular operator. $\boldsymbol{E} \in \mathbb{R}^{[n+1,n]}$ is built by stacking $\boldsymbol{Q}'$ and $[0, \cdots 0]$ such that $\boldsymbol{F} \in \mathbb{R}^{[n+1,1]} = [\sqrt{\mathcal{V}_{\text{target}}}, 0, \cdots]$. Then, eq. 2 can be reformulated as follow:

$$
\boldsymbol{\phi} := \text{minimize } -\boldsymbol{R}^\top \boldsymbol{x} \text{ subject to } \boldsymbol{A} \leq \boldsymbol{B} \text{ and } \boldsymbol{E} \leq \boldsymbol{F}. \tag{13}
$$

The complete optimal allocation of eq. 3 can be summarized by the following python script:

```python
"""EF evaluation """
import copy
import logging
import os

import cvxopt
import numpy as np

scalar = 10000

def cvxopt_solve_qp(P, q, G=None, h=None, **kwargs):
    P = 0.5 * (P + P.T)  # make sure P is symmetric
    args = [cvxopt.matrix(P), cvxopt.matrix(q)]
    if G is not None:
        args.extend([cvxopt.matrix(G), cvxopt.matrix(h)])
    sol = cvxopt.solvers.qp(*args, **kwargs)
    if sol["status"] != "optimal":
        raise ValueError("QP SOLVER: sol.status != 'optimal'")
    return np.array(sol["x"]).reshape((P.shape[1],)), num_iterations


def cvxopt_solve_socp(c, Gl, hl, Gq, hq, **kwargs):
    args = [cvxopt.matrix(c), cvxopt.matrix(Gl), cvxopt.matrix(hl), [cvxopt.matrix(Gq)],
    ↪  [cvxopt.matrix(hq)]]
    sol = cvxopt.solvers.socp(*args, **kwargs)
    num_iterations = sol["iterations"]
    if sol["status"] != "optimal":
        raise ValueError("SOCP SOLVER: sol.status != 'optimal'")
    return np.array(sol["x"]).reshape((Gl.shape[1],)), num_iterations


def efficient_frontier(max_weights, vol_target, conditions, condition_max, vol, ret, correl):
    n_assets = len(max_weights)
    min_weights = [0] * n_assets
    v_t = vol_target * vol_target * scalar
    G = np.vstack([np.eye(n_assets), -np.eye(n_assets), conditions])
    H = np.hstack([max_weights, min_weights, condition_max])
    cov = scalar * np.matmul(np.matmul(np.diag(vol), correl), np.diag(vol))
    wt = cvxopt_solve_qp(cov, np.zeros_like(ret), G, H) #eq 1
    wt = np.minimum(list(np.maximum(list(wt), min_weights)), max_weights)
    wt1d = wt.reshape([n_assets, 1])
```

```python
variance = np.matmul(np.matmul(wt1d.T, cov), wt1d)[0, 0]
if variance < v_t:
    ret = np.array(ret)
    chol = np.linalg.cholesky(cov).T
    Gq = np.vstack([np.zeros(n_assets), chol])
    hq = np.zeros(n_assets + 1)
    hq[0] = np.sqrt(v_t)
    wt = cvxopt_solve_socp(-ret, Gl=G, hl=H, Gq=Gq, hq=hq) #eq 2
    wt = np.minimum(list(np.maximum(list(wt), min_weights)), max_weights)
return wt
```

B Preprocessing

We encounter ambiguity in optimization problems due to various combinations of inputs representing the same problem. To address this, we provide three examples where we discuss the ambiguity and propose a standardized solution for processing inputs in an optimized manner prior to token projection.

When the i-th asset belongs to the j-th asset class and $x_i^{\text{MAX}} > \zeta_{c_j}$, the constraint x_i^{MAX} is overridden by ζ_{c_j}. This means that there is no combination of assets where the allocation of the i-th asset can be higher than ζ_{c_j}. To address this constraint, we clip x_i^{MAX} to ζ_{c_j} by using the formula: $x_i'^{\text{MAX}} = \min(\max(x_i^{\text{MAX}}, \zeta_{c_j}), 0)$ for all i-th assets belonging to the j-th class.

