Faster Than Fast: Pushing the Limits of Simulation with Generative Models

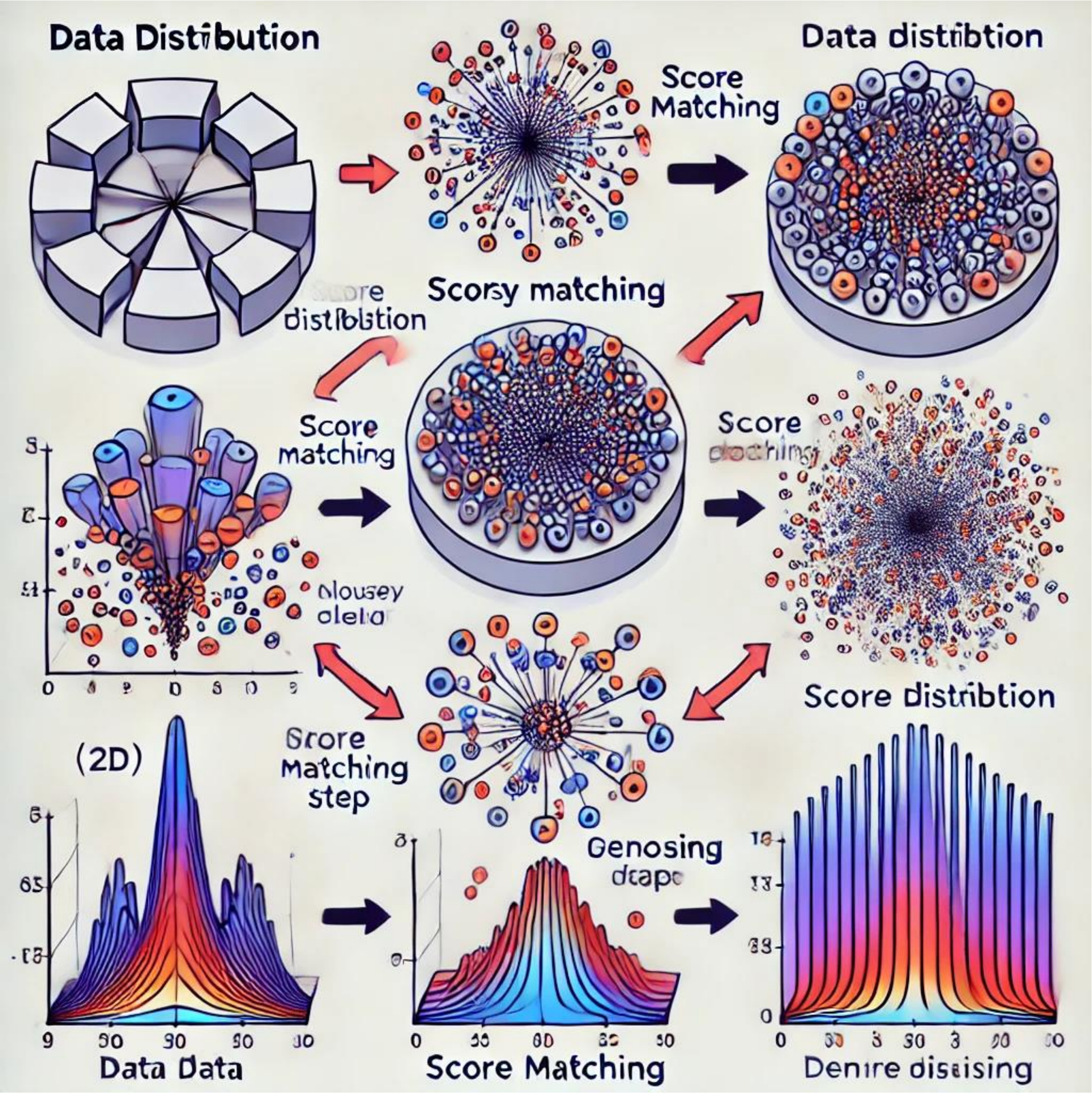
ML4Jets 2024, LPNHE, Paris

Cheng Jiang
Sitian Qian
Huilin Qu

THE UNIVERSITY of EDINBURGH

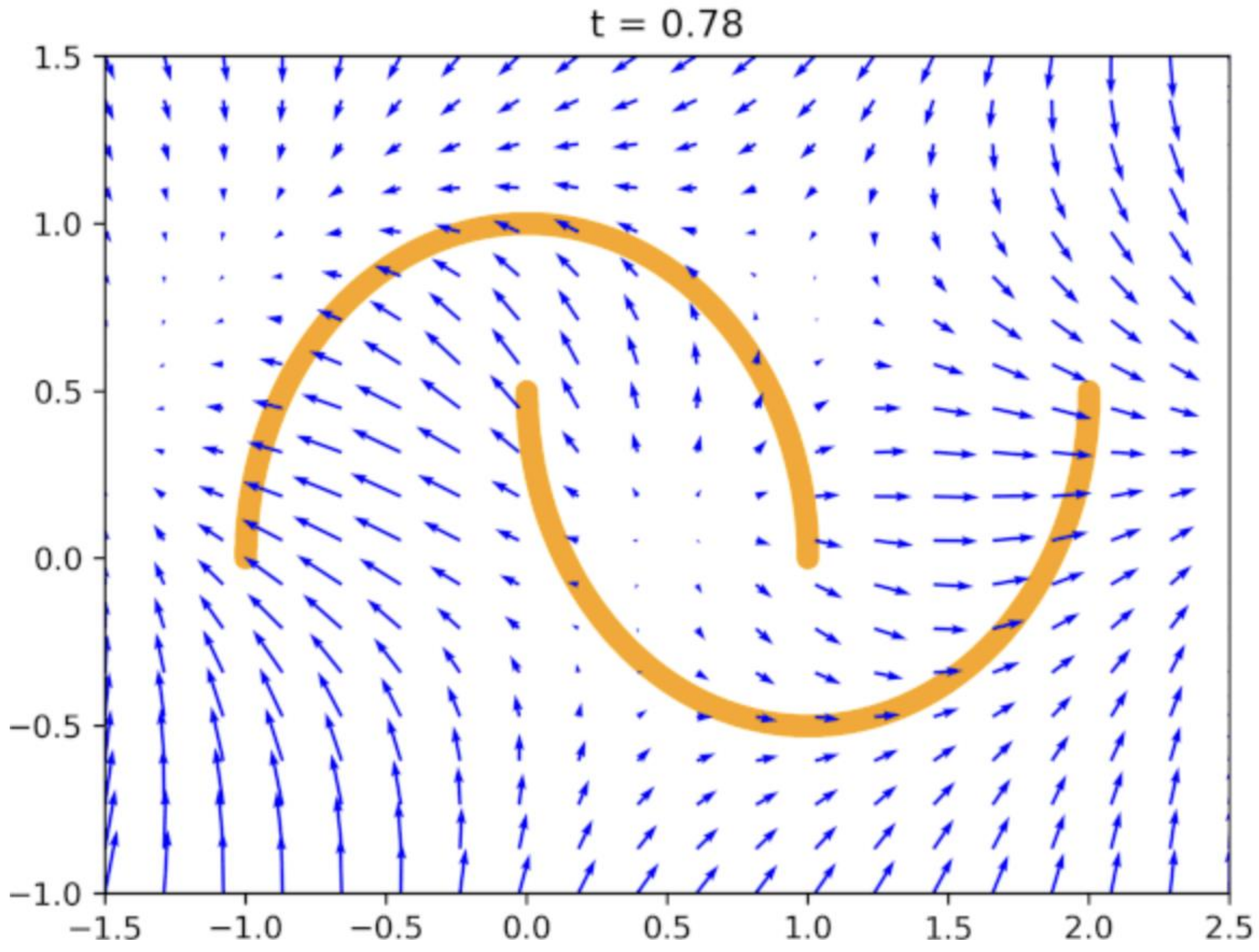
Disclaimer

- Generative models are of various kinds (DM, NF, GAN, AR models)
- We are focusing on DM and NF today in a unified context: Score Matching (SM) mechanism
- This talk will be summarizing two aspects:
- A faster backbone for SM models
- And acceleration of SM implementation



Generative

- The focus would be flow matching (or vector field matching diffusion) model.
- Mitigate the dimensionality curse than normal NF (integral → matching vector), currently one of the latest/most robust way to do generative study.



