One-Step Generative Modeling with Mean Flows for Molecular Structures

Silpa Soni Nallacheruvu ssnal@kth.se
Tove Nordmark toveno@kth.se
Lena Marie Weyer weyer@kth.se

Abstract

We study one-step generative modeling with Mean Flows under realistic compute constraints and explore its extension to molecular design. Mean Flows learns average velocity fields between two time points rather than instantaneous velocities, enabling high-quality sample generation with a single neural network evaluation. We reimplement the Diffusion Transformer (DiT) backbone in JAX/Flax and reproduce the Mean Flows training pipeline on a 5,000-image ImageNet subset. Through a series of ablation studies, we observe scale-dependent but logically consistent trends relative to the original paper and obtain reasonable 1-NFE ImageNet samples. Beyond replication, we adapt Mean Flows to operate on the latent space of a Junction Tree VAE (JT-VAE), using a molecule-specific DiT to model flows over 56-dimensional molecular latents. One-step sampling in this space produces valid molecules, providing a proof of concept that Mean Flows can be transferred from images to structured scientific data and suggesting potential for efficient molecular design workflows.

1 Introduction

Generative models have improved rapidly in recent years, but many state-of-the-art methods still require dozens or even hundreds of neural network evaluations to produce a single sample. This makes them expensive to run and difficult to scale. Mean Flows aim to solve this problem by offering a way to generate images in just one step. Instead of learning how data changes at a single moment in time, Mean Flows learn an average transformation between two time points, which makes the generation process far more efficient while still maintaining good sample quality.[1]

In this project, we reproduce the core experiments from the Mean Flows paper to understand how various modeling choices affect one-step generation. We evaluate the impact of time-pair sampling, JVP formulation, time embeddings, loss weighting, and classifier-free guidance. To support these ablations, we reimplemented the DiT architecture and paired it with a pretrained ImageNet VAE to match the original setup. Despite using far fewer images and limited compute, our results still logically align with the main trends reported in the paper, strengthening confidence in both our reproduction and the robustness of the method.

A key motivation for our work is exploring whether one-step generative models like Mean Flows could be useful beyond image synthesis. Molecular design often requires generating and exploring large numbers of chemical structures quickly, a setting where fast generation is highly valuable. As an initial step in this direction, we adapted the Mean Flows framework to operate in the latent space of a molecular generative model and produced preliminary molecule samples. This serves as a first exploration of whether the efficiency of Mean Flows can benefit molecular generation tasks.

2 Code repository

The link to the GitHub public repository is: https://github.com/tofsi/mean-flows

HT20 Deep Learning Advanced, DD2412, KTH.

3 Related Work

Modern generative modeling has been shaped by diffusion models and continuous-time flow-based approaches. Flow Matching provides a unified framework for learning time-dependent vector fields, enabling stable training and high-quality generation. Mean Flows [1] extend this idea by predicting average rather than instantaneous velocities through a two-time conditioning formulation and a Jacobian–vector-product objective, allowing high-fidelity image synthesis in a single function evaluation. The original paper conducts extensive ablations on ImageNet, studying the effects of time-pair sampling, positional embeddings, loss metrics, and classifier-free guidance. Our work reproduces these experiments under reduced compute, making this methodology central to our study.

The Diffusion Transformer (DiT) architecture has become a strong backbone for diffusion and flow-based models. We follow the open-source implementation of Peebles and Xie [2] to construct a JAX/Flax DiT compatible with Mean Flows, including patch embeddings, sinusoidal position encodings, and adaLN-Zero conditioning. Image generation quality is evaluated using Fréchet Inception Distance (FID), computed from Inception-V3 features [3], comparing generated samples to the full ImageNet distribution.

Beyond images, generative modeling of molecular structures is an expanding research frontier. Models such as Proteina [4] demonstrate how flow-based methods can learn complex structural manifolds in biomolecular systems. Graph-based molecular generation frequently relies on structured latent spaces, and the Junction Tree Variational Autoencoder (JT-VAE) [5] is among the most widely adopted methods for ensuring chemical validity. We use the open-source JT-VAE implementation[1] [6] to obtain latent representations suitable for flow-based generation.

Building on these foundations, our project explores whether the Mean Flows formulation, originally proposed for large-scale image synthesis can be adapted to molecular latent spaces. By integrating DiT and JT-VAE within the Mean Flow framework, we provide an initial demonstration of one-step molecular generation, connecting image-based flow research to emerging scientific applications.

4 Data

For our replication experiments, we used a reduced version of ImageNet (ILSVRC2012), selecting 5,000 images uniformly across 1,000 classes to fit computational limits. Images were resized, normalized, and converted into latent representations using the pretrained StabilityAI VAE tokenizer (`sd-vae-ft-mse`), consistent with the Mean Flows setup.

