Project report: Equivarariant diffusion for molecule generation

Marco Andreis — s243116

1 Overview

In the field of drug discovery, a major bottleneck is represented by the resources necessary to create and inspect large libraries of chemical compounds. These libraries are the starting point for a long and iterative process of selection ultimately leading to only a few good candidates, which will actually advance to clinical trials and, if approved, on the market. Following this premise, generative machine learning approaches to explore the chemical space have been developed and shown to be capable of effectively producing plausible molecules.

The aim of the project was the replication of the results obtained by Hoogeboom et al. in the paper "Equivariant diffusion for molecule generation" [1]. This was pursued starting from foundational papers at the base of the above-mentioned work, the denoising diffusion probabilist model (DDPM) introduced by Ho et al. [2] and work on equivariant graph neural networks (EGNN) by Satorras et al. [3]. More in detail, I implemented a baseline model with moderate performance, then performed multiple experiments aimed at familiarizing with the impact caused by changing hyperparameters and other slight modifications to the model architecture.

2 Models

2.1 DDPM

Denoising diffusion probabilistic models (DDPM) are latent generative models defined by a destructive forward process and a reverse generative process. In the forward process, the original data distribution is corrupted to a standard gaussian prior via iterative application of the following transition kernel:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t|\sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbf{I}) \tag{1}$$

with β being the noise schedule, a parameter balancing how much information is retained at any step t.
This formulation has the convenient property of allowing sampling at arbitrary timestep through:

$$q(x_t|x_0) = \mathcal{N}(x_t|\sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{2}$$

defining $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$.

The reverse process is instead defined by:

$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}\big(x_{t-1};\, \tilde{\mu}_t(x_t, x_0),\, \tilde{\beta}_t\mathbf{I}\big) \tag{3}$$

$$\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\,\beta_t}{1-\bar{\alpha}_t}\, x_0 + \frac{\sqrt{\alpha_t}\,(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\, x_t \qquad \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\,\beta_t \tag{4}$$

In practice, during sampling of new data instances, x_0 is unknown. This means that learning the generative task equals to learning to predict the transition kernel in Eq. 3 based on a noisy datapoint and the current timestep:

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}\big(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)\big) \sim q(x_{t-1} \mid x_t, x_0) \tag{5}$$

with μ_θ and Σ_θ being neural networks.
Allowing the simplification of fixing $\Sigma_\theta(x_t, t) = \sigma_t^2 \mathbf{I}$ with $\sigma_t^2 = \beta_t$, Ho et al. showed that the model can be trained using a simplified loss function:

$$\text{Loss} = \mathbb{E}_{x_0,\epsilon}\left[\frac{\beta_t^2}{2\sigma_t^2\,\alpha_t(1-\bar{\alpha}_t)}\left\|\epsilon - \epsilon_\theta\big(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon, t\big)\right\|^2\right] \tag{6}$$

where the μ_θ learns to predict the noise added at each time step, and is therefore referred to as noise network.
Training and sampling can be performed using the algorithms reported below:

Algorithm 1 - Training	**Algorithm 2 - Sampling**

repeat
$x_0 \sim q(x_0)$
$t \sim$ Uniform$(\{1, \dots, T\})$
$\epsilon \sim \mathcal{N}(0, \mathbf{I})$
Take gradient descent step on
$\nabla_\theta \|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\|^2$
until converged
	$x_T \sim \mathcal{N}(0, \mathbf{I})$
	for $t = T, \dots, 1$ **do**
	$z \sim \mathcal{N}(0, \mathbf{I})$ if $t > 1$, else $z = \mathbf{0}$
	$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right) + \sigma_t z$
	end for
	return x_0

2.2 EGNN

Following work Hoogeboom et al. [1], molecules are represented as 3D point clouds, i.e. graphs having each node defined by features ($h \in \mathbb{R}^{\text{nf}}$) and 3D coordinates ($x \in \mathbb{R}^3$). A desirable property for a neural network in this context is to have its output not be altered if all of the input nodes are translated or rotated by the same amount. Not only this reflects how the property of a molecule do not change if it rigidly moves in the space, but also restrict the function space that the network has to explore.
A function having this property is called equivariant to SE(3) transformations, formally defined as:

$$f(\mathbf{R}x + t) = \mathbf{R}f(x) + t \tag{7}$$

where $\mathbf{R} \in \mathbb{R}^{3\times3}$ is a rotation matrix and $t \in \mathbb{R}^{1\times3}$ is a translation vector.

