Property Driven Molecule Generation using Conditional Normalizing Flows

Omkar Chaudhari, Akshay Gurumoorthi, Rishabh Puri, Matthew Too (Group 9)

12th December, 2024

Abstract

This work presents a method to generate molecules with target properties using Conditional Normalising Flows. The properties of the generated molecules are further validated using xtb, and ORCA. Using the QM9 dataset, an autorgressive normalising flow model is trained on the molecules using TensorFlow and DeepChem, with the one hot encodings of their SELFIES strings. The model is conditioned during training with DFT computed properties of the molecules from the dataset. By utilising the learned bijections, new molecules are sampled by passing a condition vector with properties of interest. Using the models outputted SELFIES strings, these are converted back to SMILES, created into ORCA input files using RDKit and Python, and validated using ORCA. This conditioning approach is tested to see if the model can be directed towards generating molecules with desired properties. Such models will be useful to generate new molecules in various domains like drug discovery, the semiconductor industry, renewable materials etc.

1. Introduction

Generative model have been gaining traction to be powerful models for modeling distributions of various datasets, and encoding them into high dimensional latent spaces, allowing for simplied training for generative tasks. These model attempt to find patterns in the underlying data distributions, and using conditional probability, produce new samples that lie within the data space, ensuring similarity to the original data. Normalising Flows have emerged as one of the newer model architectures used for generative modelling. In this work we have explored conditional normalising flows (CNFs) to evaluate directed generation of molecules towards certain DFT properties of interest.

Early methods for molecular generation, like variational autoencoders (VAEs) and generative adversarial networks (GANs), were used to transform molecular structures, such as SMILES strings, into simpler spaces where properties could be optimized. These methods often had problems like unstable training and difficulties ensuring that the generated molecules were valid. GANs, for example, sometimes fail to explore all possible chemical spaces due to a problem called mode collapse. Conditional normalizing flows (CNFs) solve these issues by using a reversible and probabilistic approach to generate molecules. Unlike VAEs and GANs, CNFs apply a series of reversible transformations that connect molecular structures to their properties, ensuring accuracy and chemical validity. CNFs also allow precise control over the properties of the molecules they generate, making them a useful tool for tasks like drug discovery and materials design [1, 2].

Computational methods for studying molecules have also improved over time. Classical force fields, such as AMBER and CHARMM, use simple equations to model molecular interactions. These methods are fast and work well for large systems, but they cannot accurately describe electronic properties or chemical reactions because they do not use quantum mechanics [3]. Density functional theory (DFT), on the other hand, is a quantum mechanical method that can predict properties like dipole moments and reactivity with high accuracy. However, DFT is very slow and expensive, especially for large datasets [4]. Semi-empirical methods, like PM6, are faster but less accurate, depending on the situation. The xTB (extended Tight Binding) method is an improvement because it balances speed and accuracy. It can model molecular properties efficiently while still capturing important quantum effects, making it ideal for screening many molecules or designing new ones [5].

This project uses conditional normalizing flows (CNFs) to generate molecules based on the QM9 dataset, which contains information about the quantum properties of small molecules. The dataset includes molecular structures in SMILES format and properties like dipole moment, polarizability, and HOMO-LUMO gap. To ensure the molecular data is valid, SMILES strings are converted into SELFIES (Self-referencing Embedded Strings), which follow chemical rules. The goal is to train a CNF model to learn how these molecules are structured and then generate new molecules with specific properties.

The generated molecules are validated using a simple quantum chemistry workflow. First, the SMILES strings are converted into 3D geometries using RDKit and optimized with the Universal Force Field (UFF). These optimized structures are then used to calculate molecular properties, like potential energy, using the GFN2-xTB method, along with computing other properties using ORCA 6.0. These calculated properties are compared to the QM9 dataset to check how well the model works. This approach combines CNFs for molecule generation with xTB and ORCA for validation, ensuring that the generated molecules are both chemically valid and match the desired properties.

2. Methodology

2.1 Mathematical Framework for Multi-Dimensional Normalizing Flows

Conditional Normalizing Flows (CNFs) are a generative modeling method for converting simple base distributions into complex target distributions, especially in multi-dimensional environments. The key characteristic of CNFs is in their application of bijective transformations that are both invertible and differentiable mappings. These modifications guarantee that every point in the base distribution uniquely aligns with a point in the target distribution, maintaining the ability to determine precise likelihoods and enabling efficient sampling. In contrast to conventional generative models such as VAEs or GANs, which frequently depend on approximate mappings, CNFs utilize invertibility to represent intricate interactions among variables while maintaining mathematical tractability. Bijective transformations are essential for enabling lossless information compression, for CNFs to accurately characterize high-dimensional data distributions. By integrating conditional variables into these transformations, CNFs can be used to generate molecules with specific properties. This ability makes them a useful tool for property-oriented molecular synthesis. The bijective framework guarantees stability throughout training and alleviates mode collapse problems typically encountered in GANs, providing substantial benefits for structured data modeling. [1, 6]. We provide an extensive analysis of their mechanics, conditioning, and training goals.