For molecular generation, we used the QM9 and ZINC datasets, which provide small organic molecules in SMILES format with atom-level information. These data required conversion into the junction-tree representation needed by JT-VAE. The molecular datasets were not used to train Mean Flows directly; instead, they supplied latent vectors to the JT-VAE pipeline, which we integrated with a Mean-Flows style sampler for molecule generation.

5 Methods

5.1 Mean Flows for One-Step Generation

Mean Flows extend Flow Matching by training a model to predict the *average velocity* between two times (r, t) rather than the instantaneous velocity at time t. For a trajectory z_τ under an unknown field v, the average velocity is $u(z_t, r, t) = \frac{1}{t-r} \int_r^t v(z_\tau, \tau)\, d\tau$.

The key identity enabling training is: $u(z_t, r, t) = v(z_t, t) - (t - r)\, \frac{d}{dt} u(z_t, r, t)$,

which provides a computable target using a Jacobian–vector product (JVP). The JVP evaluates

$\frac{d}{dt} u(z_t, r, t) = \frac{dz_t}{dt} \partial_z u + \frac{dr}{dt} \partial_r u + \frac{dt}{dt} \partial_t u = v(z_t, t) \partial_z u + \partial_t u$

without explicitly forming the Jacobian, where $dz_t/dt = v$, $dr/dt = 0$, and $dt/dt = 1$.

Training therefore minimizes $\mathcal{L} = \mathbb{E}_{(r,t)} \left\| u_\theta(z_t, r, t) - \tilde{u}(z_t, r, t) \right\|_2^2$,

where $\tilde{u}$ is computed from the identity above.

[1] https://github.com/VldKnd/vae_qm9

This formulation directly supports the ablations we study: varying the (r, t) sampling ratio, changing the JVP tangent, adjusting positional embeddings, modifying loss weighting p, and applying classifier-free guidance.

We had modified the adaptive loss weighting by normalizing the weights $w = 1/(\|\Delta\|_2^2 + c)^p$ across each batch (see subsection 13.1).

5.2 DiT Backbone and Integration into Mean Flows

To parameterize the velocity field u_θ, we implemented a compact DiT (Diffusion Transformer) backbone as seen in subsection 13.2.

Integrating DiT into the Mean Flow training loop required modifying the forward pass to accept the two-time conditioning and to support JVP computation inside Algorithm 1 of the Mean Flows paper. Our implementation replicates the full training pipeline, including t, r sampling, velocity target construction, and one-step sampling (Algorithm 2) as explained in Figure 1.

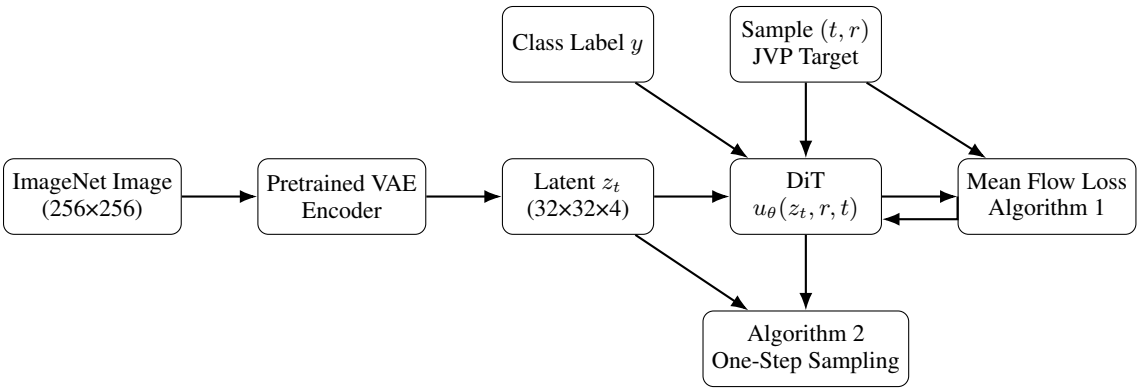

Figure 1: End-to-end pipeline for Mean Flows training and sampling using a DiT backbone and a pretrained VAE tokenizer.

5.3 Molecular Structure Generation

To encode and decode molecular structures, a junction tree variational autoencoder (JT-VAE) was used [6] [7]. The JT-VAE was trained following a two-stage protocol: first, a pretraining stage of three epochs without KL-divergence regularization ($\beta = 0$) to establish an initial latent structure, followed by a variational training stage of fifteen epochs with full KL regularization ($\beta = 0.005$). Hyperparameters were adopted from the original publication [6],[7] and detailed information about hyperparameters and training is provided in the Appendix (see subsection 13.6).

