Exploration on Dynamic Simulation: USTC-MSRA Innovation Project Report

Haoxiang Guan
Department of Computer Science and Technology
University of Science and Technology of China
jerrykwan@mail.ustc.edu.cn

Jiyan He*
University of Science and Technology of China
hejiyan@mail.ustc.edu.cn

Shuxin Zheng
Microsoft Research AI4Science
shuz@microsoft.com

Tie-Yan Liu
Microsoft Research AI4Science
tyliu@microsoft.com

Abstract

This report is a presentation of USTC-MSRA Innovation Project, with the topic of dynamic simulation. The article is mainly divided into six parts: numerical methods for solving ODEs, n-body problem, MLP and Transformer implementation with vanilla Python, molecular dynamics simulation from scratch, accelerating MD simulation with deep learning and foundation model. Codes and data are freely available at `https://github.com/Jerry-Kwan/MSR-DynamicSimulation` after the report submission deadline (2023-05-30).

1 Introduction

Dynamic simulation is a computational modeling technique used to study the time-dependent behavior of systems. It involves simulating the motion, interactions, and changes in state of objects or entities within a system over a specified period. The objective of dynamic simulation is to gain insights into how the system evolves and behaves over time, under different conditions and forces.

Dynamic simulation finds applications in various fields, such as Robotics and Automation, Aerospace Engineering and so on. In the field of computational chemistry and materials science, molecular dynamics (MD) simulation is a powerful tool to investigate the dynamics and thermodynamics of complex molecular systems, and hence provides valuable insights into various molecular processes, including the folding of proteins, chemical reactions, diffusion, phase transitions, and the properties of materials.

There are mainly two methods for MD simulation, one is ab initio molecular dynamics (AIMD), the other is empirical potentials. The former method obtains the potential energy of the system by solving the Schrödinger Equation of the system, thus obtaining the force acting on the molecules. Although this method is precise, the computational complexity is so high. The latter method consists of a summation of bonded forces associated with chemical bonds, bond angles, and bond dihedrals, and non-bonded forces associated with van der Waals forces and electrostatic charge (Wikipedia, 2023b). This method is faster, but less accurate.

*Intern at Microsoft Research AI4Science.

Currently, artificial intelligence (AI) plays a significant role in many fields, and the field of molecular science is no exception. Machine learning (ML) potentials have become especially attractive with the advent of deep neural network (DNN) architectures, which enable the example-driven definition of arbitrarily complex functions and their derivatives. As such, DNNs offer a very promising avenue to embed fast-yet-accurate potential energy functions in MD simulations, after training on large-scale databases obtained from more expensive approaches (Doerr et al., 2020).

In the following sections of this report, we will explore various aspects of dynamic simulation, including:

1. **Numerical methods for solving ODEs.** Numerical integration is an important step in dynamic simulation, where the equations governing the system dynamics are solved numerically using integration techniques. Therefore, we will firstly investigate some numerical methods.
2. **N-body problem.** Three-body problem is a famous problem in physics and classical mechanics. In this section, we will create a simulation of a dynamical system with any number of particles interacting with each other gravitationally, using different numerical integration techniques.
3. **MLP and Transformer implementation with vanilla Python.** Since PyTorch is used in the subsequent sections, we will firstly implement an MLP and a Transformer with vanilla Python in order to deepen the understanding of the main processes in deep learning frameworks.
4. **Molecular dynamics simulation from scratch.** In this section, we will implement an MD simulation from scratch using Python.
5. **Accelerating MD simulation with deep learning.** In this section, we will train a deep neural network and utilize it to accelerate MD simulation.
6. **Foundation model.** In this section, we will delve into the theories and usage of foundation models, and introduce our attempt to apply foundation models in MD simulation.

2 Numerical methods for solving ODEs

Numerical integration is a crucial component of MD simulation, which is used to approximate the time evolution of the system by dividing it into small time steps and iteratively updating the system state based on the calculated forces and interactions. The smaller the time step, the more accurate the simulation, but at the cost of increased computational resources.

There are various methods for numerical integration, such as the Euler method, Verlet method, Runge-Kutta method and leapfrog method. Here, we primarily analyze the Euler method and the modified Euler method.

2.1 Euler method

The Euler method (also called the forward Euler method) is a first-order numerical procedure for solving ordinary differential equations (ODEs) with a given initial value, which is the most basic explicit method for numerical integration of ordinary differential equations and is the simplest Runge-Kutta method (Wikipedia, 2023a).

When given the values for x_0 and $y(x_0)$, and the derivative of y is a given function of x and y denoted as $y'(x) = f(x, y(x))$. Begin the process by $y_0 = y(x_0)$. Next, choose a value h for the size of every step and set $x_n = x_0 + nh$ (or equivalently $x_{n+1} = x_n + h$). Now, one step of the Euler method from x_n to x_{n+1} is:

$$y_{n+1} = y_n + hf(x_n, y_n). \tag{1}$$

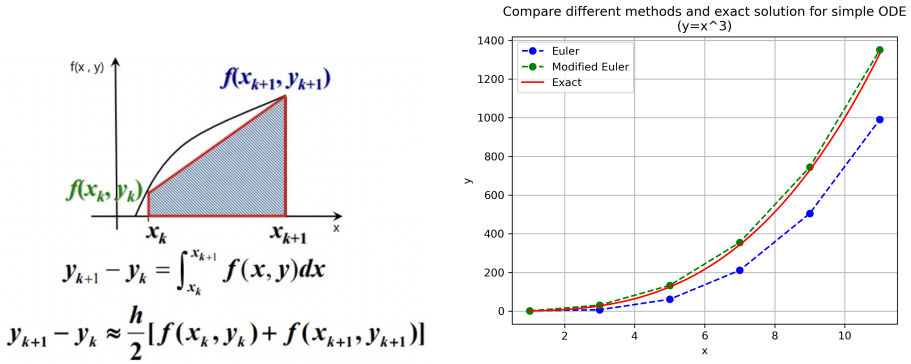
The value of y_n is an approximation of the solution to the ODE at time x_n, i.e. $y_n \approx y(x_n)$. The Euler method is explicit, i.e. the solution y_{n+1} is an explicit function of y_i for $i \leq n$.

Algorithm 1 provides the pseudocode of Euler method.

Algorithm 1: Euler method

Input: Initial condition (x_0, y_0), size of every step $step$, end condition x_{end}, a function named `Derivative` that accepts coordinate (x, y) and returns the corresponding derivative.
Output: Approximation $\hat{\mathbf{y}} = (\hat{y}_1, \ldots, \hat{y}_n)$.

$n \leftarrow (x_{end} - x_0)/step + 1$
$x' \leftarrow x_0$
$\hat{\mathbf{y}} = (y_0, 0, 0, \ldots, 0)_n$
for $i \leftarrow 2$ **to** n **do**
  $\hat{\mathbf{y}}_i \leftarrow \hat{\mathbf{y}}_{i-1} + \texttt{Derivative}(x', \hat{\mathbf{y}}_{i-1}) \times step$
  $x' \leftarrow x' + step$
end
return $\hat{\mathbf{y}}$



(a) (b)

Figure 1: (a) Theory behind modified Euler method, i.e. use trapezoidal area instead of the curvilinear trapezoidal area (TecrayC, 2022). (b) Comparison between Euler method and modified Euler method in the case of a cubic function $y = x^3$.

2.2 Modified Euler method

The Euler forward method may be very easy to implement, but it can't give accurate solutions. One improved approach is to use trapezoidal area instead of the curvilinear trapezoidal area, as shown in Figure 1(a).

The formulae for the modified Euler method is:

$$\hat{y}_{n+1} = y_n + hf(x_n, y_n) \tag{2}$$

$$y_{n+1} = y_n + \frac{h}{2}\left[f(x_n, y_n) + f(x_{n+1}, \hat{y}_{n+1})\right] \tag{3}$$

Algorithm 2 provides the pseudocode of modified Euler method.

2.3 Comparison between Euler method and modified Euler method

We illustrate the differences between Euler method and modified Euler method using the example of a cubic function $y = x^3$. Figure 1(b) shows the comparison. We can observe that as x increases, the error in the Euler method becomes larger, while the results of the modified Euler method remain close to the true values. We calculated the mean absolute error (MAE) between the two methods and the true values, and found that the error for the Euler method is 0.367, while the error for the modified Euler method is 0.047. It is evident that the modified Euler algorithm performs better, which is mainly attributed to its more refined treatment of derivatives.

Algorithm 2: Modified Euler method

Input: Initial condition (x_0, y_0), size of every step $step$, end condition x_{end}, a function named `Derivative` that accepts coordinate (x, y) and returns the corresponding derivative.
Output: Approximation $\hat{\mathbf{y}} = (\hat{y}_1, \dots, \hat{y}_n)$.

$n \leftarrow (x_{end} - x_0)/step + 1$
$x' \leftarrow x_0$
$\hat{\mathbf{y}} = (y_0, 0, 0, \dots, 0)_n$
for $i \leftarrow 2$ **to** n **do**
  $pred \leftarrow \hat{\mathbf{y}}_{i-1} + \texttt{Derivative}(x', \hat{\mathbf{y}}_{i-1}) \times step$
  $\hat{\mathbf{y}}_i \leftarrow \hat{\mathbf{y}}_{i-1} + [\texttt{Derivative}(x', \hat{\mathbf{y}}_{i-1}) + \texttt{Derivative}(x' + step, pred)] \times step/2$
  $x' \leftarrow x' + step$
end
return $\hat{\mathbf{y}}$

3 N-body problem

In this section we will create a simulation of a dynamical system with any number of particles interacting with each other gravitationally, using different numerical integration techniques. Several n-body simulation animations are available at `https://github.com/Jerry-Kwan/MSR-DynamicSimulation/tree/main/projs/x_body/data`.

3.1 Problem definition

In physics and classical mechanics, the n-body problem is the problem of taking the initial positions and velocities (or momenta) of n point masses and solving for their subsequent motion according to Newton's laws of motion and Newton's law of universal gravitation (Wikipedia, 2023h,d).

Let's consider n point masses $m_i, i = 1, 2, \dots, n$ in an inertial reference frame in three-dimensional space $\mathbb{R}^3$ moving under the influence of mutual gravitational attraction. Each mass m_i has a position vector $\mathbf{r}_i$. With Newton's second law and Newton's law of gravity, the n-body equations of motion is given by

$$m_i \frac{d^2\mathbf{r}_i}{dt^2} = \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{G m_i m_j (\mathbf{r}_j - \mathbf{r}_i)}{\|\mathbf{r}_j - \mathbf{r}_i\|_2^3}, \tag{4}$$

where G is the gravitational constant and $\|\mathbf{r}_j - \mathbf{r}_i\|_2$ is the magnitude of the distance between $\mathbf{r}_j$ and $\mathbf{r}_i$.

This is a second-order ODE, and to facilitate solving it, we can decompose it into two first-order ODE:

$$\frac{d\mathbf{r}_i}{dt} = \mathbf{v}_i \tag{5}$$

$$\frac{d\mathbf{v}_i}{dt} = \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{G m_j (\mathbf{r}_j - \mathbf{r}_i)}{\|\mathbf{r}_j - \mathbf{r}_i\|_2^3} \tag{6}$$

3.2 Time Integration

To simulate a dynamic system with n particles, we need to numerically integrate the differential equations of motion, where Euler method and modified Euler method mentioned in Section 2 are used.

Since the n-body equations of motion is a second-order ODE, the form of Euler method for this problem is:

Algorithm 3: Vectorized version of acceleration calculation (in a NumPy style)

Input: An $N \times 3$ matrix of positions P, an $N \times 1$ matrix of masses M, softening length that used to prevent two particles from getting too close to each other $soft$.