As noise network for the diffusion model, a good fit are Equivariant graph neural networks [3], an extension of message-passing neural networks [4] satisfying SE(3) equivariance through the following update rules:

$$m_{ij} = \phi_e(h_i^l, h_j^l, \|x_i^l - x_j^l\|_2^2, a_{ij}) \tag{8}$$

$$x_i^{l+1} = x_i^l + C\sum_{j\in\mathcal{N}(i)}(x_i^l - x_j^l)\cdot\phi_x(m_{ij}) \tag{9}$$

$$m_i = \sum_{j\in\mathcal{N}(i)} m_{ij} \tag{10}$$

$$h_i^{l+1} = \phi_h(h_i^l, m_i) \tag{11}$$

An important detail is that, while in the paper from Satorras et al. the feature update rule is defined by Eq. 11, in the actual implementation found in the github repository of the project the update is instead a residual update (as indicated and suggested in the appendix):

$$h_i^{l+1} = h_i^l, +\phi_h(h_i^l, m_i) \tag{12}$$

applying ϕ_h only to the aggregated messages and implementing residual connections.

While the equivariance of the network implies that the its predictions do not depend on the rotation or translation of the input, it does not guarantee them to be zero mean. Therefore we follow the common practice of enforcing the predicted noise to have zero center of gravity (CoM), avoiding accumulation of drift during the diffusion process.

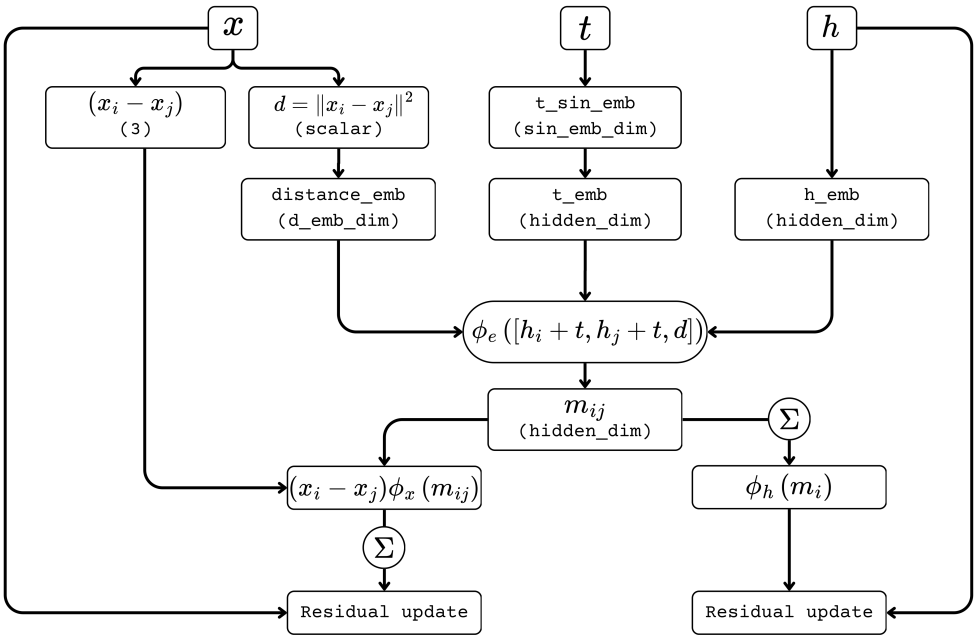

Figure 1: *Diagram of a message passing update of the implemented EGNN. Σ represent message aggregation.*

3 Model structure and training setup

The architecture of the DDPM follows closely the proposed by Ho et al. [2], therefore datapoints are diffused for $T = 1000$ timesteps using a linear noise schedule which increases from a 10^{-4} to 0.02. For all experiments the DDPM structure is not modified.
The noise network is an EGNN taking as input the nodes feature h, their coordinates x and the conditioning timestep t. The message passing updates are predicted following closely Satorras formulation as described in Figure 1. The model used follows Eq. 12 for updating features.
A small deviation from the original implementation is the use of fully connected graphs for message passing.