5.4 Integration of JT VAE into Mean Flows

After training, the JT-VAE latent space was combined with a Mean-Flow style sampler to explore molecular generation, forming an integration between the two frameworks.

To extend Mean Flows from images to molecules, we replaced the ImageNet DiT backbone with a molecule-specific DiT operating directly in the latent space of a pretrained JTNNVAE. Each molecule is first encoded into a 56-dimensional latent vector (28 dimensions tree + graph), which we reinterpret as a short sequence of two tokens using a small embedding network. We then add fixed 1D sinusoidal positional embeddings and condition the transformer only on time (no class labels), using the same multi-time embedding and adaLN-Zero conditioning as in the image model.

The molecular DiT predicts an updated latent vector of the same dimension, which is trained with the Mean Flows Algorithm 1 in exactly the same way as for images. For sampling, Algorithm 2 is applied in JTNNVAE latent space, and the resulting latent vectors are decoded back to SMILES strings with the original JTNNVAE decoder, yielding valid molecular structures generated by a one-step mean flow model.

6 Experiments and findings

6.1 Ablation Studies

Using the hyper parameter settings listed in subsection 13.3, we ran ablation studies to see how different parts of the model influence one-step performance as is shown in Table 1.

Table 1: Ablation study on 1-NFE ImageNet 256×256 generation. **FID-1K** is reported. Default configs are light gray. Best FID in bold.

(a) Ratio $r \neq t$

% $r \neq t$	FID
0% (=FM)	131.94
25%	126.31
50%	121.98
100%	**114.48**

(b) JVP computation

Tangent	FID
$(v, 0, 1)$	**126.31**
$(v, 0, 0)$	138.45
$(v, 1, 0)$	138.15
$(v, 1, 1)$	139.02

(c) Positional embedding

Embedding	FID
(t, r)	127.35
$(t, t-r)$	126.31
$(t, r, t-r)$	**125.86**
$t-r$ only	131.08

(d) Time samplers

Sampler	FID
uniform(0,1)	126.76
lognorm$(-0.2, 1.0)$	122.03
lognorm$(-0.2, 1.2)$	127.78
lognorm$(-0.4, 1.0)$	126.31
lognorm$(-0.4, 1.2)$	**119.56**

(e) Loss metrics

p	FID
0.0	128.99
0.5	**124.90**
1.0	126.31
1.5	130.47
2.0	128.25

(f) CFG scale

ω	FID
1.0 (no cfg)	126.31
1.5	**118.39**
2.0	129.94
3.0	128.69
5.0	129.33

From Flow Matching to Mean Flows:

Table 1a shows how performance varies with the proportion of $r \neq t$ samples. The 0% case reduces to standard Flow Matching, which performs worst (FID 131.94), confirming that instantaneous velocities alone are insufficient for effective one-step generation. FID improves monotonically as the $r \neq t$ fraction increases, with 100% achieving the best result (FID 114.48). This suggests that under our training regime, the model benefits more from learning average velocities across diverse time intervals than from mixing in instantaneous velocity information, which may introduce noise without sufficient data to learn precise estimates.

JVP Computation:

Table 1b demonstrates the critical importance of correct Jacobian–vector product computation. The proper tangent $(v, 0, 1)$ corresponds to the time derivative decomposition $(\partial_z u,\ \partial_r u,\ \partial_t u)$ from subsection 5.1. All three incorrect variants produce significantly worse FID, with performance degrading by approximately 10%. This confirms that even small deviations from the theoretically correct JVP formulation corrupt the velocity field estimates and severely impact generation quality.

Conditioning on (r, t):

Table 1c shows that the model is robust to different positional embedding schemes, with all variants achieving reasonable performance. Using all three terms $(t, r, t-r)$ performs marginally best, followed closely by $(t, t-r)$. Embedding only the interval $t-r$ performs worst, suggesting that absolute time information is valuable. The small differences indicate that as long as both time variables are encoded, the specific representation matters less than their presence.

Time Samplers:

Table 1d confirms that the time-sampling distribution significantly affects training. Logit-normal samplers consistently outperform uniform sampling, with lognorm$(-0.4, 1.2)$ achieving the best result. These distributions bias sampling toward intermediate and later times, which may provide

more informative training signals for learning the average velocity field. The sensitivity to sampler parameters ($\sigma = 1.0$ vs. 1.2) suggests this was a critical hyperparameter in reduced training dataset.

