

# ENZYMefLOW: GENERATING REACTION-SPECIFIC ENZYME CATALYTIC POCKETS THROUGH FLOW MATCHING AND CO-EVOLUTIONARY DYNAMICS

Anonymous authors

Paper under double-blind review

## ABSTRACT

Enzyme design is a critical area in biotechnology, with applications ranging from drug development to synthetic biology. Traditional methods for enzyme function prediction or protein binding pocket design often fall short in capturing the dynamic and complex nature of enzyme-substrate interactions, particularly in catalytic processes. To address the challenges, we introduce EnzymeFlow, a generative model that employs flow matching with hierarchical pre-training and enzyme-reaction co-evolution to generate catalytic pockets for specific substrates and catalytic reactions. Additionally, we introduce a large-scale, curated, and validated dataset of enzyme-reaction pairs, specifically designed for the catalytic pocket generation task, comprising a total of 328,192 pairs. By incorporating evolutionary dynamics and reaction-specific adaptations, EnzymeFlow becomes a powerful model for designing enzyme pockets, which is capable of catalyzing a wide range of biochemical reactions. Experiments on the new dataset demonstrate the model’s effectiveness in designing high-quality, functional enzyme catalytic pockets, paving the way for advancements in enzyme engineering and synthetic biology. The EnzymeFlow code can be found at <https://anonymous.4open.science/r/EnzymeFlow-7420>.

## 1 INTRODUCTION

Proteins are fundamental to life, participating in many essential interactions for biological processes (Whitford, 2013). Among proteins, enzymes stand out as a specialized class that serves as catalysts, driving and regulating nearly all chemical reactions and metabolic pathways across living organisms, from simple bacteria to complex mammals (Kraut, 1988; Murakami et al., 1996; Copeland, 2023) (visualized in Fig. 1). Their catalytic power is central to biological functions, enabling the efficient production of complex organic molecules in biosynthesis (Ferrer et al., 2008; Liu & Wang, 2007) and the creation of novel biological pathways in synthetic biology (Girvan & Munro, 2016; Keasling, 2010; Hodgman & Jewett, 2012). Examining enzyme functions across the tree of life deepens our understanding of the evolutionary processes that shape metabolic networks and enable organisms to adapt to their environments (Jensen, 1976; Glasner et al., 2006; Campbell et al., 2016; Pinto et al., 2022). Consequently, studying enzyme-substrate interactions is essential for comprehending biological processes and designing effective products.

Traditional methods have primarily focused on enzyme function prediction, annotation (Gligorić et al., 2021; Yu et al., 2023), or enzyme-reaction retrieval (Mikhael et al., 2024; Hua et al., 2024b; Yang et al., 2024). These approaches lack the ability to design new enzymes that catalyze specific biological processes. Recent studies suggest that current function prediction models struggle to generalize to unseen enzyme reaction data (de Crecy-Lagard et al., 2024; Kroll et al., 2023a), limiting their utility in enzyme design. To effectively design enzymes, it is crucial not only to predict protein functions but also to identify and generate enzyme catalytic pockets specific to particular substrates, thereby enabling potentially valuable biological processes.

On the other hand, recent advances in deep generative models have significantly improved pocket design for protein-ligand complexes (Stärk et al., 2023; Zhang et al., 2023b; 2024e; Krishna et al., 2024), generating diverse and functional binding pockets for ligand molecules. However, these models cannot generalize directly to the design of enzyme catalytic pockets for substrates involved in catalytic processes. Unlike protein-ligand complexes, where ligand binding typically does not lead to

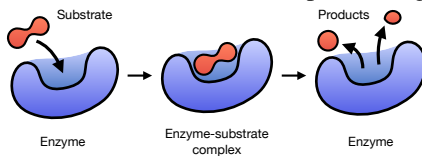


Figure 1: Enzyme-substrate Mechanism.

a chemical transformation, **enzyme-substrate interactions result in a chemical change where the substrate is converted into a product**, which has significantly different underlying mechanisms. More specifically, in protein-ligand binding, the ligand may induce a conformational change in the protein, affect its interactions with other molecules, or modulate its activity; in contrast, the formation of an enzyme-substrate complex is a precursor to a catalytic reaction, where the enzyme lowers the activation energy, facilitating the transformation of the substrate into a product. After the reaction, the enzyme is free to bind another substrate molecule. Therefore, current generative models for pocket design are restricted and limited to static ligand-binding interactions, failing to describe such dynamic transformations and the complex nature of enzyme-substrate interactions.

To address these limitations, we propose EnzymeFlow (demonstrated in Fig. 2), a flow matching model (Lipman et al., 2022; Liu et al., 2022; Albergo & Vanden-Eijnden, 2023) with enzyme-reaction co-evolution and structure-based pre-training for enzyme catalytic pocket generation. Our major contributions follow: **(1) EnzymeFlow—Flow Model for Enzyme Catalytic Pocket Design:** We define conditional flows for enzyme catalytic pocket generation based on backbone frames, amino acid types, and Enzyme Commission (EC) class. The generative flow process is conditioned on specific substrates and products, enabling potential catalytic processes. **(2) Enzyme-Reaction Co-Evolution:** Since enzyme-substrate interactions involve dynamic chemical transformations of substrate molecules, which is distinct from static protein-ligand interactions, we propose enzyme-reaction co-evolution with a new co-evolutionary transformer (*coEvoFormer*). The co-evolution is used to capture substrate-specificity in catalytic reactions. It encodes how enzymes and reactions evolve together, allowing the model to operate on evolutionary dynamics, which naturally comprehends the catalytic process. **(3) Structure-Based Hierarchical Pre-Training:** To leverage the vast data of geometric structures from existing proteins and protein-ligand complexes, we propose a structure-based hierarchical pre-training. This method progressively learns from protein backbones to protein binding pockets, and finally to enzyme catalytic pockets. This hierarchical learning of protein structures enhances geometric awareness within the model. **(4) EnzymeFill—Large-scale Pocket-specific Enzyme-Reaction Dataset with Pocket Structures:** Current enzyme-reaction datasets are based on full enzyme sequences or structures and lack precise geometry for how enzyme pockets catalyze the substrates. To address this, we construct a structure-based, curated, and validated enzyme catalytic pocket-substrate dataset, specifically designed for the catalytic pocket generation task.

## 2 RELATED WORK

### 2.1 PROTEIN EVOLUTION

Protein evolution learns how proteins change over time through processes such as mutation, selection, and genetic drift (Pál et al., 2006; Bloom & Arnold, 2009), which influence protein functions. Studies on protein evolution focus on understanding the molecular mechanisms driving changes in protein sequences and structures. Zuckerkandl & Pauling (1965) introduce the concept of the molecular clock, which postulates that proteins evolve at a relatively constant rate over time, providing a framework for estimating divergence times between species. DePristo et al. (2005) show that evolutionary rates are influenced by functional constraints, with regions critical to protein function (*e.g.*, active sites, binding interfaces) evolving more slowly due to purifying selection. This understanding leads to the development of methods for detecting functionally important residues based on evolutionary conservation. Understanding protein evolution has practical applications in protein engineering. By studying how natural proteins evolve to acquire new functions, researchers design synthetic proteins with desired properties (Xia & Levitt, 2004; Jäckel et al., 2008). Additionally, deep learning models increasingly integrate evolutionary principles to predict protein function and stability, design novel enzymes, and guide protein engineering (Yang et al., 2019; AlQuraishi, 2019; Jumper et al., 2021).