$$[||\nabla_{\mathbf{x}} \log p(\mathbf{x}) - \mathbf{s}_\theta(\mathbf{x})||_2^2]$$

$$\left[||v_\theta(\mathbf{x}_t, t) - v(\mathbf{x}_t, t)||^2\right]$$

Source: blog

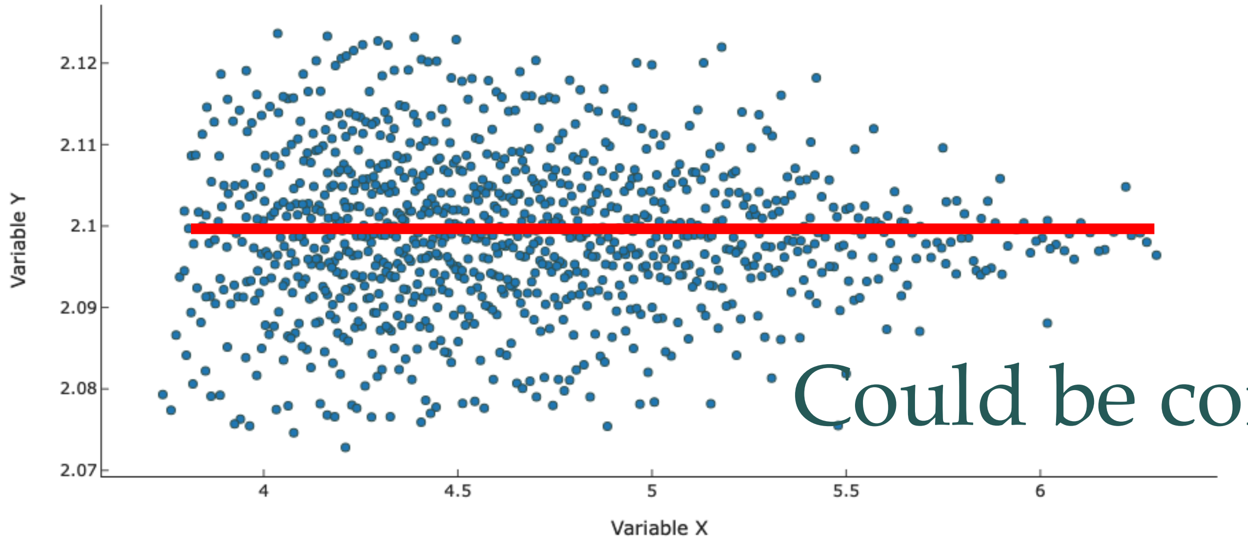
Tabular Data

- One of the common and oldest types of data seen in physics → event level quantities (MET, pT of 1st Jet etc.)

- Not like common pixelated/PC, hard to utilize the blooming CV techniques designed for image/common object detection/segmentations)

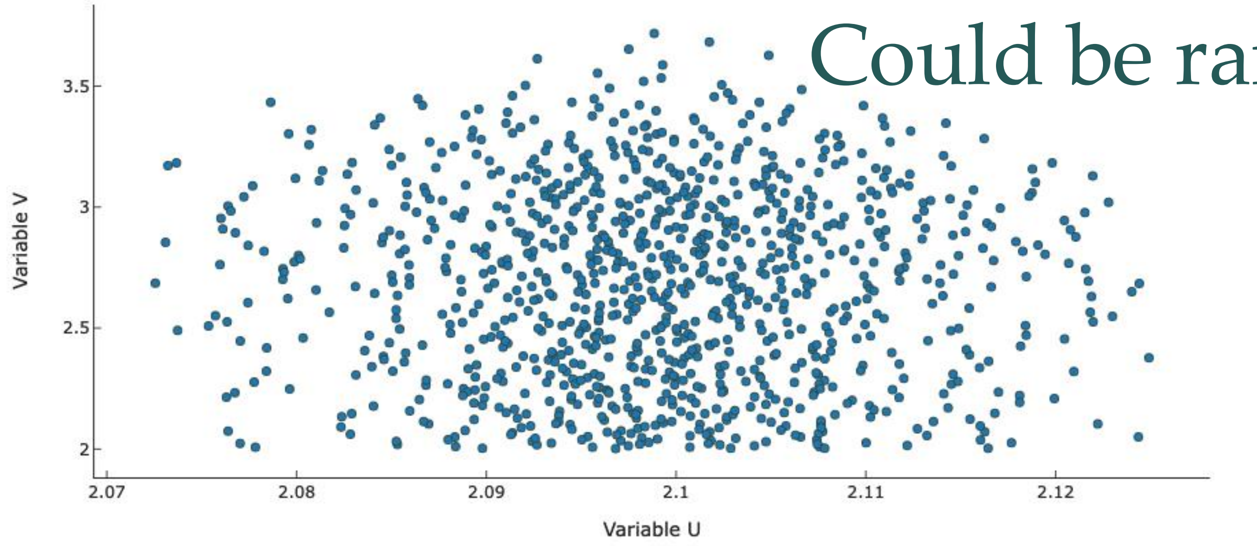
- Hard to capture the correlation among high-dimensional tabular data, if we just unroll/flatten the data (i.e. first cell readout…)

- And seems that BDT roles…? Reference study case 1 and 2.

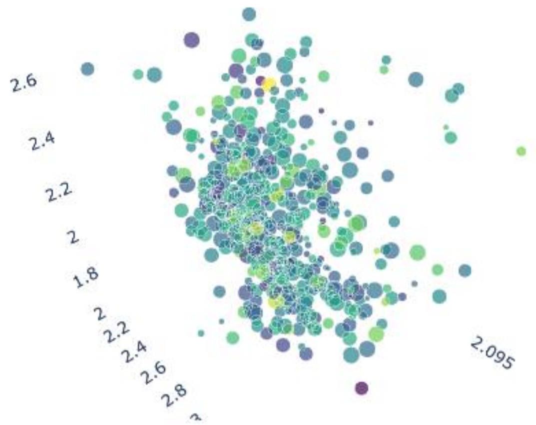


Could be correlated

Could be random



Very hard to be understood if high dim



BDT or NN ?

- Reference study case [1](#) and [2](#).
- Already intensive study and endless debate through all kinds of tabular data.
- Two classes are intrinsically different, would have different behaviours in many datasets. In their study, GBDT overall give slighter better results in classification/regression. But certainly not explore the full potential of NN (training tricks, feature engineering, tailored architectures..)
- The key is **GBDT inference time**

Table 15: Performance of algorithms across 98 datasets, where the algorithms include two modified versions of TabPFN. Columns show the algorithm family (GBDT, NN, baseline, or PFN), rank over all datasets, the average normalized F1 score loss (Mean F1), the std. dev. of normalized F1 across folds (Std. F1), and the train time in seconds per 1000 instances. Min/max/mean/median of these quantities are taken over all datasets.

Algorithm	Class	Rank min	max	mean	med.	Mean F1 mean	med.	Std. F1 mean	med.	Time /1000 inst. mean	med.
CatBoost	GBDT	1	19	6.43	5	0.87	0.93	0.29	0.22	21.0	2.08
TabPFN-3k	PFN	1	20	6.46	5.5	0.83	0.93	0.25	0.19	0.25	0.01
CatBoost-1k	GBDT	1	21	7.09	6	0.85	0.91	0.29	0.22	6.94	2.43
XGBoost	GBDT	1	19	7.78	6	0.82	0.89	0.32	0.22	0.83	0.37
TabPFN-1k	PFN	1	21	8.01	7	0.8	0.91	0.26	0.19	0.25	0.01
ResNet	NN	1	21	8.74	8.5	0.76	0.82	0.29	0.19	16.04	9.34
NODE	NN	1	21	9.23	9	0.75	0.81	0.25	0.19	140.71	117.04
SAINT	NN	1	21	9.26	9	0.73	0.85	0.3	0.23	171.14	144.37
FTTransformer	NN	1	19	9.32	9	0.76	0.82	0.3	0.19	27.94	18.4
RandomForest	base	1	21	9.46	9	0.77	0.83	0.31	0.21	0.36	0.25
LightGBM	GBDT	1	21	9.62	9	0.76	0.83	0.35	0.21	0.86	0.31