Loss Metrics:

Our normalized adaptive weighting shows clear sensitivity to the power parameter p (Table 1e). Standard MSE ($p = 0$) performs reasonably, but modest adaptive weighting ($p = 0.5$) improves performance, likely by allowing the model to focus more on challenging samples without completely ignoring easy ones. Higher values ($p \geq 1.5$) degrade performance, suggesting excessive downweighting of large errors causes the model to neglect important learning signals. The narrow optimal range around $p = 0.5$–1.0 indicates that balanced weighting is crucial under our training constraints.

Guidance Scale:

Table 1f shows that CFG substantially improves generation quality, with $\omega = 1.5$ achieving the best FID (118.39)—a 6% relative improvement over no guidance (FID 126.31). However, stronger guidance ($\omega \geq 2.0$) degrades performance. This suggests our model has learned a useful but noisy distinction between conditional and unconditional predictions. Moderate amplification strengthens class-specific features, but excessive amplification ($\omega \geq 3.0$) magnifies noise, likely because our limited training data (5 images per class) prevents learning sharp class boundaries.

Scalability:

Across model sizes (B/2, M/2, L/2), FID remains stable for 7–8 epochs before diverging sharply (Figure 2), with larger models degrading earlier and more severely. This pattern indicates overfitting: with only 5,000 training images, larger models quickly memorize the dataset rather than learning generalizable flows. Since FID is evaluated against the full ImageNet distribution, this train-test distribution mismatch becomes increasingly penalized. The instability demonstrates that under severe data constraints, model capacity must be carefully matched to dataset size—a regime where standard scaling laws break down.

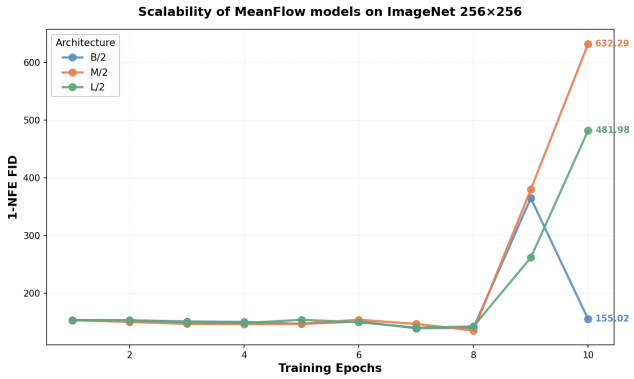

Figure 2: Scalability of Mean Flow models (DiT-B/2, M/2, L/2) on Imagenet 256 × 256

6.2 Imagenet Generation

Using the combined best settings (listed in subsection 13.4) from our ablation studies, we trained a final model for 30 epochs to inspect one-step generated samples. The resulting FID of 120.22, slightly worse than the best isolated FID of 114.48, suggests that these hyperparameters interact in non-additive ways and that individually optimal choices do not necessarily form an optimal joint configuration. As shown in Figure 3, the generated images lack fine details and sharp structure but still capture coarse shapes, colour patterns, and approximate object layouts. This indicates that, even with highly reduced training, the model learns a meaningful one-step mapping from noise toward ImageNet-like images.

6.3 Molecular Structure Generation

The molecular DiT model (see subsection 13.5) was trained for 90 epochs on a 90/10 train/val split of the encoded data set. A training curve is shown in the appendix (see subsection 13.7). Although

Figure 3: Comparison between random batches of real ImageNet images (left) and our one-step generated samples (right) with FID-1K = 120.22

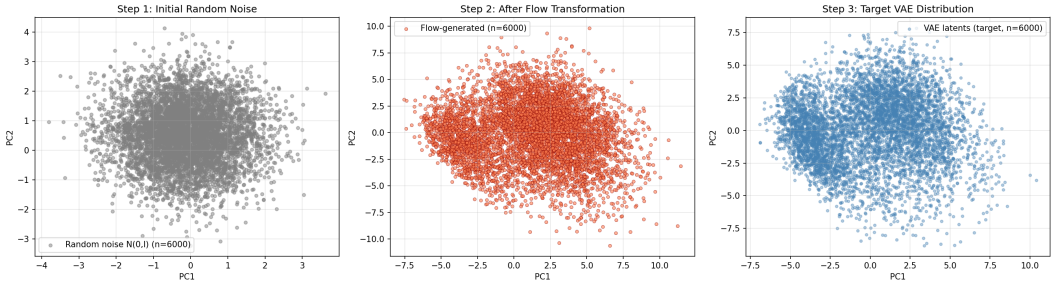

Figure 4: Plots of priors and latent spaces (n = 6000) after applying PCA. By sampling from the $N(0, I)$ prior (left), by generative modeling using the trained mean-flow model (middle) and by sampling from the data set and encoding the molecules (right).

the JT-VAE achieved only 22% exact reconstruction accuracy, molecules generated by the trained MeanFlow model were chemically valid with 99.6% accuracy, which is slightly higher than the 99.2% validity of samples drawn directly from the prior (see subsection 13.8). This indicates that one-step sampling in latent space produces structurally sound outputs.

