K. Scalability with respect to Dimension and Constraints

We first present the results of exact sampling, computing closed-form loss, and gradient estimator with respect to varying dimensions. The number of constraints is half of the dimension (n). All results are averaged over 100 runs and conducted on a Nvidia A800 80GB GPU. All numbers are measured in seconds. We highlight that even though increasing dimension slows down the algorithm, the additional computational cost can be mitigated by choosing a larger batch size. For example, the additional time taken to run with `batch_size`=16 and `batch_size`=32 is almost negligible for n = 32, 64, and 128. In the challenging n = 2048 case, using `batch_size`=32 only takes 1.5 times more time to run compared to `batch_size`=16. Larger batch sizes can significantly reduce computation time per sample thanks to the parallelization of our method.

	16	32	64	128	256
$n = 32$	0.00249	0.00250	0.00252	0.00253	0.00255
$n = 64$	0.00348	0.00348	0.00348	0.00352	0.00366
$n = 128$	0.00467	0.00473	0.00479	0.00483	0.00498
$n = 256$	0.00760	0.00769	0.00806	0.00916	0.01052
$n = 512$	0.00760	0.01770	0.02159	0.02921	0.04500
$n = 1024$	0.04843	0.06266	0.09138	0.13943	0.23699
$n = 2048$	0.20515	0.31654	0.46224	0.78970	1.24126

Table 11. Computation time (in seconds) for varying dimensions and batch sizes.

Algorithm 2 Varying Dimensions

```
 1: for each $n \in ls\_n$ do
 2:     $num\_constraints \leftarrow n/2$
 3:     $Total\_time \leftarrow 0$
 4:     for $i \leftarrow 1$ to $repeat\_num$ do
 5:         Generate unconstrained parameters and constraints
 6:         Conduct exact sampling
 7:         Compute closed-form loss
 8:         Compute gradient estimator
 9:         $Total\_time \leftarrow Total\_time +$ time
10:     end for
11:     $Total\_time \leftarrow Total\_time/repeat\_num$
12:     print($Total\_time$)
13: end for
```

We next investigate the impact of the number of constraints (`num_k`). We conduct experiments on the challenging n = 2048 setting. Similar to the findings in scaling the number of dimensions, using larger batch sizes significantly reduces computation time per sample. Additionally, we also found that increasing the number of constraints does not significantly increase the computational time. For example, under `batch_size` = 64, increasing the number of constraints from 256 to 512 only increases computational time by 32%.

	16	32	64	128	256
`num_k` = 32	0.05109	0.09899	0.19531	0.29093	0.47706
`num_k` = 64	0.05343	0.10259	0.20241	0.30316	0.48106
`num_k` = 128	0.05894	0.11060	0.21666	0.32456	0.50057
`num_k` = 256	0.07336	0.13211	0.24871	0.38556	0.52001
`num_k` = 512	0.10885	0.18119	0.32935	0.52796	0.84375
`num_k` = 1024	0.20515	0.31654	0.46224	0.78970	1.24126

Table 12. Computation time (in seconds) for varying number of constraints and batch sizes at $n = 2048$.

Algorithm 3 Exact Sampling and Gradient Estimation over Varying Number of Constraints

```
 1: for each $num\_k \in ls\_k$ do
 2:     $Total\_time \leftarrow 0$
 3:     for $i \leftarrow 1$ to $repeat\_num$ do
 4:         Generate unconstrained parameters and constraints
 5:         Conduct exact sampling
 6:         Compute closed-form loss
 7:         Compute gradient estimator
 8:         $Total\_time \leftarrow Total\_time +$ time
 9:     end for
10:     $Total\_time \leftarrow Total\_time/repeat\_num$
11:     print($Total\_time$)
12: end for
```

L. Additional Experiment on CL and Constrained Reparametrization

Additional experiment results on CL and Constrained Reparametrization are highlighted with gray.

Table 13. Comparison on VAE generative performance. The constrained VAE models achieve similar or better generative ability while strictly satisfying the constraints, whereas the unconstrained counterparts have a high constraint violation rate.

MODEL	LL ↑	ELBO ↑	RL ↓	VIOLATION ↓
VAE	-22.42 ± 0.29	-23.41 ± 0.22	15.00 ± 0.46	0.30 ± 0.06
VAE + Constr Layer	-34.45 ± 2.64	-40.89 ± 9.37	37.11 ± 9.35	**0.00** ± **0.00**
VAE + Constr Reparam	-22.33 ± 0.48	-23.83 ± 0.49	14.54 ± 0.58	**0.00** ± **0.00**
ours	**-21.48** ± **0.18**	**-22.62** ± **0.07**	**12.79** ± **0.11**	**0.00** ± **0.00**
Ladder VAE	-24.25 ± 0.07	-30.84 ± 0.51	**23.06** ± **0.54**	0.38 ± 0.02
Ladder VAE + Constr Layer	-36.83 ± 0.56	-39.59 ± 0.56	37.46 ± 0.55	**0.00** ± **0.00**
Ladder VAE + Constr Reparam	-25.27 ± 0.19	-31.56 ± 0.48	25.20 ± 0.62	**0.00** ± **0.00**
ours	**-23.86** ± **0.06**	**-30.78** ± **0.08**	**23.40** ± **0.16**	**0.00** ± **0.00**
Graph VAE	-22.74 ± 0.11	-23.54 ± 0.18	15.45 ± 0.41	0.29 ± 0.09
Graph VAE + Constr Layer	-33.27 ± 3.60	-33.27 ± 5.84	28.29 ± 6.40	**0.00** ± **0.00**
Graph VAE + Constr Reparam	-22.96 ± 0.80	-23.55 ± 0.86	16.08 ± 1.38	**0.00** ± **0.00**
ours	**-21.61** ± **0.20**	**-22.53** ± **0.06**	**12.73** ± **0.21**	**0.00** ± **0.00**