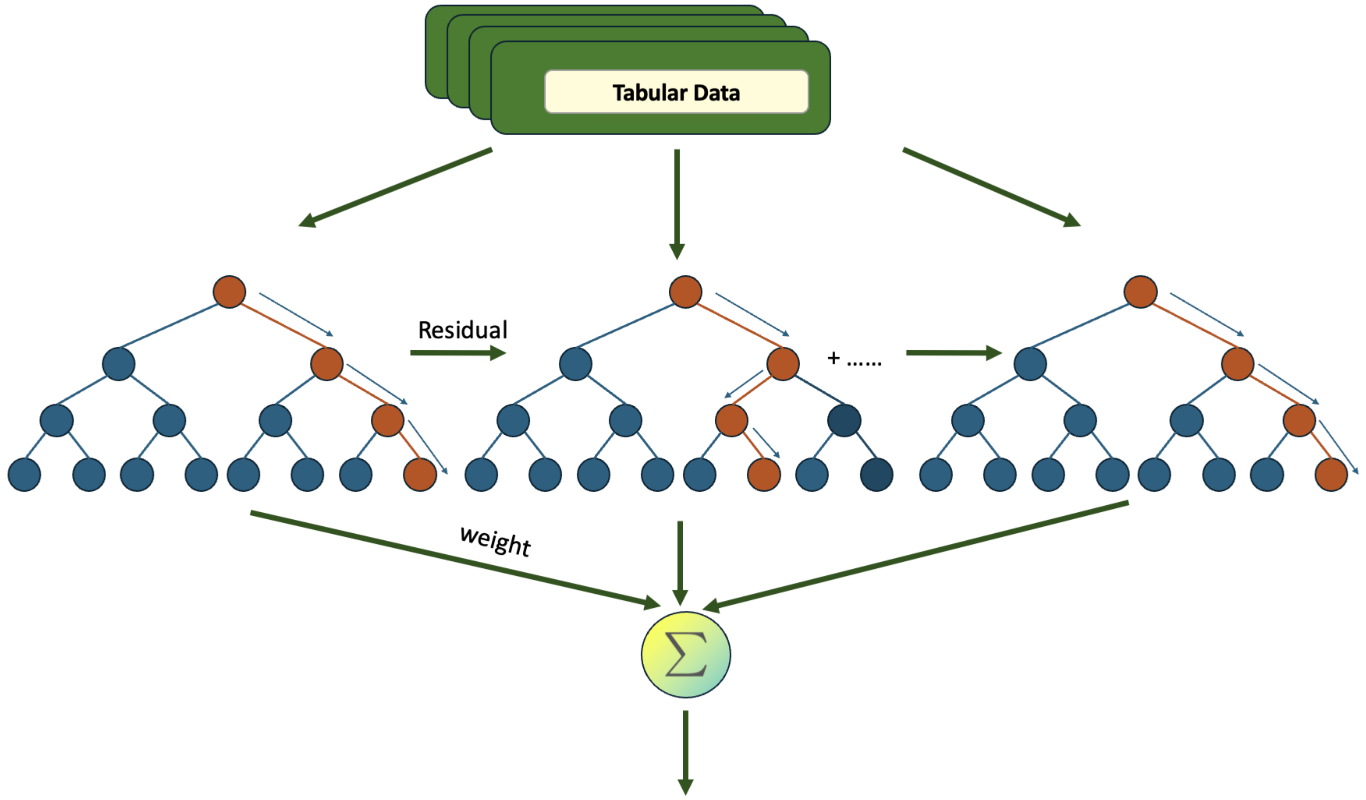


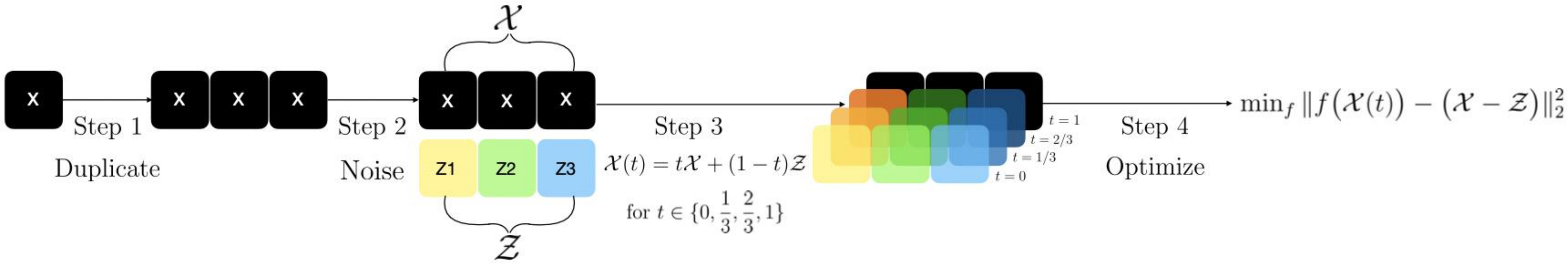
Table Source: [When Do Neural Nets Outperform Boosted Trees on Tabular Data?](#)

How to?

- Idea originally from [this paper].

- NN can use SGD with random sampling to minimize the expectation over mini-batch.

- Duplicate the partitioned input space, and precompute the in-coming and out-going vector field for each time step, then feed into a GBDT regression model.



- Is this fast? Yes! Is this efficient enough? No!
- High memory requirement (more duplicated datapoints and many GBDT model initialized).

Improvements

- Many works from 2404.18219 and 2408.16046 to reduce training/inference cost.

- Fewest tree size and duplicate times (use HPO K-fold or early stopping)
$\rightarrow$ faster inference/training

- Treat each time step independently, individual CPU training for small model
$\rightarrow$ faster training and suitable for High Throughputs Computing

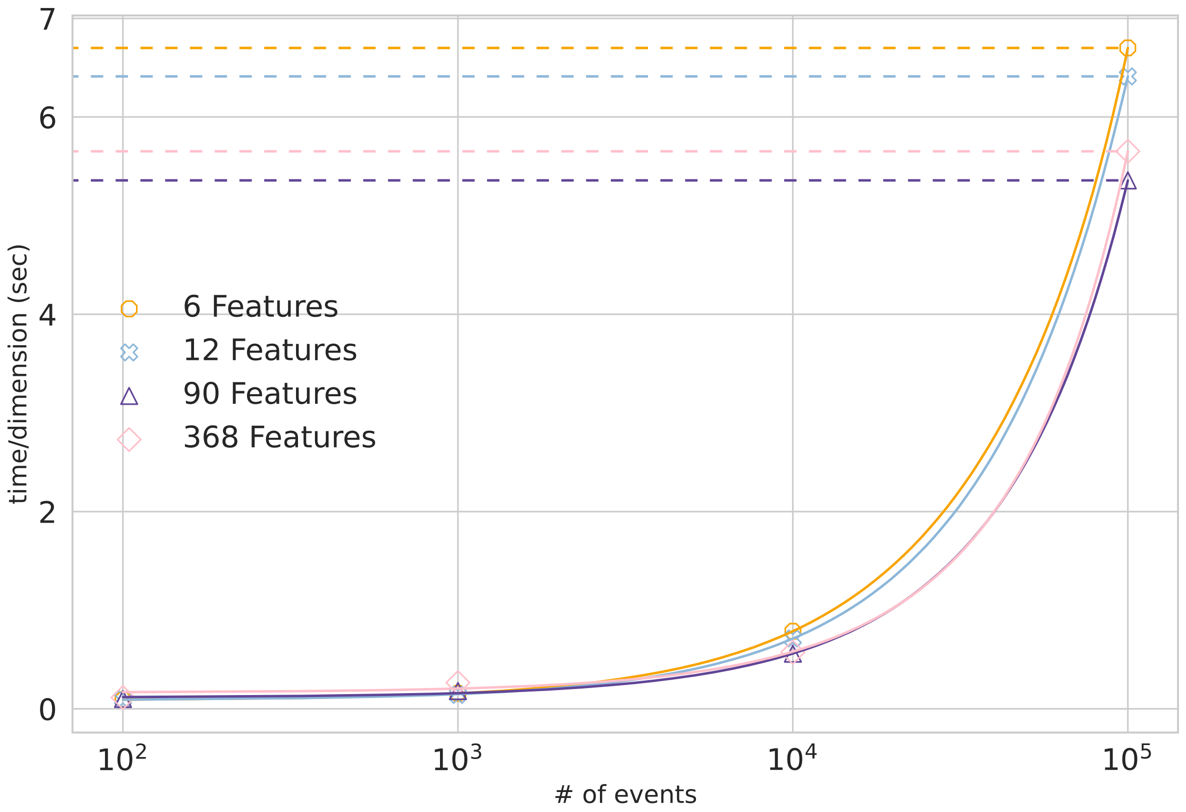
- Exploring precomputed vector field from Euler to the higher order ODE
$\rightarrow$ faster inference/training

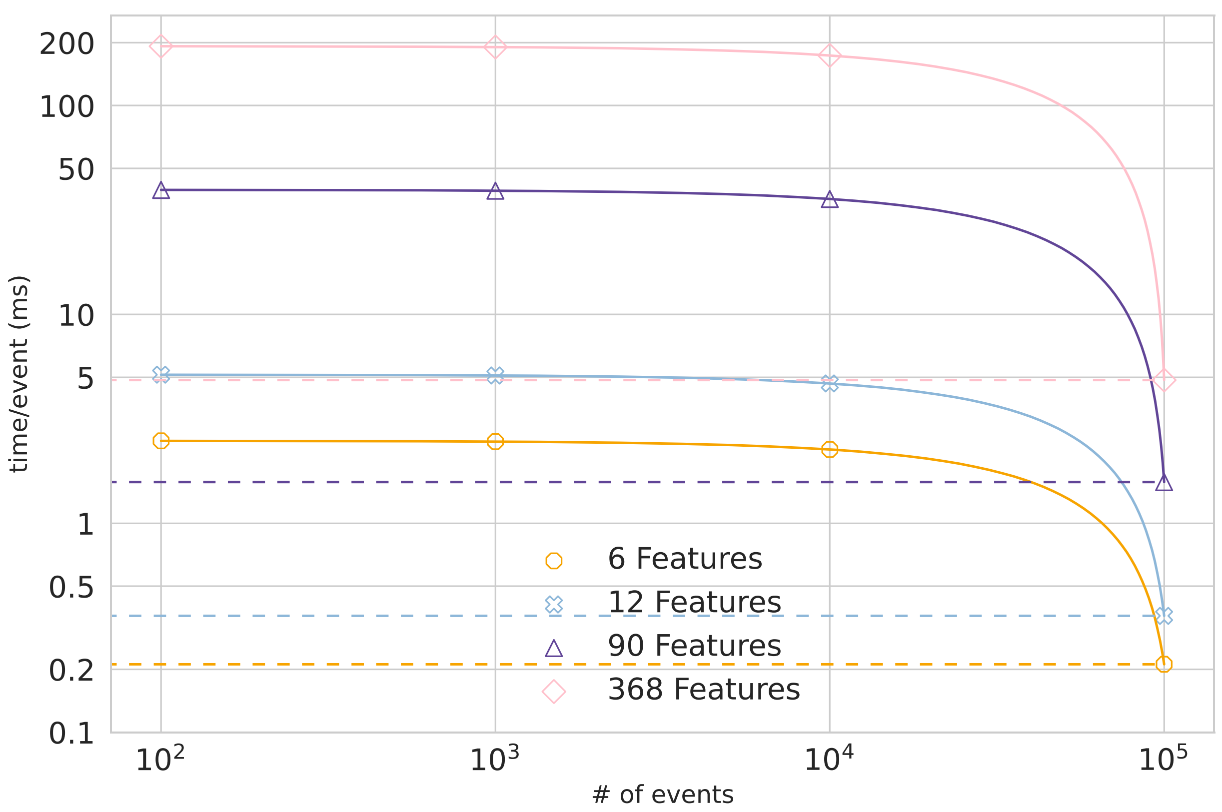
- Changing to multiple output per tree as training objectives
$\rightarrow$ faster inference especially when scaling up the dimension

Why?