2.1.1 Flow Mechanisms: Multi-Dimensional Unconditional Normalizing Flows

Unconditional Normalizing Flows rely on the principle of normalizing flows where a simple base distribution $p_{\boldsymbol{Z}}(\boldsymbol{z})$ (e.g., a standard Gaussian) is transformed into a complex target distribution $p_{\boldsymbol{X}}(\boldsymbol{x})$ using a sequence of invertible and differentiable transformations. Given invertible functions $\boldsymbol{f}_1, \boldsymbol{f}_2, \ldots, \boldsymbol{f}_k : \mathbb{R}^d \to \mathbb{R}^d$, the transformation from $\boldsymbol{z} \in \mathbb{R}^d$ to $\boldsymbol{x} \in \mathbb{R}^d$ occurs by:

$$\boldsymbol{x} = (\boldsymbol{f}_k \circ \boldsymbol{f}_{k-1} \circ \cdots \circ \boldsymbol{f}_1)(\boldsymbol{z}) = \boldsymbol{f}(\boldsymbol{z}).$$

Each transformation $\boldsymbol{f}_k$ ensures invertibility, allowing the computation of the probability density of $\boldsymbol{x}$ via the change of variables formula:

$$p_{\boldsymbol{X}}(\boldsymbol{x}) = p_{\boldsymbol{Z}}(\boldsymbol{f}^{-1}(\boldsymbol{x})) \left| \det \frac{\partial \boldsymbol{f}^{-1}(\boldsymbol{x})}{\partial \boldsymbol{x}} \right|,$$

where $\det \frac{\partial \boldsymbol{f}^{-1}(\boldsymbol{x})}{\partial \boldsymbol{x}}$ represents the determinant of the Jacobian matrix and corrects for the volume change arising from the transformation such that the resulting probability density integrates to unity. In this equation, the flow $\boldsymbol{f}^{-1}$ passes each $\boldsymbol{x}$ to its corresponding representation $\boldsymbol{z}$ in latent space while normalizing the result to obtain a valid probability density. This is the origin for how normalizing flows get their name [1].

Computational difficulty arises when trying to evaluate the inverse and determinant for general multidimensional inputs. The inverse and determinant are both of complexity $O(d^3)$ and are intractable to calculation for large d. To combat this, different architectures exist to simplify the matrix representing the bijective map. For example, some architectures like Masked Autoencoders for Density Estimation (MADE) utilize a lower triangular map which results in the inverse and determinant being calculated in $O(d^2)$ and $O(d)$ respectively. A compromise has to be made to pick a model which has enough expressiveness to properly generate the density $p_{\boldsymbol{X}}(\boldsymbol{x})$ at reasonable computational cost.

2.1.2 Conditional Normalizing Flows: Enhancing Generative Capabilities

Conditional Normalizing Flows (CNFs) extend this framework by incorporating conditional variables $y \in \mathbb{R}^m$ to adapt the generative process. These variables, such as molecular properties in chemistry, guide the transformations, resulting in conditional mappings:

$$\boldsymbol{x} = (\boldsymbol{f}_k \circ \boldsymbol{f}_{k-1} \circ \cdots \circ \boldsymbol{f}_1)(\boldsymbol{z}, \boldsymbol{y}) = \boldsymbol{f}(\boldsymbol{z}, \boldsymbol{y}),$$

with the conditional density computed as:

$$p_{\boldsymbol{X}}(\boldsymbol{x} \mid \boldsymbol{y}) = p_{\boldsymbol{Z}}(\boldsymbol{f}^{-1}(\boldsymbol{x}, \boldsymbol{y})) \left| \det \frac{\partial \boldsymbol{f}^{-1}(\boldsymbol{x}, \boldsymbol{y}))}{\partial \boldsymbol{x}} \right|.$$

This flexibility enables CNFs to generate outputs tailored to desired properties by modulating transformation parameters based on $\boldsymbol{y}$[7].

2.1.3 Autoregressive Models as Conditioned Normalizing Flows

To enhance model expressiveness, autoregressive transformations are employed to capture dependencies between variables, while permutations reorder variables at each layer, ensuring robust learning of correlations across dimensions. Models like Masked Autoregressive Flow (MAF) and Inverse Autoregressive Flow (IAF) are very effective in CNFs for handling high-dimensional data [7]. MAFs and IAFs are made up of a series of flow layers stacked upon one

another to increase expressiveness. Each flow layer contain (1) a MADE feed-forward neural network layer, and (2) a permutation layer.

MADE Layers: Masked Autoencoder for Density Estimation (MADE) layers are feed-forward neural networks that satisfy the autoregressive property and learn the parameters necessary for calculating a conditioned probability density $p(\boldsymbol{x}|\boldsymbol{y})$ as described in Figure 1. Before understanding the architecture, it is first necessary to understand how $p(\boldsymbol{x}|\boldsymbol{y})$ would be found in general.

