Riskfuel

Learning the Efficient Frontier

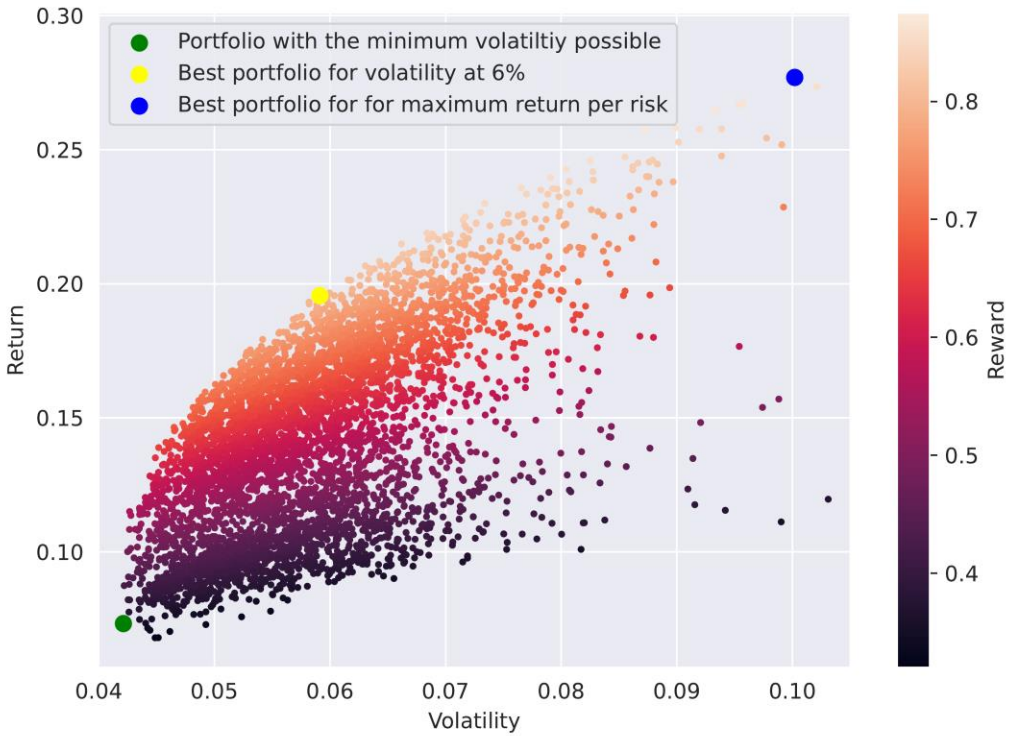
Philippe Chatigny, Ivan Sergienko, Ryan Ferguson, Jordan Weir, Maxime Bergeron

https://riskfuel.com

The Efficient Frontier

Problem: The efficient frontier (EF) is a resource allocation problem where one has to find an optimal portfolio maximizing a reward at a given level of risk.

How it is Solved: Traditionally, by solving a convex optimization problem (optimal solution)



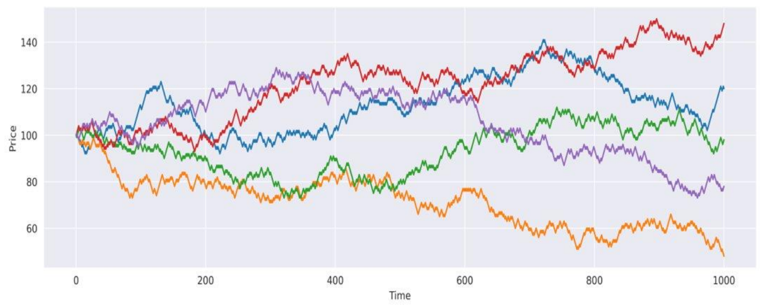
A convex optimization problem

The convex optimization problems is easy to solve:

1. Compute the portfolio with minimal risk (*Eq. (1)*)
2. if we have budget for risk, increase till we can't *(Eq. (2))*

In Practice: The parameters of the convex optimization are stochastic.