Output: An $N \times 3$ matrix of acceleration A

Split P into P_x, P_y, P_z representing the $x/y/z$ coordinates
$D_x \leftarrow P_x^T - P_x$
$D_y \leftarrow P_y^T - P_y$
$D_z \leftarrow P_z^T - P_z$
$R \leftarrow (D_x \odot D_x + D_y \odot D_y + D_z \odot D_z + soft^2)^{-1.5}$
$A_x \leftarrow G \cdot (D_x \odot R) \mathbin{@} M$
$A_y \leftarrow G \cdot (D_y \odot R) \mathbin{@} M$
$A_z \leftarrow G \cdot (D_z \odot R) \mathbin{@} M$
Stack A_x, A_y, A_z to form A
return A

$$\mathbf{v}_{n+1}^i = \mathbf{v}_n^i + \frac{\mathbf{F}_n^i}{m^i} dt \tag{7}$$

$$\mathbf{r}_{n+1}^i = \mathbf{r}_n^i + \mathbf{v}_{n+1}^i dt \tag{8}$$

The form of modified Euler method is:

$$\hat{\mathbf{v}}_{n+1}^i = \mathbf{v}_n^i + \frac{\mathbf{F}_n^i}{m^i} dt \tag{9}$$

$$\mathbf{r}_{n+1}^i = \mathbf{r}_n^i + \frac{\mathbf{v}_n^i + \hat{\mathbf{v}}_{n+1}^i}{2} dt \tag{10}$$

$$\mathbf{v}_{n+1}^i = \mathbf{v}_n^i + \frac{\mathbf{F}_n^i + \mathbf{F}_{n+1}^i}{2m^i} dt \tag{11}$$

This form of modified Euler method is also known as **leapfrog integration method**.

3.3 Acceleration calculation

Based on the aforementioned equations of motion, we can employ a nested loop to calculate the forces acted on each particle, thereby determining the acceleration. However, this approach often suffers from low computation efficiency. The performance can actually be improved in Python by vectorization using packages such as NumPy, i.e. formulating the problem in terms of vector and matrix operations (Mocz, 2020). This improvement can often lead to $100\times$ speedup. The downside is that storing intermediate calculations inside matrices uses significant memory.

Algorithm 3 provides the pseudocode of vectorized version of computing the acceleration on all the particles.

4 MLP and Transformer implementation with vanilla Python

Since PyTorch is used in the subsequent sections, here we will firstly implement an MLP and a Transformer with vanilla Python to enhance the understanding of the theories and frameworks of deep learning. NumPy and CuPy (for GPU acceleration) are used in this section for vectorized computation.

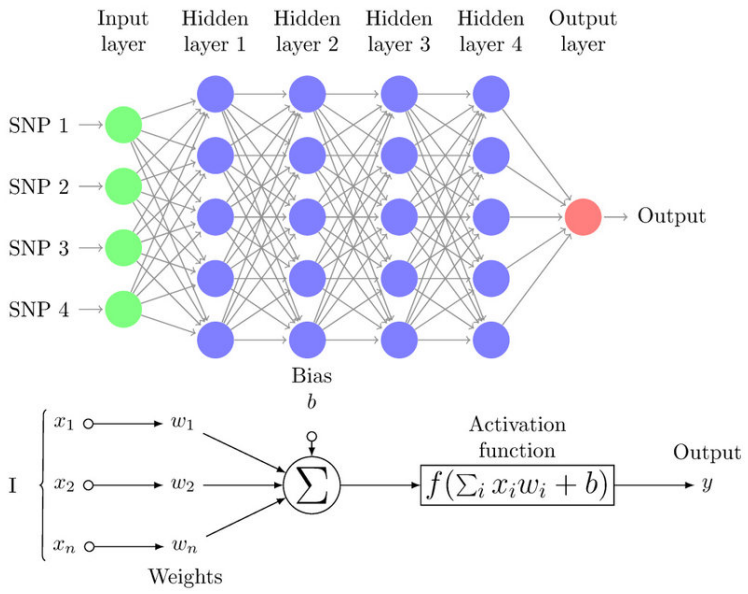


Figure 2: Multi-Layer Perceptron (MLP) diagram with four hidden layers and a collection of single nucleotide polymorphisms (SNPs) as input and illustrates a basic "neuron" with n inputs. One neuron is the result of applying the nonlinear transformations of linear combinations (x_i, w_i, and biases b) (Pérez-Enciso et al., 2019).

4.1 MLP

4.1.1 MLP Architecture

The multilayer perceptron (MLP) is a fundamental artificial neural network architecture and a type of feedforward neural network. It is composed of three or more layers (an input and an output layer with one or more hidden layers) of nonlinearly-activating nodes (Wikipedia, 2023c), as shown in Figure 2.

The input layer of an MLP receives the raw input data, which could be features extracted from a given problem. The hidden layers, located between the input and output layers, perform intermediate computations. These layers help the network learn complex patterns and extract higher-level representations from the input data. The number of hidden layers and the number of neurons in each layer are hyperparameters that can be adjusted based on the problem's complexity and available computational resources.

The output layer of an MLP provides the final results of the network's computations. The number of neurons in the output layer depends on the nature of the problem. For example, in a binary classification task, a single neuron with a sigmoid activation function can be used to produce a probability value between 0 and 1. In a multi-class classification task, the output layer typically consists of multiple neurons, with each neuron representing the probability of a specific class.

4.1.2 MLP forward and backward propagation

An MLP contains two operations: activation functions (ReLU and Softmax are used here) and linear transformations. Cross-entropy is used as a loss function after the output of MLP.

ReLU The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero.

$$\mathrm{ReLU}(x) = \max(0, x) \tag{12}$$

We can compute its derivative as follows:

$$\mathrm{ReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \tag{13}$$

Softmax The softmax function takes as input a vector $\mathbf{z} \in \mathbb{R}^n$ and normalizes it into a probability distribution $\mathbf{o} \in \mathbb{R}^n$ consisting of n probabilities proportional to the exponentials of the input numbers.

$$o_i = \mathrm{Softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{k=1}^{n} e^{z_k}} \tag{14}$$

For backward propagation, we need to firstly compute:

$$\begin{aligned} \frac{\partial o_i}{\partial z_i} &= \frac{e^{z_i} \sum_{k=1}^{n} e^{z_k} - e^{2z_i}}{\left(\sum_{k=1}^{n} e^{z_k}\right)^2} \\ &= o_i(1 - o_i) \end{aligned} \tag{15}$$

When $i \neq j$, we have:

$$\begin{aligned} \frac{\partial o_i}{\partial z_j} &= \frac{-e^{z_i + z_j}}{\left(\sum_{k=1}^{n} e^{z_k}\right)^2} \\ &= -o_i o_j \end{aligned} \tag{16}$$

Assume that $\mathbf{g} \in \mathbb{R}^n$ is a vector with each element g_i representing $\frac{\partial \text{loss}}{\partial o_i}$, then we can compute the derivative of softmax as follows:

$$\begin{aligned} \frac{\partial \text{loss}}{\partial z_i} &= \sum_{k=1}^{n} \frac{\partial \text{loss}}{\partial o_k} \frac{\partial o_k}{\partial z_i} \\ &= g_i o_i (1 - o_i) - \sum_{\substack{k=1 \\ k \neq i}}^{n} g_k o_k o_i \\ &= o_i \left(g_i - \sum_{k=1}^{n} g_k o_k \right) \end{aligned} \tag{17}$$

Linear transformation Linear transformation takes as input a matrix $X \in \mathbb{R}^{n \times p}$ and output $Y \in \mathbb{R}^{n \times q}$ as follows:

$$Y = XW + B, \tag{18}$$

where $W \in \mathbb{R}^{p \times q}$ is a weight matrix and $B \in \mathbb{R}^{n \times q}$ is a bias matrix.

Assume that $G \in \mathbb{R}^{n \times q}$ is a matrix with each element g_{ij} representing $\frac{\partial \text{loss}}{\partial y_{ij}}$, then we can compute the derivative of linear transformation as follows:

$$\begin{aligned}\frac{\partial loss}{\partial x_{ij}} &= \sum_{\substack{1\le s\le n\\1\le t\le q}} \frac{\partial loss}{\partial y_{st}}\frac{\partial y_{st}}{\partial x_{ij}} \\ &= \sum_{1\le t\le q} g_{it}\frac{\partial y_{it}}{\partial x_{ij}} \\ &= \sum_{1\le t\le q} g_{it}w_{jt} \\ &= \sum_{1\le t\le q} g_{it}\left(W^T\right)_{tj} \\ &= \left(GW^T\right)_{ij}\end{aligned} \tag{19}$$

Similarly, we have

$$\frac{\partial loss}{\partial w_{ij}} = \left(X^T G\right)_{ij} \tag{20}$$

As for the bias matrix, we have

$$\begin{aligned}\frac{\partial loss}{\partial b_{ij}} &= \sum_{\substack{1\le s\le n\\1\le t\le q}} \frac{\partial loss}{\partial y_{st}}\frac{\partial y_{st}}{\partial b_{ij}} \\ &= g_{ij}\frac{\partial y_{ij}}{\partial b_{ij}} \\ &= g_{ij}\end{aligned} \tag{21}$$

Cross-entropy Cross-entropy is a measure of the difference between two probability distributions. Assume that we are facing a classification task with n classes, $\mathbf{p} = (0, \ldots, 0, 1, 0, \ldots, 0)$ is a label with only the k-th element being 1, $\mathbf{q}$ is the prediction, then cross-entropy loss would be

$$\begin{aligned}H(\mathbf{p}, \mathbf{q}) &= -\sum_{i=1}^{n} p_i \ln q_i \\ &= -\ln q_k\end{aligned} \tag{22}$$

We can compute its derivative as follows:

$$\frac{\partial H(\mathbf{p}, \mathbf{q})}{\partial \mathbf{q}} = (0, \ldots, 0, -\frac{1}{q_k}, 0, \ldots, 0) \tag{23}$$

4.1.3 Training and testing MLP on MNIST database

Dataset We trained MLP model on MNIST database, which is a large database of handwritten digits. There are 60000 training examples and 10000 test examples in MNIST, whose examples are all 28×28 grayscale image with corresponding label (0-9).

Optimizer Stochastic gradient descent (SGD) is used as the optimizer.

Hyperparameters We use a batch size of 256 examples, a learning rate of 0.0001 and totally 100 epochs. There are 3 hidden layers in MLP, and the number of neurons in each hidden layer is 400, 200 and 100.

Results Training loss curve and testing accuracy curve are shown in Figure 3. From the curve, we can see that there is no overfitting during the training. Our MLP model ultimately achieved a testing accuracy of 0.837 on the MNIST database.

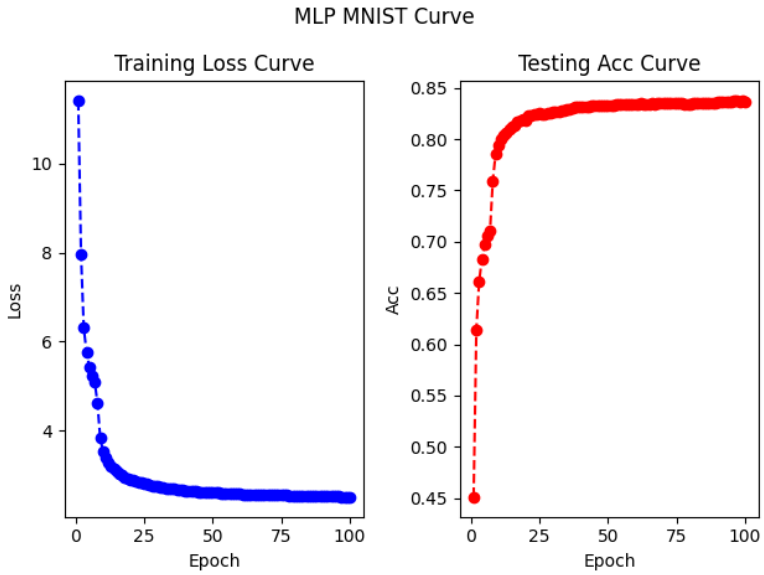


Figure 3: MLP training loss curve and testing accuracy curve on MNIST database.

4.2 Transformer

4.2.1 Transformer Architecture

Transformer (Vaswani et al., 2017) is a groundbreaking neural network architecture that has revolutionized many domains of AI, including natural language processing (NLP), computer vision (CV), and more. This architecture is designed to handle sequential data efficiently and has achieved remarkable success in tasks such as machine translation, text generation, and language understanding. Figure 4 shows the Transformer model architecture, an encoder-decoder design that utilizes self-attention mechanisms.