All the models are trained for 1000 epochs on a representative subset of the QM9 dataset (Figure 2) of 10000 samples with a batch size of 128. Optimization is carried out using AdamW, with a learning rate which is warmed up for 50 epochs up from 10^{-5} to 10^{-4} and then decreased until the end of training using cosine annealing.

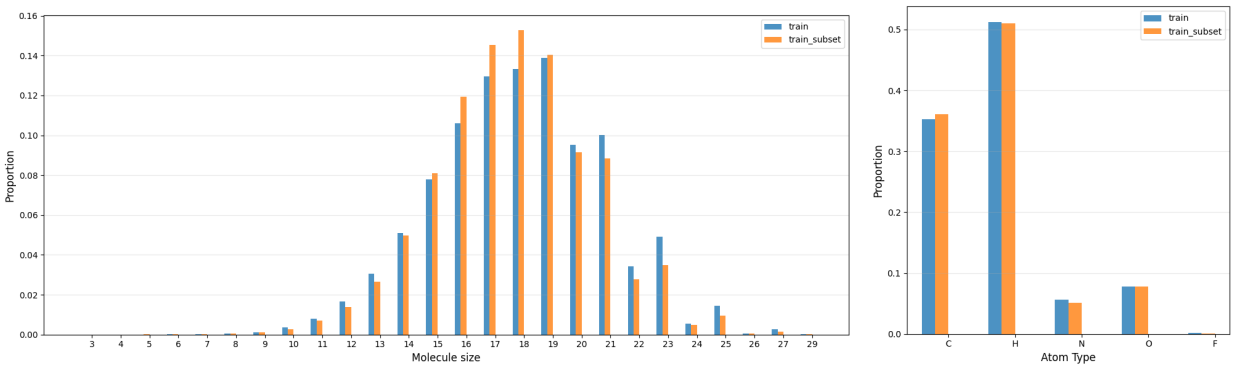

Figure 2: *Comparison between the full QM9 dataset and the subset used for the experiments*

4 Metrics

The model performance is monitored using the metrics considered by Cornet et al. [5], from which I also use the approach for bond inference based on the look-up table used by Hoogeboom et al.[1]. The metric are define as follows:

- **atom stability**: an atom is considered stable if it has 0 charge.
- **molecule stability**: a molecule is considered stable if all of its atoms are stable.
- **molecule validity**: a molecule is considered valid if it can be successfully sanitized by rdkit [6].
- **molecule uniqueness**: the fraction of all generated samples that are valid and have a unique SMILES string [7].

During the experiments, the focus is put mainly on atom stability and molecule validity as they provide a robust reference for the quality of the generated samples at atomic scale and molecule scale. I gave priority to molecule validity over stability as the latter is significantly stricter given and a single unstable atom compromises the entire molecule, therefore making the proportion of stable molecules be relatively constant until high atom stability is reached.
Metrics are evaluated on 3 sampling runs of 1024 samples each.

5 Experiments on model structure

During the implementation of the baseline model I explored changes to the implementation itself. These experiments did not follow the same rigor of those carried out on the hyperparameters (for example the testing is done on a single sampling run). I include them anyway as were essential in understanding how the model works.

5.1 Time conditioning

The first implementation performed conditioning by adding the time embeddings to the feature embeddings only once before the message passing rounds. While the model was still capable of learning, changing to adding the time conditioning at the beginning of every rounds of updates resulted in an improvements from 0.800 to 0.850 in atom stability and from 0.329 to 0.382 in molecule validity. This is because the influence of the conditioning signal on the updates becomes weaker in further rounds slowing down the model training.

5.2 Coordinate update normalization

In the paper from Satorras et al. [3], an tanh activation is applied to the output of ϕ_x, bounding the scalar weights as this stabilizes the updates preventing them from assuming large values. In my tests I saw that this hold true, without the tanh activation another form of normalization to the aggregates is needed to stabilize the model. This is especially true with an higher number of message passing rounds as the updates explode and often leads to the training crashing due to NaN values. The baseline model therefore uses the tanh activation.