- We like it because.

1. All the implementation could be minimally simple (only need some external package like *XGBoost* and *torchCFM,* and few lines of data transformations).
2. Fast to validate (train with # of steps CPUs, clearly inference advantages as light GBDT).
3. Just as the usual XGBoost, it might not be the strongest eventually, but certainly something convenient and quick to try.

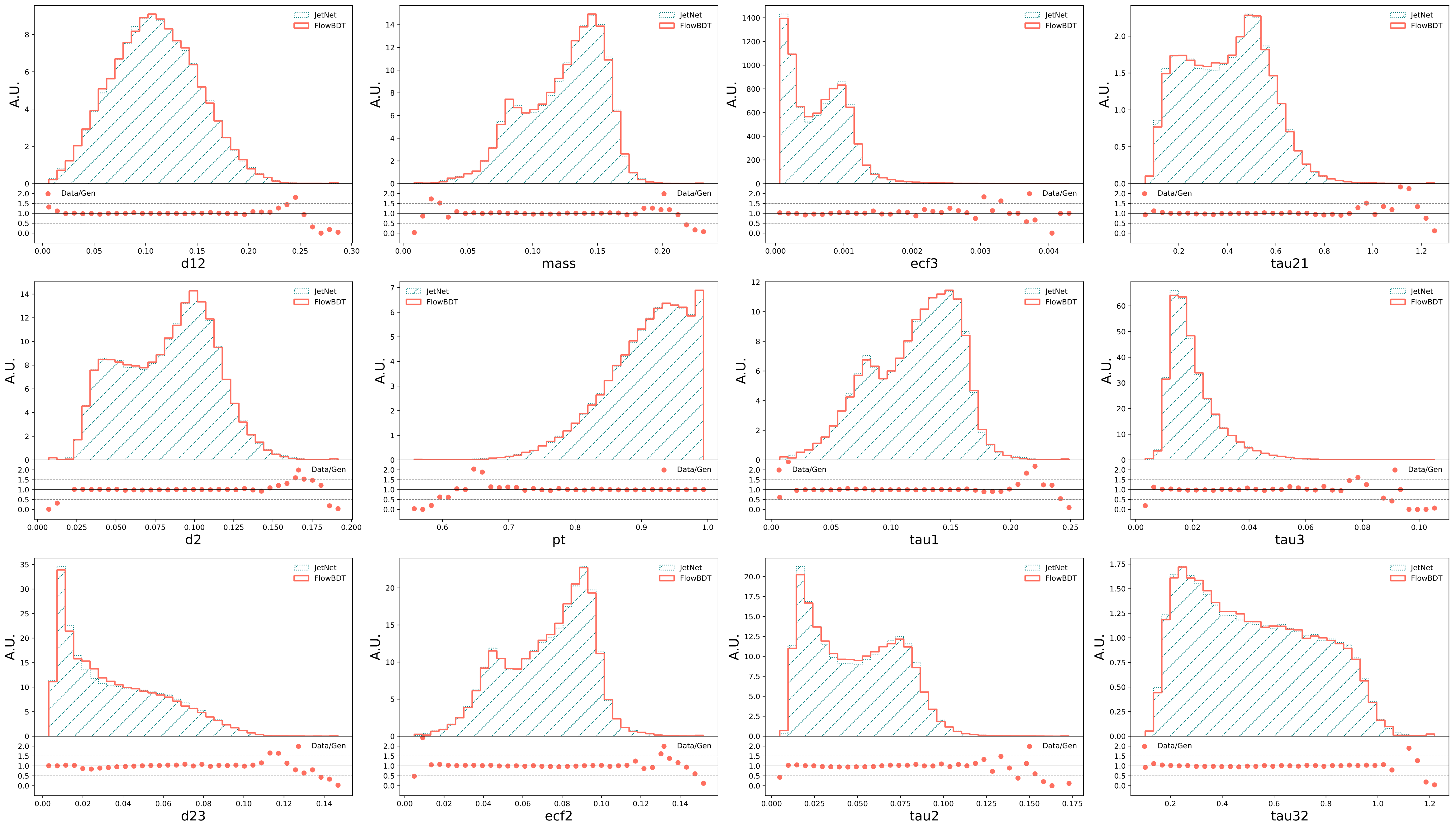


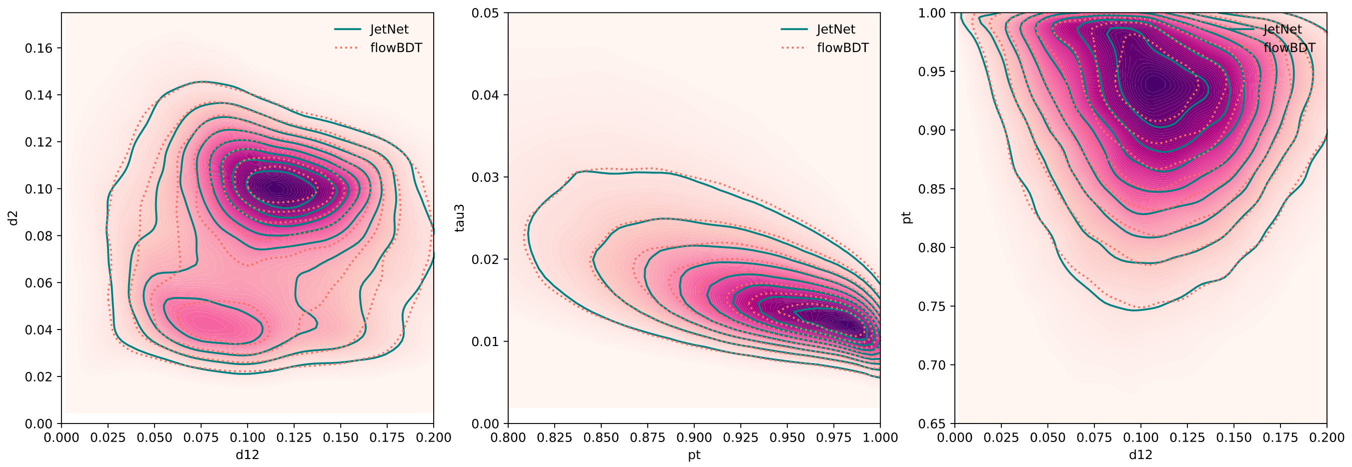


Sampling time per feature (left) and per event (right)

Use-case 1: End to end high level

- 12 dim high-level top jet observables simultaneous generation in JetNet
- 20-30 sampling steps Dormand-Prince and Midpoint solver, 0.8 ms/event