L.1. Constrained Generation using VAE

L.2. Constrained Generation using Diffusion Models

Table 14. Comparison of constrained and unconstrained models across datasets under DDPM. We report FID (Fréchet Inception Distance), IS (Inception Score), and Violation metrics.

DATASET	MODEL	FID ↓	IS ↑	VIOLATION ↓
CIFAR	Ours	**3.811**	9.223 ± 0.130	**0**
	Constr Layer	4.234	8.535 ± 0.157	**0**
	Constr Reparam	3.881	9.126 ± 0.122	**0**
	DDPM	4.173	**9.278** ± **0.116**	0.999
CelebA	Ours	**10.193**	**2.360** ± **0.016**	**0**
	Constr Layer	12.067	2.345 ± 0.039	**0**
	Constr Reparam	10.305	2.334 ± 0.028	**0**
	DDPM	10.345	2.358 ± 0.030	0.999
LSUN Church	Ours	**4.779**	**2.471** ± **0.020**	**0**
	Constr Layer	6.695	2.319 ± 0.028	**0**
	Constr Reparam	4.895	2.431 ± 0.032	**0**
	DDPM	4.945	2.460 ± 0.028	1.0
LSUN Cat	Ours	**12.489**	**4.711** ± **0.054**	**0**
	Constr Layer	13.472	4.642 ± 0.081	**0**
	Constr Reparam	12.696	4.707 ± 0.062	**0**
	DDPM	12.913	4.705 ± 0.047	1.0

L.3. Charge-Neutral Predictions

Table 15. Performances of different methods for estimating partial charges on metal ions are presented. Compared to the baseline MPNN (variance), both the closed-form loss function and likelihood objective yield superior mean absolute deviation (MAD) results. The same holds for their ensemble counterpart. We find that ensemble methods (second block) notably boost the predictive performance in general.

METHOD	MAD ↓	NLL ↓
neutrality enforcement	mean ± std	mean ± std
Constrained Layer	0.327 ± 0.004	103.522 ± 3.018
Constant Prediction	0.324 ± 0.007	—
Element-mean (uniform)	0.154 ± 0.002	—
Element-mean (variance)	0.153 ± 0.002	—
MPNN (KKThPINN)	0.0260 ± 0.0008	109.8 ± 6.9
MPNN (Constr Reparam)	0.0256 ± 0.0007	3340.03 ± 2398
MPNN (variance)	0.0251 ± 0.0010	-19.9 ± 71.1
Closed-form (ours)	**0.0245** ± **0.0009**	$> 1e+7$
Likelihood (ours)	0.0248 ± 0.0008	**-252** ± **24.7**
Constrained Layer (ens)	0.319 ± 0.002	99.236 ± 2.3
MPNN (ens, KKThPINN)	0.0244 ± 0.0006	57.29 ± 12.8
MPNN (ens, Constr Reparam)	0.0242 ± 0.0006	4883.46 ± 1695
MPNN (ens, variance)	0.0238 ± 0.0007	-45.2 ± 55.8
Closed-form (ens, ours)	**0.0230** ± **0.0008**	$> 1e+7$
Likelihood (ens, ours)	**0.0231** ± **0.0007**	**-180** ± **38.3**

L.4. Chemical Process Units and Subsystems

Table 16. Comparison of models across **CSTR**, **plant**, and **distillation** tasks. The mean and standard deviation of MSE scaled by 10^{-4} are reported. All experiments are averaged for 10 times.

MODEL	CSTR	PLANT	DISTILLATION
ECNN	20.6 ± 27.0	0.31 ± 0.23	1.94 ± 0.70
KKThPINN	11.7 ± 20.3	0.11 ± 0.04	2.02 ± 0.94
NN	18.3 ± 20.8	0.34 ± 0.64	1.99 ± 0.67
PINN	260.8 ± 20.4	3.62 ± 1.94	40.9 ± 10.7
CL	9.28 ± 3.56	0.58 ± 0.64	2.26 ± 1.19
Constr Reparam	7.13 ± 3.22	0.14 ± 0.07	2.22 ± 0.94
Ours	$\mathbf{4.31} \pm \mathbf{1.58}$	$\mathbf{0.09} \pm \mathbf{0.05}$	$\mathbf{1.73} \pm \mathbf{0.70}$

L.5. Stock Investment

Table 17. Comparison of models based on Sharpe ratio. We report the mean and standard deviation averaged across 10 runs.

MODEL	SHARPE RATIO $\uparrow$
StemGNN	1.5576 ± 0.3405
StemGNN-KKThPINN	1.8092 ± 0.7055
StemGNN-CL	1.5018 ± 0.3318
StemGNN-Constr Reparam	1.8162 ± 0.2966
Ours	$\mathbf{1.9041} \pm \mathbf{0.2329}$