Challenge: Monte Carlo simulation is often used to measure the expected reward for a given scenario. **<u>the cost of running the optimization become the principal bottleneck</u>**



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

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

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

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

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

$$\boldsymbol{\psi} := \text{minimize } \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} \text{ subject to } \boldsymbol{a}_i^\top \leq \boldsymbol{b}_i \ \ \forall i \in 1, \cdots, w, \tag{1}$$

$$\boldsymbol{\phi} := \text{minimize } -\boldsymbol{R}^\top \boldsymbol{x} \text{ subject to } \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} \leq \mathcal{V}_{\text{target}} \text{ and } \boldsymbol{a}_i^\top \leq \boldsymbol{b}_i \forall i \in 1, \cdots, w. \tag{2}$$

$$\boldsymbol{Z}_{\text{output}} = \text{EF}(\boldsymbol{Z}_{\text{input}}) = \boldsymbol{\psi} \text{ if } \mathcal{V}_{\min} > \mathcal{V}_{\text{target}} \text{ else } \boldsymbol{\phi}. \tag{3}$$

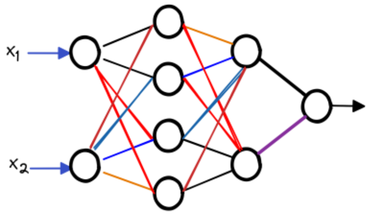
Learning the Efficient Frontier

Need for speed 🏁:

1. Rewrite the optimization to work on GPU: <u>mostly impractical</u>

2. Learn the optimization directly:
   Robustness 💪,
   Accuracy 🎯,
   Speed 🏃 &
   Flexibility 🤸

Can

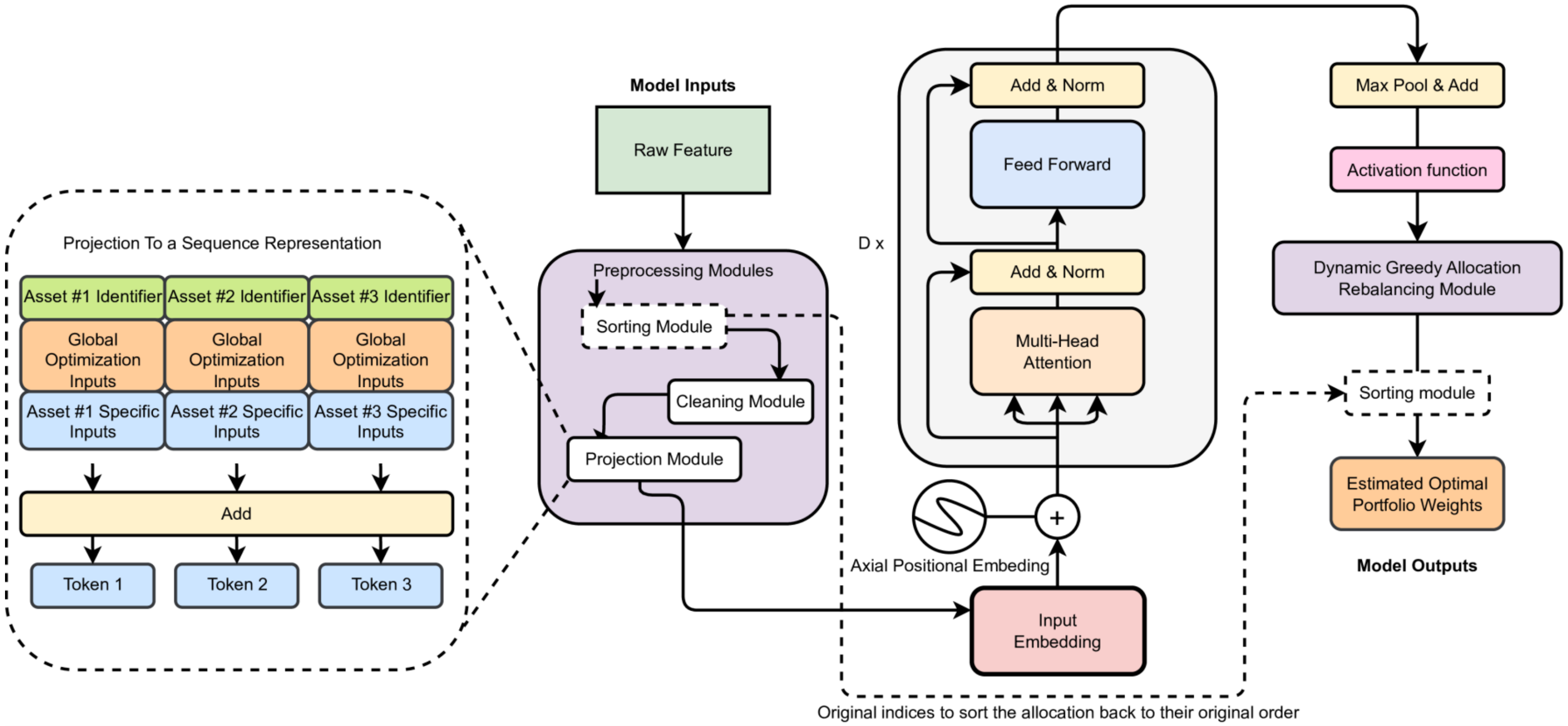
Z_{input}  Z_{output}

Be faster than

Z_{input} **CVXOPT [LAPACK, BLAS]** Z_{output}

NeuralEF: Reformulating the EF problem…

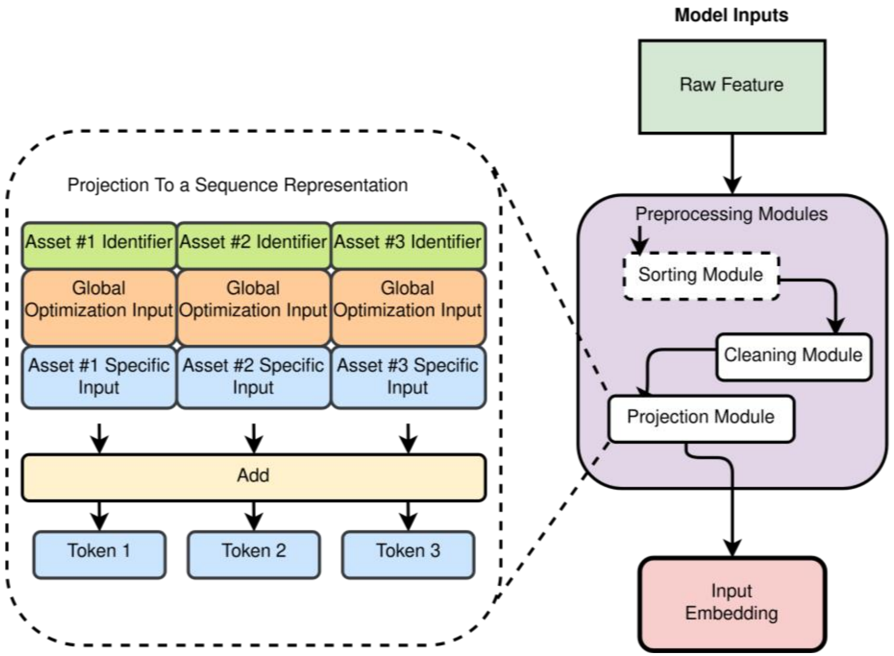
R



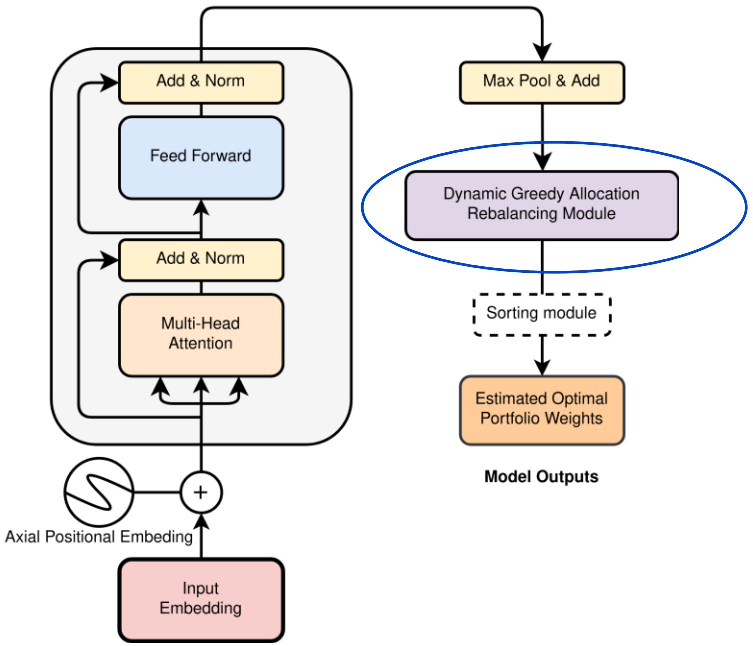
$$\theta_{\text{NeuralEF}} = \underset{\theta^*_{\text{NeuralEF}}}{\operatorname{argmin}} \frac{1}{N} \sum_{i=0}^{N} \mathcal{L}(\text{NeuralEF}(\boldsymbol{Z}_{\text{input},i}; \theta_{\text{NeuralEF}}), \text{EF}(\boldsymbol{Z}_{\text{input},i})). \tag{11}$$