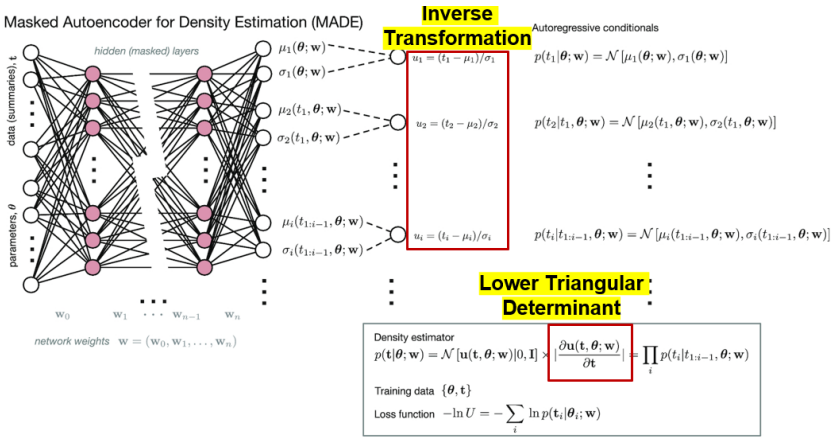

Figure 1: Full MADE architecture including inverse transform and determinant. Image taken from Ref. [8].

Given $\boldsymbol{x} = [x_1, x_2, \ldots, x_d]^T \in \mathbb{R}^d$ from the sampling distribution, $\boldsymbol{z} = [z_1, z_2, \ldots, z_d]^T \in \mathbb{R}^d$ from the base distribution $N(0, 1)$, and a conditioning vector $\boldsymbol{y} \in \mathbb{R}^m$, $p(\boldsymbol{x}|\boldsymbol{y})$ is calculated using the chain rule as:

$$p(\boldsymbol{x}|\boldsymbol{y}) = \prod_{i=1}^{d} p(x_i|\boldsymbol{x}_{1:i-1}, \boldsymbol{y})$$

where $\boldsymbol{x}_{1:i-1} = [x_1, x_2, \ldots, x_{i-1}]^T$ is the vector upon which the current value x_i is conditioned. For MAFs, each conditional probability is represented as a normal distribution with parameters conditioned upon prior dimensional inputs $\boldsymbol{x}_{1:i-1}$:

$$p(x_i|\boldsymbol{x}_{1:i-1}, \boldsymbol{y}) = N(\mu_i, (\exp \alpha_i)^2), \qquad \mu_i = f_{\mu_i}(\boldsymbol{x}_{1:i-1}, \boldsymbol{y}),\ \alpha_i = f_{\alpha_i}(\boldsymbol{x}_{1:i-1}, \boldsymbol{y}).$$

IAFs are similar to MAFs but the parameters are conditioned upon random numbers $\boldsymbol{z}_{1:i-1}$ instead of the data variables $\boldsymbol{x}_{1:i-1}$:

$$p(x_i|\boldsymbol{x}_{1:i-1}, \boldsymbol{y}) = N(\mu_i, (\exp \alpha_i)^2), \qquad \mu_i = f_{\mu_i}(\boldsymbol{z}_{1:i-1}, \boldsymbol{y}),\ \alpha_i = f_{\alpha_i}(\boldsymbol{z}_{1:i-1}, \boldsymbol{y}).$$

In principle, each $p(x_i|\boldsymbol{x}_{1:i-1}, \boldsymbol{y})$ should be found sequentially starting with $i = 1$ to satisfy the autoregressive property. However, it is possible to satisfy this property in a feed-forward neural network in a single pass by dropping out specific connections, which is what MADE accomplishes.

MADE takes in a data vector $\boldsymbol{x}$ and a conditioned vector $\boldsymbol{y}$ and outputs the set of parameters $\{\mu_i, \alpha_i\}_{i=1}^d$. The autoregressive property is satisfied by applying binary masks to weight matrices to drop out specific connections. Specifically, an output layer neuron corresponding to x_i has any connections to neurons in the input and hidden layers corresponding to $\{x_i, x_{i+1}, \ldots, x_d\}$ dropped, assuming that the sequence order is $i = 1, 2, \ldots, d$. Figure 2 shows

an example of this dropout process where the order is $\{x_2, x_3, x_1\}$ rather than $\{x_1, x_2, x_3\}$, in which case the dropout is based off the actual permutation of the x_i's (i.e., the x_2 output neuron will dropout connections involving $\{x_2, x_3, x_1\}$, x_3 output neuron will dropout $\{x_3, x_1\}$ connections, and x_1 output neuron will dropout x_1 connections). By accounting for the autoregressive property within the neural network itself, all parameters necessary for the calculation of all the autoregressive probability densities can be determined in a single pass through the network.

