Yan Zhang, Jonathon Hare, and Adam Prügel-Bennett. Fspool: Learning set representations with featurewise sort pooling. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJgBA2VYwH.

A PROOF OF LEMMA 1

Generative adversarial networks Given a set generator f, a discriminator function d, and $\boldsymbol{X}_1, ..., \boldsymbol{X}_m$ a training dataset, the standard loss function for GANs is formulated as

$$l(f, d, \boldsymbol{X}_1, ..., \boldsymbol{X}_m) = \frac{1}{m} \sum_{i=1}^{m} \log(d(\boldsymbol{X}_i))] + \mathbb{E}_{\mathbb{Z}}[\log(1 - d(f(\boldsymbol{z}))).$$

In order to obtain $l(f, d, \boldsymbol{X}_1, ..., \boldsymbol{X}_m) = l(f, d, \pi_1.\boldsymbol{X}_1, ..., \pi_m.\boldsymbol{X}_m)$ for every choice of π_i, it is therefore sufficient to choose a permutation invariant discriminator.

Autoencoder based models We assume that the autoencoder is made of a permutation invariant encoder *enc* and an arbitrary decoder f. For any set size n, set $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ and permutation $\pi \in \mathbb{S}_n$ we have

$$\begin{aligned} l(\pi.\boldsymbol{X}, \hat{\boldsymbol{X}}) = l(\boldsymbol{X}, \hat{\boldsymbol{X}}) &\implies l(\pi.\boldsymbol{X}, f(\textit{enc}(\boldsymbol{X})) = l(\boldsymbol{X}, f(\textit{enc}(\boldsymbol{X})) \\ &\implies l(\pi.\boldsymbol{X}, f(\textit{enc}(\pi.\boldsymbol{X})) = l(\boldsymbol{X}, f(\textit{enc}(\boldsymbol{X})) \quad (\textit{enc} \text{ is invariant}) \\ &\implies \nabla_\theta \, l(\pi.\boldsymbol{X}, f(\textit{enc}(\pi.\boldsymbol{X}))) = \nabla_\theta \, l(\boldsymbol{X}, f(\textit{enc}(\boldsymbol{X}))) \end{aligned}$$

B LOSS FUNCTIONS FOR SETS

Chamfer's loss and the Wasserstein 2 distance are defined as

$$d_{\text{Cham}} = \sum_{1 \leq i \leq n} \min_{j \leq n'} ||\boldsymbol{x}_i - \boldsymbol{x}'_j||_2^2 + \sum_{1 \leq j \leq n'} \min_{i \leq n} ||\boldsymbol{x}_i - \boldsymbol{x}'_j||_2^2$$

$$d_{\mathcal{W}_2} = \inf_{u \in \{\Gamma(\boldsymbol{X}, \boldsymbol{X}')\}} \sum_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n'}} u(\boldsymbol{x}_i, \boldsymbol{x}'_j) \, \|\boldsymbol{x}_i - \boldsymbol{x}'_j\|_2^2$$

where Γ is the set of couplings (i.e., bistochastic matrices) between $\boldsymbol{X}$ and $\boldsymbol{X}'$. Both loss functions solve a matching problem over the space of permutations, which makes them invariant to permutations of one argument, as required by Lemma 1. Chamfer's loss runs in quadratic time, while the standard implementation of Wasserstein distance runs in $O(n^3)$ if both sets have the same size. However, efficient algorithms exist for approximating the Wasserstein distance, and the computations can be parallelized on GPUs (Cuturi, 2013; Feydy et al., 2019).

Note however that the equation $l(\pi.\boldsymbol{X}, \hat{\boldsymbol{X}}) = l(\boldsymbol{X}, \hat{\boldsymbol{X}})$ is may be difficult to satisfy in other settings: matching graphs up to permutations, or sets up to the $SE(3)$ group, leads to difficult problems for which no polynomial time algorithm is known (Mémoli, 2007). In these settings, the design of VAE is harder and other architectures may be better suited.

