Supplementary Information
Biomimetic Fusion: Platyper's Dual Vision for Predicting Protein-Surface Interactions

Chuhang Hong
Xiaopei Wu
Jian Huang
Honglian Dai

Supplementary Notes

Supplementary Note 1. Gromacs implements SMD simulation of surface system

Molecular dynamics is a method of modeling the motion of molecular systems using Newtonian classical mechanics. The dynamic simulation of the system under study enables the understanding of the motion and biological functions of biological macromolecules, protein-small molecule interaction mechanisms, and self-assembly processes of nanomaterial molecules at the molecular level. A large number of molecules with identical properties and structures can be synthesized under certain macroscopic conditions, in various states of motion and independent of each other. This collection of independent systems with identical properties and structures, in various states of motion, is called a system.

1.1. Steered molecular dynamics

In this paper, the complexes to be used before SMD calculations include the NVT system and the NPT system. The NVT system refers to the molar number of molecules (N), volume (V), and temperature (T) remaining constant, with energy exchange with the outside world. As the system has an energy exchange with the outside world, certain methods are needed to maintain a constant temperature. On the other hand, the NPT system refers to the number of molecules (N), pressure (P), and temperature (T) remaining constant. In addition to maintaining constant temperature, the system also requires controlling constant pressure.

In this paper, we employ the Umbrella sampling technique for performing SMD simulations using Pull Code. It is worth mentioning that Pull Code encompasses three types: AFM pulling, Umbrella sampling, and constraint. We add a resonance potential to the center of mass of a group of atoms. This potential helps in maintaining the position of the pull group relative to the reference group. The setting of Pull_geometry is crucial, as it encompasses four modes: position, distance, direction, and cylinder. In this experiment, the distance mode is employed, whereby the pull is always directed along the vector connecting the reference group to the pull group. Consequently, if the reference group moves, the pull group moves along with it. In cases where the reference group is not defined, the pull group is positioned in absolute coordinates and the reference group is assigned as [0 0 0]. By utilizing this mode, the spring point will exclusively move in the direction of the line linking the reference group to the pull group.

1.2. Umbrella sampling

For biological macromolecules, some large conformational changes are difficult or impossible to reach within the existing range of simulation scales. In such cases, one often employs the means of adding external biasing forces to the system to accelerate the process. This is where umbrella sampling, an accelerated sampling method, comes into play. The process of stretching dynamics (SMD) involves applying a simple harmonic force, similar to a spring, to a specific part of the macromolecule. This force pulls the macromolecule at a constant velocity. As

a result, the region being pulled is displaced, and a number of conformations are selected during this period. Independent simulations are then performed using these conformations as the initial frames. Finally, the simulations are integrated using weighted histograms to derive the potential of mean force (PMF) for the energy change during the traction process. The PMF is a valuable tool for calculating the effective interaction between two complex molecules in a liquid medium. The standard approach to calculating the PMF along a reaction coordinate is as follows: simulations are conducted individually along the reaction coordinate ξ and influenced by the umbrella potential $w_i(\xi)$ described in Eq. l, where the system's position ξ_i is constrained by the force constant K_i.

$$w_i(\xi) = \frac{K_i}{2}\left(\xi - \xi_i\right)^2 \tag{1}$$

The PMF can be obtained by applying Equation 2 to the unbiased probability distribution of the system, with the condition that the PMF is zero at an arbitrary point ξ_0.

$$W(\xi) = -k_{\mathrm{B}}T \ln\left(\frac{P(\xi)}{P\left(\xi_0\right)}\right) \tag{2}$$

The Weighted Histogram Analysis Method (WHAM) in the GROMACS simulation package was used to derive the unbiased probability distribution. To calculate the statistical error, we employed the Bayesian bootstrap method, utilizing the complete histograms provided by the g_wham program in GROMACS. This analysis demonstrates the accurate estimation of the standard deviation function of the Potential of Mean Force (PMF) using the Bayesian bootstrap method in umbrella sampling.

Supplementary Note 2. Attention Mechanisms

Attention Mechanisms, also known as the ability to focus attention on important areas of an image and discard irrelevant ones, play a crucial role in computer vision. Inspired by the human visual cortex, Attention Mechanisms were introduced to computer vision by researchers to enhance performance in analyzing complex scene information. There are four major categories of attention mechanisms: Channel Attention, Spatial Attention, Temporal Attention, and Branch Attention.

2.2 Self-Attention

Self-attention mechanism is a kind of Attention mechanism. Unlike the general attention mechanism, the self-attention mechanism is not an Attention mechanism between an input feature and an output feature, but an Attention mechanism that occurs between elements within an input feature or between elements within an output feature.

Self-attention models often use the QKV (Query-Key-Value) model, for each vector a, multiply three coefficients , respectively, to get three values of Q,K,V :

$$\begin{aligned} Q &= W^q \cdot 1 \\ K &= W^k \cdot 1 \\ V &= W^v \cdot 1 \end{aligned} \tag{3}$$

The obtained Q,K,V denote query, key and value respectively.The weight parameter W^q,W^k, W^v are iterated during the network update, as shown in Supplementary Figure 3. This process can be summarized in the formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{4}$$

When the mean of each component is determined, the variance of the dot product of the individual components of vectors Q and K, denoted as d_k, can be calculated. Dividing by $\sqrt{d_k}$ allows the variance to be scaled appropriately, ensuring that the components generating the largest variance are not excessively magnified. Consequently, the larger the variance, the more skewed is the component that produces the largest.

Despite considering all input vectors, the self-attention mechanism fails to incorporate positional information, hindering our ability to approach the problem from an image perspective.