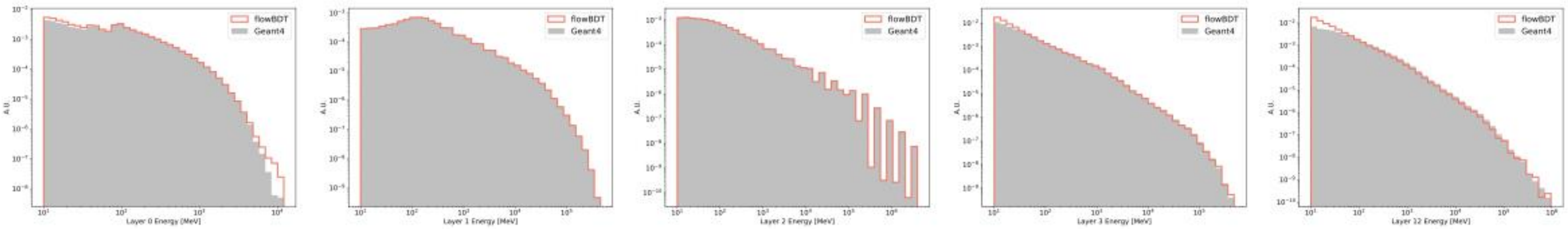


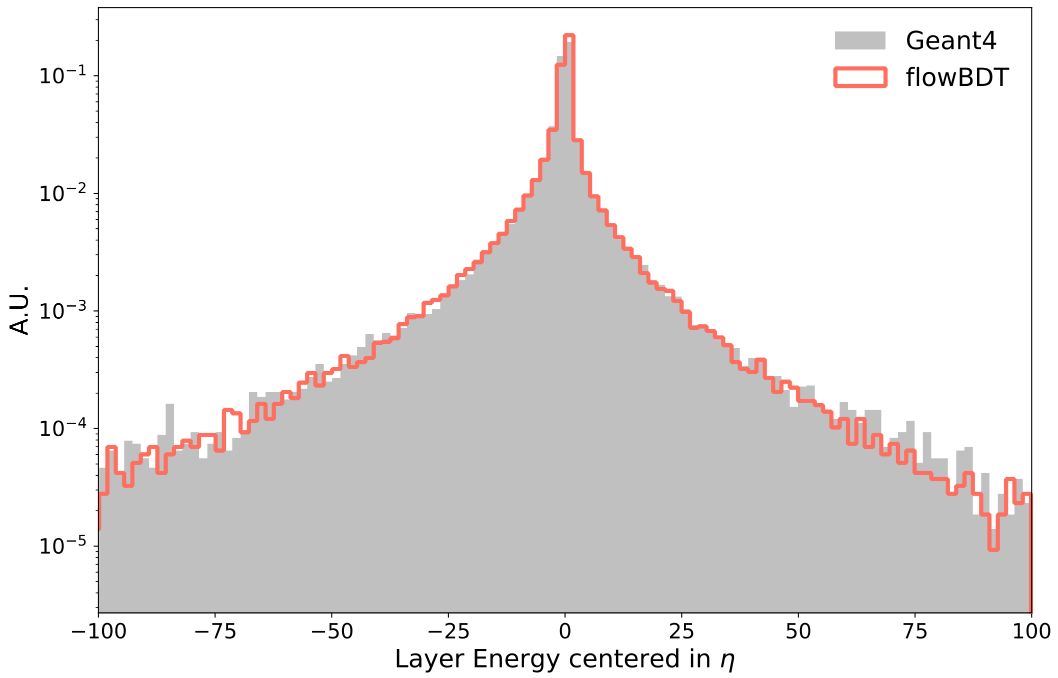


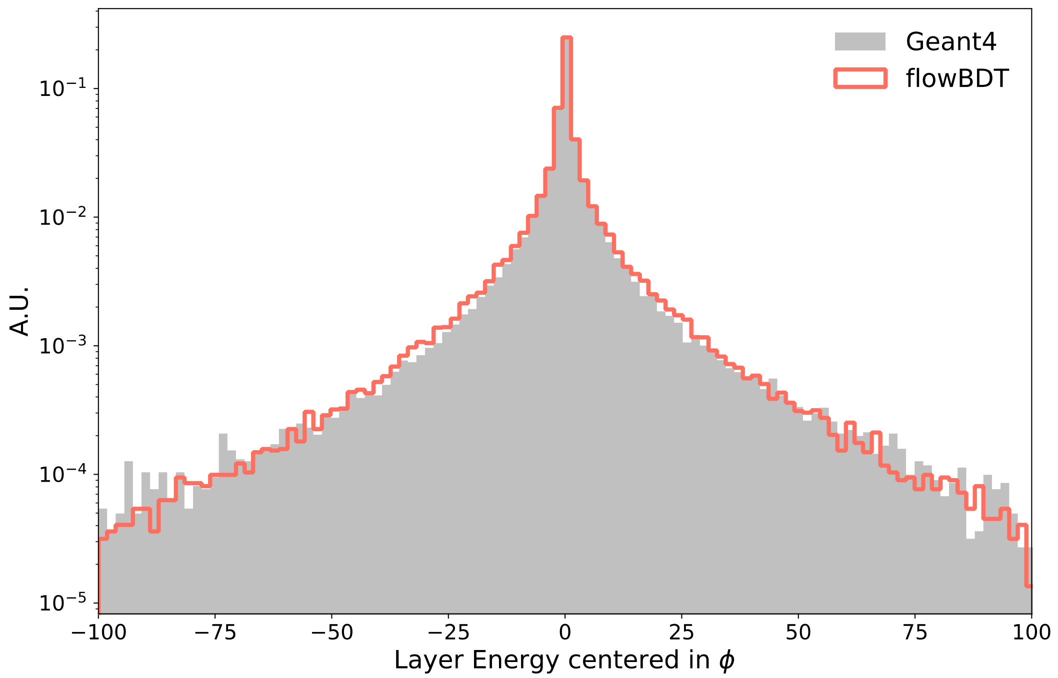
Metrics	Sep power×100	EMD×10
d_{12}	0.059	0.0030
d_2	0.14	0.0037
d_{23}	0.0057	0.095
τ_1	0.096	0.042
τ_2	0.15	0.0070
τ_3	0.065	0.009
$mass$	0.086	0.0030
p^{T}	0.051	0.0085
τ_{21}	0.045	0.020
ecf_2	0.084	0.0027
ecf_3	0.0001	0.15
τ_{32}	0.0097	0.031

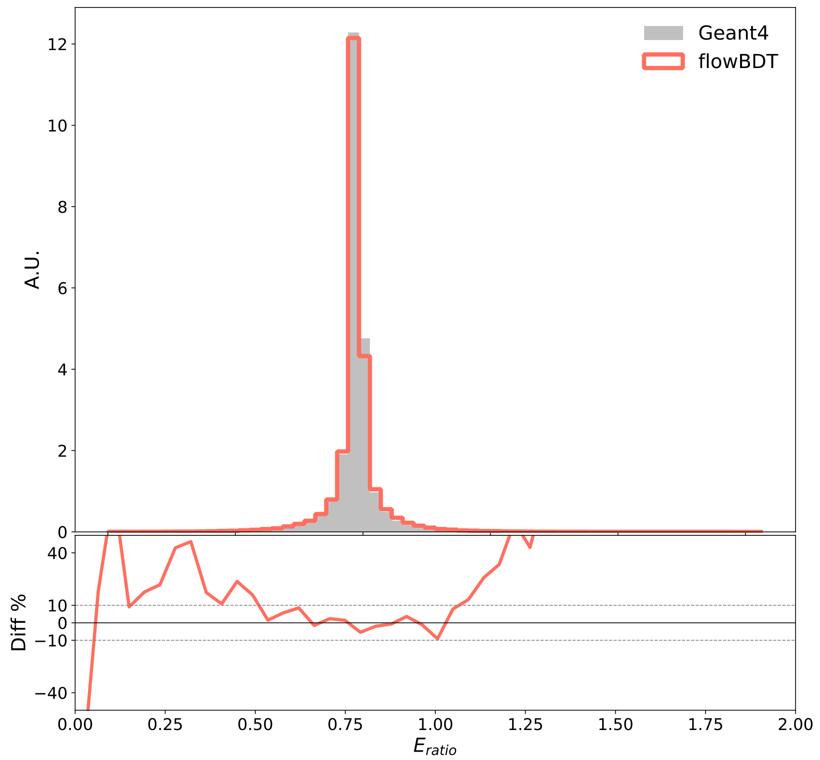
Use-case 2: Scaling up to calo

- GBT regression model by default need to be (N,1), changed to multiple output per trees (368 features for photon shower)
- Surprisingly good high-level features alignment from low-level simulations.