C PROOF OF PROPOSITION 2

Proof. We first give the proof for First-n creation:

First-n Given a set $\boldsymbol{Y} = \{\boldsymbol{y}_1, ..., \boldsymbol{y}_n\}$ of points in $\mathbb{R}^d$, we propose to sort the points $\boldsymbol{y}_i$ by alphanumeric sort (sort the values using the first feature, then sort points that have the same first features along the second feature, etc.). We denote the resulting matrix by $\boldsymbol{Y}'$. We choose as a latent vector $\boldsymbol{z} = \textit{flatten}(\boldsymbol{Y}') \in \mathbb{R}^{nd}$. It is the vector that contains the representation of $\boldsymbol{y}'_1$ in the first d features, $\boldsymbol{y}'_2$ is the next d features... By construction, $\boldsymbol{z}$ is a permutation invariant representation of the set $\boldsymbol{Y}$ (this reflects the fact that there are canonical ordering for sets, which is not the case for general graphs).

We choose as a reference set the canonical basis in $\mathbb{R}^n$ ($\boldsymbol{R} = \boldsymbol{I}_n$). After the latent vector is appended to the representation of each point, we have $\boldsymbol{r}_i = (\boldsymbol{e}_i, \boldsymbol{z})$. We are now looking for a function f that allows to approximate the set $\boldsymbol{Y}$, i.e., that satisfies $\forall i \leq n, f(\boldsymbol{e}_i, \boldsymbol{z}) = \boldsymbol{z}[i\ d : (i+1)d]$. We can choose $f(\boldsymbol{e}_i, \boldsymbol{z}) = \boldsymbol{e}_i^T \boldsymbol{W}_1 \boldsymbol{W}_2 \boldsymbol{z}$, where

$$\boldsymbol{W}_1 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & ... & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & & 1 & 1 & 1 \end{bmatrix} \in \mathbb{R}^{n \times nd} \quad \text{and} \quad \boldsymbol{W}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ & ... & \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{nd \times d}$$

.

This function is continuous, and its output is $\boldsymbol{Y}' = \textit{reshape}_{n \times d}(\boldsymbol{z})$. $\boldsymbol{Y}'$ is equal to $\boldsymbol{Y}$ up to a permutation of the rows, so that $||\boldsymbol{Y} - \boldsymbol{Y}'||_{\mathcal{W}_2} = 0$. If the entries of $\boldsymbol{z}$ are bounded, we can use standard approximation results for continuous functions over a compact space (Cybenko, 1989) to conclude that it be uniformly approximated by a 2-layer MLP.

Top-n Consider a Top-n network with n reference points such that:

- The angles of the reference points are 2d vectors such that $\phi_i = (\cos(\frac{i}{n}\frac{\pi}{4}), \sin(\frac{i}{n}\frac{\pi}{4}))$.
- The representations are $r_i = e_i / \cos(\frac{i}{n_0}\frac{\pi}{4})$, where (e_i) is the canonical basis in $\mathbb{R}^n$.
- The MLP of equation 1 (that predicts an angle from the latent vector) always outputs $(1, 0)$.

Then this Top-n creation module is equivalent to the First-n module build previously: for any set, it selects the first n points and returns $\boldsymbol{X}^0[i] = \boldsymbol{e}_i$. The same MLP that is built for First-n can therefore be used for this network. □

D Details about the experimenal setting

D.1 Set MNIST and CLEVR

Since we use existing code for this task, we refer to the respective papers (Zhang et al., 2019) and (Kosiorek et al., 2020) for details on the model used and the loss function. The code used for TSPN is not the original code (which is not available) but a reimplementation (not by one of the authors of the present paper). The reader will notice that our results for DSPN are approximately 3 times worse than in the original paper. The reason is that we fixed what we believe to be a bug in the implementation of Chamfer's loss in DSPN: a mean over channels was used instead of a sum, which explains this difference of a factor 3. We also note that the results for TSPN are around 3 times worse than the original paper. One possible reason could be that the authors of TSPN used the code of DSPN and had a similar bug.