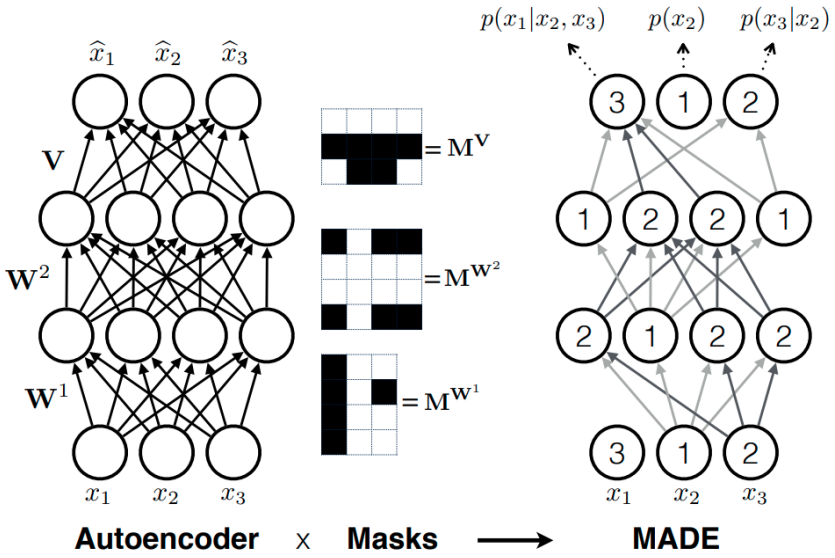

Figure 2: Example of MADE using binary masks to drop connections to satisfy the autoregressive property. Image taken from Ref. [9].

Transformations: For both MAFs and IAFs, given the set of parameters $\{\mu_i, \alpha_i\}_{i=1}^d$ determined from MADE, recursion can be used to generate an element $\boldsymbol{x}$ of the sampling distribution using an element $\boldsymbol{z}$ of the base distribution; this is the generative direction. Starting with a predefined vector $\boldsymbol{z} \sim N(0, 1)$, the set $\{x_i\}_{i=1}^d$ is calculated recursively using:

$$x_i = \mu_i + z_i \cdot \exp \alpha_i.$$

In a similar manner, starting with an element $\boldsymbol{x}$ of the output distribution, the set $\{z_i\}_{i=1}^d$ can be calculated recursively in the normalizing direction which is necessary for estimation of the density $p(\boldsymbol{x})$ via:

$$z_i = (x_i - \mu_i) \cdot \exp(-\alpha_i).$$

Despite both MAFs and IAFs using the same recursive forms, these two models differ on the speed of the generative versus normalizing directions. MAFs can determine $p(\boldsymbol{x})$ in a single step because μ_i and α_i are directly connected to $\boldsymbol{x}_{1:i-1}$, thereby allowing for the immediate determination of $\boldsymbol{z}$ from $\boldsymbol{x}$. However, MAFs are slow for sampling because each x_i must be generated recursively starting with a vector $\boldsymbol{z}$. In contrast, IAFs have μ_i and α_i directly connected to $\boldsymbol{z}_{1:i-1}$ and are therefore slow at density estimation while being fast at sampling by sampling $\boldsymbol{x}$ from $\boldsymbol{z}$ in a single step.

Determinant: Due to the autoregressive property, the bijection $\boldsymbol{f}$ mapping $\boldsymbol{z} \in \mathbb{R}^d$ to $\boldsymbol{x} \in \mathbb{R}^d$ is a lower triangular matrix. The determinant is therefore the product of the diagonal elements

of the inverse matrix Jacobian. For both MAFs and IAFs, the determinant is:

$$\det \frac{\partial \boldsymbol{f}^{-1}(\boldsymbol{x}, \boldsymbol{y})}{\partial \boldsymbol{x}} = \prod_{i=1}^{d} \exp\left(-\alpha_i\right) = \exp\left(-\sum_{i=1}^{d} \alpha_i\right)$$

and is amenable to calculation even for large d.

Permutations: Permutations further ensure that dependencies across dimensions are learned effectively by reordering variables between layers:

$$\boldsymbol{x}_{\text{perm}} = \Pi(\boldsymbol{x}),$$

where Π is the permutation matrix [10]. Because permutations are volume preserving, the magnitude of the determinant for a permutation layer is unity.

2.1.4 Training Objective

The goal of training CNFs is to maximize the log-likelihood, or equivalently minimize the negative-log-likelihood, of observed data $\mathcal{D} = \{\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\}_{i=1}^{N}$. For a single sample, the likelihood is:

$$p_{\boldsymbol{X}}(\boldsymbol{x}^{(i)} \mid \boldsymbol{y}^{(i)}) = p_{\boldsymbol{Z}}(\boldsymbol{f}^{-1}(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})) \left| \det \frac{\partial \boldsymbol{f}^{-1}(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}))}{\partial \boldsymbol{x}^{(i)}} \right|,$$

where the first term accounts for the base distribution $p_{\boldsymbol{Z}}$, and the second term represents the determinant of the Jacobian, capturing the complexity of the transformations. The overall loss function is:

$$\mathcal{L} = -\sum_{i=1}^{N} \log p_{\boldsymbol{X}}(\boldsymbol{x}^{(i)} \mid \boldsymbol{y}^{(i)}),$$

which is minimized to find the best parameters for some functional form of $\boldsymbol{f}$ [11]. In the case of MAFs and IAFs, the loss is used during the training of the MADE neural network layers to optimize the parameters $\{\mu_i, \alpha_i\}_{i=1}^{d}$.