The remaining two cases are additional edge cases related to the previous condition. If only one asset is assigned to the j-th class, ζ_{c_j} and x_j^{MAX} should be equal because it is equivalent to having no class constraint for that class. Also, if a class constraint is set but no assets belong to that class, it is equivalent to setting $\zeta_{c_j} = 0$. By processing the optimization inputs in this manner, we ensure that any ambiguity on the class constraints are standardized, allowing for equivalent linear projections into token before the transformer encoder part of the network.

C Experimental Section

C.1 Dataset

Dataset name	Size	Description
$\mathcal{D}_{\text{train}}$	1.2B samples	Sampled at random over the domain in table. 1
$\mathcal{D}_{\text{test}}$	990K samples	Sampled at random over the domain in table. 1
$\mathcal{D}_{\text{ood}}$	990K samples	Sampled following the indication of sec. 4.3

Table 5: Description of the dataset used

The size and description of the dataset we used are presented in table. 5. We used an asymmetric weighting scheme to generate all datasets, favoring more complex optimizations (see Table 6). As the number of assets increases, the number of unstable regions also increases, where allocation can significantly change. To ensure training and evaluation encompass these unstable regions, we generated a higher proportion of optimization inputs with more assets.

C.2 Accuracy with half-precision floating-point format

The results obtained using single-precision floating-point (FP32) and the model quantized to half-precision floating-point (FP16) are on the same order of accuracy as shown in table 7 and fig. 10. The quantization process to FP16 maintains the necessary precision for the calculations, resulting in equivalent outcomes as the FP32 counterpart. While a degradation in the ability to rank assets[5] and respect the volatility and class constraint occurs, we observe that this does not impact the overall distributional properties and the downstream

[5]The ranking of the results has been computed with a tolerance of $1e-4$, where a slight deviations are permissible and don't hurt the accuracy. This was made such that negligible allocation made by NeuralEF which can be disregarded for practical purpose are neglected.

Asset case	Proportion (%)
2	2.517
3	2.551
4	2.559
5	2.603
6	6.834
7	6.879
8	10.346
9	10.377
10	13.826
11	13.850
12	27.658

Table 6: Proportion of asset in each datasets.

application that would benefit from it. As such, there is no discrepancy in the results between the two representations, demonstrating the viability of using the lower-precision FP16 for computational efficiency.

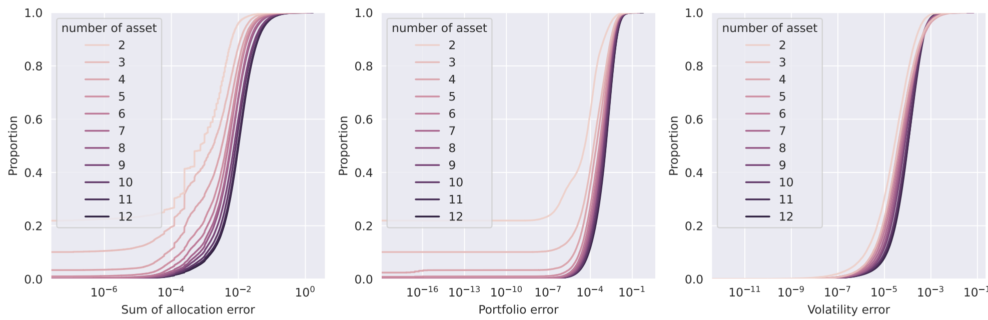

Figure 10: Cumulative distributions of the sum of absolute allocation error of allocations and portfolio returns per assets for the FP16 quantized NeuralEF.