For Top-n generation, we set the number of points in the reference set to twice the cardinality n of the generated sets. We also experimented with $n_0 = n$ which resulted in better performance for DSPN (with a Chamfer loss of 6.14 ± 0.56 e-5), but not for TSPN (16.07 ± 0.47). We observed that reducing the learning rate improves results for all methods: TSPN was therefore trained for 100 epochs with a learning rate of 5e-4, and DSPN with a learning rate of 1e-4 for 200 epochs. No other hyper-parameter was tuned.

D.2 Synthetic set generation

Dataset generation procedure Each set is created via rejection sampling: points are drawn iteratively from a uniform distribution within a bounding box in $\mathbb{R}^3$. The first point is always accepted, and the next ones are accepted only if they satisfy a predefined set of constraints: i) they are not closer to any other point than a given threshold *min-distance*. ii) they are connected to the rest of the set, i.e., have at least one neighbor than a distance *neighbor-distance* iii) they do not have too many neighbors. Our dataset is made of 2000 sets that have between 2 and 35 points, with 9 points per set on average. It is a simplification of real molecules in several aspects: there are only single

Table 6: Train reconstruction error and valency loss in the generated sets over 5 runs for a modified version of our dataset, where the cardinality varies less across sets. We observe a tradeoff between reconstruction performance and generalization.

Reference points	11	13	15	20	30	50
$\mathcal{W}_2$ train loss	$\mathbf{0.75}_{\pm.04}$	$0.78_{\pm.03}$	$0.79_{\pm.04}$	$0.87_{\pm.05}$	$0.93_{\pm.04}$	$1.03_{\pm.06}$
Valency loss (e-1)	$2.8_{\pm.7}$	$2.2_{\pm0.7}$	$2.1_{\pm.9}$	$2.4_{\pm1.2}$	$\mathbf{1.6}_{\pm.2}$	$2.8_{\pm.8}$

bonds, angles between bonds are not constrained and the atom types are only defined by the valency, which reflects the fact that atoms with the same valency tend to play a similar role and be more interchangeable.

Model The set encoder is made of a 2-layer MLP, 3 transformer layers followed by a PNA global pooling layer (that computes the sum, mean, max and standard deviation over each channel) (Corso et al., 2020) and a 2-layer MLP. The decoder is made of a set generator followed by a linear layer, 3 transformer layers and a 2-layer MLP. We use residual connections when possible and batch normalization between each layer. The reference set contains 35 points.

We experimented with the two ways to sample the number of points presented in Section 2, but we found the results to be quite similar. We therefore opted sampling the number of points from the data distribution, which is the simplest method.

Loss function We use a standard variational autoencoder loss with a Wasserstein reconstruction term and two additional regularizers. Given an input set $\boldsymbol{X}$ and its reconstruction $\hat{\boldsymbol{X}}$, the loss can be written:

$$L(\boldsymbol{X}, \hat{\boldsymbol{X}}) = d_{W_2}(\boldsymbol{X}, \hat{\boldsymbol{X}}) + \lambda_1 \, KL(p(\boldsymbol{z} \mid \boldsymbol{X}), \mathcal{N}(0, \boldsymbol{I}_l)) + \lambda_2 reg_2\,(\hat{\boldsymbol{X}}) + \lambda_3 \, reg_3(\hat{\boldsymbol{X}})$$

where

$$reg_2(\boldsymbol{X}) = \sum_{1 \leq i < j \leq n} (d_0 - ||\boldsymbol{x}_i - \boldsymbol{x}_j||_2)_+ \quad \text{with} \quad d_0 = 1$$

prevents atoms from being generated too close to each other. $reg_3(\boldsymbol{X})$ penalizes atoms that have either no neighbor, or a too large valency. It is computed in the following way:

- for each point i, compute $s^i = \text{sort}((d_{ij})_{j \leq n,\, j \neq i})$. This vector contains the sorted distances between i and all other points. Points that are at distance less than $d_1 =$ *neighbor-distance* from i are considered as its neighbours.
- Compute $l_1(i) = (s^i_0 - d_1)_+$. This term penalizes atoms that have no neighbour.
- Compute $l_2(i) = \sum_{j=\textit{max-valency}}^{n-1} (d_1 - s^i_j)_+$. This term penalizes atoms that have too many neighbors.
- $reg_3(X)$ is defined as $\sum_{1 \leq i \leq n} l_1(i) + l_2(i)$.