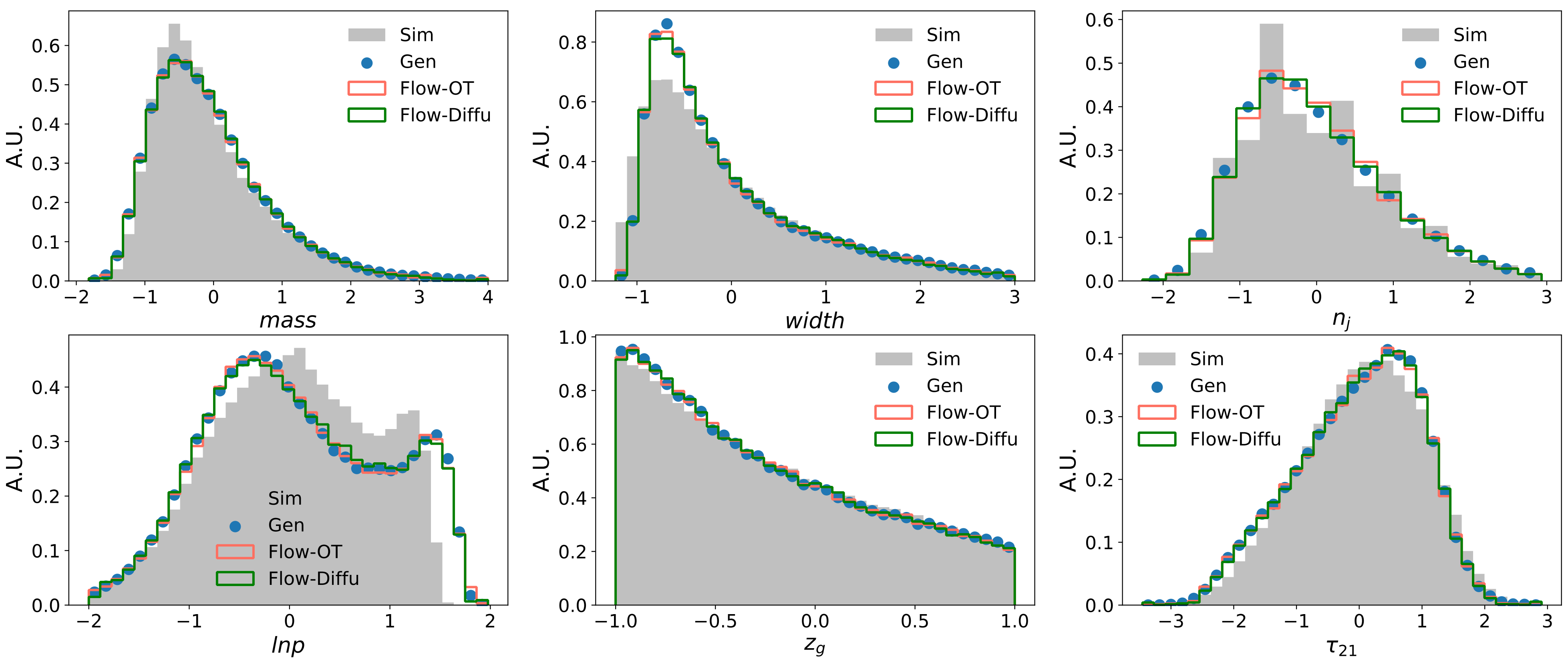


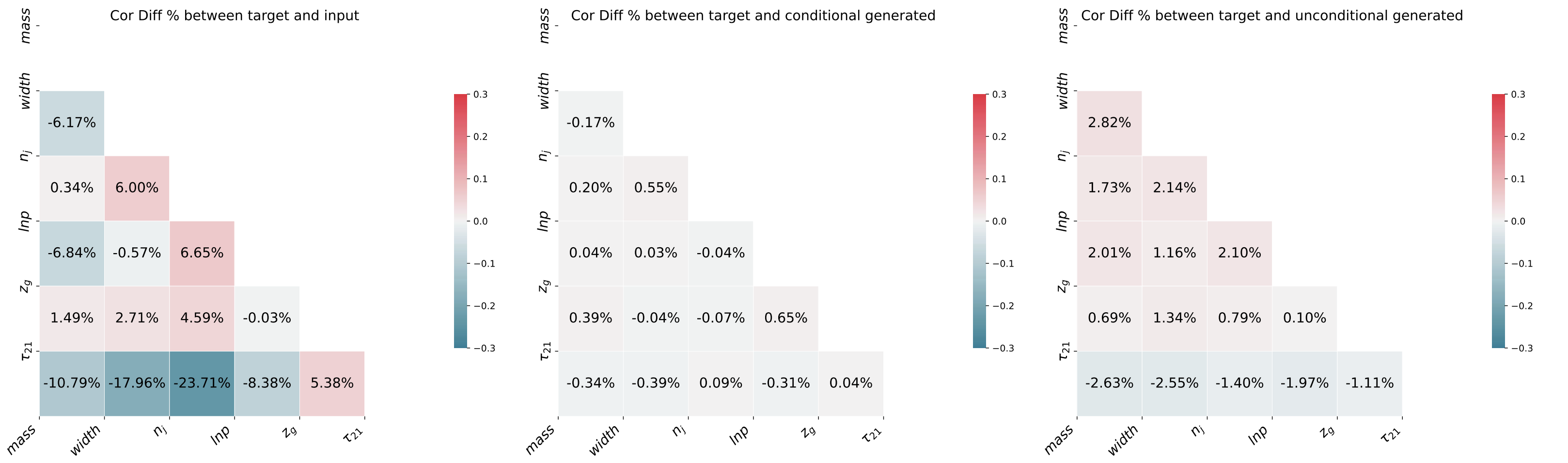




Use-case 3: Generative unfolding

- QCD jets from Z + jets events, standard OmniFold dataset.
- Compared results sampled from Gaussian and conditional (OT).
- Many times improvement in the correlation from conditional generation.





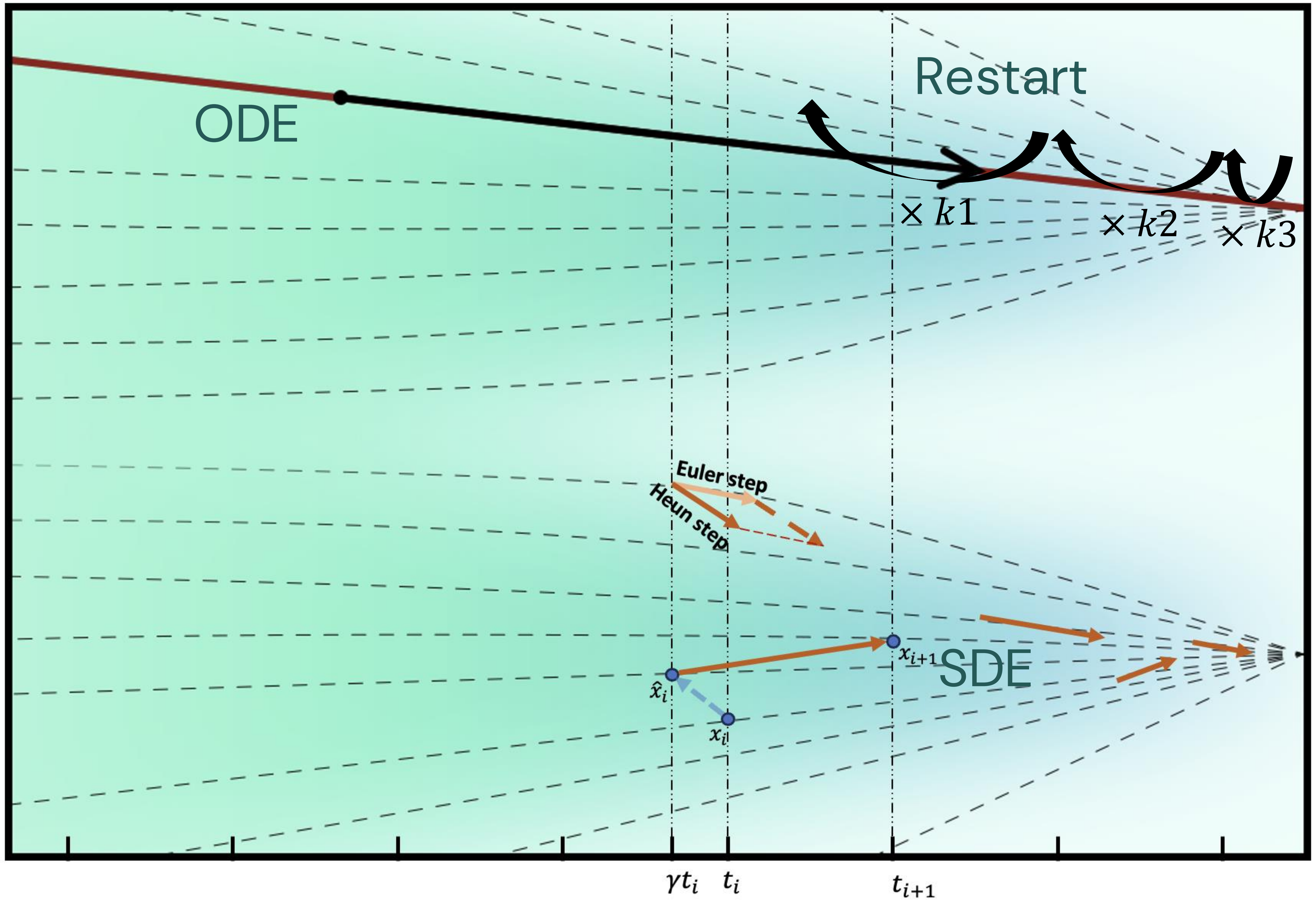
Observables(Sep power×100 / EMD×10)	Simulation	Conditional	Unconditional
Jet mass	0.496/0.369	**0.072/0.038**	0.091/0.084
Jet width	1.770/0.406	0.044/**0.031**	**0.034**/0.098
N constituents	1.129/0.405	0.10/0.28	**0.07/0.26**
$\ln_\rho$	3.684/0.916	**0.053/0.046**	0.10/0.082
z_g	0.047/0.173	**0.010/0.029**	0.013/0.030
τ_{21}	0.558/0.593	**0.095/0.054**	0.14/0.093

Faster Score Matching

- Choose your diffusion is based on the CaloDiffusion architecture (GLaM and CylindricalConv).

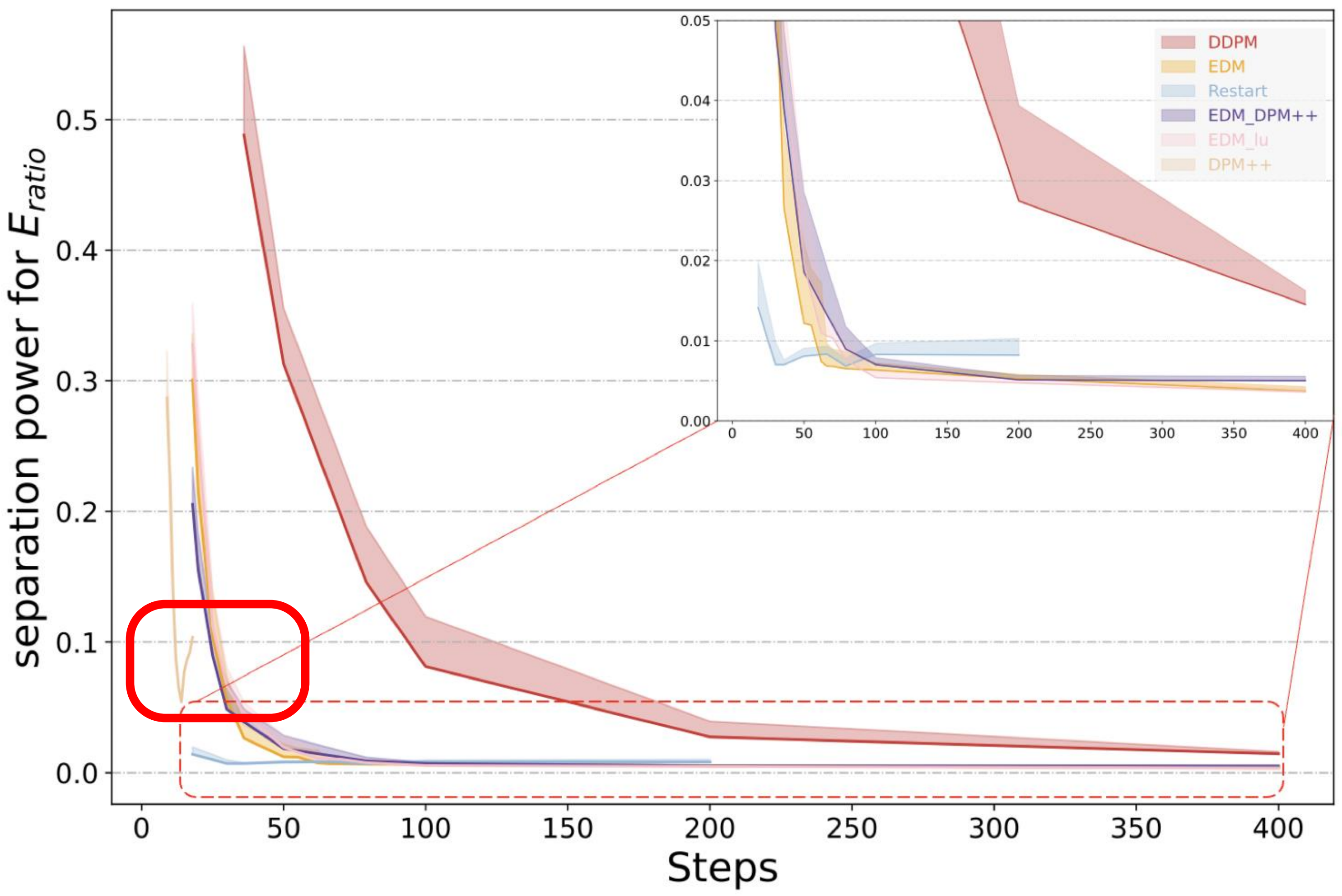
- Few training-based implementations: Min-SNR (signal-noise ratio) weight, post-hoc EMA..