The encoder part of the architecture processes the input sequence, such as a sentence, word by word. Each word representation is enriched by attending to all other words in the input sequence. This self-attention mechanism allows the encoder to capture dependencies and relationships between words, regardless of their positions in the sequence. The feed-forward design in the encoder introduces additional processing to further refine the representations. It applies a series of fully connected layers to each word's contextualized representation obtained from the attention mechanism. These feed-forward layers help capture and model more complex patterns and relationships within the input sequence. Consequently, the encoder can generate highly informative and context-aware representations for each word.

The decoder part of the architecture takes the encoder's representations as input and generates an output sequence, typically a translation or a response to an input. The self-attention mechanism in the decoder, similar to the encoder, allows the model to capture dependencies and relationships between words in the input sequence. The difference is that the decoder employs masking in its self-attention mechanism to ensure that during the decoding process, each word attends to only previous positions in the output sequence and not future positions, preserving the autoregressive property. Additionally, the decoder employs an attention mechanism called encoder-decoder attention, which enables the decoder to attend to the representations generated by the encoder for the purpose of generating coherent and contextually rich output sequences. Feed-forward design is also incorporated in the decoder.

The positional encoding in the Transformer is a technique used to incorporate information about the position or order of words in the input sequence. Since the Transformer architecture does not have any inherent notion of word order, positional encoding helps the model understand the sequential nature of the input data.

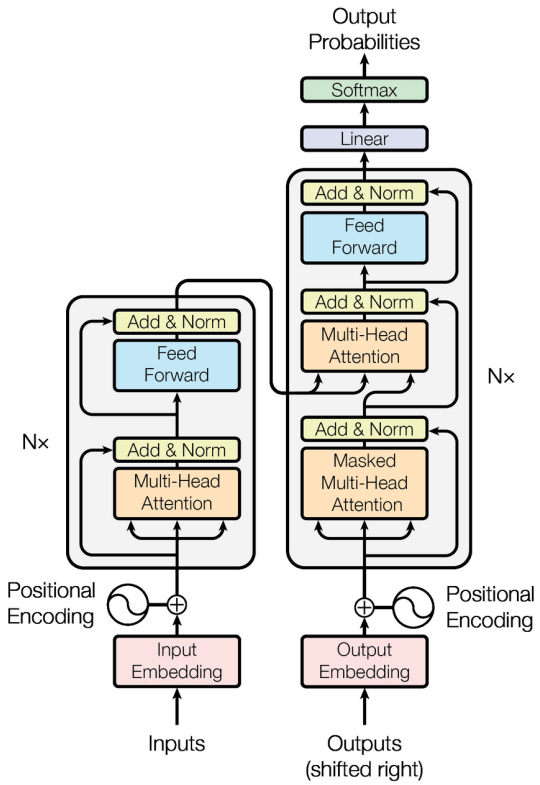


Figure 4: The Transformer - model architecture.

4.2.2 Transformer forward and backward propagation

There are mainly nine operations in Transformer: embedding, positional encoding, multi-head attention, add, layer normalization, position-wise feed-forward, linear transformations, activation functions (ReLU and Softmax are used) and dropout. Cross-entropy is used as the loss function. We will analyze operations that are not mentioned in the MLP.

Embedding Embeddings are used to convert the input tokens and output tokens to vectors of dimension d_{model}. This operation takes as input a vector $\mathbf{x} \in \mathbb{N}^n$ with each element representing the index of word, and will firstly convert it to a one-hot encoding matrix $X \in \{0,1\}^{n \times s}$ where s is the size of vocabulary, then output $Y \in \mathbb{R}^{n \times d_{\text{model}}}$ as follows:

$$Y = XW, \tag{24}$$

where $W \in \mathbb{R}^{s \times d_{\text{model}}}$ is the matrix of word embeddings.

Since the operation is a linear transformation, the computation of derivative is the same as the linear transformation mentioned in MLP.

Positional encoding The positional encoding is added to the input word embeddings at the bottoms of the encoder and decoder stacks. Transformer use sine and cosine functions of different frequencies as positional encoding:

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{\text{model}}}\right), \tag{25}$$

$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{\text{model}}}\right), \tag{26}$$

where pos is the position and i is the dimension.

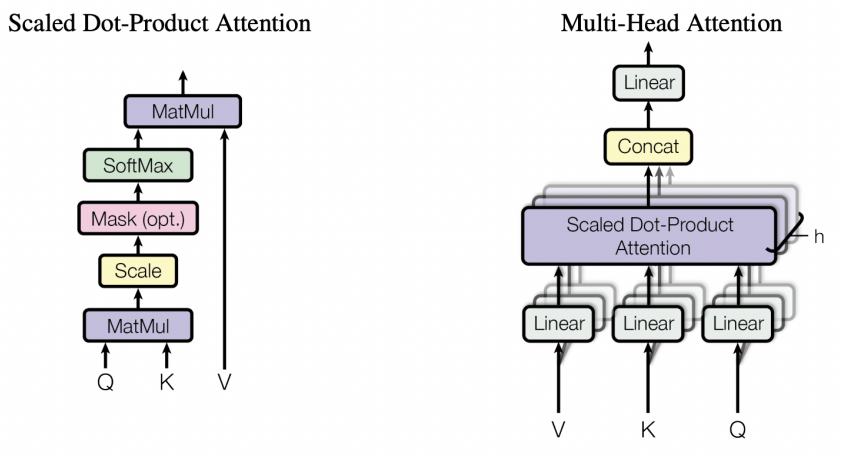


Figure 5: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Since positional encoding is just an addition operation, the derivative of the input embeddings is equal to the derivative of the output embeddings.

Multi-head attention The operation of multi-head attention is shown in Figure 5, which takes as input $Q \in \mathbb{R}^{l_q \times d_{\text{model}}}, K \in \mathbb{R}^{l_{kv} \times d_{\text{model}}}, V \in \mathbb{R}^{l_{kv} \times d_{\text{model}}}$ and output $O \in \mathbb{R}^{l_q \times d_{\text{model}}}$ as follows:

$$Q_i = QW_i^Q, K_i = KW_i^K, V_i = VW_i^V, \tag{27}$$

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i) = \text{softmax}\left(\text{mask}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right)\right) V_i, \tag{28}$$

$$O = \text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) W^O, \tag{29}$$

where $i = 1, 2, \ldots, h$ represents the i-th head and $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_q}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ are corresponding weight matrices used for projection. Usually, we have $hd_q = hd_k = hd_v = d_{\text{model}}$.

Since the derivative of the error is calculated layer by layer during backpropagation, we only need to focus on the derivative of each individual sub-operation to calculate the derivative of the entire multi-head attention.

Whether it is the projection operation of each head on Q, K, V or the scaled dot-product attention of each head, they are all linear transformations. Therefore, the calculation of derivatives is the same as the previously mentioned calculation in MLP. Taking the scaled dot-product attention of head i as an example, assuming that

$$X_i = \frac{Q_i K_i^T}{\sqrt{d_k}}, \tag{30}$$

and $G_i \in \mathbb{R}^{l_q \times l_{kv}}$ is a matrix with each element g_{ij}^i representing $\frac{\partial \text{loss}}{\partial x_{ij}^i}$, then we can calculate the derivative of scaled dot-product attention as follows:

$$\begin{aligned} \frac{\partial \text{loss}}{\partial Q_i} &= \frac{G_i K_i}{\sqrt{d_k}} \\ \frac{\partial \text{loss}}{\partial K_i} &= \frac{G_i^T Q_i}{\sqrt{d_k}} \end{aligned} \tag{31}$$

The derivative calculation of mask operation is simply setting derivatives that have been masked out to 0, and keeping the other derivatives unchanged.

Add The addition operation (He et al., 2016) is also known as shortcut connection or residual connection, where an input X is simply added to the output of the subsequent stacked layers $F(X)$ to form a final output $O = X + F(X)$. Since this is just an addition operation, the derivative of loss with respect to X is the sum of derivative with respect to O and derivative with respect to X considering only $F(X)$.

Layer normalization Layer Normalization (Ba et al., 2016) was proposed by researchers Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Unlike batch normalization, which operates on a batch of samples, layer normalization normalizes the activations on a per-sample basis. Assume that $\mathbf{x} \in \mathbb{R}^d$ is the input vector, layer normalization output $\mathbf{y} \in \mathbb{R}^d$ as follows:

$$\mathbf{y} = \frac{\mathbf{x} - \mu}{\sqrt{\sigma^2 + \epsilon}} \times \boldsymbol{\gamma} + \boldsymbol{\beta}, \tag{32}$$

where $\boldsymbol{\gamma} \in \mathbb{R}^d$ and $\boldsymbol{\beta} \in \mathbb{R}^d$ are learnable parameters, ϵ is a hyperparameter, $\mu = \mathrm{E}[\mathbf{x}]$ and $\sigma^2 = \mathrm{Var}[\mathbf{x}]$.

For backward propagation, we need to firstly calculate:

$$\frac{\partial \sigma^2}{\partial x_i} = \frac{2}{d}(x_i - \mu) \tag{33}$$

Next, we need to calculate:

$$\begin{aligned} \frac{\partial y_i}{\partial x_i} &= \frac{\partial y_i}{\partial x_i}\frac{\partial x_i}{\partial x_i} + \frac{\partial y_i}{\partial \mu}\frac{\partial \mu}{\partial x_i} + \frac{\partial y_i}{\partial \sigma^2}\frac{\partial \sigma^2}{\partial x_i} \\ &= \frac{\gamma_i}{\sqrt{\sigma^2 + \epsilon}} - \frac{\gamma_i}{d\sqrt{\sigma^2 + \epsilon}}\left(1 + \left(\frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}\right)^2\right) \end{aligned} \tag{34}$$

When $i \neq j$, we have:

$$\begin{aligned} \frac{\partial y_j}{\partial x_i} &= \frac{\partial y_j}{\partial \mu}\frac{\partial \mu}{\partial x_i} + \frac{\partial y_j}{\partial \sigma^2}\frac{\partial \sigma^2}{\partial x_i} \\ &= -\frac{\gamma_j}{d\sqrt{\sigma^2 + \epsilon}}\left(1 + \frac{(x_i - \mu)(x_j - \mu)}{\sigma^2 + \epsilon}\right) \end{aligned} \tag{35}$$

Assume that $\mathbf{g} \in \mathbb{R}^d$ is a vector with each element g_i representing $\frac{\partial \mathrm{loss}}{\partial y_i}$, then we can compute the derivative of layer normalization as follows:

$$\begin{aligned} \frac{\partial \mathrm{loss}}{\partial x_i} &= \sum_{j=1}^{d} \frac{\partial \mathrm{loss}}{\partial y_j}\frac{\partial y_j}{\partial x_i} \\ &= \frac{\gamma_i g_i}{\sqrt{\sigma^2 + \epsilon}} - \sum_{j=1}^{d} \frac{\gamma_j g_j}{d\sqrt{\sigma^2 + \epsilon}}\left(1 + \frac{(x_i - \mu)(x_j - \mu)}{\sigma^2 + \epsilon}\right) \end{aligned} \tag{36}$$

As for the derivative of $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$, we have:

$$\frac{\partial \mathrm{loss}}{\partial \boldsymbol{\gamma}} = \frac{\mathbf{x} - \mu}{\sqrt{\sigma^2 + \epsilon}} \odot \mathbf{g} \tag{37}$$

$$\frac{\partial \mathrm{loss}}{\partial \boldsymbol{\beta}} = \mathbf{g} \tag{38}$$

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Figure 6: Pseudocode for Adam.

Position-wise feed-forward Position-wise feed-forward consists of two linear transformations with a ReLU activation in between:

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{39}$$

The calculation of derivative is the same as linear transformation and ReLU mentioned in MLP.

Dropout Dropout is a regularization technique commonly used in deep learning to prevent overfitting and improve the generalization ability of neural networks. It involves randomly deactivating a proportion of neurons or connections in a network during training.

In training:

$$\mathrm{Dropout}(x) = \begin{cases} 0, & p, \\ \frac{x}{1-p}, & 1-p, \end{cases} \tag{40}$$

where p is the dropout rate.