Training details In order to train the model efficiently, mini-batches have to be used. This may not be easy when dealing with sets and graphs, since they do not have all the same shape. To circumvent this issue, we reorganise the training data in order to ensure that all sets inside a mini-batch have the same size. At generation time, this method cannot be applied, so we simply generate sets one by one.

The optimizer is Adam with its default parameters. We use a learning rate of $2e^{-4}$ and a scheduler that halves it when reconstruction performance does not improve significantly after 750 epochs. Experimentally, we found the learning rate decay to be important to achieve good reconstruction.

We also run a study with different reference set sizes. For this purpose, we slightly modify our dataset so that each set has only up to 11 points (still with 9 points on average). The reason is that there is more flexibility in the choice of the reference size if the maximal size is not too large. By training a Top-n network with several reference set sizes, we obtain the results of Table 6.

D.3 MOLECULAR GRAPH GENERATION ON QM9

Model Our encoder is a graph neural network is made of 3 message-passing layers followed by a PNA global pooling layer and a final MLP. For the decoder, we use a set creation method followed by transformer layers. The resulting representation is then processed by i) a Set2Graph layer (Serviansky et al., 2020) followed by two MLPs to generate edge probabilities and edge features a MLP which generates node features ii) a MLP that takes as input the set representation and the valencies predicted for each atom, and returns an atom type.

Graph matching and loss function As explained in Appendix B, the loss function of the variational autoencoder should solve a graph matching problem, which is hard in general. Instead of using a proper graph matching method, we propose to use the atom types to perform an imperfect but much cheaper alignment between the target and the predicted molecules.

For both molecules, we compute a score for each atom i defined as:

$$s(i) = 10^5 \textit{atom-type}(i) + 10^4 \textit{num-edges}(i) + \sum_{j \in \mathcal{N}(i)} \textit{edge-type}(i,j) * \textit{atom-type(j)}$$

This score cannot differentiate between all atoms in each molecule, but it reduces drastically the number of permutations that can represent the same input. It is motivated by the fact that empirically, we observe that our method quickly learns to reconstruct the molecular formula very well. Once they are all computed, we use these scores to sort the atoms reorder the adjacency matrix and the atom and edge types.

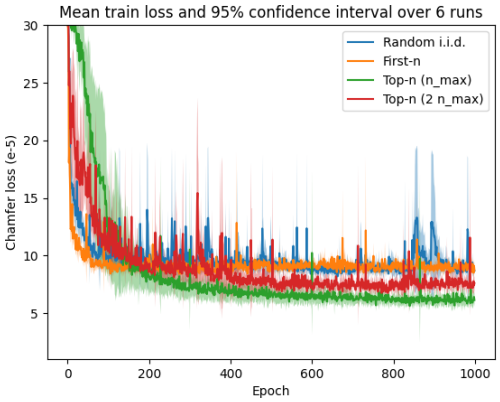
Our model learns to predict a probabilistic model for the atom types, edge presence and edge types. For this purpose, we use standard cross entropy loss between the predicted probabilities and the ground truth. However, these metrics can be hard to optimize because of the imperfect graph matching algorithm. We therefore regularize these metrics with several other measures at the graph level, that do not depend on matching:

- The mean squared error between the real atomic formula and the predicted one.
- The mean squared error between the average number of edges per atom in the input and predicted molecule.
- The mean squared error between the distribution of edge types in the real and predicted molecule.