2.2 Convolutional Block Attention Module (CBAM)

CBAM, which combines channel and spatial attention, can be considered as a mixture of these two types of attention.The CBAM module is a lightweight module that can be embedded into any backbone network to improve performance. It is able to sequentially generate attention feature map information in both channel and spatial dimensions, given a feature map. The feature map information is then multiplied with the previous original input feature map for adaptive feature correction to produce the final feature map. CBAM produces 1D channel attention feature maps $M_c \in R^{C \times 1 \times 1}$ and 2D spatial attention feature maps $M_s \in R^{1 \times H \times W}$ for the feature maps $F \in R^{C \times H \times W}$ generated by the network backbone, respectively. This process can be described as Eq.3, which denotes the element-level multiplication, with the intermediate dimensional transformation and matching using the broadcast mechanism.

$$F' = M_c(F) \otimes F$$
$$F'' = M_s(F') \otimes F' \tag{5}$$

The channel attention mechanism is based on the relationship between the features. Each channel of the feature map is treated as a feature detector, allowing the channel features to concentrate on identifying the useful information in the image. To enhance the efficiency of computing the channel attention features, the spatial dimension of the feature map is compressed using two methods: average pooling F_{avg}^{c} and maximum pooling F_{max}^{c}. The corresponding average pooling feature and maximum pooling feature are denoted as and , respectively. Subsequently, the feature is inputted into a shared multilayer perceptron (MLP) network to generate the final channel attention feature map $M_c \in R^{C \times 1 \times 1}$. To reduce the computational parameters, a dimensionality reduction coefficient, r ($M_c \in R^{C/r \times 1 \times 1}$), is employed in the MLP. Equation 4 summarizes the channel attention computation formula.

$$\begin{aligned} M_c(F) &= \sigma(MLP(AvgPool(F)) + MLP(MaxPool(F))) \\ &= \sigma\left(W_1\left(W_0\left(F_{\text{avg}}^{c}\right)\right) + W_1\left(W_0\left(F_{\max}^{c}\right)\right)\right) \end{aligned} \tag{6}$$

The spatial attention mechanism, unlike channel attention, directs attention towards the "where" of the effective information on the feature map. To compute spatial attention, the feature map undergoes average pooling and maximum pooling in the channel dimension. The resulting feature maps are then concatenated, and a convolution operation is performed on the concatenated feature map to generate the final spatial attention feature map $M_s(F) \in R^{H,W}$. Similar to the channel attention mechanism, two pooling methods are employed in the channel dimension to produce 2D feature maps, $F_{avg}^{s} \in R^{1 \times H \times W}$ and $F_{avg}^{s} \in R^{1 \times H \times W}$. The spatial attention formulation can be summarized in Equation 5.

$$\begin{aligned} M_s(F) &= \sigma\left(f^{7\times7}([AvgPool(F); \max Pool(F)])\right) \\ &= \sigma\left(f^{7\times7}\left(F_{\text{avg}}^{s}; F_{\max}^{s}\right)\right) \end{aligned} \tag{7}$$

Supplementary Note 3. Platyper model's training parameters

Understanding tuning techniques and common optimization strategies is crucial to training efficient models because deep learning models have many hyperparameters that need to be experimented with and tuned to optimize their performance. These hyperparameters, including learning rate, batch size, and number of layers, among others, play a crucial role in determining the effectiveness of the model. Therefore, researchers and practitioners must carefully select and tune these hyperparameters to achieve optimal results. The training process was conducted using an NVIDIA RTX 4080 graphics card and a Dell Precision 7920 workstation with dual Intel Xeon Silver 4210R CPUs.

3.1 Gradient descent

The weights θ in the model are updated through stochastic gradient descent, represented as Equation 6, where α is the learning rate. The objective function $J(\theta)$, which is minimized or maximized, is referred to as the loss function when minimizing, shown in Equation 7 and Equation 8.

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{8}$$

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} \left(h_\theta(x) - y \right)^2 \tag{9}$$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots\ldots + \theta_n x_n \tag{10}$$

During training, multiple iterations on the entire dataset will be performed, known as epochs. The gradient $\frac{\partial}{\partial \theta_j} J(\theta)$ is obtained through backpropagation.

3.2 Batch Size

Batch Size is an important concept in deep learning, referring to the number of samples inputted simultaneously during each model training. In the training process of neural networks, data is usually divided into several batches, with each batch containing a specific number of samples. Batches denote a collection of samples simultaneously inputted into the neural network, enabling the utilization of the parallel computing capabilities of hardware acceleration, such as GPUs, to enhance training efficiency. Compared to the complete batch training with all samples inputted at once, batch processing is more effective in utilizing computational resources. A larger Batch Size typically enhances training speed as it allows for the simultaneous processing of multiple samples in each training step, thereby better utilizing hardware acceleration, which is particularly crucial for large-scale data and complex models. Conversely, a smaller Batch Size may contribute to better generalization to unseen data as it introduces some noise similar to the regularization effect, thus preventing overfitting. In deep learning, parameter updates are typically achieved through the gradient descent algorithm, and Batch Size influences the calculation of gradients in each update step. A larger Batch Size usually provides more stable gradients, although at times, it may lead to convergence to local minima.