### 2.2 GENERATIVE MODELS FOR PROTEIN AND POCKET DESIGN

Recent advancements in generative models have advanced the field of protein design and binding pocket design, enabling the creation of proteins or binding pockets with desired properties and functions (Yim et al., 2023a;b; Chu et al., 2024; Hua et al., 2024a; Abramson et al., 2024). For example, RFDiffusion (Watson et al., 2023) employs denoising diffusion in conjunction with RoseTTAFold (Baek et al., 2021) for *de novo* protein structure design, achieving wet-lab-level generated structures that can be extended to binding pocket design. RFDiffusionAA (Krishna et al., 2024) extends RFDiffusion for joint modeling of protein and ligand structures, generating ligand-binding proteins and further leveraging MPNNs for sequence design. Additionally, FAIR (Zhang et al., 2023b) and

PocketGen (Zhang et al., 2024e) use a two-stage coarse-to-fine refinement approach to co-design pocket structures and sequences. Recent models leveraging flow matching frameworks have shown promising results in these tasks. For instance, FoldFlow (Bose et al., 2023) introduces a series of flow models for protein backbone design, improving training stability and efficiency. FrameFlow (Yim et al., 2023a) further enhances sampling efficiency and demonstrates success in motif-scaffolding tasks using flow matching, while MultiFlow (Campbell et al., 2024) advances to structure and sequence co-design. These flow models, initially applied to protein backbones, have been further generalized to binding pockets. For example, PocketFlow (Zhang et al., 2024d) combines flow matching with physical priors to explicitly learn protein-ligand interactions in binding pocket design, achieving stronger results compared to RFDiffusionAA. While these models excel in protein and binding pocket design, they primarily focus on static protein(-ligand) interactions and lack the ability to model the chemical transformations involved in enzyme-catalyzed reactions. This limitation may reduce their accuracy and generalizability in designing enzyme pockets for catalytic reactions. In EnzymeFlow, we aim to address these current limitations. An extended discussion of related works on AI-driven protein engineering can be found in App. C.

**Discussion regarding PocketFlow.** PocketFlow (Zhang et al., 2024d) has demonstrated strong performance in protein-ligand design, showing generalizability across various protein pocket categories. However, it falls short when applied to the design of enzyme catalytic pocket with specific substrates. One key limitation is that protein-ligand interactions are static, meaning that the training data and model design do not capture or describe the chemical transformations, such as the conversion or production of new molecules, that occur during enzyme-catalyzed reactions. This dynamic aspect of enzyme-substrate interactions is missing in current models. Another limitation is that PocketFlow fixes the overall protein backbone structure before designing the binding pocket, treating the pocket as a missing element to be filled in. This approach may not align with practical needs, as the overall protein backbone structure is often unknown before pocket design. Ideally, the design process should be reversed: the pocket should be designed first, influencing the overall protein structure. Despite these challenges, PocketFlow remains a good and leading work in pocket design. With EnzymeFlow, we aim to address these limitations, particularly in the context of catalytic pocket design.

### 3 ENZYMEFLOW

We introduce EnzymeFlow, a flow matching model with hierarchical pre-training and enzyme-reaction co-evolution to generate enzyme catalytic pockets for specific substrates and catalytic reactions. We demonstrate the pipeline in Fig. 2, discuss the EnzymeFlow with co-evolutionary dynamics in Sec. 3.1, further introduce the structure-based hierarchical pre-training for generalizability in Sec. 3.2

#### 3.1 ENZYME CATALYTIC POCKET GENERATION WITH FLOW MATCHING

**EnzymeFlow on Catalytic Pocket.** Following Yim et al. (2023a), we refer to the protein structure as the backbone atomic coordinates of each residue. A pocket with number of residues  $N_r$  can be parameterized into SE(3) residue frames  $\{(x^i, r^i, c^i)\}_{i=1}^{N_r}$ , where  $x^i \in \mathbb{R}^3$  represents the position (translation) of the  $C_\alpha$  atom of the  $i$ -th residue,  $r^i \in \text{SO}(3)$  is a rotation matrix defining the local frame relative to a global reference frame, and  $c^i \in \{1, \dots, 20\} \cup \{\times\}$  denotes the amino acid type, with an additional  $\times$  indicating a *masking state* of the amino acid type. We refer to the residue block as  $T^i = (x^i, r^i, c^i)$ , and the entire pocket is described by a set of residues  $\mathbf{T} = \{T^i\}_{i=1}^{N_r}$ . Additionally, we denote the graph representations of substrate and product molecules in the catalytic reaction as  $l_s$  and  $l_p$ , respectively. An enzyme-reaction pair can therefore be described as  $(\mathbf{T}, l_s, l_p)$ .

Following flow matching literature (Yim et al., 2023a; Campbell et al., 2024), we use time  $t = 1$  to denote the source data. The conditional flow on the enzyme catalytic pocket  $p_t(\mathbf{T}_t|\mathbf{T}_1)$  for a time step  $t \in (0, 1]$  can be factorized into the probability density over continuous variables (translations and rotations) and the probability mass function over discrete variables (amino acid types) as:

$$p_t(\mathbf{T}_t|\mathbf{T}_1) = \prod_{i=1}^{N_r} p_t(x_t^i|x_1^i) p_t(r_t^i|r_1^i) p_t(c_t^i|c_1^i), \quad (1)$$

where the translation, rotation, and amino acid type at time  $t$  are derived as:

$$\begin{aligned} x_t^i &= (1-t)x_0^i + tx_1^i, \quad x_0^i \sim \mathcal{N}(0, I); \quad r_t^i = \exp_{r_0^i}(t \log_{r_0^i} r_1^i), \quad r_0^i \sim \mathcal{U}_{\text{SO}(3)}; \\ c_t^i &\sim p_t(c_t^i|c_1^i) = \text{Cat}(t \delta(c_1^i, c_t^i) + (1-t) \delta(\times, c_t^i)), \end{aligned} \quad (2)$$