This ensures that $\mathrm{E}[x] = x$, so in testing we have:

$$\mathrm{Dropout}(x) = x \tag{41}$$

As for derivative, we have:

$$\mathrm{Dropout}'(x) = \begin{cases} 0, & \text{if dropout} \\ \frac{1}{1-p}, & \text{if not dropout} \end{cases} \tag{42}$$

4.2.3 Training and testing Transformer

Dataset We trained Transformer model on an English-French dataset that consists of bilingual sentence pairs from the Tatoeba Project. Each line in the dataset is a tab-delimited pair consisting of an English text sequence and the translated French text sequence. There are totally 135842 pairs in the dataset.

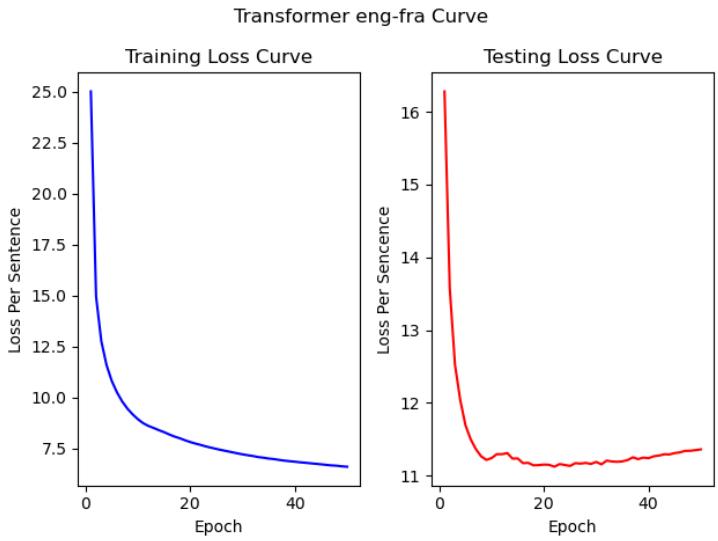


Figure 7: Transformer training loss curve and testing loss curve on an English-French dataset (loss is calculated per sentence).

Optimizer Adam optimizer is used in training. This optimizer calculates the learning rate for each parameter based on the estimates of both the first and second moments of the gradients. Figure 6 shows the pseudocode of Adam optimization algorithm.

Hyperparameters Transformer was built with a 3-layer encoder and a 3-layer decoder. The dimensionality of model is $d_{\text{model}} = 256$, and the inner-layer of position-wise feed-forward has dimensionality $d_{ff} = 512$. Dropout rate is set to 0.1, the number of heads is 8, the ratio of testing data is 0.1. We use a batch size of 64 and trained the model for totally 50 epochs. Adam optimizer is used with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. The number of warmup steps is 4000.

Results Training loss curve and testing loss curve are shown in Figure 7. From the curve, we can observe that overfitting occurs after 25 epochs during the training. The Transformer model achieved the best testing loss of 11.122 on the English-French dataset.

Table 1 displays three translation test examples and their comparison with Google Translate. Figure 8 shows the attention matrices of Transformer for the first translation test example that suggest what the model focus on in the encoder-decoder attention.

Table 1: Three translation test examples and comparison with Google Translate.

Src	**Tgt (Transformer)**	**Tgt (Google)**	**Tgt (Transformer) to Src Using Google**
I love study.	J'adore étudier.	J'aime l'étude.	I love to study.
The whether is good today.	Le temps est beau aujourd'hui.	Le si est bon aujourd'hui.	The weather is nice today.
Do you want to play with me tomorrow?	Voulez-vous jouer avec moi demain?	Tu veux jouer avec moi demain?	Do you want to play with me tomorrow?

5 Molecular dynamics simulation from scratch

In this section, we will implement an MD simulation from scratch using Python, and compare its accuracy and performance with those of OpenMM.

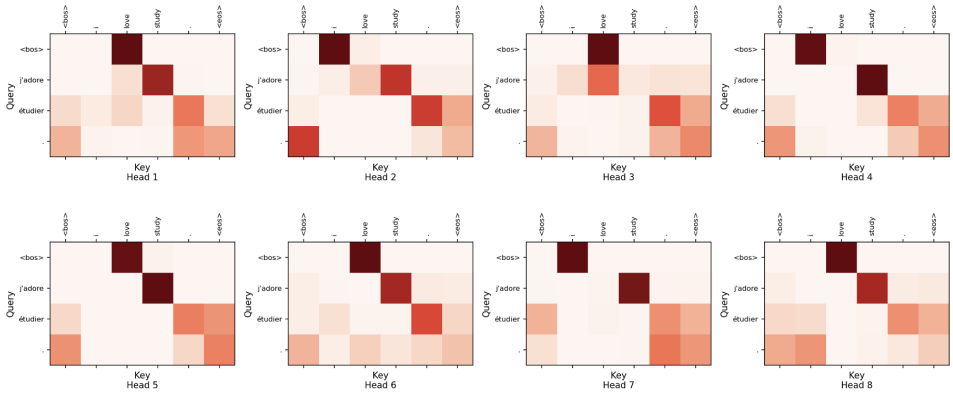


Figure 8: Attention matrices of Transformer for the first translation test example in the encoder-decoder attention. The darker the color, the higher the weight.

5.1 An overview of MD simulation

Molecular dynamics (MD) simulation is a powerful computational technique used to study the behavior and motion of atoms and molecules over time. It provides a detailed understanding of the dynamic properties and interactions within a system at the atomic level, offering valuable insights into various scientific disciplines, including chemistry, materials science, biophysics, and nanotechnology.

At its core, molecular dynamics simulation employs classical mechanics principles to numerically solve the equations of motion for a collection of interacting particles, where the interatomic potentials $U(R)$ are described as a function of the atoms' coordinates R, and forces are derived from interatomic potentials according to:

$$F(R) = -\frac{\partial U}{\partial R} \tag{43}$$

There are mainly two methods for obtaining potentials, one is ab initio molecular dynamics (AIMD), the other is empirical potentials.

AIMD AIMD obtains the potential energy of the system by solving the Schrödinger Equation of the system. Although theoretically it is possible to obtain an exact solution for the potential energy function of a system using AIMD, in reality, the Schrödinger equation cannot be solved exactly for systems with more than two particles. Instead, various approximation methods must be employed. The computational requirements for solving the equation differ significantly depending on the chosen approximation method. Even for the computationally least demanding methods like Hartree-Fock (HF) or density functional theory (DFT), the computational resources required are still substantial (MindScience, 2021).

Empirical potentials Empirical potentials used in chemistry are frequently called force fields, while those used in materials physics are called interatomic potentials. Most force fields in chemistry are empirical and consist of a summation of bonded forces associated with chemical bonds, bond angles, and bond dihedrals, and non-bonded forces associated with van der Waals forces and electrostatic charge. Empirical potentials represent quantum-mechanical effects in a limited way through ad hoc functional approximations. These potentials contain free parameters such as atomic charge, van der Waals parameters reflecting estimates of atomic radius, and equilibrium bond length, angle, and dihedral; these are obtained by fitting against detailed electronic calculations (quantum chemical simulations) or experimental physical properties such as elastic constants, lattice parameters and spectroscopic measurements (Wikipedia, 2023b).

OpenMM is a library for performing molecular dynamics simulations on a wide variety of hardware architectures, behind which the theory is mainly empirical potentials.

In the following section, we will build an MD simulation from scratch based on empirical potentials, and compare its accuracy and performance with those of OpenMM.

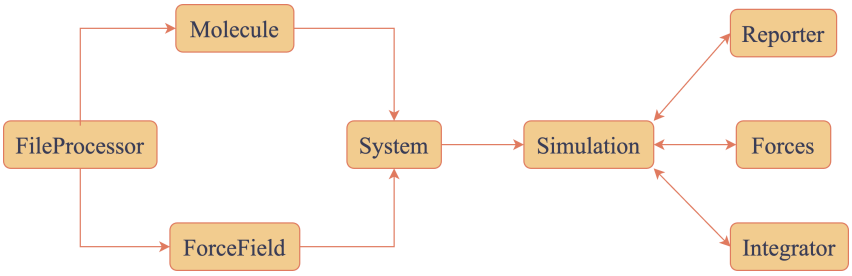


Figure 9: Module design of MD simulation program.

5.2 Module design

Taking inspiration from OpenMM's design, we divide a complete MD simulation program into eight modules: FileProcessor, Molecule, ForceField, System, Simulation, Reporter, Forces and Integrator. Figure 9 illustrates the division and interconnection of these modules.

FileProcessor A complete MD simulation workflow requires multiple files, each with different file formats, such as .pdb, .prmtop, and so on. Different files have different meanings. For example, a PSF file, also called a protein structure file, contains all the molecule-specific information needed to apply a particular force field to a molecular system. If we want to perform a molecular dynamics simulation, we often need a combination of different files in different formats. Therefore, it is often necessary to perform file format conversions and fill in missing information in the files. This module is designed to accomplish these tasks.

Molecule An object of the Molecule class is a representation of the topology of the molecule, which is similar to that of a PDB file, including atom types, atom names, bonds and so on. In addition, the Molecule object stores the shape of the periodic box and the atomic coordinates.

ForceField The ForceField module provides a mapping between bonded and non-bonded interactions involving the various combinations of atom types and specific spring constants and similar parameters. These parameters are used to calculate the bonded and non-bonded potentials according to their functional form.

System A System object represents a molecular system constructed from the combination of a certain Molecule object and a certain ForceField object. In other words, when we apply a force field on a certain molecule, we get a molecular system, which contains all the information except for atomic positions and velocities (these two things would be determined in Simulation).

Simulation The Simulation module is responsible for controlling the entire process of the simulation, which ties together various objects used for running a simulation, including System, Reporter, Forces and Integrator. Initial positions of atoms should be passed to this module. The initial velocities are set to random values chosen from a Maxwell-Boltzmann distribution at a given temperature. This sampling can be obtained by transforming the standard normal distribution since:

$$\begin{aligned} f(v_z) &= \sqrt{\frac{m}{2\pi kT}} \exp\left(-\frac{mv_z^2}{2kT}\right) \\ &= \frac{1}{\sqrt{2\pi}\sqrt{\frac{kT}{m}}} \exp\left(-\frac{(v_z-0)^2}{2\left(\sqrt{\frac{kT}{m}}\right)^2}\right), \end{aligned} \tag{44}$$

which indicates that $v_z \sim N(0, \frac{kT}{m})$.

Usually, before simulation, minimizing the energy of a system is a crucial step to achieve a stable and accurate representation of molecular behavior. Energy minimization aims to find the positions of

atoms that corresponds to the lowest potential energy state, which corresponds to a stable equilibrium structure. Without energy minimization, the simulation may not converge to a stable equilibrium structure, or lead to distorted structures. Various optimization algorithms like steepest descent are commonly employed. In our implementation, L-BFGS-B algorithm is used.

Reporter The Reporter module is used to record various information during the simulation, including potential energy, kinetic energy, system temperature, time and so on.

Forces This module is used to perform calculation of various bonded and non-bonded potentials and corresponding derivatives, including bonds, angles, dihedrals, impropers, Lennard-Jones potential and electrostatics. We will analyze the calculation in the next section.

Integrator This module is used to perform numerical integration during simulation using Velocity Verlet algorithm:

$$\mathbf{r}_i(t+\Delta t) = \mathbf{r}_i(t) + \mathbf{v}_i(t)\Delta t + \frac{\mathbf{F}_i(t)}{2m_i}\Delta t^2 \tag{45}$$

$$\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i(t) + \frac{\mathbf{F}_i(t+\Delta t) + \mathbf{F}_i(t)}{2m_i}\Delta t \tag{46}$$

5.3 Calculation of potentials and forces

In this section we will analyze the calculation of potentials and forces, including bonded terms (harmonic bond, harmonic angle, proper torsion and improper torsion) and non-bonded terms (Lennard-Jones interaction and electrostatic) (OpenMM, 2017).

Harmonic bond Each harmonic bond is represented by an energy term of the form

$$E = \frac{1}{2}k(x - x_0)^2, \tag{47}$$

where x is the distance between the two particles, x_0 is the equilibrium distance, and k is the force constant.