Three main ideas:

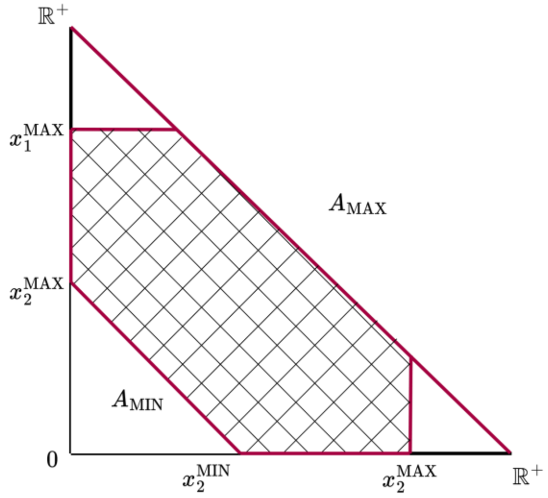
1) From optimization inputs to Seq2Seq



2) Solving the Seq2Seq problem with attention



3) Adjust greedily



Flexibility

Robustness

Evaluated At scale

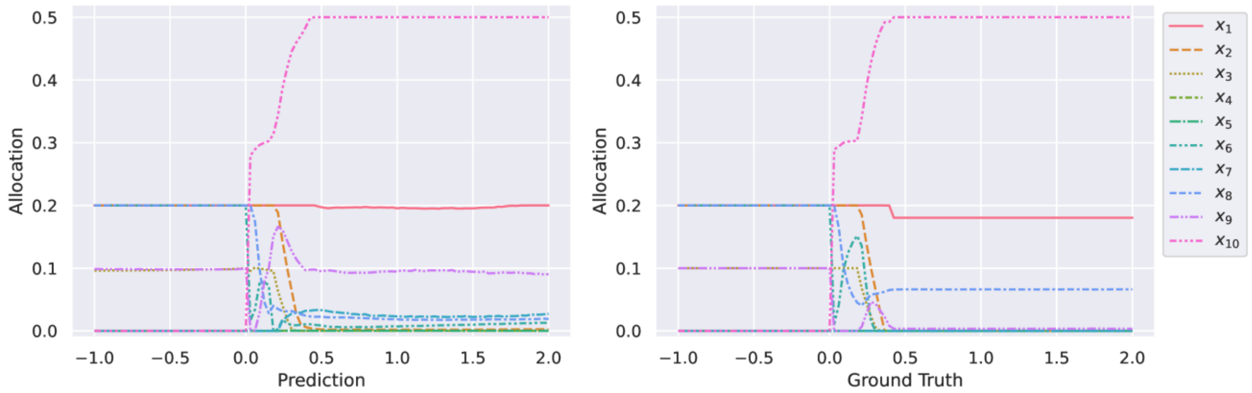
Trained on a large synthetic datasets

Feature	Range	Feature	Range
($\mathcal{V}_{\text{target}}$) volatility target	[0.05, 0.15]	($\boldsymbol{V}$) volatility	[0, 2]
($\boldsymbol{P}$) Correlation matrix	[-1, 1]	($\boldsymbol{R}$) returns	[-1,2]
(ζ_{MAX}) maximum class allocation	[0.2, 1.0]	(wt_{MAX}) maximum asset allocations	[0.01, 1.0]
(α_{MIN}) Allocation lower bound	[0.6, 1.0]	(α_{MAX}) Allocation upper bound	1.0
(n) Number of asset sampled	[2,12]	(m) Possible class	[0,1,2]

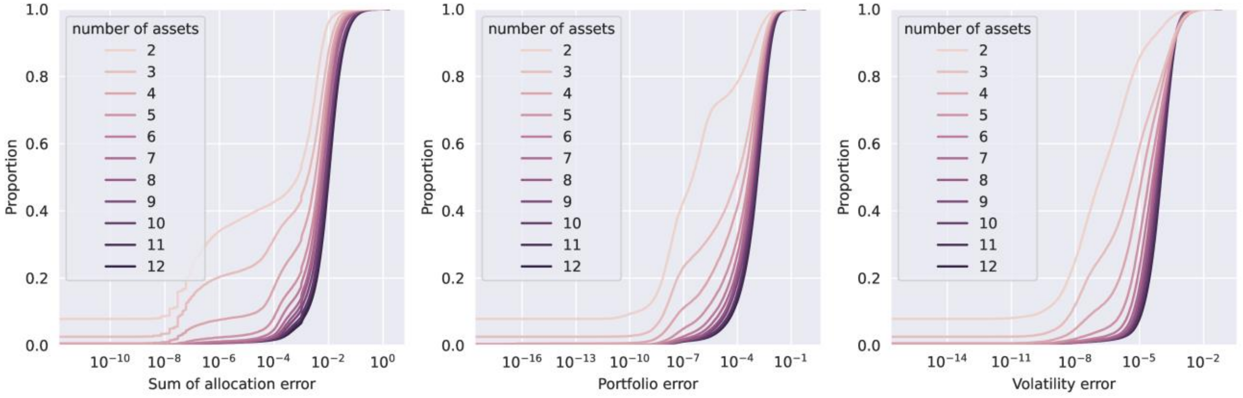
Table 1: Input Domain of optimization input used for training.