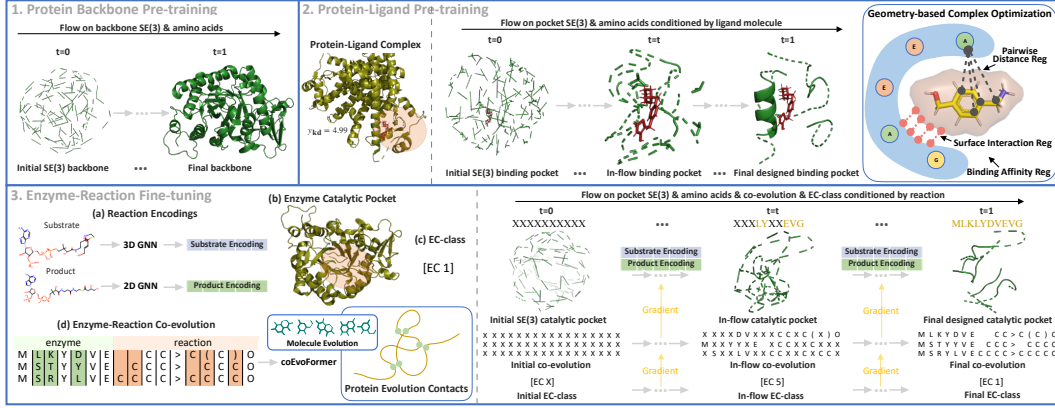


Figure 2: Overview of EnzymeFlow with hierarchical pre-training and enzyme-reaction co-evolution. (1) Flow model pre-trained on protein backbones and amino acid types. (2) Flow model further pre-trained on protein binding pockets, conditioned on ligand molecules with geometry-specific optimization. (3) Flow model fine-tuned on enzyme catalytic pockets, and conditioned on substrate and product molecules, with enzyme-reaction co-evolution and EC-class generation.

where  $\delta(a, b)$  is the Kronecker delta, which equals to 1 if  $a = b$  and 0 if  $a \neq b$ ; Cat is a categorical distribution for the sampling of discrete amino acid type, with probabilities  $t\delta(c_1^i, c_t^i) + (1-t)\delta(\times, c_t^i)$ . The discrete flow interpolates from the *masking state*  $\times$  at  $t = 0$  to the actual amino acid type  $c_1^i$  at  $t = 1$  (Campbell et al., 2024). In a catalytic process, enzymes interact with substrates to produce specific products. In practical enzyme design, we typically know the substrates  $l_s$  (as 3D atom point clouds) and the desired products  $l_p$  (as 2D molecular graphs or SMILES). Therefore, the formation of the enzyme catalytic pocket should be conditioned on both substrates and products. Our enzyme flow matching model is conditioned on these two ligand molecules  $l_s, l_p$ , ensuring that the predictions of vector fields  $v_\theta(\cdot)$  and loss functions account for the substrate and product molecules:

$$\begin{aligned} \mathcal{L}_{\text{trans}} &= \sum_{i=1}^{N_r} \|v_\theta^i(x_t^i, t, l_s, l_p) - (x_1^i - x_0^i)\|_2^2; \quad \mathcal{L}_{\text{rot}} = \sum_{i=1}^{N_r} \|v_\theta^i(r_t^i, t, l_s, l_p) - \frac{\log_{r_t^i} r_1^i}{1-t}\|_{\text{SO}(3)}^2; \\ \mathcal{L}_{\text{aa}} &= - \sum_{i=1}^{N_r} \log p_\theta(c_1^i | v_\theta^i(c_t^i, t, l_s, l_p)). \end{aligned} \quad (3)$$

To design the enzyme pocket and model protein-ligand interactions, we implement 3D and 2D GNNs to encode the substrate and product, respectively (implemented in App. E). The main vector field network applies cross-attention to model protein-ligand interactions and incorporates Invariant Point Attention (IPA) (Jumper et al., 2021) to encode protein features and make predictions. Following tricks in Yim et al. (2023a); Campbell et al. (2024), we let the model predict the final structure at  $t = 1$  and interpolates to compute the vector fields (discussed in App. F).

**EnzymeFlow on EC-Class.** The Enzyme Commission (EC) classification is crucial for categorizing enzymes based on the reactions they catalyze. Understanding the EC-class of an enzyme-reaction pair can help predict its function in various biochemical pathways (Bansal et al., 2022). Given its importance, EnzymeFlow leverages EC-class to enhance its generalizability across various enzymes and catalytic reactions. Therefore, our model incorporates EC-class,  $y_{\text{ec}} \in \{1, \dots, 7\} \cup \{\times\}$ , as a discrete factor in the design process. The EC-class is sampled from a Categorical distribution with probabilities  $t\delta(y_{\text{ec}_1}, y_{\text{ec}_t}) + (1-t)\delta(\times, y_{\text{ec}_t})$ . The discrete flow on EC-class interpolates from the *masking state*  $\times$  at  $t = 0$  to the actual EC-class  $y_{\text{ec}_1}$  at  $t = 1$ . The prediction and loss function are conditioned on the pocket frames and the substrate and product molecules:

$$\mathcal{L}_{\text{ec}} = - \log p_\theta(y_{\text{ec}_1} | v_\theta(\mathbf{T}_t, t, l_s, l_p, y_{\text{ec}_t})). \quad (4)$$

The model predicts the final EC-class at  $t = 1$  and interpolates to compute its vector field. For EC-class prediction, we first employ a EC-class embedding network to encode  $y_{\text{ec}_t}$ . The final predicted EC-class is obtained by pooling cross-attention between the encoded enzyme and EC-class features.

### 3.1.1 ENZYMEFLOW WITH ENZYME-REACTION CO-EVOLUTION

Enzyme (protein) evolution refers to the process by which enzyme structures and functions change over time due to genetic variations, such as mutations, duplications, and recombinations. These changes can lead to alterations in amino acids, potentially affecting the enzyme structure, function, stability, and interactions (Pál et al., 2006; Sikosek & Chan, 2014). Reaction evolution, on the other





Figure 3: Catalytic pocket design example using EnzymeFlow (UniProt: Q7U4P2). The pocket generation is conditioned on reaction CN[C@H](C(=O)C)CS.C/C=C\1/C(=C/c2[nH]c(c(c2C)CCC(=O)O)/C=C/2\N=C(C(=C2CCC(=O)O)C)C[C@H]2NC(=O)C(=C2C)C=C)/NC(=O)[C@H]1C>>CN[C@H](C(=O)C)CSC(C1=C(C)C(=O)N[C@H]1Cc1[nH]c(c(c1C)CCC(=O)O)/C=C/1\N=C(C(=C1CCC(=O)O)C)C[C@H]1NC(=O)C(=C1C)C=C)C of EC4 (ligase enzyme), from  $t = 0$  to  $t = 1$ .

hand, refers to the process by which chemical reactions or substrates, particularly those catalyzed by enzymes, change and diversify within biological systems over time (illustrated in Fig. 2(3)(d)).

**Co-Evolutionary Dynamics.** Enzymes can co-evolve with the metabolic or biochemical pathways they are part of, adapting to changes in substrate availability, the introduction of new reaction steps, or the need for more efficient flux through the pathway. As pathways evolve, enzymes within them may develop new catalytic functions or refine existing ones to better accommodate these changes (Noda-Garcia et al., 2018). This process frequently involves the co-evolution of enzymes and their substrates. As substrates change—whether due to the introduction of new compounds in the environment or mutations in other metabolic pathways—enzymes may adapt to catalyze reactions with these new substrates, leading to the emergence of entirely new reactions. Understanding enzyme-substrate interactions, therefore, requires considering their evolutionary dynamics, as these interactions are shaped by the evolutionary history and adaptations of both enzymes and their substrates. This co-evolutionary process is crucial for explaining how enzymes develop new functions and maintain efficiency in response to ongoing changes in their biochemical environment.

