Lecture 16

Molecular Docking with Diffusion Models

Contents

In this lecture we introduce the *molecular docking* problem, which concerns predicting the binding structure between a protein and a small-molecule ligand. Given the three-dimensional structures of the protein and the ligand, the goal is to determine how the ligand can bind within the protein binding pocket and form a stable complex.

We will discuss how diffusion models, in particular the Denoising Diffusion Probabilistic Models (DDPM), can be applied to this problem.

1 Molecular Docking

Molecular docking is a central topic in structural biology and computational drug discovery. The problem can be summarized schematically as

$$\text{protein structure} + \text{ligand structure} \;\rightarrow\; \text{geometric pose of the protein–ligand complex}$$

In other words, docking attempts to predict the spatial orientation and conformation of a ligand when bound to a protein binding pocket.

An example illustration is shown in Fig. 1.

In most cases, the ligand binds within a protein pocket through a collection of non-covalent interactions, including hydrogen bonding, electrostatic interactions, van der Waals (dispersion) interactions, and π–π stacking. Because these interactions are relatively weak individually, the geometric complementarity between the ligand and the pocket plays a crucial role in stabilizing the complex.

For favorable poses, the ligand binds to the pocket with a decrease in free energy, referred to as the *binding free energy*. This quantity is directly related to the **binding affinity** between the protein

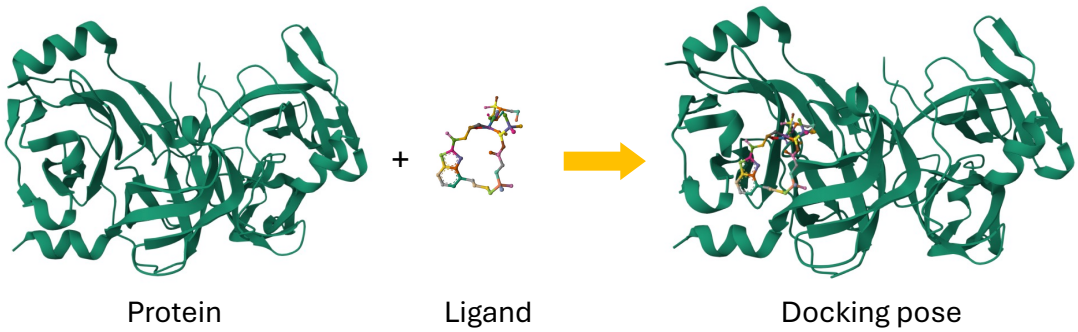

Figure 1: Docking of a macrocyclic inhibitor to the hepatitis C virus (HCV) NS3 protease. The structures are obtained from the Protein Data Bank (PDB ID: 2XNI).

and the ligand. Consequently, accurately predicting docking poses is an important step toward estimating binding affinity and evaluating the effectiveness of a ligand as a potential inhibitor.

1.1 Molecular Docking Representation

From the problem description, we know that the input is the structures of the protein and the ligand, and the output is the pose of ligand relative to the protein. The ideal case is representing the whole system with the 3D geometry, but that would be overly complicated and computationally unfavorable. In the following, we describe a simplified representation proposed by the DiffDock work [1].

1.1.1 Protein Structure

For the docking problem, we usually assume that the binding pocket is known. Therefore, instead of representing the entire protein, we model only the structure around the pocket, which drastically reduces complexity.

Additionally, since only the relative orientation of the ligand to the pocket matters, we assume that the pocket's position is fixed in 3D space. In summary, we make two main assumptions:

- Only the pocket and its immediate neighborhood are considered.
- The protein pocket is fixed in 3D space.