3.3 Number of epochs

The number of epochs has a significant impact on the learning process of a model as it controls the number of times the model learns about the entire dataset. This repeated exposure to the dataset allows the model to gradually learn patterns and features, thus improving its performance over time. The assessment of performance on both the training and validation sets is critical in determining whether the model is overfitting. If the model performs well on the training set but its performance deteriorates on the validation set, it is likely overfitting. Proper control of epochs is necessary to prevent overfitting. Moreover, the number of epochs is also closely linked to the convergence of the model. As epochs increase during training, there comes a point where the model's performance stabilizes and stops showing significant improvement. Consequently, further increasing the number of epochs may not yield substantial performance gains.Batch Size and number of training steps for Epochs N can be expressed by the following equation:

$$N = \frac{S}{B} \times E \tag{11}$$

,where S is the number of samples in the training set, B is Batch Size, and E is Epochs.

3.4 Learning Rate

The learning rate is an essential hyperparameter in machine learning and deep learning, as it controls the magnitude of updates to the model parameters during each iteration. It determines the step size while moving along the gradient direction in gradient descent, thereby influencing the tuning of model parameters. Consequently, the choice of learning rate directly impacts the convergence speed of the model. Using a large learning rate might lead to oversized parameter updates, causing the model to oscillate and struggle to converge. Conversely, a smaller learning rate may result in slow model convergence. Thus, an appropriate learning rate can expedite convergence and enhance training efficiency. The choice of learning rate also influences the generalization performance of the model, with a smaller learning rate contributing to better generalization by providing greater stability during training and reducing the risk of overfitting. Maintaining the stability of the model is crucial, as an excessively large learning rate can lead to divergence during training while an overly small learning rate might trap the model in local minima. Consequently, practitioners commonly experiment with various learning rate values to identify the optimal one for a given task and model structure. Additionally, learning rate decay, such as exponential decay or cosine annealing, is often employed to further fine-tune the parameters as training progresses.

3.5 Activation Function

The activation function in a neural network is a crucial nonlinear operation within a neuron or layer, receiving inputs and producing outputs. This function introduces nonlinear properties that enable the neural network to learn and represent complex patterns and relationships. By nonlinearly mapping the inputs, the activation function allows the network to automatically learn and characterize features in the data, enhancing its ability to handle different types of features. Moreover, it introduces the necessary nonlinear transformations, enabling the network to learn and represent nonlinear relationships, which are essential for modeling real-world data that contains complex and non-linear patterns. Consequently, the choice of activation function significantly influences the expressive power of the neural network, making it easier to train and helping to overcome issues such as gradient vanishing or explosion. Furthermore, the output range of an activation function can be tailored to meet specific task requirements, for instance, the sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

(12)

having an output range between 0 and 1, is suitable for binary classification problems, while the tanh function

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

(13)

, with an output range of -1 to 1, serves a similar purpose but offers a different range. The Rectified Linear Unit (ReLU) function

$$ReLU(x) = max(0,x) \quad (14)$$

, widely employed in hidden layers, introduces nonlinearity. It is essential for increasing the model's expressive power. This function effectively addresses the issue of vanishing gradients associated with linear activation functions, contributing to improved accuracy and convergence in deep learning models. Furthermore, the Leaky ReLU function

Leaky $ReLU(x) = max(\alpha x,x)$ (15)

, which introduces a small positive number (α), resolves the problem of negative values that can occur with the standard ReLU function. This enhancement enables the Leaky ReLU to maintain gradient flow and address the "dying ReLU" problem, a limitation of the standard ReLU function. In contrast, the Softmax function

$$Softmax(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

(16)

, commonly utilized in the output layer of a multi-classification problem, is instrumental in transforming the output into a probability distribution. By producing normalized class probabilities, Softmax facilitates the interpretation of the model's predictions, supporting effective decision-making in practical applications.

Supplementary Note 4. Platyper model's evaluation protocols

Seven metrics are used to indicate the difference between the model predictions and the simulated values of the test dataset as a means of assessing model performance . $y_i - \hat{y}_i$ represents the discrepancy between the true value and the test value on the test set.

4.1 R-squared (R^2)

R-squared (R^2) is a metric used to assess the goodness-of-fit of a regression model. It takes values between 0 and 1 and indicates the proportion of the variance of the dependent variable (target variable) that is explained by the model. The closer R^2 is to 1, the better the model explains the data. The model's ability to explain the data is weaker when R^2 is closer to 0. In contrast, a higher R^2 value suggests a stronger explanation of the data by the model.

$$\mathbf{R}^2 = 1 - \frac{\sum_i (\hat{y}_i - y_i)^2}{\sum_i (\bar{y}_i - y_i)^2} \tag{17}$$

4.2 Mean Squared Error(MSE)

Mean Squared Error(MSE) measures the difference between the predicted value and the true value by taking the square of the mean error. A smaller MSE indicates a better model accuracy, as it signifies that the predicted value is closer to the true value.

$$\mathrm{MSE} = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i) \tag{18}$$

4.3 Root Mean Squared Error(RMSE)

Root Mean Squared Error(RMSE) is the square root of MSE, representing the mean prediction error. It shares the same units as the original data and, like MSE, lower values indicate smaller prediction errors of the model.

$$\mathrm{RMSE} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2} \tag{19}$$

4.4 MAE (Mean Absolute Error)

MAE (Mean Absolute Error) calculates the average of the absolute differences between predicted and true values. Unlike MSE, which squares the error, MAE is more sensitive to large errors because it does not incorporate the squaring step.

$$\mathrm{MAE} = \frac{1}{m}\sum_{i=1}^{m}\left|\left(y_i - \hat{y}_i\right)\right| \tag{20}$$

4.5 The Median Absolute Error (MedianAE)

The Median Absolute Error (MedianAE) is a measure of the absolute difference between predicted and true values, taking the median value. Unlike MAE, MedianAE is resistant to outliers and prevents outliers from influencing the measure.