To capture the evolutionary dynamics, we introduce the concept of enzyme-reaction co-evolution into EnzymeFlow. We compute the enzyme and reaction evolution by applying multiple sequence alignment (MSA) to enzyme sequences and reaction SMILES, respectively (Steinegger & Söding, 2017). The co-evolution of an enzyme-reaction pair is represented by a matrix  $U \in \mathbb{R}^{N_{\text{MSA}} \times N_{\text{token}}}$ , which combines the MSA results of enzyme sequences and reaction SMILES (illustrated in Fig. 2(3)(d) & Fig. 8), where  $N_{\text{MSA}}$  denotes the number of MSA sequences and  $N_{\text{token}}$  denotes the length of the MSA alignment preserved. And each element  $u^{mn} \in \{1, \dots, 64\} \cup \{\times\}$  in  $U$  denotes a tokenized character from our co-evolution vocabulary, with additional  $\times$  indicating the *masking state*.

**EnzymeFlow on Co-Evolution.** The flow for co-evolution follows a similar approach to that used for amino acid types and EC-class, treating it as a discrete factor in the design process. The co-evolution is sampled from a Categorical distribution, where each element has probabilities  $t\delta(u_1^{mn}, u_t^{mn}) + (1 - t)\delta(\times, u_t^{mn})$ . Each element flows independently, reflecting the natural independence of amino acid mutations (Boyko et al., 2008). The discrete flow on co-evolution interpolates from the *masking state*  $\times$  at  $t = 0$  to the actual character  $u_1^{mn}$  at  $t = 1$ . The prediction and loss function are conditioned on the pocket frames and the substrate and product molecules:

$$\mathcal{L}_{\text{coevo}} = - \sum_{m=1}^{N_{\text{MSA}}} \sum_{n=1}^{N_{\text{token}}} \log p_{\theta}(u_1^{mn} | v_{\theta}(\mathbf{T}_t, t, l_s, l_p, u_t^{mn})). \quad (5)$$

The model predicts the final co-evolution at  $t = 1$  and interpolates to compute its vector field. For co-evolution prediction, we first introduce a co-evolutionary MSA transformer (coEvoFormer) to encode  $U_t$  (implemented in App. D). The final predicted co-evolution is obtained by computing cross-attention between the encoded enzyme and ligand, and the encoded co-evolution features.

We can therefore express EnzymeFlow with co-evolutionary dynamics for catalytic pocket design as:

$$p_t(\mathbf{T}_t, U_t, y_{\text{ec}_t} | \mathbf{T}_1, U_1, y_{\text{ec}_1}, l_s, l_p) = p_t(y_{\text{ec}_t} | y_{\text{ec}_1}, \mathbf{T}_t) p_t(U_t | U_1, \mathbf{T}_t) p_t(\mathbf{T}_t | \mathbf{T}_1, l_s, l_p). \quad (6)$$

The final EnzymeFlow model performs flows on protein backbones, amino acid types, EC-class, and enzyme-reaction co-evolution. Given the SE(3)-invariant prior and the main SE(3)-equivariant network in EnzymeFlow, the pocket generation process is also SE(3)-equivariant (proven in App. G).

### 3.2 STRUCTURE-BASED HIERARCHICAL PRE-TRAINING

In addition to the standard EnzymeFlow for enzyme pocket design, we propose a hierarchical pre-training strategy to enhance the generalizability of the model across different enzyme categories. The

term *hierarchical pre-training* is used because the approach first involves training the flow model to understand protein backbone generation, followed by training it to learn the geometric relationships between proteins and ligand molecules, which form protein binding pockets. After the flow model learns these prior knowledge, we fine-tune it specifically on an enzyme-reaction dataset to generate enzyme catalytic pockets. The term *hierarchical* reflects the progression from protein backbone generation, to protein binding pocket formation, and finally to enzyme catalytic pocket generation.

**Motivation.** A key limitation of current datasets, such as ESP (Kroll et al., 2023b), EnzymeMap (Heid et al., 2023), CARE (Yang et al., 2024), or ReactZyme (Hua et al., 2024b), is the lack of precise pocket information. These datasets typically provide enzyme-reaction data, including protein sequences and SMILES representations, which is used to predict EC numbers in practice. To incorporate geometric information and enhance the model’s generalizability through structural data, we propose a hierarchical pre-training approach to ensure the model comprehends the geometry between proteins and ligand molecules. Alongside this, we introduce a new synthetic dataset, the EnzymeFlow pocket dataset, which includes precise pocket structures with substrate conformations (discussed in Sec. 4).

Specifically, we begin by pre-training the flow model on a protein backbones. Once the model learns it, we proceed to post-train it on a protein-ligands, with the objective of generating binding pockets conditioned on the ligand molecules. Finally, the model is fine-tuned on our EnzymeFlow dataset to generate valid enzyme catalytic pockets for specific substrates and catalytic reactions.

### 3.2.1 PROTEIN BACKBONE PRE-TRAINING

The initial step involves pre-training the model on a protein backbone dataset (illustrated in Fig. 2(1)). We use the backbone dataset discussed in FrameFlow (Yim et al., 2023a). This pre-training focuses solely on SE(3) backbone frames and discrete amino acid types, allowing the flow model to acquire foundational knowledge of protein backbone geometry and structure.

### 3.2.2 PROTEIN-LIGAND PRE-TRAINING

Following the protein backbone pre-training, we proceed to pre-train the flow model on a protein-ligand dataset (illustrated in Fig. 2(2)). Specifically, we use PDBBind2020 (Wang et al., 2004). This pre-training focuses on binding pocket frames, with the flow model conditioned on the 3D representations of ligand molecules  $l$  consisting of  $N_l$  atoms. Additionally, binding affinity  $y_{kd} \in \mathbb{R}$  and atomic-level pocket-ligand distance  $D^i \in \mathbb{R}^{4 \times N_l}$  for the  $i$ -th residue frame serve as optimization factors. The parametrization is similar to Eq. 6, with conditioning on the ligand molecule as follows:

$$p_t(\mathbf{T}_t, y_{kd} | \mathbf{T}_1, l) = p_t(y_{kd} | \mathbf{T}_t, l) p_t(\mathbf{T}_t | \mathbf{T}_1, l). \quad (7)$$

In addition to the flow matching losses in Eq. 3, we introduce a protein-ligand interaction loss to prevent intersection during the binding generation process. Conceptually, this ensures that the generated pocket atoms do not come into contact with the surface of the ligand molecule. Following previous work on protein-ligand binding (Lin et al., 2022), the surface of a ligand  $\{a_j | j \in \mathbb{N}(N_l)\}$  is defined as  $\{a \in \mathbb{R}^3 | S(a) = \gamma\}$ , where  $S(a) = -\rho \log(\sum_{j=1}^{N_l} \exp(-|a - a_j|^2 / \rho))$ . The interior of the ligand molecule is thus defined by  $\{a \in \mathbb{R}^3 | S(a) < \gamma\}$ , and the binding pocket atoms are constrained to lie within  $\{a \in \mathbb{R}^3 | S(a) > \gamma\}$ . We also introduce a protein-ligand distance loss to regularize pairwise atomic distances, along with a binding affinity loss to enforce the generation of more valid protein-ligand pairs. These objectives are defined as follows:

$$\mathcal{L}_{\text{inter}} = \sum_{i=1}^{N_r} \max(0, \gamma - S(\hat{A}_t^i)), \quad \mathcal{L}_{\text{dist}} = \sum_{i=1}^{N_r} \frac{\|\mathbf{1}\{D_1^i < 8\text{\AA}\}(D_1^i - \hat{D}_t^i)\|_2^2}{\sum \mathbf{1}_{D_1^i < 8\text{\AA}}}, \quad \mathcal{L}_{\text{kd}} = \|y_{kd} - \hat{y}_{kd}\|^2, \quad (8)$$

where  $\hat{A}^i \in \mathbb{R}^{4 \times 3}$  denotes the predicted atomic positions of  $i$ -th residue frame,  $\gamma = 6$  and  $\rho = 2$  are hyperparameters, and  $\hat{y}_{kd}$  is the predicted binding affinity for a generated pair.  $\hat{D}^i \in \mathbb{R}^{4 \times N_l}$  is defined similarly to  $D^i$ , based on the distance between the predicted atomic positions and ligand positions for the  $i$ -th residue frame. The predicted affinity  $\hat{y}_{kd}$  is obtained by pooling the encoded protein and ligand features. These additional losses are incorporated to improve the model’s generalizability, enforcing more constrained geometries for more valid protein pocket design.

## 4 ENZYMEFILL: LARGE-SCALE ENZYME POCKET-REACTION DATASET

**Data Source.** We construct a curated and validated dataset of enzyme-reaction pairs by collecting data from the Rhea (Bansal et al., 2022), MetaCyc (Caspi et al., 2020), and Brenda (Schomburg et al., 2002) databases. For enzymes in these databases, we exclude entries missing UniProt IDs or protein sequences. For reactions, we apply the following procedures: (1) remove cofactors, small

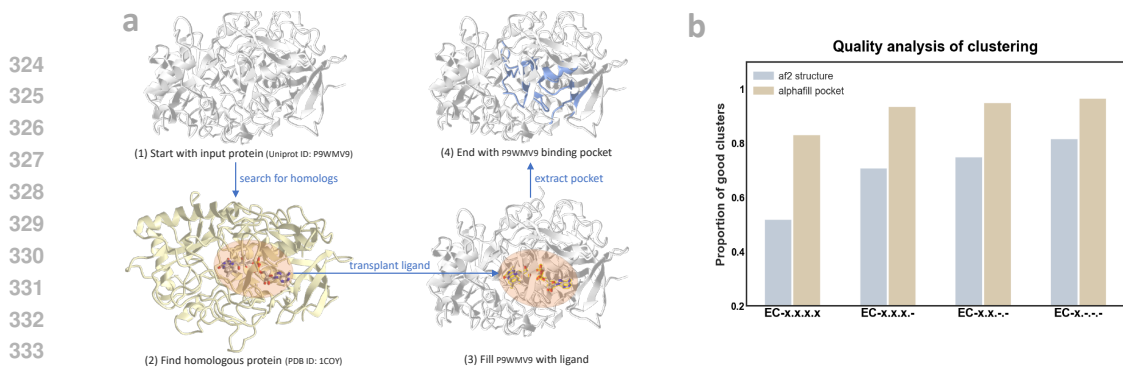


Figure 4: (a) Enzyme pocket extraction workflow with AlphaFill. (b) Quality analysis of clustering between enzyme pockets and full structures; good clusters have high functional concentration.

ion groups, and molecules that appear in both substrates and products within a single reaction; (2) exclude reactions with more than five substrates or products; and (3) apply OpenBabel (O’Boyle et al., 2011) to standardize canonical SMILES. Ultimately, we obtain a total of 328,192 enzyme-reaction pairs, comprising 145,782 unique enzymes and 17,868 unique reactions; we name it EnzymeFill.

**Catalytic Pocket with AlphaFill.** We identify all enzyme catalytic pockets using AlphaFill (Hekkelman et al., 2023), an AF-based algorithm that uses sequence and structure similarity to transplant ligand molecules from experimentally determined structures to predicted protein models. We download the AlphaFold structures for all enzymes and apply AlphaFill to extract the enzyme pockets. Simultaneously, we determine the reaction center by using atom-atom mapping of the reactions. During the pocket extraction process, AlphaFill first identifies homologous proteins of the target enzyme in the PDB-REDO database, along with their complexes with ligands (van Beusekom et al., 2018). It then transplants the ligands from the homologous protein complexes to the target enzyme through structural alignment (illustrated in Fig. 4(a)). After transplantation, we select the appropriate ligand molecule based on the number of atoms and its frequency of occurrence, and extract the pocket using a pre-defined radius of 10Å. We also perform clustering analysis on the extracted pockets using Foldseek (van Kempen et al., 2022), which reveals that enzyme catalytic pockets capture functional information more effectively than full structures (illustrated in Fig. 4(b)). For the extraction of reaction centers, we first apply RXNMapper to extract atom-atom mappings (Schwaller et al., 2021), which maps the atoms between the substrates and products. We then identify atoms where changes occurred in chemical bonds, charges, and chirality, labeling these atoms as reaction centers.

**Data Debiasing for Generation.** To ensure the quality of catalytic pocket data for the design task, we exclude pockets with fewer than 32 residues<sup>1</sup>, resulting in 232,520 enzyme-reaction pairs. Additionally, enzymes and their catalytic pockets can exhibit significant sequence similarity. When enzymes that are highly similar in sequence appear too frequently in the dataset, they tend to belong to the same cluster or homologous group, which can introduce substantial biases during model training. To mitigate this issue and ensure a more balanced dataset, it is important to reduce the number of homologous enzymes by clustering and selectively removing enzymes from the same clusters. This helps to debias the data and improve the model’s generalizability. We perform sequence alignment to cluster enzymes and identify homologous ones (Steinegger & Söding, 2017). We then revise the dataset into five major categories based on enzyme sequence similarity, resulting in: (1) 19,379 pairs with at most 40% homology, (2) 34,750 pairs with at most 50% homology, (3) 53,483 pairs with at most 60% homology, (4) 100,925 pairs with at most 80% homology, and (5) 132,047 pairs with at most 90% homology. In EnzymeFlow, we choose to use the clustered data with at most 60% homology with 53,483 samples for training. We provide more dataset statistics in App. H

## 5 EXPERIMENT — GENERATING CATALYTIC POCKET CONDITIONED ON REACTIONS AND SUBSTRATES

We compare EnzymeFlow with state-of-the-arts representative baselines, including template-matching method DEPACT (Chen et al., 2022), deep equivariant and iterative refinement model PocketGen (Zhang et al., 2024e), golden-standard diffusion model RFDiffAA (Krishna et al., 2024), and the most recent PocketFlow<sup>2</sup> (Zhang et al., 2024d). For RFDiffAA-designed pockets, we apply LigandMPNN

<sup>1</sup>32 residues are chosen based on LigandMPNN (Dauparas et al., 2023), ensuring high-quality interactions.

<sup>2</sup>PocketFlow is not open-sourced yet, we implement and train it on EnzymeFill without fixing the backbones.

Table 1: EnzymeFlow Evaluation Data Statistics.