To retain the pocket structure, we define a **cutoff distance** to the ligand, e.g., 10 Å. Atoms within this region are kept, while others are discarded. An example of this procedure is shown in Fig. 2

1.1.2 Ligand Representation

For diffusion-based docking, the ligand is represented by its 3D coordinates and chemical features. The key idea is that the ligand's internal geometry (bond lengths, angles, torsions) is

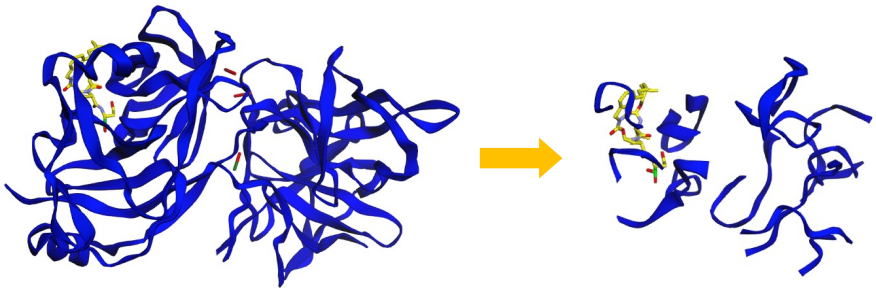

Figure 2: Reducing the protein-ligand complex to pocket-ligand complex for the 2XNI complex.

preserved, but its absolute position and orientation relative to the pocket are initially unknown.

The features used in the diffusion model include:

- Atomic positions: 3D coordinates of each atom.
- Atom types: element identity, hybridization, or other chemical descriptors.
- Torsion angles: for flexible bonds, to allow the model to predict conformational adjustments.

During the diffusion process, the model iteratively updates the ligand's coordinates and torsion angles, including translation, rotation, and torsion. The model refines the ligand pose inside the pocket. The protein pocket remains fixed, serving as a spatial reference that guides the ligand into a physically plausible binding configuration.

1.1.3 Advanced reading

In the paper, the authors used graphs to represent both pockets and ligands. Since our focus of this lecture is diffusion models, we will not dive into details of the graph representations.

1.2 Data Processing

The raw training data are derived from the Protein Data Bank (PDB) with the protein-ligand complex structures, as a `.pdb` file.

We have introduced downloading a `.pdb` file using API.

```
import requests

pdb_id = "2XNI"
url = f"https://files.rcsb.org/download/{pdb_id}.pdb"
r = requests.get(url)
r.raise_for_status()
pdb_content = r.text  # PDB file content as string
```

```
# Save to local file
output_file = f"{pdb_id}.pdb"
with open(output_file, "w") as f:
    f.write(pdb_content)
```

You can also download a batch of `.pdb` files using a shell script: `https://www.rcsb.org/docs/programmatic-access/batch-downloads-with-shell-script`.

Once you have the `.pdb` file, you can extract the ligand and the pockets. The Python Library used here is `biopython`: `https://biopython.org/`. A script is provided in the lab session.

2 Diffusion Models

A diffusion model is a generative model that learns to transform noise into structured data by modeling the reversal of a diffusion process, shown in Fig. 3. It consists of two main components:

- **Forward process**: progressively adds noise to the data until it becomes nearly random (Gaussian noise).
- **Reverse process**: a neural network learns to denoise the data step by step, reconstructing the original distribution.

Diffusion models have gained popularity in image generation, for example in OpenAI's DALL·E. They are particularly powerful for learning continuous data in 2D or 3D. Molecules, composed of atoms with continuous 3D coordinates, are analogous to image pixels with continuous RGB values. This analogy has motivated the use of diffusion models for generating chemical structures, including small molecules, materials, and ligand poses.

$$\text{image pixels} \leftrightarrow \text{atoms} \qquad \text{RGB values} \leftrightarrow \text{3D coordinates}$$