Robustness: Example with a "worst-case" prediction

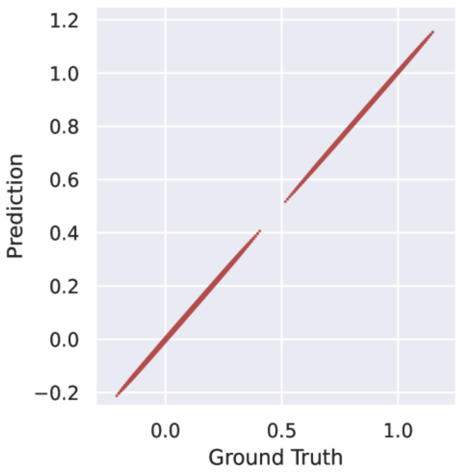
Allocation



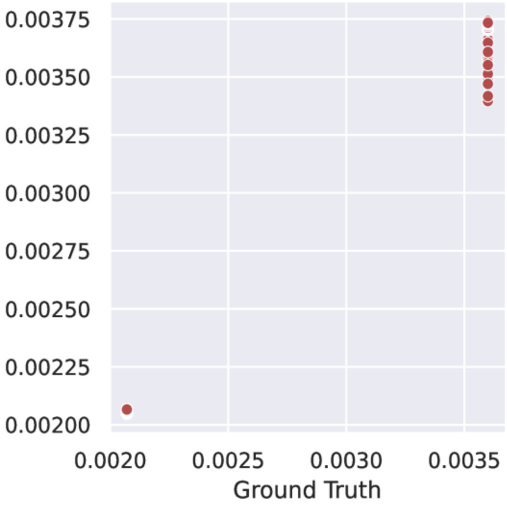
In-domain Interpolation Accuracy



Portfolio Return



Portfolio Volatility

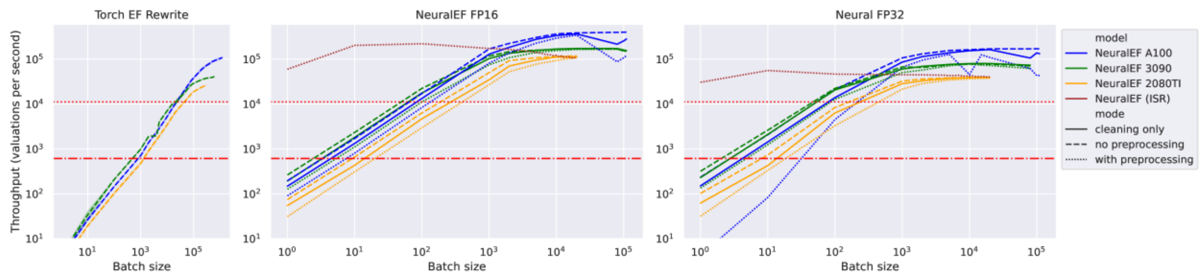


Accuracy

Throughput Evaluation and Carbon Footprint

R

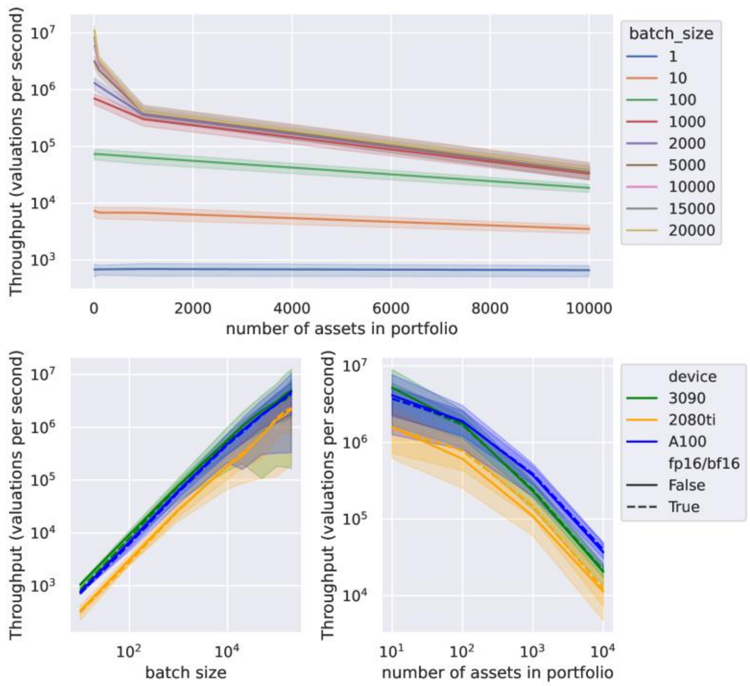
Throughput at scale



A100:	Throughput (eval./sec.)	2080TI:	Throughput (eval./sec.)
Pytorch-EF	111479.39	Pytorch-EF	26452.06
NeuralEF (fp32)	128950.46	NeuralEF (fp32)	37821.05
NeuralEF (fp16)	343760.77	NeuralEF (fp16)	107259.29
NeuralEF (fp16) clean-only	366450.96	NeuralEF (fp16) clean-only	114160.65
NeuralEF (fp16) no preprocessing	401641.81	NeuralEF (fp16) no preprocessing	118711.91
Intel Xeon Platinum 8480+ (ISR)		**3090:**	**Throughput (eval./sec.)**
NeuralEF (fp32)	56050.39	Pytorch-EF	41035.23
NeuralEF (bf16 + AMX)	221787.48	NeuralEF (fp32)	77859.95
AMD 5950X (reference)		NeuralEF (fp16)	167594.15