$$MedianAE = median\left(\left|y_1 - \hat{y}_1\right|, \ldots\ldots, \left|y_i - \hat{y}_i\right|\right) \tag{21}$$

4.6 The mean absolute percentage error (MAPE)

The mean absolute percentage error (MAPE) is a metric that is sensitive to relative error. It remains unaffected by global scaling of the target variable and is appropriate for situations with a wide difference in the target variable's magnitude.

$$\mathrm{MAPE} = \frac{1}{m}\sum_{i=1}^{m}\frac{\left|\left(y_i - \hat{y}_i\right)\right|}{\max\left(\dot{o}, |y_i|\right)} \tag{22}$$

4.7 Explained Variance assesses

Explained Variance assesses the model's ability to explain the variance of the target variable and quantifies the improvement in the model's predictive power compared to the simple mean.

$$\mathrm{Explained\_Variance} = 1 - \frac{Var\{y - \hat{y}\}}{Var\{y\}} \tag{23}$$

Supplementary Note 5. Gradient-based Class Activation Mapping

5.1 Class Activation Mapping

Class Activation Mapping (CAM) is a technique that generates a map of the same size as the original image. The pixels on this map have values ranging from 0 to 1, and it is commonly represented as a grayscale map from 0 to 255. CAM can be visualized using a heat map overlayed on the original image, as shown in Fig5 where the red highlighted area serves as the main judgment basis. The process of acquiring CAM involves several steps. Firstly, the feature layer to be visualized is extracted. Then, the weight of each channel of this tensor is obtained. Next, the tensor is linearly fused, with the weighted and summed channel dimension resulting in a smaller-sized map. Lastly, the map is normalized and resized to the original size through interpolation.

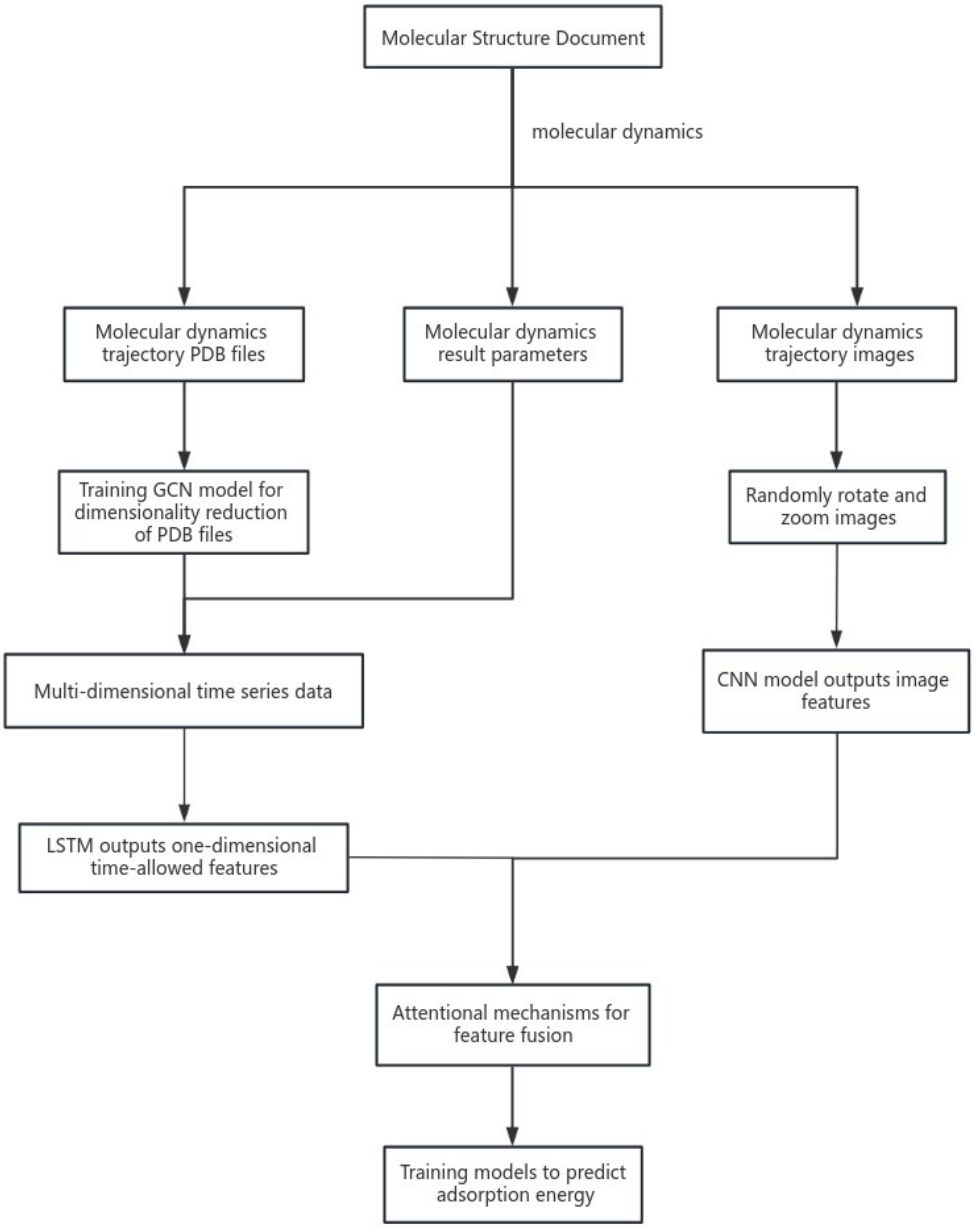
5.2 Gradient-based Class Activation Mapping