2.2 Data Preparation and Encoding

2.2.1 Dataset

The QM9 dataset comprises molecular structures and characteristics of 134,000 small organic compounds, sourced from the GDB-9 subset of the GDB-17 chemical library. It comprises neutral organic compounds with up to nine non-hydrogen atoms (C, N, O, F), hence representing a substantial segment of organic chemical space. Biomolecules, such as amino acids and nucleobases, along with medicinal substances, are depicted. Of the 621 unique formulations, $C_7H_{10}O_2$ is the predominant compound, displaying 6095 constitutional isomers. Structures were derived from SMILES strings and optimized utilizing PM7 (MOPAC) and B3LYP/6-31G(2df,p) (Gaussian 09) methodologies. Rigorous requirements guaranteed precision, and characteristics were computed using the B3LYP method, whereas G4MP2 was employed for the energetics of $C_7H_{10}O_2$ isomers. This dataset, contains molecular SMILES strings and quantum chemical properties including rotational contants, dipole moment, isotropic polarizability, HOMO-LUMO gap, internal energies, vibrational energies, enthallpy, free energies, heat capacity, electronic spatial extent, zero-point vibrational energy, and atomization energies are used as the input dataset.

2.2.2 Preprocessing

Preprocessing ensures that the molecular data is prepared in a form suitable for computational models. The QM9 dataset consists of chemically diverse molecules represented as SMILES strings, which need to be preprossed and encoded to maintain chemical validity. Proper preprocessing guarantees that the input data can work with the requirements of downstream models, such as Conditional Normalizing Flows (CNFs), enabling accurate training and property prediction.

- **Data Extraction**
  The preprocessing begins with extracting molecular structures represented as SMILES strings and their corresponding quantum chemical properties derived from Density Functional Theory (DFT). These properties, such as dipole moments, isotropic polarizability, and HOMO-LUMO gaps, are critical for conditioning molecular generation tasks. The quantum chemical properties are stored as a matrix of shape (n, t), where n is the number of molecules, and t represents the number of task features.

- **Filtering Fragmented Molecules**
  To ensure chemical validity, molecules with disconnected components (e.g., containing "`.`" in their SMILES representation) are removed. These molecules are fragmented molecules, and are removed so they dont get generated as fragments.

- **Conversion to SELFIES**
  The filtered SMILES strings are converted into SELFIES (Self-referencing Embedded Strings), a chemically robust molecular representation. SELFIES ensure that all encoded molecules are chemically valid, even under transformations or perturbations, making them a reliable input for machine learning models. The SELFIES strings are stored in a new column, and the dataset is sorted by molecular length to facilitate efficient encoding.

- **One-Hot Encoding**
  The SELFIES strings are encoded into one-hot vectors using a custom alphabet generated from the dataset. The alphabet is augmented with a `[nop]` token for padding sequences of varying lengths. The maximum sequence length in the dataset determines the size of the encoded vectors, resulting in one-hot matrices of shape (n, m), where n is the number of molecules and m is the maximum sequence length.

- **Integration with DFT Features**
  The one-hot encoded SELFIES vectors are combined with the DFT features, creating a vector representation that combines structural information with quantum properties. This combination results in a dataset of shape $(n, m + t)$, where m is the one-hot vector size and t is the number of DFT features. The combined data is then converted into a tensor for computational efficiency.

- **Dequantization**
  A uniform noise tensor is added to the input data to perform dequantization, preventing the model from learning artificial discontinuities. The dequantized data is prepared as a tensor of shape $(n, m + t)$, ensuring compatibility with downstream training.

- **Dataset Splitting**
  The dataset is split into training, validation, and test sets using a random splitting strategy. The splits ensure that molecular diversity is preserved in each subset, enabling robust training and evaluation. The splits result in three separate datasets for training, validation, and testing, each containing tensors for molecular representations and DFT features.

2.3 Data Dequantization

Data dequantization refers to the process of introducing noise to discrete one-hot encoded vectors in order to transform them into a continuous space. This transformation is essential for training Conditional Normalizing Flows (CNFs), as these models require continuous data distributions rather than discrete ones. The typical approach involves adding small amounts of noise to the one-hot encoded representations, ensuring that the input space becomes suitable for training CNFs.

Mathematically, given a one-hot encoded vector $\mathbf{x} \in \{0, 1\}^d$, where d is the dimensionality of the vector, the dequantized representation $\tilde{\mathbf{x}}$ is computed as:

$$\tilde{\mathbf{x}} = \mathbf{x} + \epsilon, \quad \epsilon \sim \mathcal{U}(0, \delta),$$

where:

- ϵ is a noise vector sampled independently for each dimension from a uniform distribution $\mathcal{U}(0, \delta)$,
- δ is a small positive constant controlling the magnitude of the noise.

This process generates a soft, continuous version of the original data:

$$\tilde{\mathbf{x}} \in [0, 1]^d,$$

making it suitable for CNFs that operate on continuous distributions.