Let $x = \|\mathbf{r}_1 - \mathbf{r}_2\|_2$, then to calculate the force of harmonic bond, we need to firstly calculate:

$$\frac{\partial x}{\partial \mathbf{r}_1} = \frac{\mathbf{r}_1 - \mathbf{r}_2}{\|\mathbf{r}_1 - \mathbf{r}_2\|_2} \tag{48}$$

$$\frac{\partial x}{\partial \mathbf{r}_2} = \frac{\mathbf{r}_2 - \mathbf{r}_1}{\|\mathbf{r}_1 - \mathbf{r}_2\|_2} \tag{49}$$

$$\tag{50}$$

Then the force of harmonic bond is:

$$\mathbf{F}_{1(2)} = -\frac{\partial E}{\partial \mathbf{r}_{1(2)}} = -k(x - x_0)\frac{\partial x}{\partial \mathbf{r}_{1(2)}} \tag{51}$$

Cutoff radius is used in harmonic bond.

Harmonic angle Each harmonic angle is represented by an energy term of the form

$$E = \frac{1}{2}k(\theta - \theta_0)^2, \tag{52}$$

where θ is the angle formed by the three particles, θ_0 is the equilibrium angle, and k is the force constant.

Let atom 2 be the middle atom, then we have

$$* = \frac{\mathbf{r}_{21} \cdot \mathbf{r}_{23}}{\|\mathbf{r}_{21}\|_2 \|\mathbf{r}_{23}\|_2} \tag{53}$$

$$\begin{aligned} \mathbf{F}_1 &= -\frac{\partial E}{\partial \mathbf{r}_1} \\ &= -k(\theta - \theta_0) \frac{\partial}{\partial \mathbf{r}_1} (\arccos(*)) \\ &= \frac{-2k(\theta - \theta_0)}{\sqrt{1 - *^2}} \frac{1}{\|\mathbf{r}_{21}\|_2} \left(* \frac{\mathbf{r}_{21}}{\|\mathbf{r}_{21}\|_2} - \frac{\mathbf{r}_{23}}{\|\mathbf{r}_{23}\|_2} \right) \end{aligned} \tag{54}$$

$$\begin{aligned} \mathbf{F}_3 &= -\frac{\partial E}{\partial \mathbf{r}_3} \\ &= \frac{-2k(\theta - \theta_0)}{\sqrt{1 - *^2}} \frac{1}{\|\mathbf{r}_{23}\|_2} \left(* \frac{\mathbf{r}_{23}}{\|\mathbf{r}_{23}\|_2} - \frac{\mathbf{r}_{21}}{\|\mathbf{r}_{21}\|_2} \right) \end{aligned} \tag{55}$$

As for the force acted on atom 2, we have

$$\mathbf{F}_2 = -\mathbf{F}_1 - \mathbf{F}_3 \tag{56}$$

Proper and improper torsion Each proper torsion is represented by an energy term of the form

$$E = k(1 + \cos(n\theta - \theta_0)), \tag{57}$$

where θ is the dihedral angle formed by the four particles, θ_0 is the phase offset, n is the periodicity, and k is the force constant.

As for improper torsions, we have

$$E = k(\theta - \theta_0)^2 \tag{58}$$

Since the calculation of the derivative of torsion is complex, we use the implementation by OpenMM in our program.

Lennard-Jones interaction The Lennard-Jones interaction between each pair of particles is represented by an energy term of the form

$$E = \frac{A}{r^{12}} - \frac{B}{r^6}, \tag{59}$$

where r is the distance between the two particles, $A = 4\epsilon\sigma^{12}$ and $B = 4\epsilon\sigma^6$ with ϵ being the well depth of the interaction of two atoms and σ the distance at which the energy is zero (Doerr et al., 2020). The calculation of σ and ϵ would use the Lorentz-Berthelot combining rule:

$$\sigma = \frac{\sigma_1 + \sigma_2}{2} \tag{60}$$

$$\epsilon = (\epsilon_1 \epsilon_2)^{1/2} \tag{61}$$

Cutoff is used in Lennard-Jones interaction to directly ignore the interactions between particles beyond that distance. However, a more reasonable approach is to use the switching function to make the energy go smoothly to 0 at the cutoff distance (OpenMM, 2017), which is not implemented in our program. Also, atom pairs including harmonic bonds and harmonic angles are not calculated for Lennard-Jones interaction.

The force of Lennard-Jones interaction can be calculated using the conclusion in harmonic bond:

$$\mathbf{F}_{1(2)} = -\frac{\partial E}{\partial \mathbf{r}_{1(2)}} = (\frac{12A}{r^{13}} - \frac{6B}{r^7})\frac{\partial r}{\partial \mathbf{r}_{1(2)}} \tag{62}$$

Electrostatic Electrostatics are implemented using the following potential:

$$E = k_e \frac{q_i q_j}{r}, \tag{63}$$

where $k_e = \frac{1}{4\pi\epsilon_0}$ is Coulomb's constant, q_i and q_j the charges of the two atoms and r the distance between them. Cutoff is used in electrostatic to directly ignore the interactions between particles beyond that distance. However, a more reasonable approach is to use the reaction field approximation, which is not implemented in our program. Also, atom pairs including harmonic bonds and harmonic angles are not calculated for electrostatic.

The force of electrostatic can be calculated using the conclusion in harmonic bond:

$$\mathbf{F}_{1(2)} = -\frac{\partial E}{\partial \mathbf{r}_{1(2)}} = k_e \frac{q_i q_j}{r^2} \frac{\partial r}{\partial \mathbf{r}_{1(2)}} \tag{64}$$

5.4 Periodic boundary conditions

Periodic boundary conditions (PBCs) are a set of boundary conditions which are often chosen for approximating a large (infinite) system by using a small part called a unit cell (Wikipedia, 2023e). In MD simulation, a periodic box is used as the unit cell that contains the molecules of interest and is replicated periodically in all three dimensions. This replication creates an artificial boundary condition, simulating an infinite system. It effectively mimics the bulk environment, allowing the simulation to capture important collective phenomena, such as phase transitions, diffusion, and long-range interactions.

In our implementation, the interactions between atoms in different boxes are calculated using a **minimum image convention**, which ensures that the closest image of an atom is considered for calculating interactions.

5.5 Experiments and results

In this section we will compare our simulation program with OpenMM in the following four aspects: accuracy, rationality of structure, simulation visualization and performance. We use chemical units consistent with classical MD codes such as ACEMD, namely kcal/mol for energies, K for temperatures, g/mol for masses and Å for distances (Doerr et al., 2020).

Accuracy We utilized our program and OpenMM to calculate the potential energies of three substances, namely, ALA15, benzamidine and 1YTC, under their respective structures, with a cutoff of 9 Å and $T = 300K$. The size of periodic box for ALA15 and benzamidine is 20 Å, and 200 Å for 1YTC.

- **ALA15** is a peptide chain consisting of 15 alanine residues.
- **Benzamidine** is the organic compound with the formula $C_6H_5C(NH)NH_2$.
- **1YTC** is a substance introduced by WA et al. (1996).

Figure 10 shows the results. Our results of bonded terms for benzamidine are very close to those of OpenMM, while there are slight differences for ALA15 and 1YTC. As for non-bonded terms, there are also differences between our results and those of OpenMM.

The differences between the results of our program and those of OpenMM stem from the fact that our calculations for potentials lack the level of precision that OpenMM offers. When calculating Lennard-Jones interaction, OpenMM uses a switching function to make the energy go smoothly to 0 at the cutoff distance, which is not implemented in our program. When calculating electrostatics, OpenMM uses the reaction field approximation to deal with cutoff, which is derived by assuming

		Bond	Angle	Dihedral (proper + improper)	Non-bonded	Total
ALA15	Ours	2.678	22.301	114.447	6068.609	6208.035
	OpenMM	0.766	7.905	101.976	6273.186	6383.833
Benzamidine	Ours	5.918	2.962	2.681	-2.714	8.846
	OpenMM	5.917	2.962	2.681	-6.781	4.779
1YTC	Ours	9543.93	4640.024	2309.266	-1279.307	15213.913
	OpenMM	9562.423	4642.426	2021.954	1602.121	17828.924

Figure 10: Comparison of potential energies (kcal/mol) for three substances between our program and OpenMM.

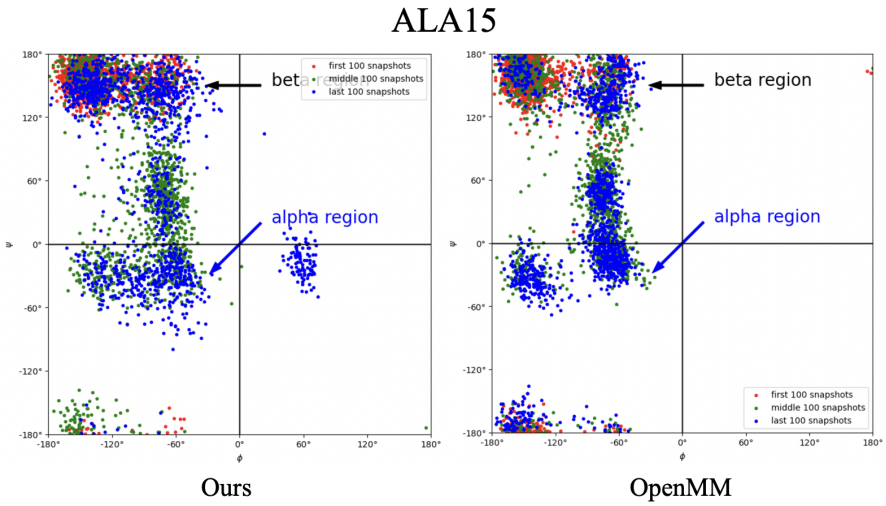


Figure 11: Comparison of the Ramachandran plot for ALA15.

everything beyond the cutoff distance is a solvent with a uniform dielectric constant (OpenMM, 2017). Also, a long range dispersion correction is used in OpenMM. As for the bonded terms, OpenMM has its own definition of chemical bonds for all standard residue types, which could probably lead to differences between the results of our program and those of OpenMM.

Rationality of structure The rationality of molecular structure is an important criterion for evaluating the correctness of MD simulation programs. Therefore, we use our program and OpenMM to run simulations on ALA15 and 1YTC, and compare two results: Ramachandran plots and distance between two atoms during simulation. A Ramachandran plot is a way to visualize energetically allowed regions for backbone dihedral angles of amino acid residues in protein structure, which can reflect the rationality of structure (Wikipedia, 2023f).

In simulation, temperature is set to $300.15K$, cutoff is not used, size of periodic box is 100 Å, time step is 2 femtoseconds and the number of steps is 50000.

Figure 11 shows the Ramachandran plot for ALA15, indicating that the structure gained by our simulation program is similar to that of OpenMM. In addition, we can also observe that our program successfully simulated **alpha-helix** and **beta-sheet** phenomena, which is consistent with chemical principles. Figure 12 shows the distance between an N atom in ALA-1 and an C atom in ALA-5 during simulation. The results of our program show that the distance between these two atoms tends to stabilize and fluctuate between 4 Å and 6 Å, which is similar to the results of OpenMM.

Figure 13 shows the Ramachandran plot for 1YTC. This simulation was started under a stable structure of 1YTC. The result of our program is similar to that of OpenMM.

Simulation visualization See the jupyter notebook at `https://github.com/Jerry-Kwan/MSR-DynamicSimulation/tree/main/projs/md_from_scratch` for the visualization of sim-

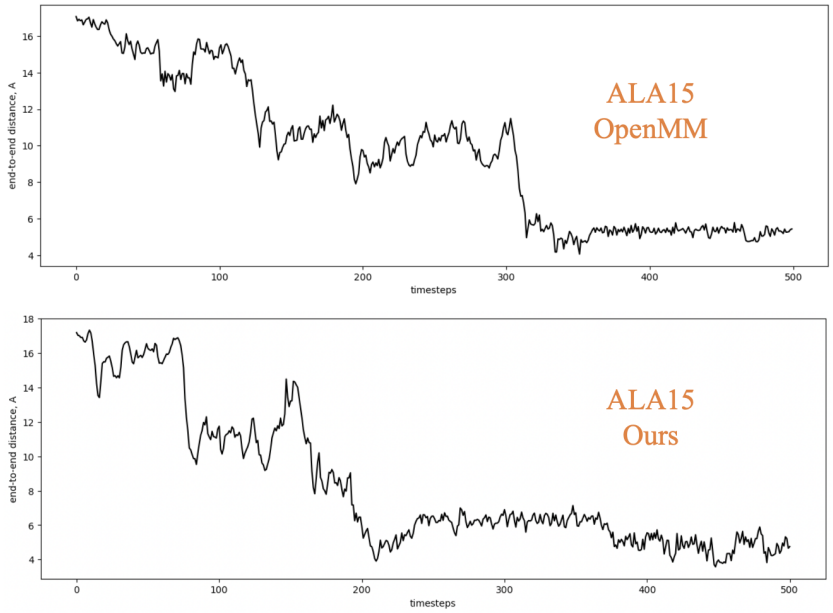


Figure 12: Distance (unit: Angstrom) between an N atom in ALA-1 and an C atom in ALA-5 during simulation.