To evaluate how well the model approximates the target distribution, we visualized the latent space using principal component analysis, and compared the trained model with the prior and the encoded data set. The resulting plots in figure 4 and figure 8 (in subsection 13.10) clearly demonstrate that the model learns to transform the prior towards the encoded data distribution. We further compared the generated molecules (see Figure 5) with those drawn from the standard Gaussian prior $\mathcal{N}(0, I)$ and with samples from the encoded dataset (see subsection 13.9).

Overall, these results show that the MeanFlow model successfully learns meaningful structure in the JT-VAE latent space and can guide samples toward the target molecular distribution. Although the molecular design capabilities are not yet fully optimized, these results demonstrate that one-step generative flows can produce chemically sensible outputs, which is an encouraging sign for future extensions toward larger and more complex molecular spaces.

7 Challenges

Reproducing Mean Flows required several practical and technical adjustments. While the paper describes the core method clearly, many implementation details, particularly for the DiT architecture,

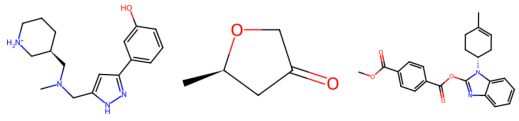

Figure 5: Example of three molecules generated using the trained model.

JT-VAE preprocessing, and FID evaluation, had to be inferred from various citied authors' code. Integrating these components into a unified JAX/Flax pipeline was nontrivial, especially given limited compute, which constrained batch sizes, training length, and scalability experiments. Interpreting ablation behaviour was also challenging, as the original work provides limited explanation for certain trends, making justification under our reduced-training regime more difficult.

8 Conclusion

Our study reproduces the main experimental findings of the Mean Flows paper and shows that its design choices are not optional: each ablated component noticeably influences one-step generation quality. Despite operating at smaller scale, many trends align with the original results, confirming the method's stability and sensitivity to its core ingredients such as correct JVP computation, time-pair sampling, and positional embeddings.

Beyond reproduction, we demonstrated that Mean Flows can be adapted beyond images. By integrating a molecule-specific DiT with a JT-VAE latent space, we produced valid molecular structures in a single sampling step. This preliminary exploration highlights that the Mean Flow framework can operate on structured scientific latent spaces, opening a path toward efficient molecular design. A natural next step is extending this approach to larger chemical spaces or to protein backbones, where fast, one-step generative models could have significant practical impact.

9 Ethical consideration, societal impact, alignment with UN SDG targets

Generative models raise ethical concerns related to misuse, biased outputs, and limited transparency about how synthetic data is produced. These issues apply to Mean Flows as well. Molecular generation represents a domain where the consequences are more severe, while such models offer promise for expediting drug discovery, they generate unsafe or biologically harmful molecules when deployed without sufficient control mechanisms. This makes expert oversight and careful evaluation essential. Despite these risks, the potential societal benefits are substantial. Faster molecular and protein design can support better healthcare and scientific innovation, aligning strongly with UN SDG 3 (Good Health and Well-Being) and SDG 9 (Industry, Innovation, and Infrastructure).

10 Limitations

The scope of this study is mainly limited by compute and time. To fit available hardware, we reduced ImageNet from 1.2M images to a 5,000-image subset, lowered the batch size from 256 to 64, and trained for only 10 instead of 80 epochs. The modification of normalizing weights in adaptive loss modified the behavior of our mean flows algorithm. These adjustments make our ablation results less directly comparable to the original Mean Flows paper, especially for components that benefit from long training. We also were unable to perform scalability experiments for larger DiT variants (e.g. DiT-XL/2) or retrain the B/2, M/2, and L/2 models with additional regularization due to resource limits. Additionally, the training instability observed at around 35 epochs of the molecular structure model (Figure 6) raises concerns about the sensitivity of the method to hyperparameter selection.

11 Self Assessment

Our project successfully reproduced the main ablation studies from the Mean Flows paper, and even with reduced compute, our results were logically consistent with the main experiment. We also replicated the scalability experiment with predictable behavior. Beyond replication, we extended the method to a new task by adapting DiT and Mean Flows to molecular latent spaces, generating valid molecular structures using JT-VAE. Additionally, we implemented full 1-NFE image generation and included visual samples to support our evaluation. Furthermore, we were able to verify that the methods were successfully extended to the realm of molecular structure data by showing that the trained model was able to transform the prior toward the encoded data distribution.

Given the fidelity of our reproductions, the technical complexity of our implementation, and our meaningful extension to molecular generation, we believe that this project meets the requirements for an **A** grade.