6 Experiments on hyperparameters

The bulk of the experiments consist in altering the following hyperparameters one at a time: dimensionality of messages, number of message passing layers, embedding size of the distance between nodes, and the weight of the feature and coordinate loss.

6.1 Baseline model

The baseline model has 5 message passing layers with hidden dimension $= 256$. The sinusoidal embeddings of the timestep is of dimension 64, while the distance between atoms is 32. Feature and coordinate losses are weighted equally at 0.5. This results in a 0.850 ± 0.003 stable atoms and 0.388 ± 0.004 valid molecules.

6.2 Model width

As shown in Figure 3, incrementing the dimensionality of the hidden node representation and messages has a significant impact on the quality of the samples, with a clear and unsurprising positive trend of larger models performing better. An interesting detail is that model scaling provides diminishing return on the fraction of stable atoms (which is a common behaviour in many ML tasks), however, even small improvements in this fraction lead to substantial improvements in the validity of the generated molecules. This is compatible with the fact that the unstable atoms are equally distributed among the molecules, which are then classified as non-valid just for a single unstable atom.

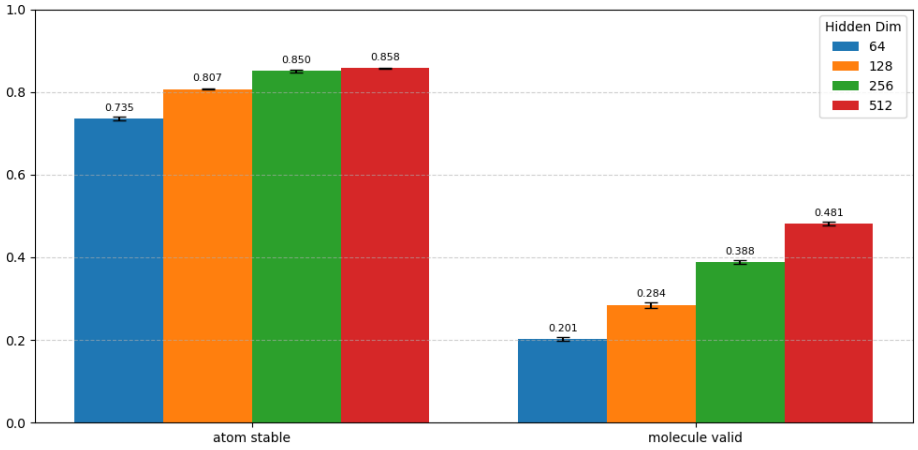

Figure 3: *Performance comparison across different hidden dimensionalities.*

Table 1 highlights that while growing model width increases the total parameters linearly by a factor of 4, training times seems to rise exponentially. This suggest that this approach to achieve better performance does not scale well to larger networks.

Table 1: *Model size comparison with increasing model width*

Hidden Dimension	Model Parameters	Training Time (minutes)
64	143,594	47.64
128	540,010	64.98
256	2,102,890	97.08
512	8,251,498	239.70

6.3 Model depth

A different approach to increasing model complexity is tuning the number of message passing rounds. The specific number depends on the number and diversity of nodes in graph, as too many updates result in oversmoothing, when all nodes representation converge to the average. While too few rounds fail to provide meaningful updates to the node representations, as the network isn't able to efficiently aggregate information from neighbouring nodes .

A problem faced when experimenting with higher number of message passing rounds (9-12) consisted in the model not improving with more rounds, finding the behaviour counterintuitive, I discovered an issue similar to what is described for coordinates updates (subsection 5.2), in this case is the output of ϕ_h which is unbounded. Instead of applying a bounding activation on the output of the network, I found that removing the network all-together and using the normalized aggregates directly for the residual update stabilized training and allowed the model to actually benefit from the added complexity.

This change is not implemented in the baseline and models from the other experiments, as this optimization was only found late in testing. Still, taken in a vacuum, these experiments provide valuable insights on the effect of deepening the EGNN. In addition, comparing the baseline with the model having 5 message passing layers, the results show that the absence of ϕ_h only has a minimal influence on the performances.

Figure 4 shows that the number of message passing rounds has a impact on performance superior to making the model wider. Importantly, despite working on fully connected graphs, even with 12 layers, no negative effects of oversmoothing are present, instead the model continues to improve.