Gradient-based Class Activation Mapping (Grad-CAM) is a versatile method used for obtaining Class Activation Maps (CAMs) in CNN-based networks. Initially, CAM is obtained by modifying the global average pooling (GAP) form of the network, utilizing the weights of the GAP layer and the global connection as the feature fusion weights[37]. This linear fusion of feature maps allows for the creation of CAM. However, this method is limited to networks that solely consist of CNNs and are used for classification purposes. On the other hand, gradient-based CAM allows for greater flexibility. The core concept behind gradient-based CAM is to express the fusion weights of target feature maps as gradients. Additionally, in order to focus on features that positively impact the classification, ReLU is incorporated to remove negative values in the heatmap. Notably, Grad-CAM can be utilized in various problems beyond classification, as long as the activation function can be derived. The feature fusion weights are calculated according to Equation 16.

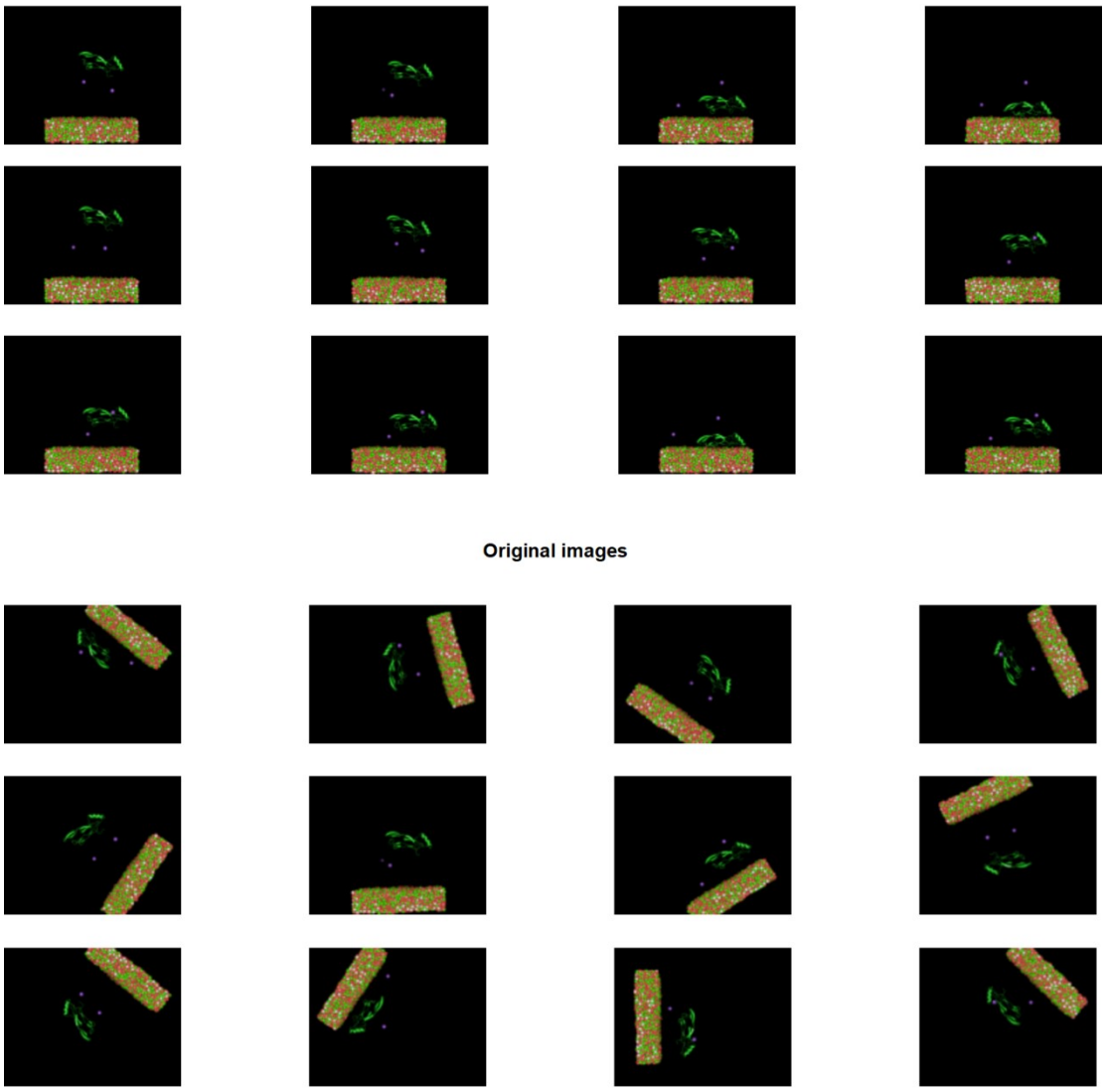
$$\alpha_k^c = \frac{1}{Z}\sum_i\sum_j \frac{\partial y^c}{\partial A_{ij}^k} \tag{24}$$

The variable y^c represents the score of the target category, which is the weight in the network that has not been passed through the softmax function. A_{ij}^k refers to the target feature map that needs to be visualized. $\frac{1}{Z}\sum_i\sum_j$ performs global average pooling. $\frac{\partial y^c}{\partial A_{ij}^k}$ calculates the gradient through backpropagation. The grad-CAM method described here is applicable to network structures without GAP connections, and it can extract the heatmap of any layer's feature map.