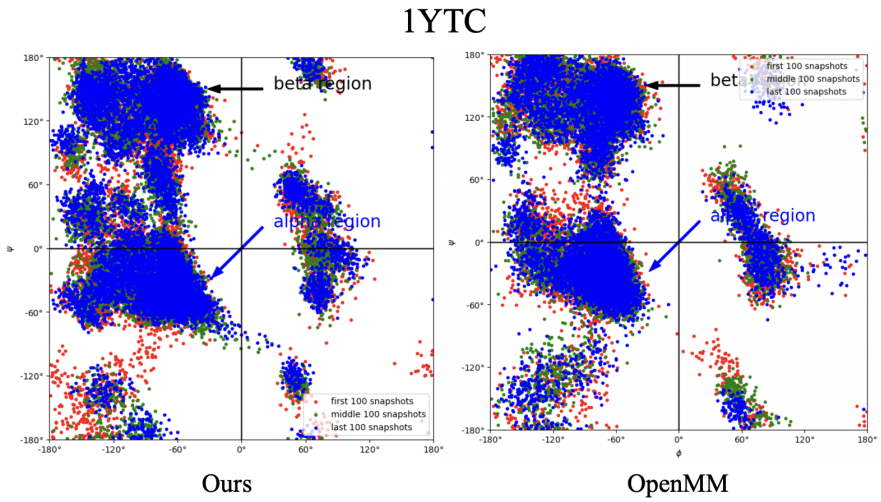


Figure 13: Comparison of the Ramachandran plot for 1YTC.

ulation in our program. The visualization of ALA15 simulation shows **alpha-helix** and **beta-sheet** phenomena, and the visualization of 1YTC simulation indicates that the stable structure of 1YTC can be well maintained.

Performance In terms of time performance, our program falls far behind OpenMM. On one hand, OpenMM utilizes the neighbor list technique in the calculation of non-bonded interactions, which reduces the cost of distance calculations. However, our program does not employ this technique. On the other hand, OpenMM is specialized for MD simulation, while PyTorch (the library we use for the majority of calculation in our program) is a general deep learning framework, which is not as specialized as OpenMM in MD simulation.

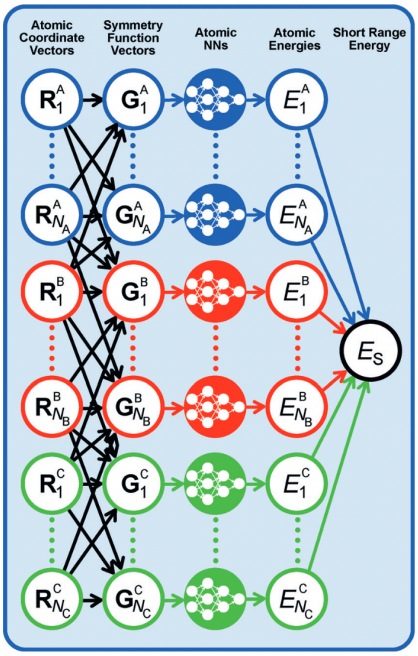


Figure 14: High-dimensional neural network for a ternary system containing elements A, B and C. The numbers of atoms per element are N_A, N_B and N_C, respectively. The total short range energy E_s is the sum of all atomic energies $E_i^X(X = A, B, C)$, which are provided by individual atomic NNs. For a given element, the architecture and parameters of the atomic NNs are the same. The symmetry function vectors $\mathbf{G}_i^X$ provide the information about the local chemical environments of the atoms to the atomic NNs. Consequently, $\mathbf{G}_i^X$ depends on the Cartesian position vectors $\mathbf{R}_i^X$ of all atoms within the cutoff spheres, which is represented by the black arrows. For clarity, only some of these arrows are shown (J, 2017).

6 Accelerating MD simulation with deep learning

6.1 An overview of deep learning in MD simulation

In section 5 we have mentioned two methods for obtaining potentials and forces in MD simulation, i.e. AIMD and empirical potentials. Although AIMD is precise, the computational complexity is so high. In contrast, the latter method is faster, but less accurate.

Currently, artificial intelligence (AI) plays a significant role in many fields, and the field of molecular science is no exception. Machine learning (ML) potentials have become especially attractive with the advent of deep neural network (DNN) architectures, which enable the example-driven definition of arbitrarily complex functions and their derivatives. As such, DNNs offer a very promising avenue to embed **fast-yet-accurate** potential energy functions in MD simulations, after training on large-scale databases obtained from more expensive approaches (Doerr et al., 2020).

There are currently several neural network models available for predicting molecular potential based on molecular configurations. These models can be mainly classified into two categories based on their underlying theory: descriptor-based models and GNN models (MindScience, 2021).

Descriptor-based models Descriptor-based models divide the system into different "descriptors" based on the types of central atoms and their local environments. Each descriptor is associated with a neural network that predicts its local potential energy. The predicted potential energies of all descriptors are then summed to obtain the total potential energy of the system. Representative models in this category include the Behler-Parrinello (BP) model (J and M, 2007) and the DPMD model (Zhang et al., 2017). Figure 14 shows the BP model for a ternary system containing elements A, B, and C.

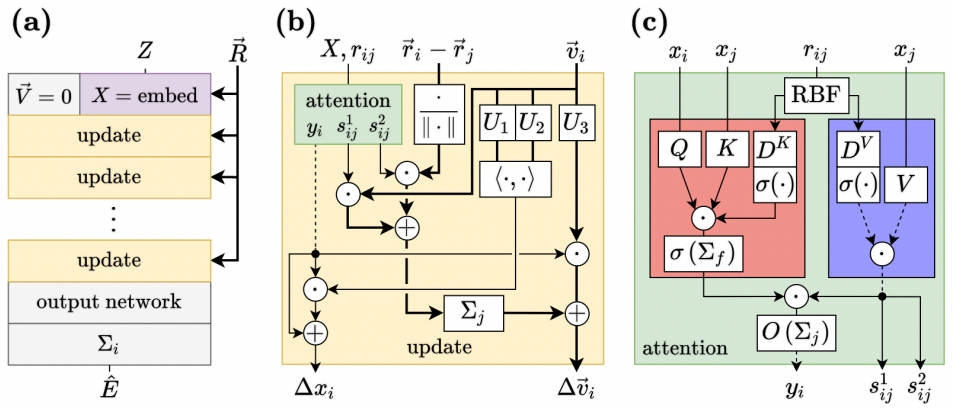


Figure 15: Overview of the equivariant Transformer architecture. Thin lines: scalar features in $\mathbb{R}^F$, hick lines: vector features in $\mathbb{R}^{3\times F}$, dashed lines: multiple feature vectors. **(a)** Transformer consisting of an embedding layer, update layers and an output network. **(b)** Residual update layer including attention based interatomic interactions and information exchange between scalar and vector features. **(c)** Modified dot-product attention mechanism, scaling values (blue) by the attention weights (red). Thölke and Fabritiis (2022)

GNN models GNN models consider a molecular system as a graph, where each atom is represented as a vertex and the distances between atoms are represented as edges. In GNN models, each vertex and edge are represented by embedding vectors. The point vectors of central atoms are updated based on the surrounding point and edge vectors through several iterations. By performing these updates iteratively, the potential energy experienced by each atom is obtained. The total potential energy of the system is then calculated by summing up the potential energies of all atoms. Unlike descriptor-based models that use different neural networks for different types of descriptors, GNN models utilize the same neural network parameters for all atoms during each update iteration. Representative models in this category include MPNN model (Gilmer et al., 2017), Graphormer (Ying et al., 2021) and so on.

6.2 Equivariant Transformer

Equivariant Transformer is a GNN model proposed by Thölke and Fabritiis (2022) for neural network based molecular potentials. It leverages attention mechanism, which is modified in order to include information stored in the graph's edges, corresponding to interatomic distances in the context of molecular data. The model is made up of three main blocks, as shown in Figure 15. An embedding layer encodes atom types Z and the atomic neighborhood of each atom into a dense feature vector x_i. Then, a series of update layers compute interactions between pairs of atoms through a modified multi-head attention mechanism, with which the latent atomic representations are updated. Finally, a layer normalization followed by an output network computes scalar atom-wise predictions using gated equivariant blocks, which get aggregated into a single molecular prediction. This can be matched with a scalar target variable or differentiated against atomic coordinates, providing force predictions.

6.3 Experiments and results

In this section we will use the equivariant Transformer model to run an MD simulation on aspirin.

6.3.1 Dataset

The equivariant Transformer model was trained on the MD17 dataset (Chmiela et al., 2017), which contains ab-initio molecular dynamics trajectories of eight organic molecules. For every trajectory, the dataset contains the Cartesian positions of atoms (in Angstrom), their atomic numbers, as well as the total energy (in kcal/mol) and forces (kcal/mol/Angstrom) on each atom. Trajectories of aspirin were used for training.

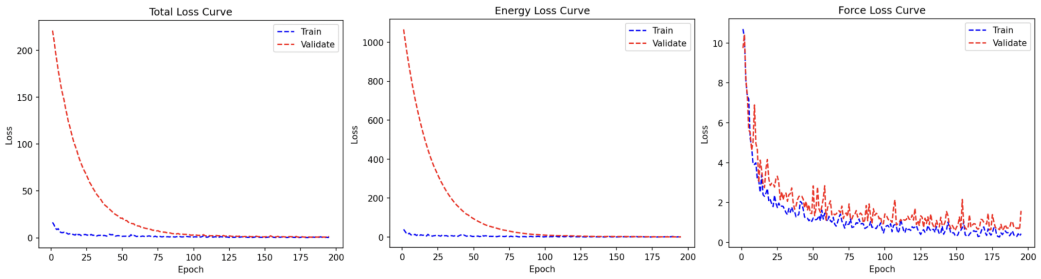


Figure 16: Training loss curve and validating loss curve for total loss, energy loss and force loss (first 200 epochs).

6.3.2 Loss function

Considering that there are two regression targets, namely energies and forces, the loss function used was a combination of the loss of energy predictions and the loss of force predictions. More exactly,

$$\text{loss} = \text{loss}_e + w\text{loss}_f, \tag{65}$$

where loss_e and loss_f are the loss of energy predictions and force predictions, and w is the combination coefficient.

6.3.3 Hyperparameters

The model was trained with an upper cutoff of $5\,\text{Å}$ and a lower cutoff of 0. The batch size is 4 and the number of maximum epochs is 3000. Other hyperparameters can be seen at `https://github.com/Jerry-Kwan/MSR-DynamicSimulation/blob/main/projs/acc_md_with_ai/data/aspirin/et-md17/hparams.yaml`.

6.3.4 Results

The training was performed by the authors of equivariant Transformer. Figure 16 shows the training loss curve and validating loss curve for total loss, energy loss and force loss (first 200 epochs). From the curve, it can be observed that the energy loss converges quickly and steadily, while the force loss converges slower and exhibits larger fluctuations. This is because force is derived from energy. If the neural network is only required to fit the function values, it is relatively easier. However, if fitting the derivatives of the function is also required, the difficulty increases.

We use the equivariant Transformer model to run an MD simulation on aspirin, and record the distance between two O atoms during simulation. The simulation uses a lower cutoff of 10^{-5} Å and an upper cutoff of $5\,\text{Å}$. Temperature is set to $300K$, time step is 2 femtoseconds and the number of steps is 50000. The result shows that the distance fluctuates around $2.8\,\text{Å}$, which is close to that of OpenMM. The visualization of simulation and results can be seen at `https://github.com/Jerry-Kwan/MSR-DynamicSimulation/tree/main/projs/acc_md_with_ai`.