Probability Diffusion

At each forward step, we add noise to the data, so that after T steps, the final state is essentially Gaussian noise. For example, in images, each pixel's RGB value becomes approximately standard Gaussian. Reversing the process reconstructs the original data probabilistically. Therefore, the diffusion model learns the **distribution** of the original data:

$$\mathbf{x} \sim p(\mathbf{x}) \tag{1}$$

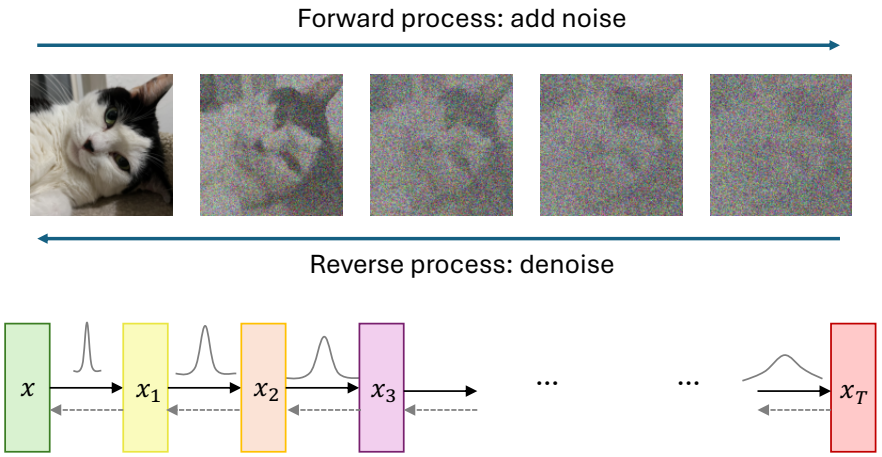

Figure 3: Diffusion model learning an image.

2.1 Forward Process

Starting from the original data $\mathbf{x}$, we gradually transform it into a noisy vector over T steps:

$$\mathbf{x} \longrightarrow \mathbf{x}_1 \longrightarrow \mathbf{x}_2 \longrightarrow \cdots \longrightarrow \mathbf{x}_t \longrightarrow \cdots \longrightarrow \mathbf{x}_T \tag{2}$$

At step t, the forward process assumes a Gaussian transition from the previous state:

$$\mathbf{x}_t \sim \mathcal{N}\left(\sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t I\right), \tag{3}$$

where the mean is a rescaled $\mathbf{x}_{t-1}$ and the variance is controlled by $\beta_t \in [0,1]$, a *hyperparameter* that determines the noise added at this step.

Equivalently, using the *reparameterization trick*:

$$\mathbf{x}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\epsilon, \quad \epsilon \sim \mathcal{N}(0, I) \tag{4}$$

Starting from $\mathbf{x}_0 = \mathbf{x}$, we can relate $\mathbf{x}_t$ to $\mathbf{x}_0$ directly

$$\begin{aligned} \mathbf{x}_t &= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, \quad \epsilon \sim \mathcal{N}(0, I), \\ \text{where} \quad \bar{\alpha}_t &= \prod_{s=1}^{t}(1-\beta_t). \end{aligned} \tag{5}$$

Intuitively, if we think of $\mathbf{x}$ as the position of a particle and t as time, each step moves the particle slightly toward zero (the $\sqrt{1-\beta_t}\mathbf{x}_{t-1}$ term) while randomly diffusing it according to the Gaussian

noise (the $\sqrt{\beta_t}\epsilon$ term). This is why the model is called a *diffusion* model.

Note: the forward process does not involve any neural networks. Each $\mathbf{x}_t$ is sampled directly from the well-defined Gaussian distributions defined by the variance schedule $\{\beta_t\}_{t=1}^{T}$.

2.2 Reverse Process

The forward diffusion is straightforward, but how do we reverse it? How can we model the conditional distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$? Sampling directly from this distribution is impractical because it would require many samples to reconstruct the distribution.

Since the forward transition $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ is Gaussian, we assume the reverse conditional is also Gaussian, with learnable parameters $\boldsymbol{\theta}$:

$$q_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}\big(\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t, t), \Sigma(t)\big), \tag{6}$$

where $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ is the predicted mean depending on $\mathbf{x}_t$ and network parameters $\boldsymbol{\theta}$.