2.4 Model Architecture and Training

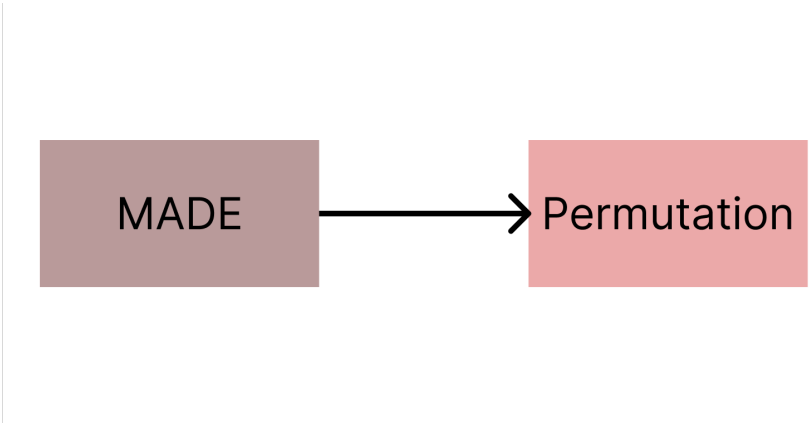

Figure 3: Base unit of the Model

2.4.1 Base Distribution:

- A multivariate normal distribution is defined as the base distribution, with a mean vector of zeros and a diagonal covariance matrix.
- The mean vector has a size equal to `dim_smiles` and is initialized with zeros.
- The diagonal of the covariance matrix is initialized with ones, both defined in 64-bit floating-point precision.

2.4.2 Flow Layers:

- The flow model consists of 8 layers, with each layer comprising a MADE bijector followed by a permutation.
- The MADE bijector is defined using `tfb.MaskedAutoregressiveFlow`, which uses an autoregressive neural network with hidden layers of size [16, 16] and ReLU activations.
- The permutation layer shuffles the input dimensions to improve expressivity, with random permutations generated by shuffling integers from 0 to `dim_smiles`.

2.4.3 Chaining and Transformed Distribution:

- A `tfb.Chain` bijector is created by chaining the reversed list of flow layers.
- A transformed distribution is then defined using the base multivariate normal distribution and the chain of bijectors.

2.5 Model Training

2.5.1 Framework

The implementation begins with TensorFlow and TensorFlowProbability. The training integrates seamlessly with molecular datasets using DeepChem.

2.5.2 Model Construction

- The conditional normalizing flow model is constructed using TensorFlow Keras. The model takes two inputs: molecular features with dimensions `dim_smiles` and conditional quantum features with dimensions `dim_dft`.
- The log probability of the transformed distribution is calculated as the model output. The calculation is conditioned on the quantum features using bijector-specific keyword arguments.
- The model is compiled with the Adam optimizer, using a learning rate of 1×10^{-4}, and a custom loss function defined as the negative log-likelihood.

2.5.3 Training Process

- The molecular features are normalized by subtracting the mean and dividing by the standard deviation for each feature.
- Early stopping is used to monitor the validation loss during training. The training stops if there is no improvement in validation loss for 5 consecutive epochs, and the best weights are restored automatically.
- The model is trained with a batch size of 128 for a maximum of 100 epochs. The number of steps per epoch is determined based on the size of the training dataset.
- The training and validation losses are monitored throughout the training process to evaluate the model's convergence.

2.5.4 Hyperparameter Tuning

The model was evaluated with different combinations of layers and hidden units to optimize its performance. The hyperparameters tested include the number of flow layers (4, 6, 8) and hidden unit configurations ([8, 8], [16, 16]). The results from hyperparameter tuning are summarized in Table 1.

Table 1: Hyperparameter Combinations Tested During Model Tuning

Number of Layers	Hidden Units	Activation Function
4	[8, 8]	ReLU
4	[16, 16]	ReLU
6	[8, 8]	ReLU
6	[16, 16]	ReLU
8	[8, 8]	ReLU
8	[16, 16]	ReLU

2.5.5 Training Parameters

The key parameters used for training are detailed in Table 2.

Table 2: Model Training Parameters

Parameter	Value
Batch Size	128
Learning Rate	1×10^{-4}
Maximum Epochs	100
Early Stopping Patience	5
Optimizer	Adam
Loss Function	Negative Log-Likelihood

3. Results and Discussion

3.1 Molecule Generation

- Samples are drawn from the trained flow model to generate continuous molecular representations.
- These representations are quantized into discrete binary encodings and converted back to SELFIES strings.
- The SELFIES strings are then decoded into SMILES using the SELFIES decoder.

3.2 Validation and Property Calculation

- **Chemical Validity:** The python library RDKit was used to obtain an initial guess of the 3D structure of the generated molecule, from its SMILES string. Subsequently, the RDKit generated structures were used as a starting guess for verification through DFT calculations.
- **Property Verification:**

– DFT Calculations, performed using Orca 6.0.1[12] were subsequently done to obtain the chemical and physical properties of the molecule, to validate the predictions of the Generative Model. Some of the properties calculated from DFT and used for comparison include dipole moment, zero point vibrational energy, enthalpy, Gibbs Free Energy etc. among others. The DFT calculations, to validate were done at a lower level of theory than the one used in the QM9 Dataset. This was done to ensure that the validation does not consume a high level of computational resources and to evaluate the accuracy of the generative Model. The Dipole Moments of the generated molecules, are compared to the expected dipole moment, from the input to the generative model through the evaluation of the mean absolute error, which is reported in Fig 4. An observation from the plot is that the Mean Absolute Error reduces with the increase of neurons, for the same number of layers, consistent with expectations.