7 Foundation model

AI is undergoing a paradigm shift with the rise of models (e.g., BERT, DALL-E, ChatGPT) trained on broad data (generally using self-supervision at scale) that can be adapted to a wide range of downstream tasks. These models are called foundation models (Bommasani et al., 2021), which have revolutionized the field of AI with their exceptional capabilities and vast potential. Foundation models in NLP possess an excellent ability to comprehend and generate human-like text, enabling them to navigate the complexities of language. In other fields, research on foundation models is also becoming increasingly popular.

Foundation models are pre-trained on a vast amount of publicly available data. One of the current research focuses is how to enhance the capability of these foundation models in a specific domain,

transforming them from generalists to specialists. Since currently lots of foundation models are large language models (LLMs), in this section, we will firstly introduce some solutions to obtain a domain-specific LLM. Next, we will introduce some foundation models in the field of molecular science. At last, we will introduce our attempt to apply foundation models in MD simulation.

7.1 Methods for domain-specific LLMs

Currently, methods for making a domain-specific LLM can be mainly divided into two categories: tuning and plugin-based methods.

Tuning Tuning is a crucial technique to adapt a pre-trained model to a specific task or domain, making it more specialized and effective for a particular application. Based on the strategies used for tuning the model, lots of techniques have been proposed. For example, it is quite common to freeze the parameters in the backbone networks throughout the finetuning procedure, leaving only the head being trained. Low-Rank Adaptation of Large Language Models (LoRA) is a training method that accelerates the training of large models while consuming less memory (Hu et al., 2021). It adds pairs of rank-decomposition weight matrices (called update matrices) to existing weights, and only trains those newly added weights. Reinforcement Learning from Human Feedback (RLHF) is a method proposed by OpenAI (Ouyang et al., 2022) that has achieved good results in aligning language models, whose most recent success was its use in ChatGPT.

Besides tuning the model, prompts can also be tuned. Prompt-tuning is an efficient, low-cost way of adapting an AI foundation model to new downstream tasks without retraining the model and updating its weights (IBM, 2023b). The core insight in prompt-tuning is that the prompts are designed by AI, but not human, which is called "soft prompts". These prompts are learned by backpropagation and can be tuned to incorporate signals from any number of labeled examples. The downside of this method is that it is hard to interpret the prompts designed by AI, as prompt-tuning works in the continuous embedding space rather than the discrete token space (Lester et al., 2021).

Plugin-based methods Plugin-based methods, which do not require any training, take LLMs as black box models, and focus on better handling of input and output of LLMs, as well as various interactions with them. LlamaIndex (LlamaIndex, 2023) a comprehensive toolkit to help perform data augmentation for LLMs without any training. It offers data connectors to ingest your existing data sources and data formats, provides ways to structure your data (indices, graphs) so that these data can be easily used with LLMs, and provides an advanced retrieval and query interface over your data when interacting with LLMs. Browsing is a plugin created by OpenAI to allow ChatGPT to read information from the internet, expanding the amount of content it can discuss and going beyond the training corpus to fresh information from the present day.

7.2 Foundation models in molecular science

Currently, research on foundation models in the field of molecular science is gaining significant momentum. AlphaFold (Jumper et al., 2021) is an AI system developed by DeepMind that predicts a protein's 3D structure from its amino acid sequence, which regularly achieves accuracy competitive with experiment. AlphaFold2 in an improved version of AlphaFold. HelixFold (Wang et al., 2022), proposed by Baidu, is an efficient implementation of AlphaFold2. MolE (Méndez-Lucio et al., 2022) is a molecular foundation model for drug discovery that uses self-supervised pre-training on large unlabeled datasets followed by finetuning on smaller, labeled datasets, whose motivation comes from the success of LLMs. MoLFormer-XL (Ross et al., 2022) is an AI foundation model that learns the grammar of molecules using SMILES notation system (short for Simplified Molecular Input Line Entry System). Each SMILES string describes how atoms in the types of organic small molecules targeted for drug and material discovery are bonded and arranged in a so-called molecular graph. At scale, this meager information contains a wealth of structural clues (IBM, 2023a).

7.3 Our attempt to apply foundation models in MD simulation

Considering that the current research on applying LLMs to specific domains using tuning methods has shown mediocre performance in domains that require high precision, we will seek breakthroughs in the application of foundation models in molecular science.

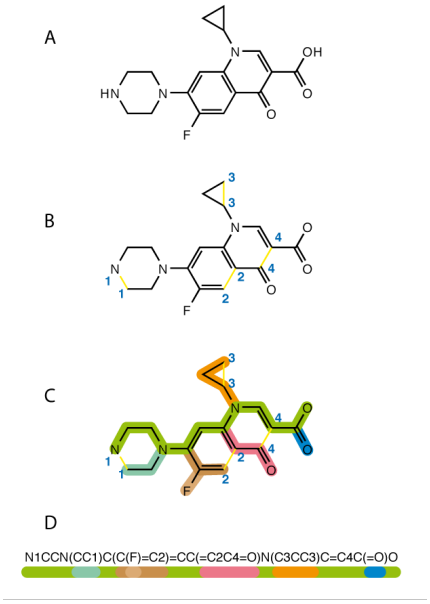


Figure 17: SMILES generation algorithm for ciprofloxacin: break cycles, then write as branches off a main backbone (Wikipedia, 2023g).

Our attempt is based on the work of MoLFormer (Ross et al., 2022), thus it will be introduced first.

7.3.1 MoLFormer

To understand the theories behind MoLFormer, we need to know what SMILES is first.

The simplified molecular-input line-entry system (SMILES) is a specification in the form of a line notation for describing the structure of chemical species using short ASCII strings (Wikipedia, 2023g). Each SMILES string describes how atoms in the types of organic small molecules targeted for drug and material discovery are bonded and arranged in a so-called molecular graph. At scale, this meager information contains a wealth of structural clues. Figure 17 shows the SMILES generation algorithm for ciprofloxacin. The main advantage of these representations of molecules is that it serializes the three-dimensional structure into a one-dimensional sentence, allowing the application of AI methods from NLP.

MoLFormer is an attention-based model that pre-trained on SMILES sequences of 1.1 billion unlabeled molecules from the PubChem and ZINC datasets in a self-supervised fashion (like the training of LLMs). The MoLFormer architecture was designed with an efficient linear attention mechanism and relative positional embeddings with the goal of learning a meaningful and compressed representation of chemical molecules. Experiments show that utilizing the learned molecular representation outperforms existing baselines on downstream tasks. Moreover, further analyzes demonstrate that MoLFormer trained on chemical SMILES indeed learns the spatial relationships between atoms within a molecule, and could distinguish molecules by their flavor, for example, or by their blood-brain barrier permeability, even though the authors never told the model about either property (IBM, 2023a). These abilities of the pre-trained MoLFormer model motivate us, leading to our idea of applying MoLFormer in MD simulation.

7.3.2 Our Idea

Since the pre-trained MoLFormer has learned the spatial relationships between atoms within a molecule and even the properties of molecules, we can deploy the model as an encoder, which can encode the three-dimensional structure of a molecule to embedding vectors that well represent it. These embedding vectors can be heuristics for those neural networks that are specialized for MD simulation or property prediction tasks.

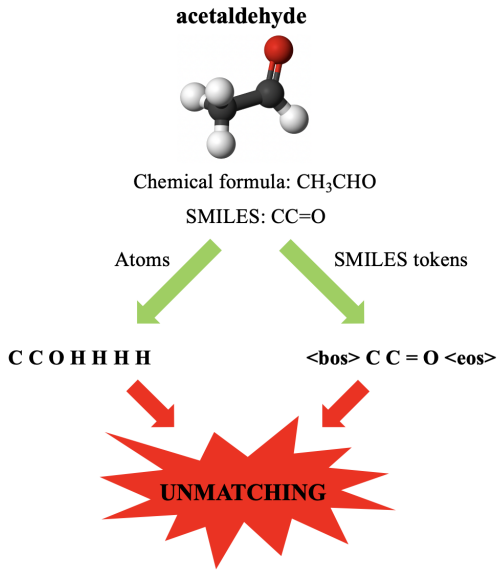


Figure 18: Unmatching between SMILES tokens and atoms.

In section 6 we introduce equivariant Transformer, whose embedding layer assigns an embedding vector to each atom in the molecular according to their element type and neighboring atoms. Currently, with the heuristics coming from MoLFormer, one basic idea is that we replace the original embeddings with the embeddings from MoLFormer, since they can well represent the structure and properties of the molecule.

An obstacle - Unmatching between SMILES tokens and atoms However, things didn't go as smoothly. Problem is that there is unmatching between SMILES tokens and atoms, as shown in figure 18. In the architecture of equivariant Transformer, we need an embedding vector for each atom. However, the tokens of SMILES do not correspond one-to-one with atoms, which prevents us from directly replacing the existing embeddings with those from MoLFormer.

Solution - Attention mechanism Here we propose our solution to the unmatching between SMILES tokens and atoms. Since there is no one-to-one correspondence between them, we keep the original embeddings and use those from MoLFormer as external information for the network. Now, the key issue is how to make full use of this external information. Motivated by the same idea as equivariant Transformer and MoLFormer, we use attention mechanics for interactions between original embeddings and external information from MoLFormer. Figure 19 shows our design. In the modified architecture of equivariant Transformer, not only the information of atomic positions will be attended to, but also the information from MoLFormer that represents the structure and properties of the molecule. In the attention of the original embeddings and external information, queries are the original embeddings, keys and values are external information. This design effectively solves the unmatching between SMILES tokens and atoms, as attention mechanics does not require that the length of queries equals the length of keys.

7.3.3 Experiments and results

Dataset Considering that the significance of foundation models lies in learning representations of numerous molecules, we need to select datasets that contain various types of molecules. QM9 is our choice, which comprises 133885 small organic molecules with up to nine heavy atoms of type C, O, N and F. It reports computed geometric, thermodynamic, energetic, and electronic properties for locally optimized geometries. Since potentials is what we focus on in MD simulation, we only use U_0 (Internal energy at 0K) as our regression target.

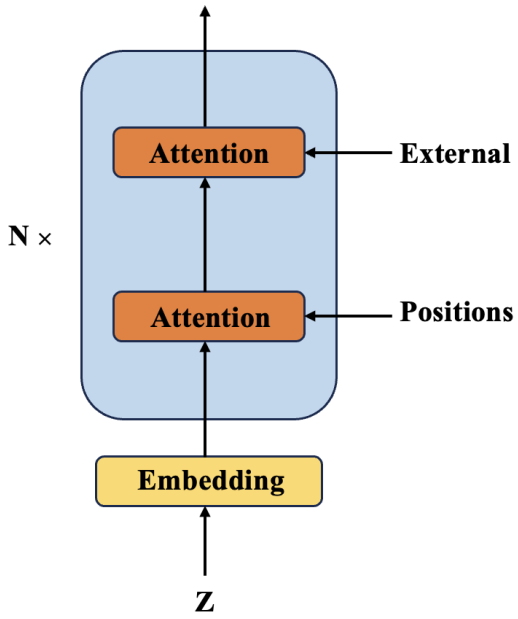


Figure 19: New design of attention mechanism in equivariant Transformer. Here, we use attention mechanism to deal with the unmatching between SMILES tokens and atoms. In the attention of the original embeddings and external information, queries are the original embeddings, keys and values are external information.

Experiment Design A simplified version of equivariant Transformer was used in experiment. As for MoLFormer, we used a checkpoint provided by IBM.

To compare the performance, we designed a control group and two experimental groups. The control group uses the simplified equivariant Transformer. The first experimental group uses the modified equivariant Transformer with all parameters (including those of MoLFormer and equivariant Transformer) being tuned. The second experimental group uses the modified equivariant Transformer with the parameters of MoLFormer frozen, leaving only the modified equivariant Transformer being trained.

Optimizer In the case of all parameters being tuned, we use fused LAMB. As for partial parameters being trained, we use AdamW.

Loss function We use MAE (L1 loss) as loss function.

Hyperparameters Hyperparameters can be seen at `https://github.com/Jerry-Kwan/MSR-DynamicSimulation/tree/main/projs/foundation_model/data/rslt`.