The covariance $\Sigma(t)$ is typically **not learned**, but derived from the forward noise schedule $\{\beta_t\}_{t=1}^{T}$:

$$\Sigma(t) = \tilde{\beta}_t I, \quad \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t, \quad \bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s). \tag{7}$$

Thus, the main task is to learn $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$. Using the relation between forward and reverse steps, the mean can be evaluated as

$$\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\right), \tag{8}$$

where $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ is the predicted noise, unknown a priori, which we model using a fully connected neural network (FNN):

$$(\mathbf{x}_t, t) \quad \longrightarrow \quad \mathrm{FNN}_{\boldsymbol{\theta}} \quad \longrightarrow \quad \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \tag{9}$$

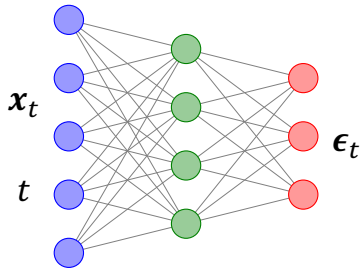

Figure 4: Neural network predicting noise for one reverse step.

Once $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ is computed, the reverse step samples

$$\mathbf{x}_{t-1} = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) + \sqrt{\tilde{\beta}_t}\,\eta, \quad \eta \sim \mathcal{N}(0, I). \tag{10}$$

2.3 Training Objective

For a known $\mathbf{x}_0$, we can compute the **true noise** ϵ added in the forward process. The network is trained to predict this noise using the loss

$$\mathcal{L}_t(\boldsymbol{\theta}) = \left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\right\|^2, \quad \epsilon \sim \mathcal{N}(0, I) \tag{11}$$

which is a simplified form of the maximum likelihood for $p_{\boldsymbol{\theta}}(\mathbf{x})$

In practice, we:

1. Sample a timestep t.
2. Sample noise $\epsilon \sim \mathcal{N}(0, I)$ and compute x_t from x_0.
3. Evaluate the loss $\mathcal{L}_t(\boldsymbol{\theta})$.

Minimizing this loss over many training steps allows the network to learn to denoise across all timesteps.

2.4 Generating New Data

To generate new samples, only the reverse process is needed:

1. Sample $\mathbf{x}_T \sim \mathcal{N}(0, I)$.
2. Iteratively compute $\mathbf{x}_{T-1}, \ldots, \mathbf{x}_0$ using the learned network.
3. Return $\mathbf{x}_0$ as the generated data.

3 Conditional Diffusion Models

In many applications, we want to generate data under certain conditions. For example, in molecular docking we want to generate ligand poses that are compatible with a given protein pocket. Diffusion models can be naturally extended to this setting by introducing conditional diffusion models.

Let $\mathbf{y}$ denote the conditioning variable. In the docking problem, $\mathbf{y}$ may represent the protein pocket structure and associated features. Instead of learning the unconditional distribution $p(\mathbf{x})$, the

goal is to learn the conditional distribution

$$p(\mathbf{x}|\mathbf{y}),$$

which represents the distribution of data $\mathbf{x}$ given the condition $\mathbf{y}$.

3.1 Architecture Changes

Forward process. The forward diffusion process remains **unchanged**. The conditioning variable $\mathbf{y}$ does not affect the forward process; it only appears in the reverse model.