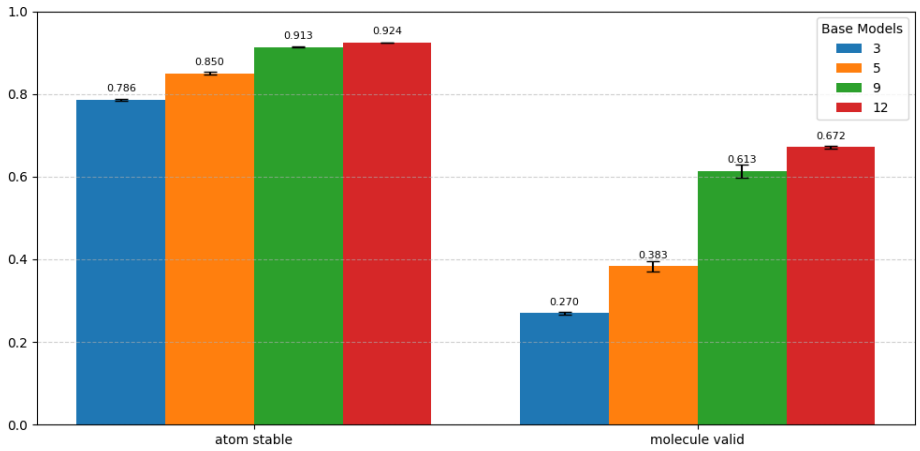

Figure 4: *Performance comparison across different number of message passing layers.*

Comparing Table 2 with Table 1, scaling model complexity by adding layers seems to be a better approach than increasing layer width as the number of parameters and training time both rise slower.

Table 2: *Model size comparison with increasing number of message passing rounds*

Hidden Dimension	Model Parameters	Training Time (minutes)
3	$1,297,514$	56
5	$1,773,930$	97
9	$3,318,890$	159
12	$4,329,578$	210

6.4 Node distance embedding

The node distance embedding is directly concatenated to the feature embeddings of the connecting nodes to build the input to the MLP producing the messages (it can be interpreted as an edge attribute). Figure 5 shows that the optimal embedding dimensionality is 16 ($\sim 1/16$ of the feature embedding). Larger embeddings gradually degrade the quality of the samples indicating that overparametrizing the distance shifts focus away from the feature embeddings, worsening training. The opposite happens with 1d embeddings, in this case atom distances have too little influence on the messages resulting in worse quality samples.

6.5 Loss weights

The loss weights ultimately govern how the network train. Since denoising atomic coordinates is a substantially more difficult task than to denoising atom types due to the much larger complexity of the possible coordinates space, I expected that assigning greater importance to the coordinate loss would would improve performance.

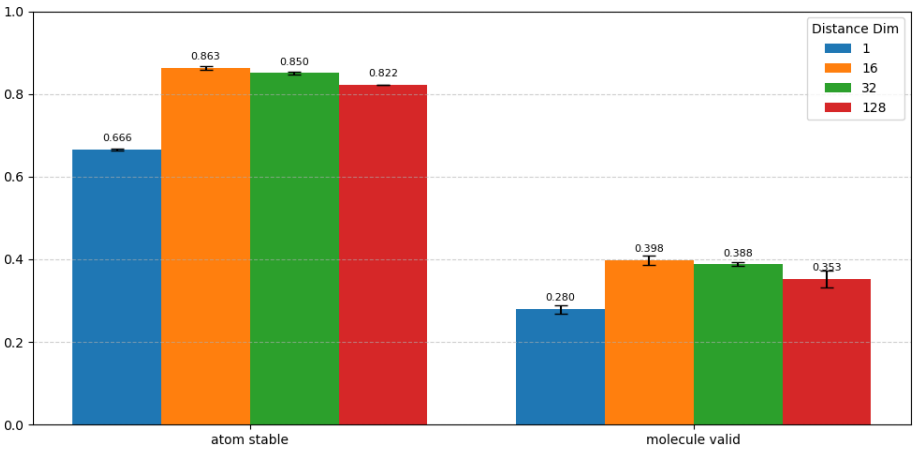

Figure 5: *Performance comparison across different atom distance dimensionality.*