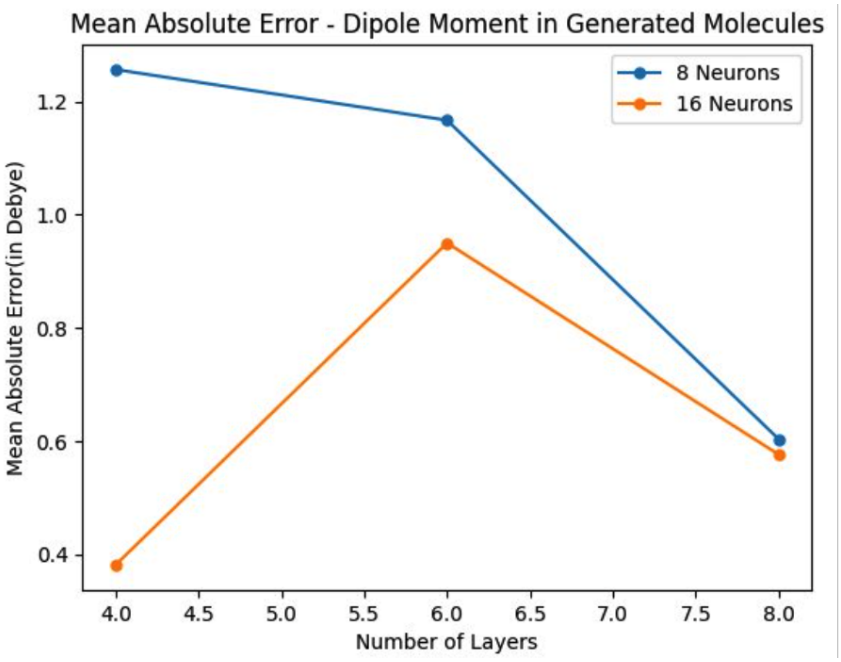

Figure 4: Mean Absolute Error of the Dipole Moment of Generated Molecules

– In Fig. 5, representation of the generated molecules, whose dipole moment is well-matched with the target values from the model are reported. Apart from successful generation of molecules with the target properties, there were also various molecules generated that were unphysical. Fig 6. has some examples, with energetically unstable 3 and 4-membered rings.

- In Fig. 7, the training and validation losses for one set of parameters of the generative model is reported. The high negative value of the losses is due to gradient instability, which leads to extreme volume elements in the determinant.

3.3 Similarity and Visualization

- **Similarity Analysis:**

  – Tanimoto similarity scores are computed between fingerprints of generated molecules and those in the QM9 dataset.

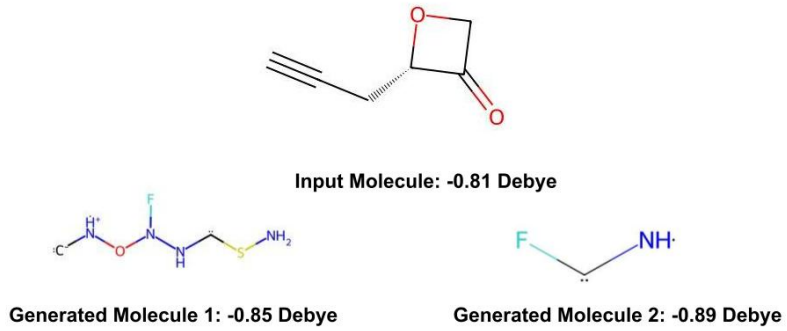

Figure 5: Input molecule and corresponding generated molecules from the Model

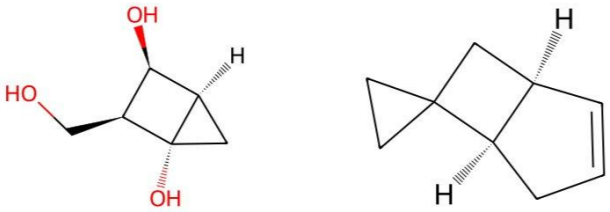

Figure 6: Unphysical Molecules generated by the model

  - Top similar molecules are visually inspected for alignment with known molecular structures.

- **Visualization:** RDKit is used to render molecular structures, and representative examples are presented for qualitative evaluation.