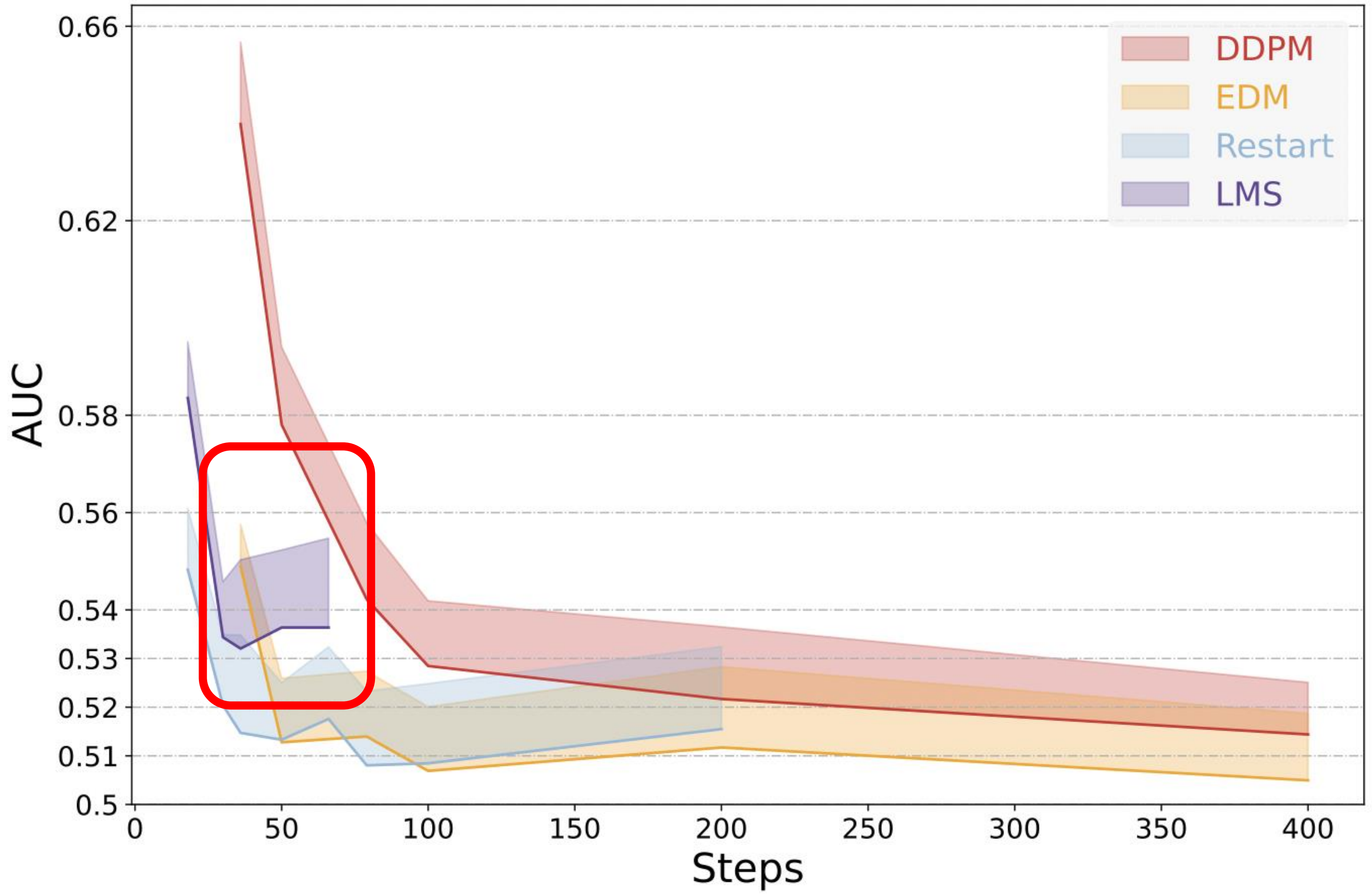
- Few training-free implementations: Validation for almost all main-stream samplers and schedulers



Previous Work

- EDM and Restart already proven as much better benchmark than DDPM, in both CV and CaloChallenge.
- Restart can be viewed as a sampler with special predefined schedulers.
- Problem happens for ODE at both small and large steps.
- Could we find a better initialization for schedulers or even time-step aware schedulers?