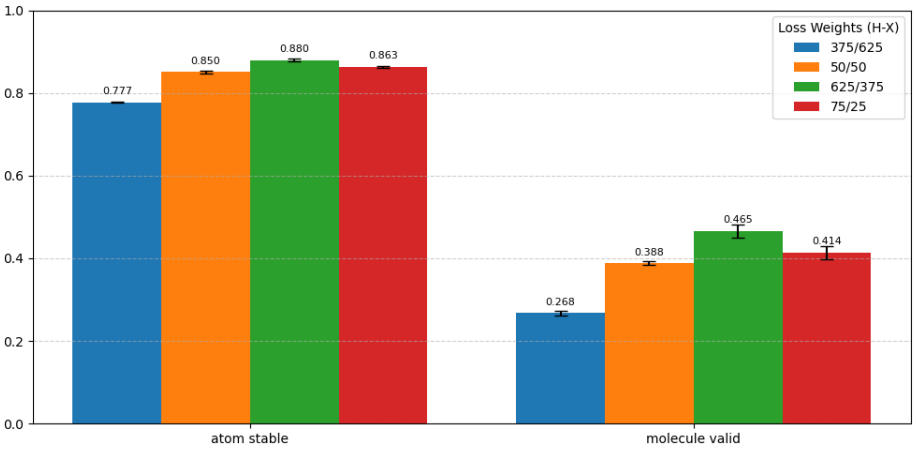

Figure 6: *Performance comparison across different loss weighting.*

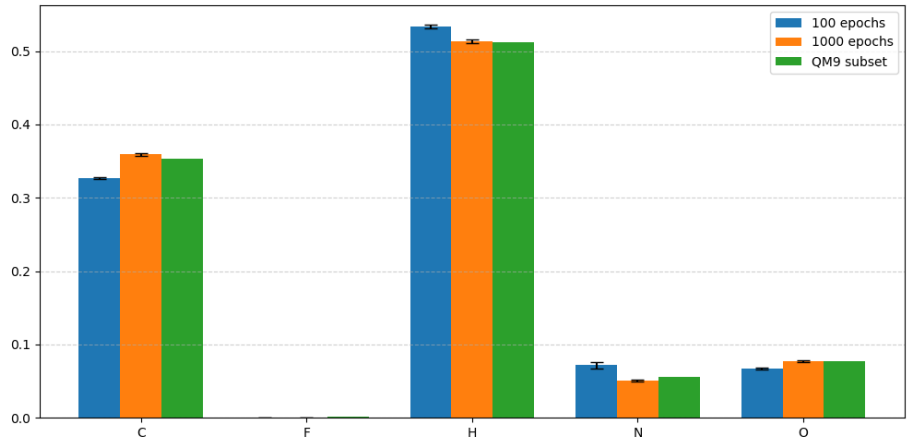

Figure 7: *Distribution of generared atoms for the 625/375 model after 100 and 1000 epochs.*

The experiments in Figure 6 show the opposite: focusing on the coordinate loss degrades drastically the samples quality, while the best results are obtained by weighting the loss on the atom types more heavily. I hypothesise the reason for this behaviour is that allowing the model to accurately learn to denoise atom types early in training provides a solid base for the model to optimise the coordinates denoising. This is also suggested by the fact that the distrbution of generated atoms matches closely that of the training set after just 100 epochs as reported in Figure 7.

7 Comparison with other EGNN architecture

As source of comparison, I trained the same DDPM from the experiments, but equipped with the EGNN architecture inspired by Hoogeboom et al. [2]. This implementation first updates the features using the same approach utilized by Satorras, then updates the coordinates with ϕ_x taking as input the concatenation of the already updated features and edge attributes. The edge attributes include both the normalized distance between nodes before the first message passing round and the normalized distance between nodes at the current round which are embedded using 12d sinusoidal embeddings. In addition, the feature updates are made on the base of weighted aggregates, where each message is multiplied by scalar predicted by an attention network.

The same boundary using tanh is applied to the scalar weights, but the scale of the updates are not given by the coordinate difference (as they are also normalize) but by multiplying by predefined coordinate range, which defines the space boundary within the atoms reasonably stay. Conditioning is also handled differently, as the raw timestep is simply concatenated to the node features before any processing.