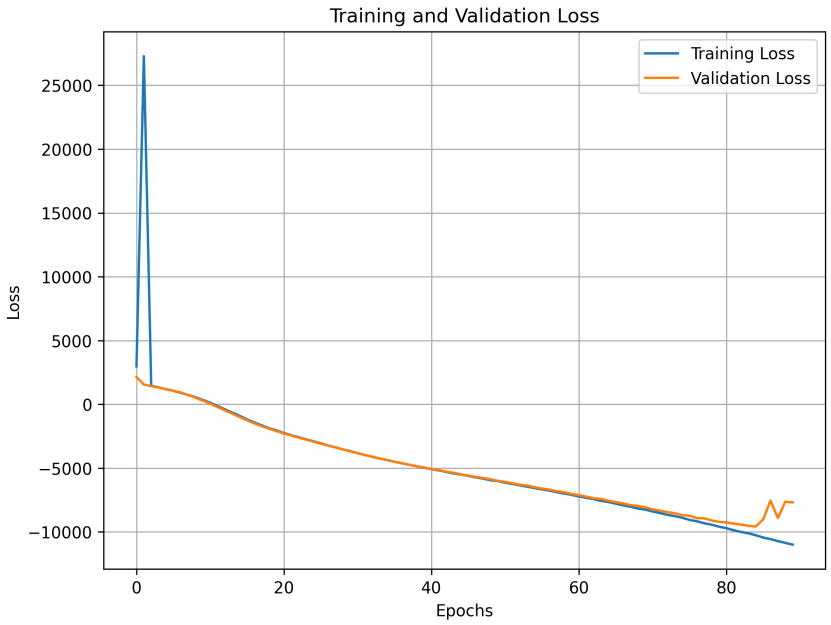

Figure 7: Training and Validation Losses for the model with 6 layers, and 8 neurons per layer.

4. Conclusion and Future Scope

4.1 Conclusion

In this project, we explored Normalising Flows as a viable model to generate molecules with specific properties. By using tensorflow, deepchem, and ORCA, we learned that the model can be directed towards generation of molecules with certain properties.

The results demonstrated that such implementations to generate molecules are possible, albeit with some considerations to be made.

4.2 Future Scope

Several practical improvements and explorations can be pursued to deepen understanding and improve the model:

- **Improving Gradient Stability:** Gradient instability was observed during training, leading to issues like extreme loss values. Exploring different initialization methods, smaller learning rates, or regularization techniques could help stabilize training.
- **Testing Other Flows:** Beyond Masked Autoregressive Flows, trying different types of flows, such as RealNVP or Neural Spline Flows, could offer insights into how flow choices affect model performance and stability.
- **Better Handling of Unphysical Outputs:** Adding constraints or penalties during training could reduce the number of unphysical molecules generated. For example, filtering outputs with known structural issues could be integrated into the pipeline.

These practical steps would not only improve the current model but also enhance understanding of CNF models, making them more accessible and easier to apply to similar problems in the future.

5. References

References

[1] D. Rezende, S. Mohamed, Variational inference with normalizing flows, in: International conference on machine learning, PMLR, 2015, pp. 1530–1538.

[2] K. Madhawa, K. Ishiguro, K. Nakago, M. Abe, Graphnvp: An invertible flow model for generating molecular graphs, arXiv preprint arXiv:1905.11600 (2019).

[3] W. D. Cornell, P. Cieplak, C. I. Bayly, I. R. Gould, K. M. Merz, D. M. Ferguson, D. C. Spellmeyer, T. Fox, J. W. Caldwell, P. A. Kollman, A second generation force field for the simulation of proteins, nucleic acids, and organic molecules, Journal of the American Chemical Society 117 (19) (1995) 5179–5197.

[4] B. Miehlich, A. Savin, H. Stoll, H. Preuss, Results obtained with the correlation energy density functionals of becke and lee, yang and parr, Chemical Physics Letters 157 (3) (1989) 200–206.

[5] L. Goerigk, A. Hansen, C. Bauer, S. Ehrlich, A. Najibi, S. Grimme, A look at the density functional theory zoo with the advanced gmtkn55 database for general main group thermochemistry, kinetics and noncovalent interactions, Physical Chemistry Chemical Physics 19 (48) (2017) 32184–32215.

[6] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, B. Lakshminarayanan, Normalizing flows for probabilistic modeling and inference, Journal of Machine Learning Research 22 (57) (2021) 1–64.

[7] G. Papamakarios, T. Pavlakou, I. Murray, Masked autoregressive flow for density estimation, Advances in neural information processing systems 30 (2017).

[8] J. Alsing, T. Charnock, S. Feeney, B. Wandelt, Fast likelihood-free cosmology with neural density estimators and active learning, Monthly Notices of the Royal Astronomical Society 488 (3) (2019) 4440–4458.

[9] M. Germain, K. Gregor, I. Murray, H. Larochelle, Made: Masked autoencoder for distribution estimation, in: International conference on machine learning, PMLR, 2015, pp. 881–889.

[10] D. P. Kingma, P. Dhariwal, Glow: Generative flow with invertible 1x1 convolutions, Advances in neural information processing systems 31 (2018).

[11] I. Kobyzev, S. J. Prince, M. A. Brubaker, Normalizing flows: An introduction and review of current methods, IEEE transactions on pattern analysis and machine intelligence 43 (11) (2020) 3964–3979.

[12] F. Neese, The orca program system, WIRES Comput. Molec. Sci. 2 (1) (2012) 73–78. doi:10.1002/wcms.81.