Data	Reaction	Enzyme	Substrate		Product		Enzyme Commission						
	#reaction	#enzyme	#substrate	#avg atom	#product	#avg atom	EC1	EC2	EC3	EC4	EC5	EC6	EC7
Raw	232520	97912	7259	30.81	7664	30.34	44881 (19.30)	75944 (32.66)	37728 (16.23)	47242 (20.32)	8315 (3.58)	18281 (7.86)	129 (0.06)
Train	53483	22350	6112	30.95	6331	30.34	11674 (21.83)	18419 (34.44)	11394 (21.30)	5555 (10.39)	2194 (4.10)	4200 (7.85)	47 (0.09)
Eval	100	100	100	30.7	94	28.84	17 (17.00)	17 (17.00)	17 (17.00)	17 (17.00)	16 (16.00)	16 (16.00)	0 (0.00)

(Dauparas et al., 2023) to inverse fold and predict the sequences post-hoc. We provide the code of EnzymeFlow at <https://anonymous.4open.science/r/EnzymeFlow-7420>.

**Evaluation Data.** We use MMseqs2 to perform clustering with a 10% homology threshold, selecting the center of each cluster as the initial dataset, resulting in a total of 3,417 pairs. After de-duplicating both repeated substrates and UniProt entries, we are left with 839 unique enzyme-reaction pairs. We then uniformly sample data across different EC classes, selecting 17 pairs from EC1 to EC4 classes and 16 pairs from EC5 and EC6 classes, respectively, resulting in a total of 100 unique catalytic pockets and 100 unique reactions. Each enzyme-reaction pair is labeled with a ground-truth EC-class from EC1 to EC6. We present the EC-class distribution in the evaluation set in Tab. 1.

**Reaction-conditioned Generation.** For pocket design and model sampling, we perform conditional generation on each reaction (or substrate), generating 100 catalytic pockets for each reaction in the evaluation set. We evaluate the generated pockets for their structures and functions (*i.e.*, EC-class).

**EnzymeFlow Scope.** In EnzymeFlow, we adhere to the philosophy that enzyme function dictates its structure. This means that an enzyme folds into a specific 3D shape to fulfill its catalytic role, and the resulting structure can then be inversely folded into a sequence—essentially, *function*  $\rightarrow$  *structure*  $\rightarrow$  *sequence*. In EnzymeFlow, the enzyme function is defined by the reaction that the enzyme will catalyze (discussed in App. B). We evaluate the structures and functions of the designed pockets.

### 5.1 CATALYTIC POCKET STRUCTURE EVALUATION

We begin by assessing the structural validity of generated catalytic pockets. While enzyme function determines whether the designed pocket can catalyze a specific reaction, the structure determines whether the substrate conformation can properly bind to the catalytic pocket. We provide some visual examples of designed pockets in Fig. 5 and App. J.

**Metrics.** We use the following metrics to evaluate and compare the structural validity of the generated pocket. Constrained-site RMSD (cRMSD): The structural distance between the ground-truth and generated pockets, as proposed in Hayes et al. (2024). TM-score: The topological similarity between the generated and ground-truth pockets in local deviations. Aggregated Chai Score (chai): The confidence and structural validity of the designed pockets by running Chai (Chai, 2024). It is calculated as  $0.2 \times pTM + 0.8 \times ipTM - 100 \times clash$ , where *pTM* is the predicted template modeling score, *ipTM* is the interface predicted template modeling score (as used in Jumper et al. (2021)), and the definition of *chai* is proposed by Chai (2024). Binding Affinity (Kd): The binding affinity between the generated catalytic pocket and the substrate conformation is computed using AutoDock Vina (Trott & Olson, 2010). Amino Acid Recovery (AAR): The overlap ratio between the predicted and ground-truth amino acid types in the generated pocket. Enzyme Commission Accuracy (ECacc): The accuracy of matching the EC-class of generated pockets with the ground-truth EC-class.

Table 2: Evaluation of structural validity of EnzymeFlow- and baseline-generated catalytic pockets. The binding affinities (Kd) and structural confidence (chai) are computed by performing docking on the catalytic pocket and substrate conformation using Vina (Trott & Olson, 2010) and Chai (Chai, 2024), respectively. We highlight top three results in **bold**, underline, and *italic*, respectively.

Model	cRMSD ( $\downarrow$ )			TM-score ( $\uparrow$ )			Kd ( $\downarrow$ )	chai ( $\uparrow$ )	AAR ( $\uparrow$ )	ECacc ( $\uparrow$ )
	Top1	Top10	Median	Top1	Top10	Median				
Eval Data	-			-			-4.65	-	-	-
DEPACT	9.25	9.75	11.16	0.238	0.206	0.149	-5.46	0.125	0.112	0.149
PocketGen	7.65	8.14	10.45	0.260	0.233	0.193	-5.01	0.121	0.176	0.152
RFDiffAA	9.13	9.77	11.92	0.269	0.245	0.198	<b>-12.71</b>	<b>0.232</b>	0.153	0.170
PocketFlow	7.42	8.09	10.01	0.268	<i>0.260</i>	0.197	-4.93	0.123	<i>0.207</i>	0.166
EnzymeFlow (T=50)	<b>6.94</b>	<b>7.57</b>	<u>9.04</u>	<b>0.290</b>	<b>0.262</b>	<b>0.209</b>	-5.03	0.129	<b>0.216</b>	<b>0.280</b>
w/o coevo	7.02	<u>7.60</u>	<i>9.15</i>	<u>0.288</u>	<i>0.260</i>	<i>0.205</i>	-4.86	0.123	0.196	0.246
w/o pretraining	<i>7.01</i>	<i>7.69</i>	9.29	<i>0.286</i>	<i>0.261</i>	<i>0.207</i>	-4.33	<i>0.134</i>	0.202	0.255
w/o coevo+pretraining	7.05	7.81	9.43	0.278	0.255	0.204	-4.72	0.125	0.154	0.221
EnzymeFlow (T=100)	<u>6.97</u>	<b>7.57</b>	<b>9.02</b>	0.283	0.258	<u>0.207</u>	<i>-5.31</i>	<u>0.135</u>	<u>0.215</u>	<u>0.273</u>

**Results.** We compare the structural validity between EnzymeFlow- and baseline-generated catalytic pockets in Tab. 2. EnzymeFlow and its ablation models outperform baseline models, including leading models like RFDiffAA and PocketFlow, with significant improvements in cRMSD, TM-score, and



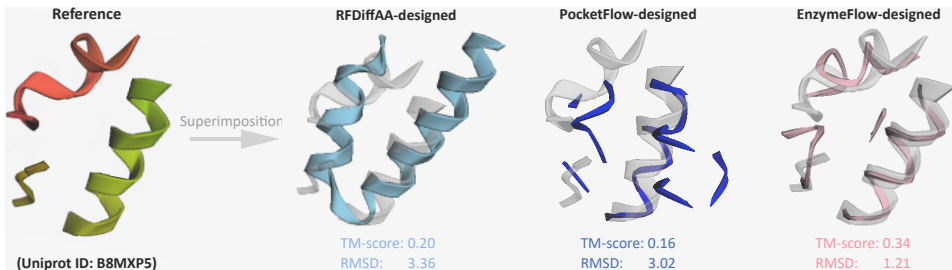


Figure 5: Case study of catalytic pocket design (UniProt: B8MXP5). We show the reference and designed pockets of different models. The pocket generation is conditioned on reaction OC[C@H]1O[C@@H](O)C(=O)C(=O)O[C@H]1O of EC3.