Supplementary Figures

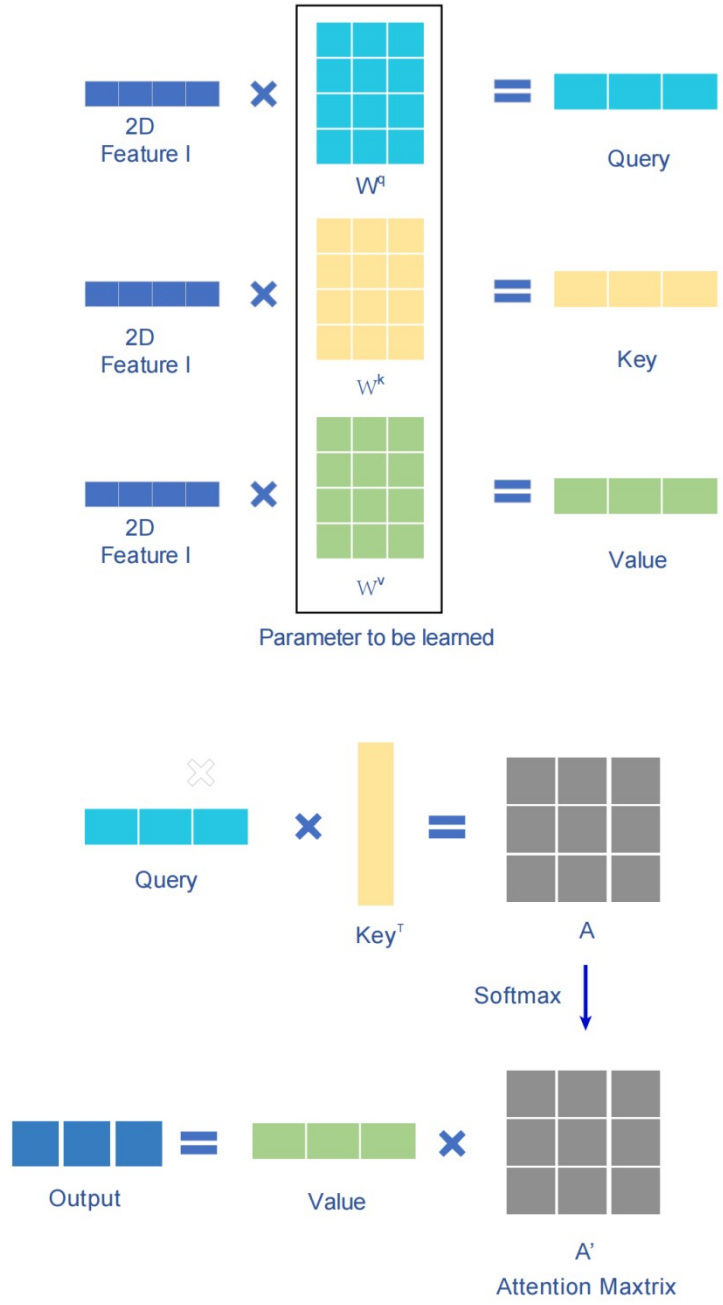


Supplementary Figure 1. Flow of the Platypus Dual Perception Neural Network



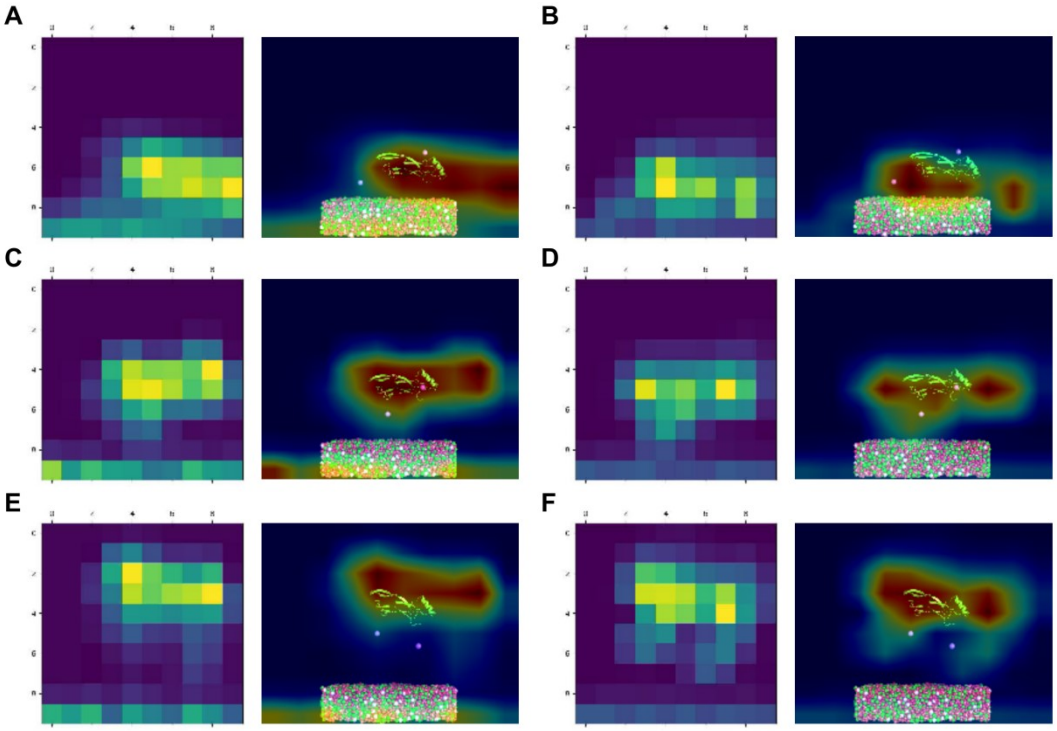
Supplementary Figure 2. Original images and randomly rotated images

Generate a series of new images by randomly rotating and scaling the original trajectory images using matlab.