12 Declaration of AI

We used generative AI tools to refine and streamline our wording, for proofreading, stylistic editing, discussing ideas, and as a coding assistant. Authors take full responsibility for all the conceptual work, experimental design, implementation decisions, analyses, and conclusions.

References

[1] Zhengyang Geng et al. *Mean Flows for One-step Generative Modeling*. 2025. arXiv: 2505.13447 [cs.LG]. URL: https://arxiv.org/abs/2505.13447.

[2] William Peebles and Saining Xie. *Scalable Diffusion Models with Transformers*. 2023. arXiv: 2212.09748 [cs.CV]. URL: https://arxiv.org/abs/2212.09748.

[3] Christian Szegedy et al. *Rethinking the Inception Architecture for Computer Vision*. 2015. arXiv: 1512.00567 [cs.CV]. URL: https://arxiv.org/abs/1512.00567.

[4] Tomas Geffner et al. *Proteina: Scaling Flow-based Protein Structure Generative Models*. 2025. arXiv: 2503.00710 [cs.LG]. URL: https://arxiv.org/abs/2503.00710.

[5] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. *Junction Tree Variational Autoencoder for Molecular Graph Generation*. 2019. arXiv: 1802.04364 [cs.LG]. URL: https://arxiv.org/abs/1802.04364.

[6] Vladimir Kondratyev et al. *Generative model based on junction tree variational autoencoder for HOMO value prediction and molecular optimization*. 2023. URL: https://doi.org/10.1186/s13321-023-00681-4.

[7] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. "Junction Tree Variational Autoencoder for Molecular Graph Generation". In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 2323–2332. URL: https://proceedings.mlr.press/v80/jin18a.html.

13 Appendix

13.1 Adaptive Loss with normalized weights

Our initial reasoning was that when $p = 1$, the weighted loss $w \cdot \|\Delta\|_2^2$ becomes approximately constant across all samples, potentially making the loss uninformative for monitoring training progress. Normalizing the weights ensures they maintain a consistent scale across different p values while preserving their relative importance between samples within each batch.

We later realized that constant loss values are not problematic since the gradients which drive optimization still vary appropriately. Nevertheless, we found that weight normalization appeared to stabilize training dynamics. However, when we tested training with unnormalized weights, the final FID improved by approximately only 5 points, 120.88 in comparison to 126.31, suggesting this modification has minimal impact on final performance.

We acknowledge this modification makes our absolute FID values and optimal p not directly comparable to the original paper. However, we maintained normalized weights throughout all experiments for consistency.

13.2 DiT Architecture

The model embeds VAE latents via a patch-embedding layer, adds fixed 2D sinusoidal positional encodings, and incorporates time and label conditioning through learned embeddings. Each transformer block uses adaLN-Zero modulation to inject (t, r, y) into both the attention and MLP paths. A final projection and unpatchify step maps tokens back to the VAE latent space, producing $u_\theta(z_t, r, t)$ for use in the Mean Flows loss.

13.3 Hyperparameters for DiT architecture

We have used the same hyper parameters as mentioned in the Appendix of [1] as shown in Table 2. The training computation details are: NVIDIA L4 GPU, g2-standard-16 (16 vCPUs, 64 GB Memory) and a CPU, n2-standard-32 (32 vCPUs, 128 GB Memory).

Hyperparameter	Value
Architecture	DiT-B-4
Epochs	10
Learning rate	1×10^{-4}
Adam β_1	0.9
Adam β_2	0.95
EMA decay	0.9999
p	1.0
CFG scale ω	1.0
Ratio $(r \neq t)$	0.25
JVP tangent	$(v, 0, 1)$
(t, r) embedding	(t, r)
Time sampler	Logit-normal $(-0.4, 1.0)$
seed	42

Table 2: Hyperparameters used for DiT-B-4 training.

13.4 Hyperparameters for the best model for 1-NFE generated samples

We trained a model using the best-performing configuration from each ablation study as shown in Table 3. This combined configuration achieved FID 120.22 after 30 epochs, compared to optimal FID 114.48 obtained during the ratio ablation with default settings for other hyperparameters. This suggests that, for instance, stronger CFG ($\omega = 1.5$) may require different loss weighting than our optimal $p = 0.5$, or the aggressive 100% $r \neq t$ sampling may benefit from different time distributions. Additionally, the longer training (30 vs. 10 epochs) may have induced overfitting, as observed in Figure 2.