Results Figure 20 shows the validating loss (MAE, unit: Hartree) curve for three experimental groups. We can observe that the network we designed does not perform well when training all the parameters. However, when only tuning the parameters of equivariant Transformer, our model indeed performs slightly better. Our model achieves the best validating loss of 0.00594 Hartree, while the best validating loss for the original model is 0.00655 Hartree.

8 Conclusion

Throughout this project, my knowledge has expanded greatly in the fields of dynamic simulation, molecular science, and deep learning. Delving into the implementation of numerical integration and n-body simulation, I acquired a profound understanding of the underlying theories within dynamic simulation. The endeavor of constructing MLP and Transformer models from scratch, without relying on PyTorch, met challenges in manually calculating derivatives. However, the results were gratifying.

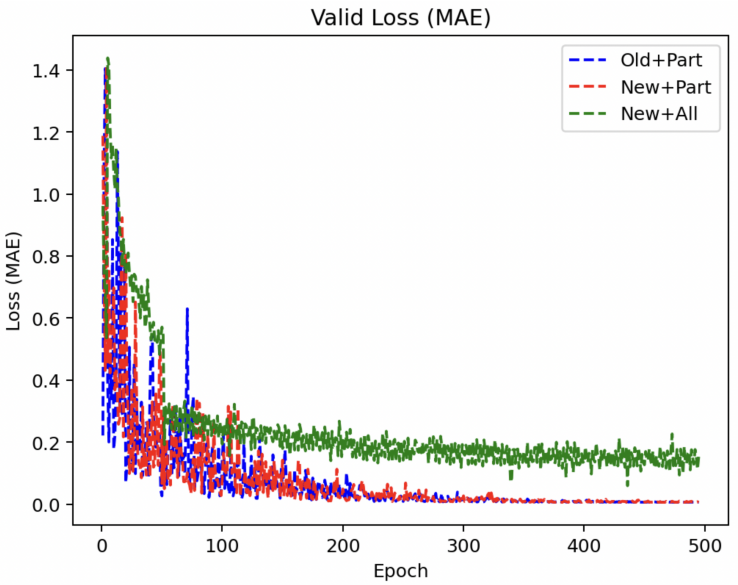


Figure 20: Validating loss (MAE, unit: Hartree) curve for three experimental groups. "Old" means using the simplified equivariant Transformer. "New" means using the modified equivariant Transformer. "Part" means training the parameters of equivariant Transformer only. "All" means training all the parameters.

Additionally, writing a program for molecular dynamics simulation from scratch exposed me to the world of molecular science, igniting my interest for this captivating field. Moreover, exploring the potential applications of deep learning and fundamental models in molecular science unveiled to me the boundless capabilities of artificial intelligence. In summary, this project has proven to be incredibly rewarding, allowing me to acquire a wealth of knowledge and insights.

9 Discussion

In this section, I will list some ideas that popped into my mind during the project. These ideas, although potentially infeasible, retain a degree of inherent value.

1. Currently, the smallest scale in MD simulations is at the atomic level. Is it possible to achieve even finer-grained simulations using AI? I believe there is some potential for that. The motion of electrons is often described using probability distributions represented by electron clouds, and deep learning excels at handling probability distributions. For example, the generator in GAN networks can transform a simple probability distribution into a high-dimensional complex distribution, such as the distribution of human faces. Therefore, deep learning can potentially be used to approximate the distribution of electron clouds.
2. In section 7 we introduce an attention mechanism that incorporate the information from the original embeddings and the pre-trained model MoLFormer. Maybe this could be extended to engage more knowledge from more foundation models. In other words, once a new foundation model in molecular science is proposed, we can add an attention layer in the model specialized for a specific downstream task to incorporate the knowledge from this new foundation model. This is the charm of attention mechanism.
3. Our design of attention mechanics that incorporates the information from the original embeddings and the pre-trained model MoLFormer only led to a slight increase in accuracy. Where can further improvements be made? In machine learning, normalization is an important concept, which means transforming features to be on a similar scale, improving the performance and training stability of the model (Google, 2022). Knowledge from foundation model may not have the similar scale with the original embeddings, therefore normalization

on knowledge from foundation model before it is inputted is essential. However, we have not implemented this.

4. To ensure that our design of attention mechanism is indeed effective, we need to check whether it is an identity mapping.

Acknowledgments

First, I would like to sincerely thank Microsoft Research Asia for providing me with this opportunity to participate in the Innovation Project, from which I have learned a lot about dynamic simulation, molecular science and AI. I would also like to express my gratitude to Mentor Tie-yan Liu and Teaching Assistants Jiyan He and Shuxin Zheng for their guidance throughout the project. I would like to extend my thanks to NeurIPS 2023 for providing the LaTeX template for this report. Lastly, I want to thank my family for their care and support, which has been a tremendous source of motivation for me. Once again, I want to express my gratitude to all my relatives, teachers, classmates, and friends who have shown care, support, and help to me.

References

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016. URL http://arxiv.org/abs/1607.06450. cite arxiv:1607.06450.

R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, E. Brynjolfsson, S. Buch, D. Card, R. Castellon, N. S. Chatterji, A. S. Chen, K. A. Creel, J. Davis, D. Demszky, C. Donahue, M. Doumbouya, E. Durmus, S. Ermon, J. Etchemendy, K. Ethayarajh, L. Fei-Fei, C. Finn, T. Gale, L. E. Gillespie, K. Goel, N. D. Goodman, S. Grossman, N. Guha, T. Hashimoto, P. Henderson, J. Hewitt, D. E. Ho, J. Hong, K. Hsu, J. Huang, T. F. Icard, S. Jain, D. Jurafsky, P. Kalluri, S. Karamcheti, G. Keeling, F. Khani, O. Khattab, P. W. Koh, M. S. Krass, R. Krishna, R. Kuditipudi, A. Kumar, F. Ladhak, M. Lee, T. Lee, J. Leskovec, I. Levent, X. L. Li, X. Li, T. Ma, A. Malik, C. D. Manning, S. P. Mirchandani, E. Mitchell, Z. Munyikwa, S. Nair, A. Narayan, D. Narayanan, B. Newman, A. Nie, J. C. Niebles, H. Nilforoshan, J. F. Nyarko, G. Ogut, L. Orr, I. Papadimitriou, J. S. Park, C. Piech, E. Portelance, C. Potts, A. Raghunathan, R. Reich, H. Ren, F. Rong, Y. H. Roohani, C. Ruiz, J. Ryan, C. R'e, D. Sadigh, S. Sagawa, K. Santhanam, A. Shih, K. P. Srinivasan, A. Tamkin, R. Taori, A. W. Thomas, F. Tramèr, R. E. Wang, W. Wang, B. Wu, J. Wu, Y. Wu, S. M. Xie, M. Yasunaga, J. You, M. A. Zaharia, M. Zhang, T. Zhang, X. Zhang, Y. Zhang, L. Zheng, K. Zhou, and P. Liang. On the opportunities and risks of foundation models. *ArXiv*, 2021. URL https://crfm.stanford.edu/assets/report.pdf.

S. Chmiela, A. Tkatchenko, H. E. Sauceda, I. Poltavsky, K. T. Schütt, and K.-R. Müller. Machine learning of accurate energy-conserving molecular force fields. *Science Advances*, 3(5):e1603015, 2017. doi: 10.1126/sciadv.1603015. URL https://www.science.org/doi/abs/10.1126/sciadv.1603015.

S. Doerr, M. Majewsk, A. Pérez, A. Krämer, C. Clementi, F. Noe, T. Giorgino, and G. D. Fabritiis. Torchmd: A deep learning framework for molecular simulations, 2020.

J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/gilmer17a.html.

Google. Normalization. https://developers.google.com/machine-learning/data-prep/transform/normalization, 2022. [Online; accessed 30-May-2023].

K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '16, pages 770–778. IEEE, June 2016. doi: 10.1109/CVPR.2016.90. URL http://ieeexplore.ieee.org/document/7780459.

E. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models, 2021.

IBM. An ai foundation model that learns the grammar of molecules. https://research.ibm.com/blog/molecular-transformer-discovery, 2023a. [Online; accessed 30-May-2023].

IBM. What is prompt-tuning? https://research.ibm.com/blog/what-is-ai-prompt-tuning, 2023b. [Online; accessed 29-May-2023].

B. J. First principles neural network potentials for reactive simulations of large molecular and condensed systems, 2017. URL https://doi.org/10.1002/anie.201703114.

B. J and P. M. Generalized neural-network representation of high-dimensional potential-energy surfaces, 2007. URL https://doi.org/10.1103/PhysRevLett.98.146401.

J. M. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Zídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. A. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583 – 589, 2021.

B. Lester, R. Al-Rfou, and N. Constant. The power of scale for parameter-efficient prompt tuning, 2021.

LlamaIndex. Welcome to llamaindex. https://gpt-index.readthedocs.io/en/latest/, 2023. [Online; accessed 29-May-2023].

MindScience. Molecular dynamics simulation based on ai force fields. https://gitee.com/mindspore/mindscience/tree/r0.1/MindSPONGE/examples/claisen_rearrangement, 2021. [Online; accessed 27-May-2023].

P. Mocz. Create Your Own N-body Simulation with Python. https://medium.com/swlh/create-your-own-n-body-simulation-with-python-f417234885e9, 2020. [Online; accessed 25-May-2023].

O. Méndez-Lucio, C. Nicolaou, and B. Earnshaw. Mole: a molecular foundation model for drug discovery, 2022.

OpenMM. The Theory Behind OpenMM. http://docs.openmm.org/latest/userguide/theory.html, 2017. [Online; accessed 27-May-2023].

L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback, 2022.

Pérez-Enciso, Zingaretti, and Laura. A guide for using deep learning for complex trait genomic prediction. *Genes*, 10:553, 07 2019. doi: 10.3390/genes10070553.

J. Ross, B. Belgodere, V. Chenthamarakshan, I. Padhi, Y. Mroueh, and P. Das. Large-scale chemical language representations capture molecular structure and properties. *Nature Machine Intelligence*, 4(12):1256–1264, 2022. doi: 10.1038/s42256-022-00580-7.

TecrayC. Numerical Integration — Euler Method, Modified Euler Method, Runge-Kutta Method, Adams. https://zhuanlan.zhihu.com/p/435769998, 2022. [Online; accessed 24-May-2023].

P. Thölke and G. D. Fabritiis. Equivariant transformers for neural network based molecular potentials. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=zNHzqZ9wrRB.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

M. WA, R. FI, L. JR, R.-G. S, L. Y, C. J, B. GD, M. AG, and N. BT. Thermodynamic cycles as probes of structure in unfolded proteins, 1996. URL https://doi.org/10.1021/bi951228f.

G. Wang, X. Fang, Z. Wu, Y. Liu, Y. Xue, Y. Xiang, D. Yu, F. Wang, and Y. Ma. Helixfold: An efficient implementation of alphafold2 using paddlepaddle, 2022.

Wikipedia. Euler method — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Euler%20method&oldid=1153911068, 2023a. [Online; accessed 24-May-2023].

Wikipedia. Molecular dynamics — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Molecular%20dynamics&oldid=1155592559, 2023b. [Online; accessed 23-May-2023].

Wikipedia. Multilayer perceptron — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Multilayer%20perceptron&oldid=1153920594, 2023c. [Online; accessed 25-May-2023].

Wikipedia. N-body problem — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=N-body%20problem&oldid=1153364270, 2023d. [Online; accessed 25-May-2023].

Wikipedia. Periodic boundary conditions — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Periodic%20boundary%20conditions&oldid=1117854491, 2023e. [Online; accessed 27-May-2023].

Wikipedia. Ramachandran plot — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Ramachandran%20plot&oldid=1149481608, 2023f. [Online; accessed 28-May-2023].

Wikipedia. Simplified molecular-input line-entry system — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Simplified%20molecular-input%20line-entry%20system&oldid=1156072880, 2023g. [Online; accessed 30-May-2023].

Wikipedia. Three-body problem — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Three-body%20problem&oldid=1155160273, 2023h. [Online; accessed 25-May-2023].

C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu. Do transformers really perform badly for graph representation? In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=OeWooOxFwDa.

L. Zhang, J. Han, H. Wang, R. Car, and W. E. Deep potential molecular dynamics: a scalable model with the accuracy of quantum mechanics, 2017.