singe-thread	559.15	NeuralEF (fp16) clean-only	173388.66
Concurent processes (23)	10377.80	NeuralEF (fp16) no preprocessing	170650.62

Table 4: Maximum average throughput achieved. Note that two Intel Xeon Platinum 8480 CPUs were used simultaneously on a dual-socket machine to achieve the best throughput.

DGAR at scale



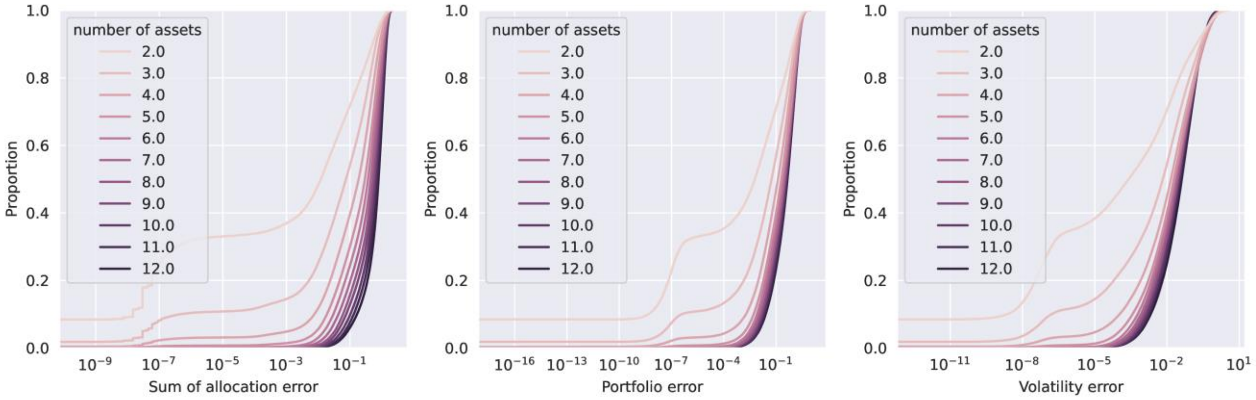
CO2 emission on inference: ≈46X time less than the base optimization on a desktop CPU 🌎

Speed 🏃

Limitations and Broader Challenges

R

1) **Out of domain Generalization**



2) Extend the model to other convex optimizations

3) Accelerate the execution of computer programs that rely on the expected value of convex optimizations problems.

Conclusion

We introduce NeuralEF, a model that can learn the EF convex optimization problem with heterogenous linear constraints robustly

We show how converting an optimization as SEQ2EQ is a viable solution to accelerate large-scale simulation while handling discontinuous behavior