Hyperparameter	Value
Architecture	DiT-B-4
Epochs	30
Learning rate	1×10^{-4}
Adam β_1	0.9
Adam β_2	0.95
EMA decay	0.9999
p	0.5
CFG scale ω	1.5
Ratio $(r \neq t)$	1.00
JVP tangent	$(v, 0, 1)$
(t, r) embedding	$(t, r, t - r)$
Time sampler	Logit-normal $(-0.4, 1.2)$
seed	3456

Table 3: Hyperparameters used for the best model DiT-B-4 training.

13.5 Hyperparameters for Molecular DiT training

The architecture "Mol-DiT-B" of the generative model used for molecular structures was a transformer of depth 8 with dimensionality of the hidden layers equal to 512, and with 8 heads. The hyperparameters for training are shown in table 4. These were chosen to be similar to the default hyperparameters used in [1]. The batch size was 8192, chosen based on available virtual memory.

Hyperparameter	Value
Architecture	”Mol-DiT-B”
Epochs	90
lr	$5 \cdot 10^{-}5$
Adam β_1	0.9
Adam β_2	0.99
EMA decay	0.9999
p	0.0
Ratio ($t \neq r$)	0.25
JVP Tangent	$(v, 0, 1)$
(t, r) Embedding	$(t, t - r)$
Time sampler	lognorm$(-0.4, 1.0)$

Table 4: Hyperparameters used for Mol-DiT-B-4 training.

13.6 JT-VAE training

13.6.1 Hyperparameters for JT-VAE training

For molecular generation, the raw dataset could not be processed directly on GPU, requiring all JT-VAE preprocessing to be performed on a dedicated CPU machine (see subsubsection 13.6.2).

Hyperparameter	Value
Hidden size	450
Latent size	56
Message passing depth	3
Batch size	64
Optimizer	Adam
Initial learning rate	10^{-3}
LR scheduler	Exponential decay ($\gamma = 0.9$)
Stereo modeling	Enabled

Table 5: Hyperparameters used for JT-VAE training.

13.6.2 Training computation details

Training of the Junction Tree Variational Autoencoder (JT-VAE) was conducted on a GPU machine (Google Cloud `g2-standard-4`, 4 vCPUs, 16 GB RAM, equipped with one NVIDIA L4 GPU). The training procedure followed the two-stage protocol introduced by Jin, Barzilay, and Jaakkola.

All preprocessing steps were performed on a separate CPU machine (Google Cloud `e2-highmem-8`, 8 vCPUs, 64 GB RAM, x86/64 architecture).

13.6.3 Details for training the JT-VAE

To encode and decode molecular structures a junction tree autoencoder was used. Junction-tree representations of the molecules are required by the JT-VAE to process molecules. In the given data smiles representations are used. Different to the original papers all molecules were first converted from their SMILES representation into a MolTree object (Junction-tree representation)together and not separate to speed up the computations. Because loading the full dataset simultaneously exceeded available system memory and resulted in process termination, the dataset was shuffled and partitioned into 16 chunks. This allowed training to be performed on one chunk at a time without exceeding RAM limits. In addition, a new vocabulary file was generated from the entire dataset, as the vocabulary provided in the original repository did not include all molecular substructures.The JT-VAE was trained in two stages. First, a pretraining stage of three epochs was performed without applying the KL-divergence term ($\beta = 0$), enabling the encoder and decoder to learn an initial latent structure. This was followed by a variational training stage of fifteen epochs with full KL regularization ($\beta = 0.005$), during which the model was trained chunk-wise on the GPU.The hyperparameter

settings reported in the original JT-VAE publication [6],[7] were used to train JT-VAE as well (see subsubsection 13.6.1).

13.7 Training Curve for Molecule Structure Generative Model

The training curve for the molecule structure generative model shown in figure 6 shows a decreasing loss, for both training and validation sets. However, training seems to momentarily destabilize at around 35 epochs. The reason for this is unclear, but possible reasons include a poorly chosen learning rate or β_2 leading to overshooting and instability in weight updates.

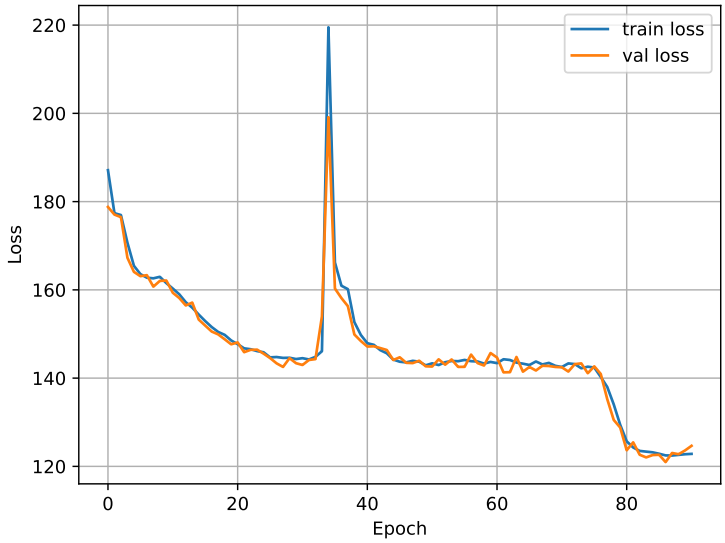

Figure 6: Training curve for the molecular structure generative model.