ECacc, and competitive performance in AAR. This demonstrates that EnzymeFlow is capable of generating more structurally valid catalytic pockets, aligning with the enzyme function analysis presented in Fig. 6. The average improvements over RFDiffAA in cRMSD, TM-score, AAR, and EC-Acc are 23.9%, 7.8%, 41.1%, and 64.7%, respectively. Additionally, EnzymeFlow slightly outperforms PocketFlow in catalytic-substrate binding, showing improved affinity scores ( $K_d$ ) and structural confidence ( $chai$ ) by 2.1% and 9.8%, respectively.

However, EnzymeFlow underperforms RFDiffAA in binding scores, reflected by lower affinities and structural confidence. Nonetheless, considering that the affinities of EnzymeFlow-generated catalytic pockets (-5.03) are close to those of enzyme-reaction pairs in the evaluation set (-4.65), the binding of EnzymeFlow remains acceptable, as enzymes and substrates do not always require tight binding to catalyze reactions because of the kinetic mechanism (Cleland, 1977; Arcus & Mulholland, 2020).

## 5.2 QUANTITATIVE ANALYSIS OF ENZYME FUNCTION

The key question is how we can *quantitatively* assess enzyme functions, *i.e.*, catalytic ability, of the generated pockets for a given reaction. To answer this, we perform enzyme function analysis on the designed catalytic pockets. Accurate annotated enzyme function is important for catalytic pocket design because it helps identify the functionality and the active sites that should be preserved or modified to improve catalytic efficiency (Rost, 2002; Barglow & Cravatt, 2007; Yu et al., 2023).

**Enzyme Function Comparison.** In EnzymeFlow, we co-annotate the enzyme function alongside the catalytic pocket design, allowing their functions to directly influence the structure generation. This integration of enzyme function annotation into EnzymeFlow ensures functionality control throughout the design. For baselines that design general proteins rather than enzyme-specific pockets, we perform enzyme function annotation post-hoc using CLEAN (Yu et al., 2023) to classify and annotate the EC-class of the generated pockets. After labeling each generated pocket with a EC-class, we compare it to the ground-truth EC-class associated with the actual reaction to compute EC-class accuracy, which quantifies how well the generated pockets align with the intended enzyme functions.

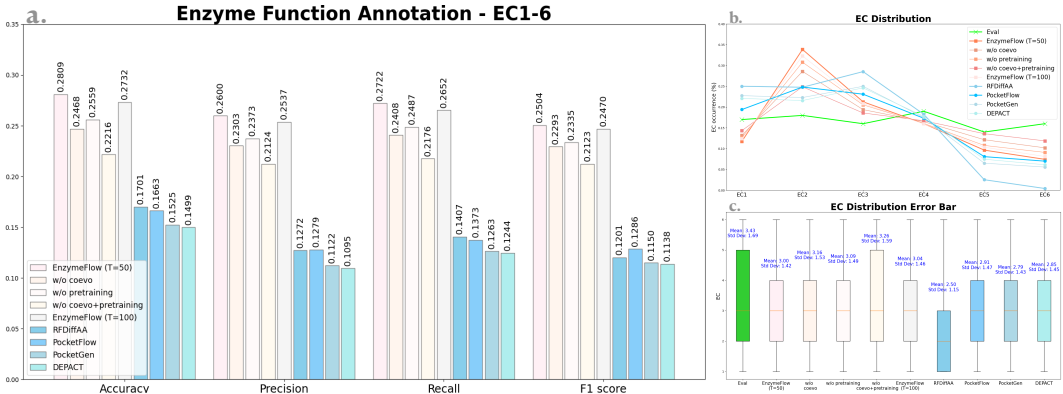


Figure 6: Quantitative comparison of annotated enzyme functions between EnzymeFlow- and baseline-generated catalytic pockets across all EC classes. Light color represents EnzymeFlow and ablation models, blue color represents baseline pocket design models, green color represents ground-truth data. (a) Evaluation of annotated functions using four multi-label accuracy metrics: accuracy, precision, recall, and F1 score. (b) Occurrence of Enzyme Commission (EC) numbers in generated catalytic pockets compared to the ground-truth occurrence. (c) Distribution of EC numbers in generated catalytic pockets with mean and standard deviation compared to the ground-truth.

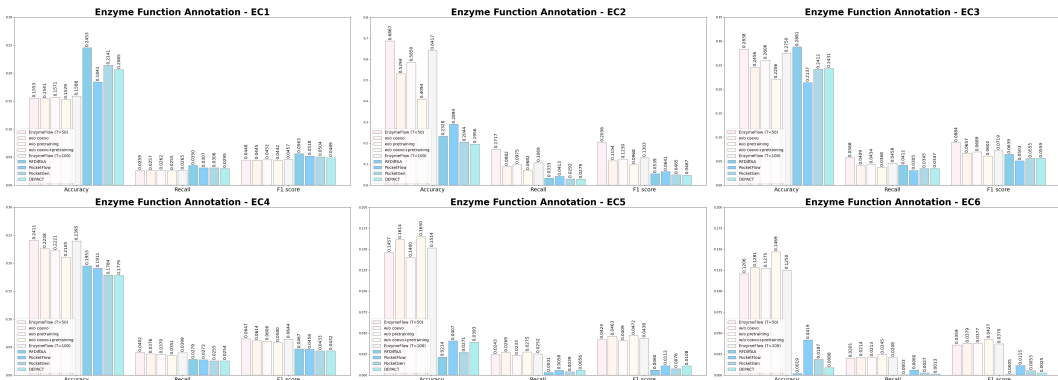


Figure 7: Quantitative comparison of annotated enzyme functions between EnzymeFlow- and baseline-generated catalytic pockets per EC-class, using accuracy, recall, and F1 score. Light color represents EnzymeFlow and ablation models, blue color represents baseline pocket design models.

**Results.** We quantitatively compare the annotated enzyme functions between EnzymeFlow- and baseline-generated catalytic pockets across all EC classes in Fig. 6, and compare the per-class performance in Fig. 7. These figures allow us to interpret the functions of enzyme catalytic pockets designed by different models. From Fig. 6(a), EnzymeFlow and its ablation models achieve the highest values across various multi-label accuracy metrics, including accuracy (0.2809), precision (0.2600), recall (0.2722), and F1 score (0.2504), outperforming models like RFDiffAA and PocketFlow. In Fig. 6(b), we observe the per-class distribution of functions in generated catalytic pockets, where EnzymeFlow generates pockets with a function distribution closer to the evaluation data, peaking at EC2. Baseline models, however, tend to notably underperform on EC5 and EC6. This trend is further highlighted in Fig. 6(c), which visualizes the mean and standard deviation of the functions in generated catalytic pockets. The evaluation data have a mean of 3.43 and a standard deviation of 1.69 for enzyme functions, and EnzymeFlow-generated pockets are much closer to these values compared to baseline models, which show less alignment with the evaluation data.