Reverse process. In the reverse diffusion process, we learn the conditional distribution

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{y}) = \mathcal{N}\left(\boldsymbol{\mu}_\theta(\mathbf{x}_t, \mathbf{y}, t), \Sigma(t)\right). \tag{12}$$

The neural network now receives both the noisy data $\mathbf{x}_t$ and the condition $\mathbf{y}$ as input:

$$(\mathbf{x}_t, \mathbf{y}, t) \quad \longrightarrow \quad \mathrm{NN}_\theta \quad \longrightarrow \quad \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{y}, t).$$

Similar to the unconditional case, the mean can be expressed using the predicted noise:

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, \mathbf{y}, t) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{y}, t)\right). \tag{13}$$

3.2 Training Objective

The training objective remains the same as before, except that the prediction now depends on the condition $\mathbf{y}$:

$$\mathcal{L}_t(\theta) = \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{y}, t)\|^2. \tag{14}$$

During training, we sample (1) a data sample $\mathbf{x}_0$, (2) a condition $\mathbf{y}$, (3) a timestep t, and (4) noise $\boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$. Then we construct $\mathbf{x}_t$ using the forward process.

3.3 Conditional Generation

After training, new samples conditioned on $\mathbf{y}$ can be generated by running the reverse diffusion process:

1. Pick a fixed $\mathbf{y}$ value.
2. Sample $\mathbf{x}_T \sim \mathcal{N}(0, I)$.

3. For $t = T, T-1, \ldots, 1$, compute

$$\mathbf{x}_{t-1} = \boldsymbol{\mu}_\theta(\mathbf{x}_t, \mathbf{y}, t) + \sqrt{\tilde{\beta}_t}\,\eta, \quad \eta \sim \mathcal{N}(0, I). \tag{15}$$

4. Return $\mathbf{x}_0$ as a generated sample satisfying the condition $\mathbf{y}$.

In the docking problem, the condition $\mathbf{y}$ corresponds to the protein pocket structure. The diffusion model then generates ligand poses that are compatible with the geometry and chemical environment of the pocket.

4 Applications

Diffusion models are widely used for many chemical problems [2]. We summarize some applications in Table 1.

Application Area	**Representative Papers / Models**
Molecule Generation	EDM [3], GeoDiff [4]
Protein Structure Design	RFdiffusion [5]
Protein-Ligand Docking	DiffDock [1]
Crystal/Materials Generation	DiffCSP [6], CDVAE [7]
Molecular Conformation Sampling	ConfGF [8], GeoDiff [4]
Reaction Modeling	Transition state prediction [9]

Table 1: Representative applications of diffusion models in chemistry, biology, and materials science.

4.1 Diffusion Models for Molecular Docking

Molecular docking can be framed as a generative problem: given a protein pocket, we aim to generate plausible ligand poses that satisfy chemical and geometric constraints. Diffusion models are well-suited for this task because they can model complex, continuous 3D structures and learn distributions over ligand configurations.

4.1.1 Problem Formulation

Let $\mathcal{P}$ denote the fixed protein pocket coordinates and features, and let $\mathbf{x}_0$ denote the ligand atomic coordinates in a reference frame. The docking problem can be written as generating a ligand configuration conditioned on the pocket:

$$\mathbf{x}_0 \sim p_{\text{dock}}(\mathbf{x}_0 | \mathcal{P})$$

where p_{dock} is the distribution of physically plausible docked poses.

We model this distribution with a conditional diffusion model:

$$\mathbf{x}_T \sim \mathcal{N}(0, I), \quad \mathbf{x}_{t-1} = \boldsymbol{\mu}_\theta(\mathbf{x}_t, \mathcal{P}, t) + \sqrt{\tilde{\beta}_t}\,\eta, \quad \eta \sim \mathcal{N}(0, I)$$

Here, the network $\epsilon_\theta(\mathbf{x}_t, \mathcal{P}, t)$ predicts the noise added to the ligand coordinates at timestep t, conditioned on the protein pocket $\mathcal{P}$.

4.1.2 Input Representation

Protein pocket: Only atoms within a cutoff (e.g., 10 Å) of the ligand are considered. The pocket coordinates are fixed in 3D space and serve as a conditioning input to the network.

Ligand: The ligand coordinates $\mathbf{x}_0$ are taken in a reference frame relative to the pocket. Additional features such as atom types, bond connectivity, or torsion angles can be incorporated as network inputs.