13.8 Ability of Models to Reconstruct Valid Molecules

JT-VAE achieved 22% accuracy, calculated as the ratio of perfectly reconstructed molecules to total molecules. However, as can be seen in table 6, the VAE was able to decode a vast majority of latent samples into molecules found in the dictionary, not only from the data set or the trained model, but also from the prior.

data set:	499 / 500 (99.80%)
prior:	496 / 500 (99.20%)
model:	498 / 500 (99.60%)

Table 6: Table showing percentage of a sample of 500 molecules from the encoded data set, the $\mathcal{N}(0, I)$ prior, and the trained model, that get decoded into any valid molecules present in the dictionary.

13.9 Images of Molecule Samples

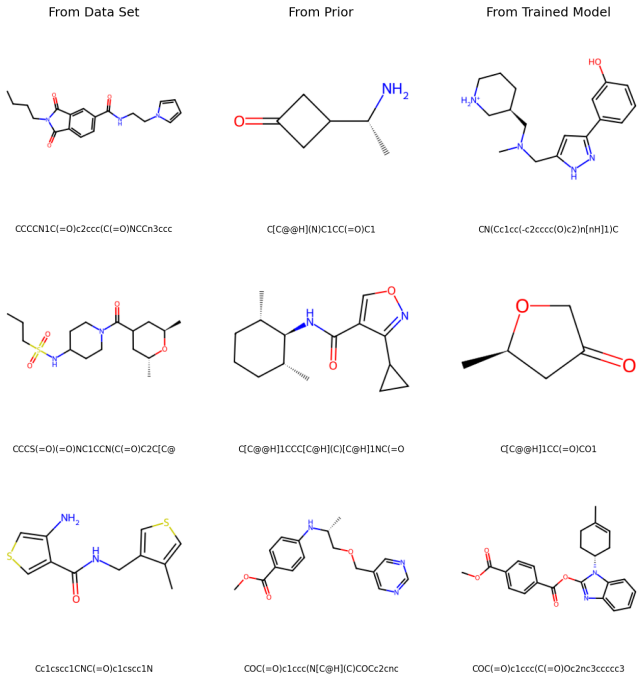

Figure 7: Examples of molecules, generated by sampling from the data set (left), by sampling from the $N(0, I)$ prior (middle) and by generative modeling using the trained model (right).

13.10 Latent space distributions

PCA was applied to the 56-dimensional JT-VAE latent vectors to project both the original VAE latents and the flow-generated latents into a shared 2D space, enabling direct visual comparison of their distributions. By examining these projections and their density histograms, we assess how closely the flow model reshapes noise toward the structure of the true molecular latent manifold.

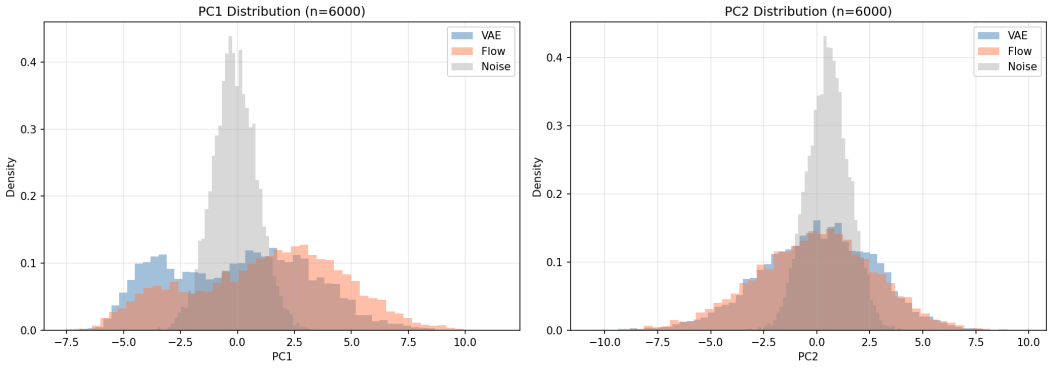

Figure 8: Plots of prior $(N(0, I))$ and latent space distributions for the 2 most relevant principle components. the latent space distributions are created by generative modeling using the trained mean-flow model and by sampling from the data set and encoding the molecules.