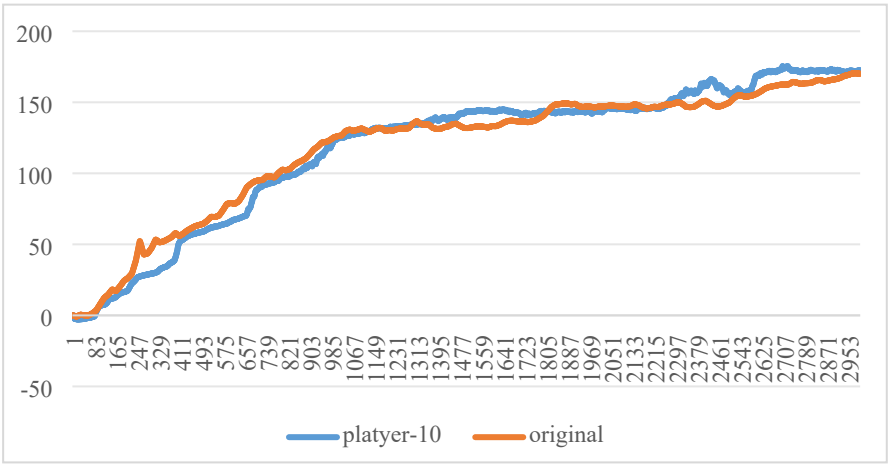


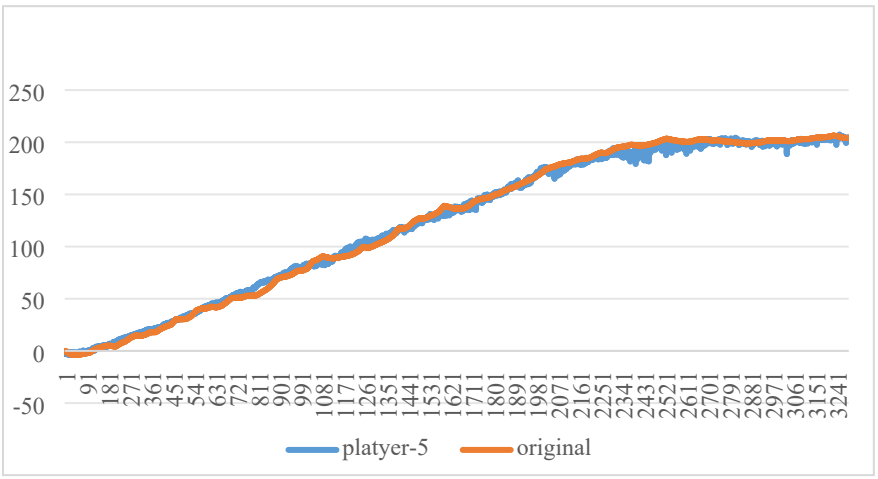
Supplementary Figure 3. Self-attention mechanism calculation method flow

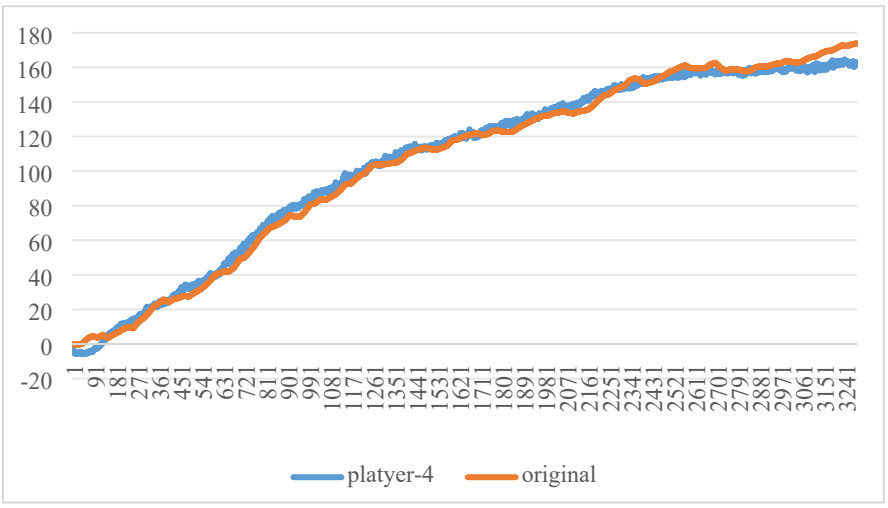
Q,K,V denote query, key and value respectively.The correlation between every two input vectors is computed using the obtained Q and K, that is, the matrix of ATTENTION is computed A. Each value in the matrix A records the magnitude of the Attention of the corresponding two input vectors. A' is obtained by performing softmax operation or relu operation on the matrix A'. The obtained A' and V are used to compute the output vector b of the self-attention layer corresponding to each input vector a, which is combined to form the output.



Supplementary Figure 4. Heatmaps for both the Platyer (left) and Fix-CNN (right) models
At three time points (200ps, 400ps, and 600ps), heatmaps for both the Platyer (left) and Fix-CNN (right) models were generated. The heatmaps visualize the areas of focus within the models. These visual representations reveal that, for the most part, the Platyer model's attention is primarily concentrated on proteins and surfaces. Only at 600ps, the region of focus for the Fix-CNN model shifts towards proteins, demonstrating a more favorable concentration compared to Platyer.







Supplementary Figure 5. Comparison of PMF curves of three different nano-resulting surfaces with the results of Platyer model predicting curves

The horizontal and vertical axes of these curves are the same as Fig.4 D-G in the manuscript.