4.1.3 Network and Loss

A neural network predicts the noise $\epsilon_\theta(\mathbf{x}_t, \mathcal{P}, t)$ for the ligand coordinates at each diffusion timestep. The network is trained using the mean squared error between the predicted noise and the true Gaussian noise added during the forward process:

$$\mathcal{L}_t(\theta) = \left\|\epsilon - \epsilon_\theta(\mathbf{x}_t, \mathcal{P}, t)\right\|^2, \quad \epsilon \sim \mathcal{N}(0, I)$$

During training, a random timestep t is sampled for each ligand-pocket pair, and the forward noisy ligand coordinates $\mathbf{x}_t$ are generated from the clean pose $\mathbf{x}_0$.

4.1.4 Pose Generation

Once trained, ligand poses can be generated as follows:

1. Input the protein structure $\mathcal{P}$
2. Sample initial ligand coordinates $\mathbf{x}_T \sim \mathcal{N}(0, I)$.
3. Iteratively apply the reverse diffusion steps conditioned on the pocket $\mathcal{P}$:

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t, \mathcal{P}, t)\right) + \sqrt{\tilde{\beta}_t}\,\eta$$

4. Return $\mathbf{x}_0$ as a predicted docked ligand pose.

This approach allows generating multiple plausible poses by sampling different initial noise vectors $\mathbf{x}_T$, effectively capturing the distribution of binding conformations for a given pocket.

5 Lab

Processing PDB files and find pockets:

`https://colab.research.google.com/drive/1Xuh8CEwGSEotuZwtrJqTUGWVEDQQBjao?usp=sharing`

A simple diffusion model:

`https://colab.research.google.com/drive/1V50QMtEcgapCn0kElcuWGvjlmVtH1VpN?usp=sharing`

References

[1] Gabriele Corso, Hannes Stärk, Bowen Jing, Regina Barzilay, and Tommi Jaakkola. Diffdock: Diffusion steps, twists, and turns for molecular docking, 2023.

[2] Liang Wang, Chao Song, Zhiyuan Liu, Yu Rong, Qiang Liu, and Shu Wu. Diffusion models for molecules: A survey of methods and tasks. *arXiv preprint arXiv:2502.09511*, 2025.

[3] Emiel Hoogeboom, Victor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3d, 2022.

[4] Minkai Xu, Lantao Yu, Yang Song, Chence Shi, Stefano Ermon, and Jian Tang. Geodiff: a geometric diffusion model for molecular conformation generation, 2022.

[5] Joseph L Watson, David Juergens, Nathaniel R Bennett, Brian L Trippe, Jason Yim, Helen E Eisenach, Woody Ahern, Andrew J Borst, Robert J Ragotte, Lukas F Milles, et al. De novo design of protein structure and function with rfdiffusion. *Nature*, 620(7976):1089–1100, 2023.

[6] Rui Jiao, Wenbing Huang, Peijia Lin, Jiaqi Han, Pin Chen, Yutong Lu, and Yang Liu. Crystal structure prediction by joint equivariant diffusion. *Advances in Neural Information Processing Systems*, 36:17464–17497, 2023.

[7] Tian Xie, Xiang Fu, Octavian-Eugen Ganea, Regina Barzilay, and Tommi Jaakkola. Crystal diffusion variational autoencoder for periodic material generation. *arXiv preprint arXiv:2110.06197*, 2021.

[8] Chence Shi, Shitong Luo, Minkai Xu, and Jian Tang. Learning gradient fields for molecular conformation generation. In *International conference on machine learning*, pages 9558–9568. PMLR, 2021.

[9] Chenru Duan, Yuanqi Du, Haojun Jia, and Heather J Kulik. Accurate transition state generation with an object-aware equivariant elementary reaction diffusion model. *Nature computational science*, 3(12):1045–1055, 2023.