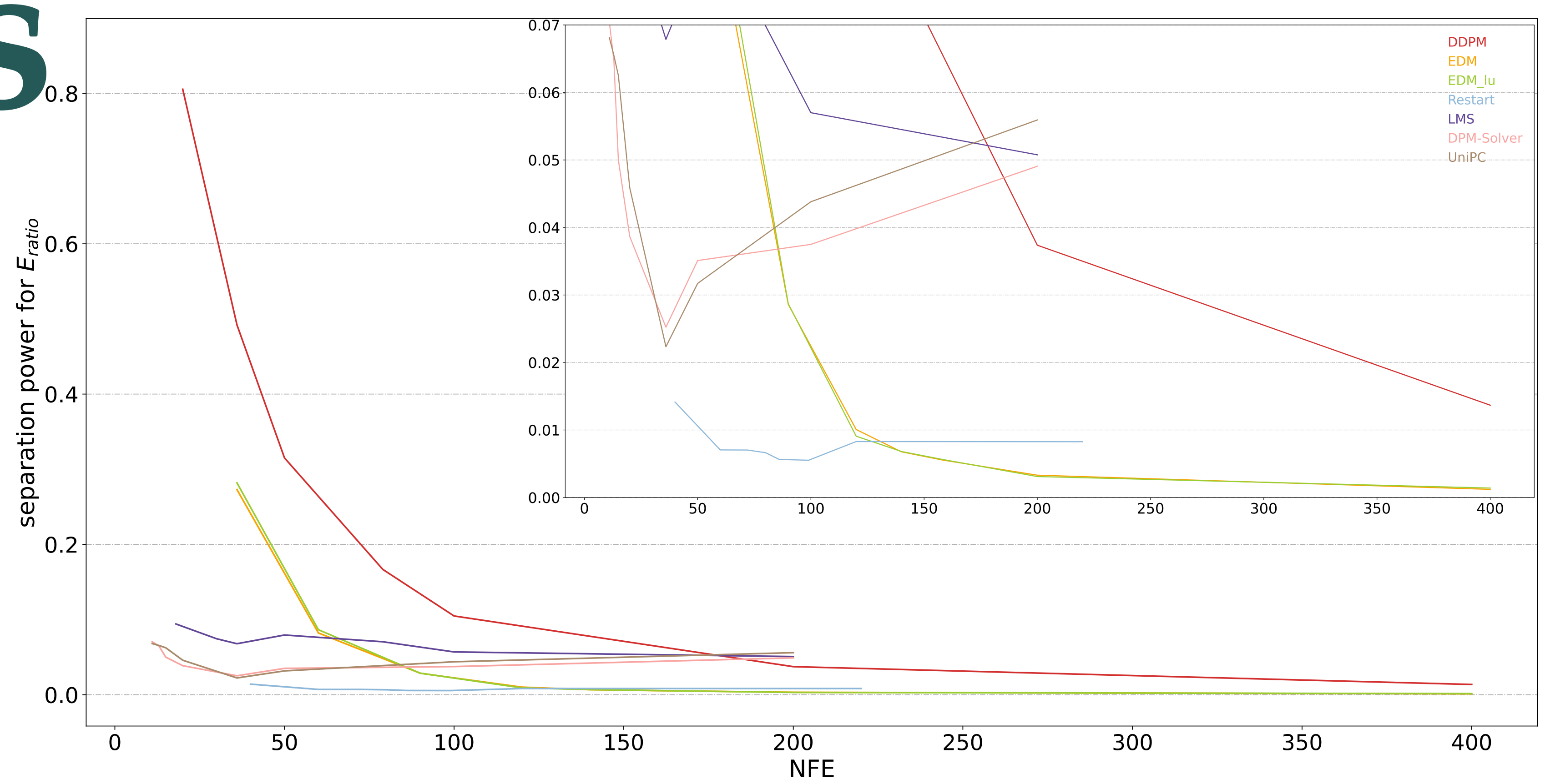




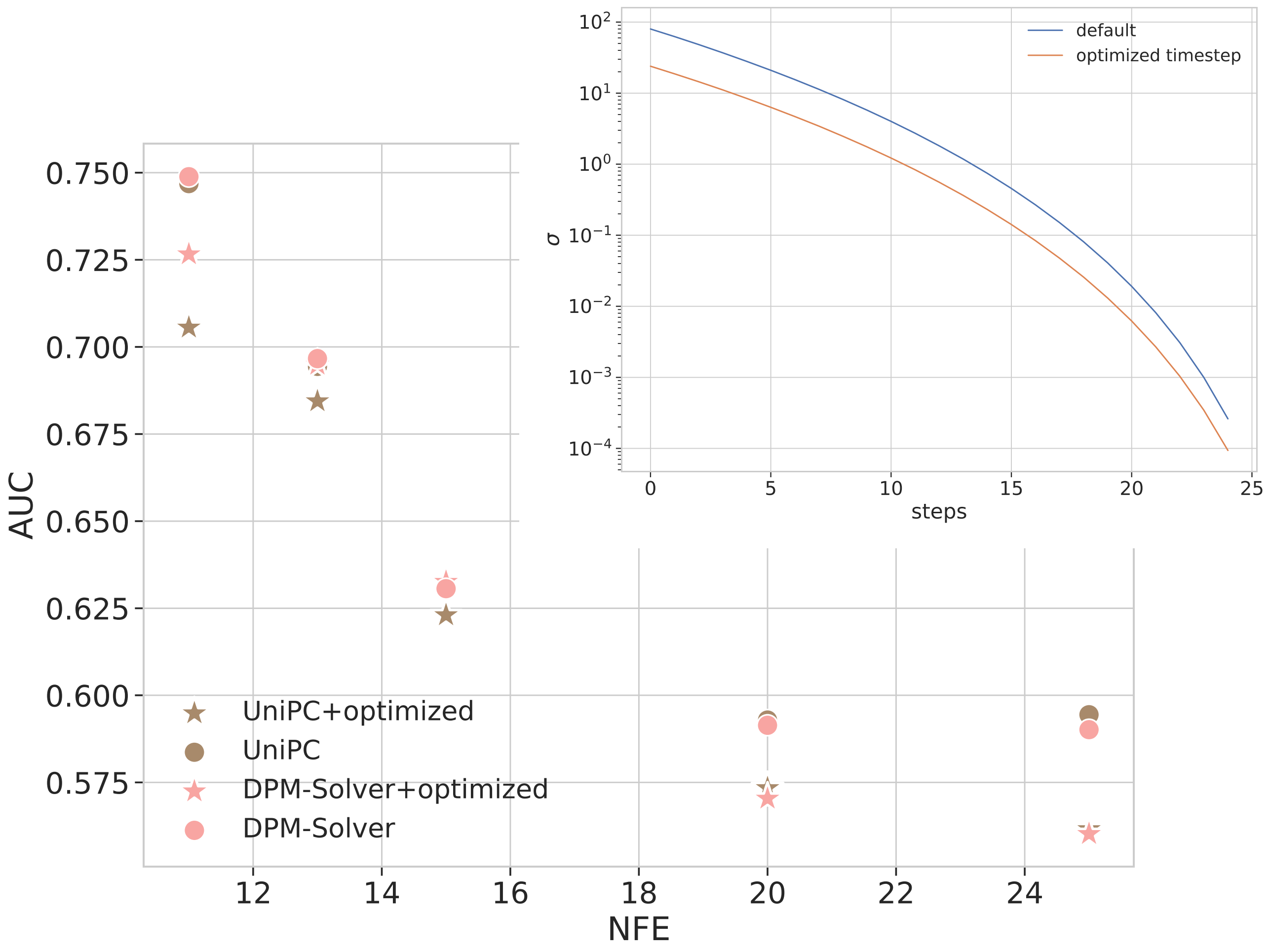
Scheduler Matters

- Ideas based on [Align your steps](), [Skip-step](), [timestep-optimizer]().
- Try to find the best sampling parameters for ODE/SDE, min/max σ initialization via LinearConstraint, to minimize high level feature FPD, divergence...

- Most improvement seen for high-level feature metrics with ODE samplers.



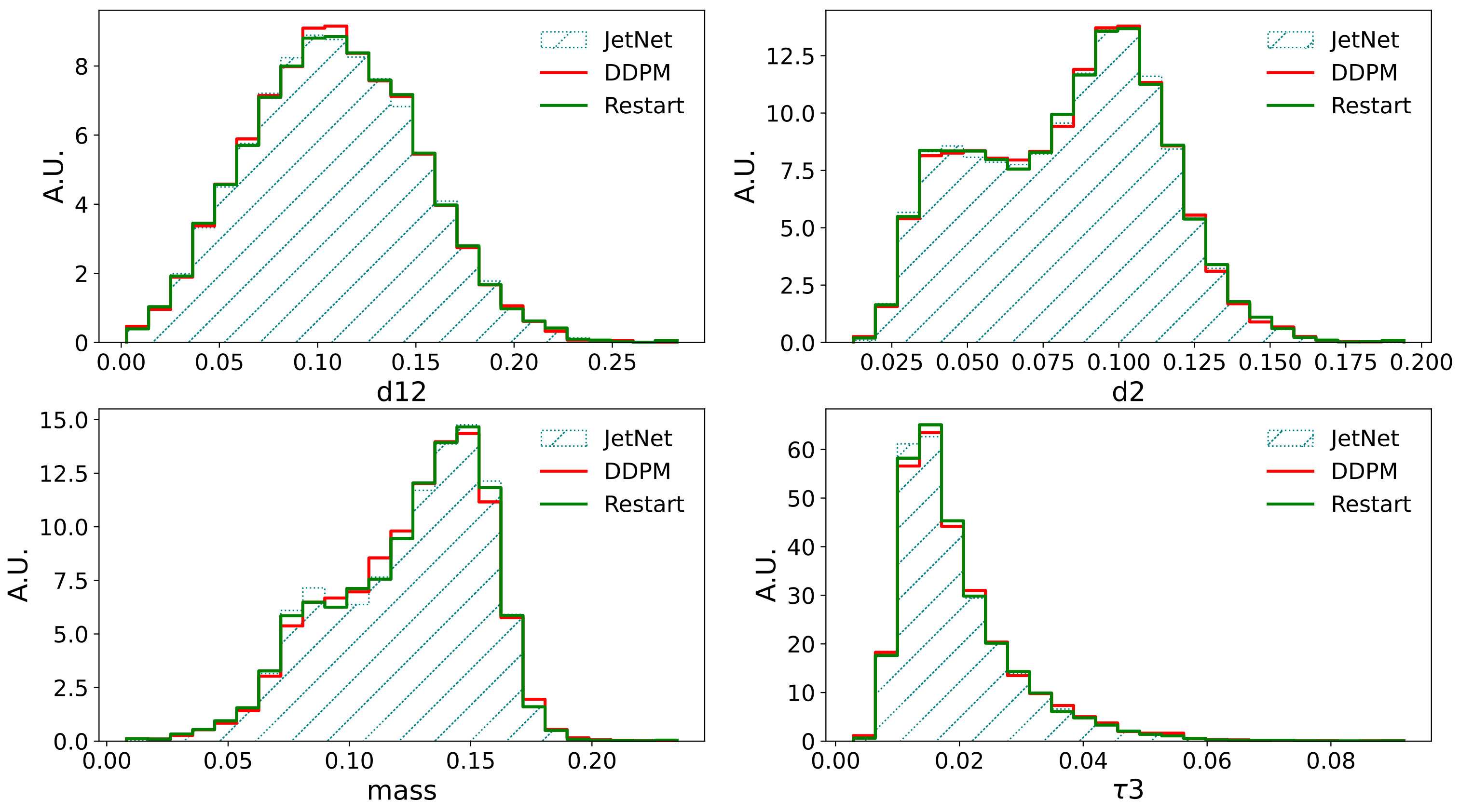
before/after	AUC	Separation	FPD
DPM++ 25	59.8/56.8	0.0534/0.0252	0.146/0.113
UniPC 25	60.2/56.9	0.1304/**0.0231**	0.152/0.112
DDPM 200	55.7	0.0344	0.046



And…?

And we can fuse!

- Could flowBDT works with diffusion?
- Yes, it can also work with diffusion.
- Quick test case: 18 steps (~40 NFE) Restart v.s. 200 steps DDPM



Separation $*10^3$ /EMD$*10^2$	d12	d2	mass	τ3
Restart 18	0.055/0.0029	0.150/0.0039	0.105/0.0041	0.045/0.0048
DDPM 200	0.0080/0.0045	0.191/0.0065	0.145/0.0045	0.070/0.0079

More interestingly

- Restart need a predefined parameters to work, usually need to manually setup the timestep interval and value k.
- Number of evaluation steps > sampling steps (higher order).
- Could flowBDT helps Restart as precomputed x_t to feed in a regression? → experimenting now

Conclusion

- Showed how could we replace the traditional NN backbone generative model with GBDT backbone to deal with tabular data.
- The advantages brought by GBDT is its fast inference, very fast in low-dim, still comparably fast in high dimension.
- We tested in various tasks like end-to-end high feature, calorimeter cell simulations, as well as the conditional generations, all shows promising result.

- A more comprehensive study about DM in fast calorimeter simulation, we implemented few training-free tricks to improve the schedulers in ODE samplers, though still the best ones are EDM and restart, those ODE samplers can now also be good candidates that provide great quality/sampling time tradeoff.