Asset case	**Portfolio weights MSE**	**Portfolio weights MAE**	**95 quantile**	**99.865 quantile**	**99.997 quantile**	**Ranking precision**
2	9.54e-07	9.77e-04	1.25e-02	3.37e-02	1.09e-01	93.012 %
3	7.95e-08	1.63e-04	1.51e-02	3.78e-02	1.06e-01	98.394 %
4	6.85e-07	5.34e-04	1.65e-02	4.59e-02	1.39e-01	97.231 %
5	2.52e-06	1.25e-03	1.48e-02	4.08e-02	1.77e-01	94.064 %
6	1.94e-05	2.00e-03	1.51e-02	4.28e-02	1.68e-01	89.798 %
7	2.84e-06	1.19e-03	1.67e-02	4.50e-02	1.95e-01	85.224 %
8	1.06e-05	2.40e-03	1.61e-02	4.49e-02	1.57e-01	81.598 %
9	7.72e-06	1.65e-03	1.99e-02	5.20e-02	2.08e-01	77.217 %
10	1.16e-05	2.21e-03	2.00e-02	5.25e-02	1.71e-01	74.899 %
11	1.11e-05	1.72e-03	2.22e-02	5.85e-02	2.36e-01	71.646 %
12	3.48e-08	1.30e-04	1.98e-02	5.31e-02	2.11e-01	68.804 %
	Portfolio return MSE	**Portfolio return MAE**	**95 quantile**	**99.865 quantile**	**99.997 quantile**	ζ_{MAX} **precision**
2	2.69e-06	1.64e-03	7.87e-03	2.26e-02	6.98e-02	97.555 %
3	9.48e-06	3.08e-03	1.24e-02	3.19e-02	1.12e-01	90.949 %
4	3.58e-06	1.89e-03	1.48e-02	4.05e-02	1.77e-01	87.576 %
5	8.79e-06	2.96e-03	1.49e-02	3.57e-02	9.32e-02	86.583 %
6	1.60e-13	3.99e-07	1.52e-02	3.58e-02	1.21e-01	85.592 %
7	2.15e-09	4.63e-05	1.68e-02	3.94e-02	1.51e-01	82.323 %
8	5.74e-06	2.40e-03	1.72e-02	4.25e-02	1.49e-01	84.808 %
9	2.11e-06	1.45e-03	2.13e-02	5.19e-02	1.74e-01	83.589 %
10	2.38e-09	4.88e-05	2.31e-02	5.36e-02	1.55e-01	83.707 %
11	1.31e-07	3.62e-04	2.49e-02	5.75e-02	1.75e-01	83.190 %
12	1.28e-06	1.13e-03	2.49e-02	5.75e-02	1.84e-01	83.197 %
	Volatility return MSE	**Volatility return MAE**	**95 quantile**	**99.865 quantile**	**99.997 quantile**	$\mathcal{V}_{\text{target}}$ **precision**
2	2.69e-06	1.64e-03	7.87e-03	2.26e-02	6.98e-02	87.649 %
3	9.48e-06	3.08e-03	1.24e-02	3.19e-02	1.12e-01	81.729 %
4	3.58e-06	1.89e-03	1.48e-02	4.05e-02	1.77e-01	83.229 %
5	8.79e-06	2.96e-03	1.49e-02	3.57e-02	9.32e-02	79.704 %
6	1.60e-13	3.99e-07	1.52e-02	3.58e-02	1.21e-01	80.056 %
7	2.15e-09	4.63e-05	1.68e-02	3.94e-02	1.51e-01	80.270 %
8	5.74e-06	2.40e-03	1.72e-02	4.25e-02	1.49e-01	76.472 %
9	2.11e-06	1.45e-03	2.13e-02	5.19e-02	1.74e-01	77.205 %
10	2.38e-09	4.88e-05	2.31e-02	5.36e-02	1.55e-01	79.268 %
11	1.31e-07	3.62e-04	2.49e-02	5.75e-02	1.75e-01	81.682 %
12	1.28e-06	1.13e-03	2.49e-02	5.75e-02	1.84e-01	79.517 %

Table 7: Accuracy of portfolio weights, implied return and resulting volatility for the FP16 quantized NeuralEF.