I trained a final model using the best hyperparameters found, while keeping the model capacity comparable to the Hoogeboom implementation. This corresponds to 256 as hidden dimension, 9 message passing layers, 16 as distance projection dimension, and loss weighted as 0.635 for features and 0.375 for coordinates.

Table 3: *Performance comparison between models*

	Stability (↑)		Val. / Uniq. (↑)	
Model	A [%]	M [%]	V [%]	V×U [%]
Data	99.0	95.2	97.7	97.7
EDM (Hoogeboom, reported by Cornet et al. [5])	98.7	82.0	91.9	90.7
DDPM (with Hoogeboom EGNN)	96.1	53.1	86.6	86.6
DDPM (with tuned EGNN)	$95.1_{\pm.0}$	$46.7_{\pm.9}$	$81.0_{\pm.8}$	$80.8_{\pm.7}$

From the results reported in Table 3, the DDPM equipped with Hoogeboom implementation of the EGNN performs significantly worse than what is reported in the literature. This deviation is most likely due to different diffusion dynamics and training approaches, for examples in the original paper is used a learned noise schedule, a different initial learning rate, and different annealing schedule from the one I utilized.
As expected the stability of the molecules is a much stricter metric compared to validity. Additionally, it can be seen how large improvements in stability are given by small in small increases in atom stability: from the tuned EGNN to the Hoogeboom EGNN, a 1% improvement in stable atoms increases the molecule stability by 6.4%, while the 2.6% from the Hoogeboom EGNN and the EDM results in a massive 28.9% jump.
Nonetheless, the sample quality achieved by the two EGNN implementations prove that both models are capable of learning effectively to generate chemically valid and unique molecules.

8 Conclusions and future directions

This project has taught me how to implement from foundation principles a diffusion model for molecule generation, taking into consideration the geometrical constraints that needs to be satisfied by the network when working with molecular data. The experiments that were carried out provide a strong understanding of which factors determine the final performance of the model, which will be essential for working with this type of models in the future.

By looking at other implementations, I had the opportunity to see how many different implementation details can be changed, even outside hyperparameters. Some examples are the different conditioning methods, where and how to apply normalization or setting output boundaries to prevent instability.

There are many slight modification which I didn't have time to test. The results from the loss weighting experiments suggest that splitting training into two phases, or at least have an initial training stage where the only the atom types are diffused could improve the sample quality.
The experiments on model width and depth both indicate that model capacity is a constrain on the quality of the samples, though depth seems to be more crucial, which poses questions such as how would a narrow but very deep network perform, and what is the upper bound to the number of message passing updates before oversmoothing becomes an issue.

Another large part which I did not explore is changing the diffusion dynamics. For example trying different noise schedules as Hoogeboom et al. [1] experiment with polynomial schedules and also a learnable schedule.

Code availability

The code and data utilized to perform the experiments are available in this github repository.

References

1. Hoogeboom, E., Satorras, V. G., Vignac, C. & Welling, M. *Equivariant Diffusion for Molecule Generation in 3D* 2022. arXiv: 2203.17003 [cs.LG]. https://arxiv.org/abs/2203.17003.
2. Ho, J., Jain, A. & Abbeel, P. *Denoising Diffusion Probabilistic Models* 2020. arXiv: 2006.11239 [cs.LG]. https://arxiv.org/abs/2006.11239.
3. Satorras, V. G., Hoogeboom, E. & Welling, M. *E(n) Equivariant Graph Neural Networks* 2022. arXiv: 2102.09844 [cs.LG]. https://arxiv.org/abs/2102.09844.
4. Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O. & Dahl, G. E. *Neural Message Passing for Quantum Chemistry* 2017. arXiv: 1704.01212 [cs.LG]. https://arxiv.org/abs/1704.01212.
5. Cornet, F., Bartosh, G., Schmidt, M. N. & Naesseth, C. A. *Equivariant Neural Diffusion for Molecule Generation* 2025. arXiv: 2506.10532 [cs.LG]. https://arxiv.org/abs/2506.10532.
6. Contributors, R. *RDKit: Open-source cheminformatics* https://www.rdkit.org. Version 2025.09.1. 2025.
7. Weininger, D. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences* **28,** 31–36 (1 1988).