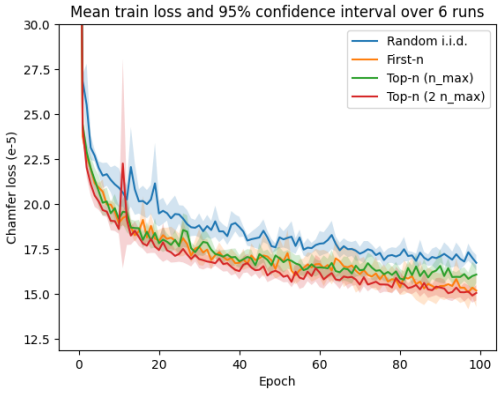
Finally, we add a matching dependent term, which is the mean squared error between the valencies of the input and the target molecule.

Training details The model is trained over 600 epochs with a batch size of 512 and a learning rate of $2e^{-3}$. It is halved after 100 epochs when the loss does not improve anymore. The optimizer is Adam with default parameters. The reference set has 12 points. When using more points, we obtained overall similar results, but with a larger variance.

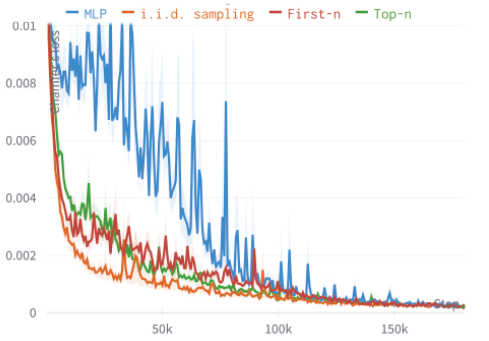
E TRAINING CURVES



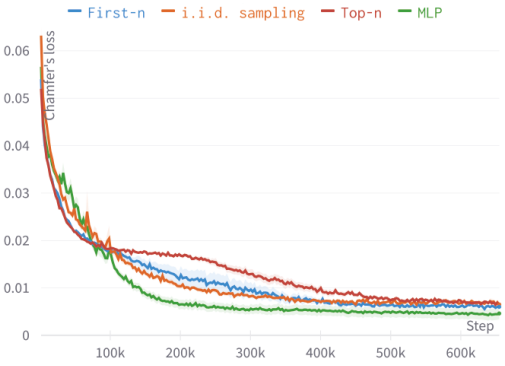
(a) DSPN training on Set-MNIST



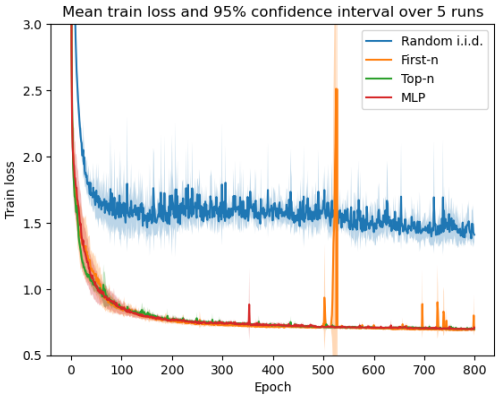
(b) TSPN training on Set-MNIST



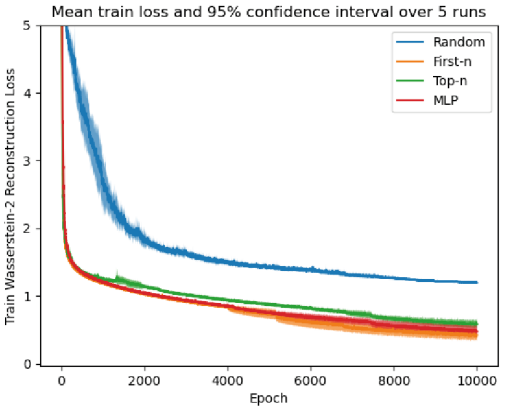
(c) DSPN training on CLEVR for bounding box prediction.



(d) DSPN training on Set-MNIST for full state prediction.



(e) Molecule generation on QM9. First-n, Top-n and MLP mostly overlap.



(f) Synthetic molecule-like dataset in 3d.

Figure 4: Training curves for all models. We observe that random i.i.d. generation is in general harder to train than the other models, while the differences between the other methods are smaller.