Supplementary Tables

Supplementary Table 1. Platyper model's evaluation results

Evaluation indicators	Fix-CNN	CNN	LSTM	Platyper
R-squared	0.9666	0.8207	0.9675	0.9748
Mean Squared Error	12.199	65.4506	11.8572	9.2027
Root Mean Squared Error	3.4927	8.0902	3.4434	3.0336
Mean Absolute Error	2.8525	5.5028	2.8103	2.4818
Median Absolute Error	2.5319	3.7127	2.4905	2.1774
Mean Absolute Percentage Error	14.257	72.1345	37.5173	21.4935
Explained Variance	0.969553828	0.820732117	0.968199909	0.975620508

Supplementary Table 2. Platyper model main network parameters

Layer	Input Size	Output Size	Parameters
Conv2d (conv1)	(3, H, W)	(6, H_out, W_out)	(3×5×5 + 1) * 6 = 456
MaxPool2d	(6, H_out, W_out)	(6, H_out/2, W_out/2)	0
Conv2d (conv2)	(6, H_out/2, W_out/2)	(9, H_out/4, W_out/4)	(6×5×5 + 1) * 9 = 409
MaxPool2d	(9, H_out/4, W_out/4)	(9, H_out/8, W_out/8)	0
Conv2d (conv3)	(9, H_out/8, W_out/8)	(6, H_out/16, W_out/16)	(9×5×5 + 1) * 6 = 456
MaxPool2d	(6, H_out/16, W_out/16)	(6, H_out/32, W_out/32)	0
Conv2d (conv4)	(6, H_out/32, W_out/32)	(3, H_out/64, W_out/64)	(6×5×5 + 1) * 3 = 228
Linear (fc1)	(3×H_out/64×W_out/64)	125	(3×H_out/64×W_out/64 + 1) * 125 = 70313
LSTM	-	100	See LSTM formula
Conv2d (query_conv)	(1, 15, 15)	(1, 15, 15)	(1×1 + 1) * 1 = 2
Conv2d (key_conv)	(1, 15, 15)	(1, 15, 15)	(1×1 + 1) * 1 = 2
Conv2d (value_conv)	(1, 15, 15)	(1, 15, 15)	(1×1 + 1) * 1 = 2
Linear (fc2)	225	84	(225 + 1) * 84 = 19044
Linear (fc3)	84	1	(84 + 1) * 1 = 85

Supplementary Table 3. Platyper Model CBAM Partial Network Parameters

Layer	Input Size	Output Size	Parameters
CBAM	(in_planes, H, W)	(out_planes, H_out, W_out)	Total parameters from above layers
BasicConv (conv)	(in_planes, H, W)	(out_planes, H_out, W_out)	(in_planes * out_planes * kernel_size^2 + 1) * out_planes
BatchNorm2d (bn)	(out_planes, H_out, W_out)	(out_planes, H_out, W_out)	2 * out_planes
ChannelGate (mlp)	(gate_channels, H_out, W_out)	(gate_channels, 1, 1)	(gate_channels + 1) * (gate_channels // reduction_ratio + 1)
Flatten	(gate_channels, 1, 1)	(gate_channels)	-
Linear	(gate_channels)	(gate_channels)	(gate_channels + 1) * gate_channels
ChannelPool	(gate_channels, H_out, W_out)	(2, H_out, W_out)	-

Supplementary Table 4. Platyper Model CBAM Partial Network Parameters

Hyperparameter	Value	Description
Epochs	10	Number of training epochs
Learning Rate	0.0001	Learning rate for the optimizer
Criterion	MSELoss	Mean Squared Error Loss
Optimizer	NAdam	NAdam optimizer (not standard in PyTorch, but assumed to be similar to Adam)
Batch Size	50	The number of samples inputted simultaneously during each model training
Activation Function	ReLU	CNN_LSTM_Attention BasicConv (conv) ChannelGate (mlp)

Supplementary Table 5. Evaluation of predicted results for different nanosurfaces

Evaluation indicators	Fix-CNN-10	LSTM-10	Platyper-10
R-squared	0.9323	0.9126	0.9478
Mean Squared Error	134.6377	173.8669	103.936
Root Mean Squared Error	11.6034	13.1859	10.1949
Mean Absolute Error	9.7243	11.6945	8.2688
Median Absolute Error	8.5735	11.2772	8.0194
Mean Absolute Percentage Error	68.0909	57.0497	57.5514
Explained Variance	0.954885859	0.930736256	0.978826284

Evaluation indicators	Fix-CNN-5	LSTM-5	Platyper-5
R-squared	0.9384	0.9242	0.9597
Mean Squared Error	261.9417	385.1792	204.8513
Root Mean Squared Error	16.1846	19.626	14.3126
Mean Absolute Error	14.2055	16.7505	12.8628
Median Absolute Error	13.7278	18.83	13.2143
Mean Absolute Percentage Error	163.9078	192.1586	94.2853
Explained Variance	0.930801175	0.938205323	0.951526425

Evaluation indicators	Fix-CNN-4	LSTM-4	Platyper-4
R-squared	0.848	0.8428	0.8883
Mean Squared Error	431.8362	446.7944	374.2619
Root Mean Squared Error	20.7807	21.1375	19.3458
Mean Absolute Error	18.3242	17.805	17.4185
Median Absolute Error	20.9332	20.1654	16.6868
Mean Absolute Percentage Error	165.4007	141.6449	71.4921
Explained Variance	0.862649047	0.85095861	0.879410326