Additionally, Fig. 7 illustrates per-class enzyme function accuracy, where EnzymeFlow demonstrates strong performance in EC2, EC4, EC5, and EC6, competitive performance in EC3, but slightly weaker performance in EC1 compared to baseline models. Baseline models tend to perform poorly in EC5 and EC6, with per-class occurrence and accuracy showing values close to 0. In contrast, EnzymeFlow generates more functionally diverse and accurate catalytic pockets, maintaining higher accuracy across different EC classes. In conclusion, EnzymeFlow generates catalytic pockets that are better compared to other pocket design models, providing more accurate and diverse enzyme functions, which suggests enhanced catalytic potential.

Overall, from both functional and structural perspectives, EnzymeFlow leverages enzyme-reaction co-evolution, which effectively captures the dynamic changes in a catalytic reaction as substrates are converted into products. This introduces function-based enzyme design, allowing for the generation of more functionally and structurally valid catalytic pockets when targeting specific reactions.

## 6 LIMITATION AND FUTURE WORK

EnzymeFlow addresses key challenges in designing enzyme catalytic pockets for specific reactions, but several limitations remain. The first limitation is that EnzymeFlow currently generates only the catalytic pocket residues, rather than the entire enzyme structure. Ideally, the catalytic pocket should be designed first, followed by the design or reconstruction of the full enzyme structure based on the pocket. While we are developing to use ESM3 (Hayes et al., 2024) to reconstruct the full enzyme structure based on the designed catalytic pocket (discussed in App. I), this is not the most ideal solution. ESM3 is not specifically trained for enzyme-related tasks, which may limit its performance in enzyme design. In future versions of EnzymeFlow, we are working to fine-tune large biological models like ESM3 (Hayes et al., 2024), RFDiffAA (Krishna et al., 2024), or Genie2 (Lin et al., 2024) to specialize them for enzyme-related tasks, particularly for inpainting functional motifs of enzymes (enzyme catalytic motif scaffolding). Additionally, we aim to create an end-to-end model that combines EnzymeFlow with these large models, enabling catalytic pocket generation and functional motif inpainting in a single step, rather than in a two-step process. The second limitation, though minor, is that EnzymeFlow currently operates only on enzyme backbones and does not model or generate enzyme side chains. In future work, we plan to incorporate models like DiffPack (Zhang et al., 2024c) or develop a full-atom model to address this.

## REFERENCES

- Josh Abramson, Jonas Adler, Jack Dunger, Richard Evans, Tim Green, Alexander Pritzel, Olaf Ronneberger, Lindsay Willmore, Andrew J Ballard, Joshua Bambrick, et al. Accurate structure prediction of biomolecular interactions with alphafold 3. *Nature*, pp. 1–3, 2024.
- Michael S. Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants, 2023. URL <https://arxiv.org/abs/2209.15571>.
- Mohammed AlQuraishi. End-to-end differentiable learning of protein structure. *Cell systems*, 8(4): 292–301, 2019.
- Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.
- Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–3402, 1997.
- Vickery L Arcus and Adrian J Mulholland. Temperature, dynamics, and enzyme-catalyzed reaction rates. *Annual review of biophysics*, 49(1):163–180, 2020.
- Minkyung Baek, Frank DiMaio, Ivan Anishchenko, Justas Dauparas, Sergey Ovchinnikov, Gyu Rie Lee, Jue Wang, Qian Cong, Lisa N Kinch, R Dustin Schaeffer, et al. Accurate prediction of protein structures and interactions using a three-track neural network. *Science*, 373(6557):871–876, 2021.
- Amos Bairoch. The enzyme database in 2000. *Nucleic acids research*, 28(1):304–305, 2000.
- Parit Bansal, Anne Morgat, Kristian B Axelsen, Venkatesh Muthukrishnan, Elisabeth Coudert, Lucila Aimó, Nevila Hyka-Nouspikel, Elisabeth Gasteiger, Arnaud Kerhornou, Teresa Batista Neto, et al. Rhea, the reaction knowledgebase in 2022. *Nucleic acids research*, 50(D1):D693–D700, 2022.
- Katherine T Barglow and Benjamin F Cravatt. Activity-based protein profiling for the functional annotation of enzymes. *Nature methods*, 4(10):822–827, 2007.
- Jesse D Bloom and Frances H Arnold. In the light of directed evolution: pathways of adaptive protein evolution. *Proceedings of the National Academy of Sciences*, 106(supplement\_1):9995–10000, 2009.
- Rosalin Bonetta and Gianluca Valentino. Machine learning techniques for protein function prediction. *Proteins: Structure, Function, and Bioinformatics*, 88(3):397–413, 2020.
- Avishek Joey Bose, Tara Akhound-Sadegh, Kilian Fatras, Guillaume Hugué, Jarrid Rector-Brooks, Cheng-Hao Liu, Andrei Cristian Nica, Maksym Korablyov, Michael Bronstein, and Alexander Tong. Se (3)-stochastic flow matching for protein backbone generation. *arXiv preprint arXiv:2310.02391*, 2023.
- Adam R Boyko, Scott H Williamson, Amit R Indap, Jeremiah D Degenhardt, Ryan D Hernandez, Kirk E Lohmueller, Mark D Adams, Steffen Schmidt, John J Sninsky, Shamil R Sunyaev, et al. Assessing the evolutionary impact of amino acid mutations in the human genome. *PLoS genetics*, 4(5):e1000083, 2008.
- Andrew Campbell, Jason Yim, Regina Barzilay, Tom Rainforth, and Tommi Jaakkola. Generative flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design. *arXiv preprint arXiv:2402.04997*, 2024.
- Eleanor Campbell, Miriam Kaltenbach, Galen J Correy, Paul D Carr, Benjamin T Porebski, Emma K Livingstone, Livnat Afriat-Jurnou, Ashley M Buckle, Martin Weik, Florian Hollfelder, et al. The role of protein dynamics in the evolution of new enzyme function. *Nature chemical biology*, 12(11):944–950, 2016.
- Ron Caspi, Richard Billington, Ingrid M Keseler, Anamika Kothari, Markus Krummenacker, Peter E Midford, Wai Kit Ong, Suzanne Paley, Pallavi Subhraveti, and Peter D Karp. The metacyc database of metabolic pathways and enzymes-a 2019 update. *Nucleic acids research*, 48(D1):D445–D453, 2020.

- Chai. Chai-1 technical report. [https://chaiassets.com/chai-1/paper/technical\\_report\\_v1.pdf](https://chaiassets.com/chai-1/paper/technical_report_v1.pdf), 2024.
- Yaoxi Chen, Quan Chen, and Haiyan Liu. Depact and pacmatch: A workflow of designing de novo protein pockets to bind small molecules. *Journal of Chemical Information and Modeling*, 62(4): 971–985, 2022.
- Alexander E Chu, Jinho Kim, Lucy Cheng, Gina El Nesr, Minkai Xu, Richard W Shuai, and Po-Ssu Huang. An all-atom protein generative model. *Proceedings of the National Academy of Sciences*, 121(27):e2311500121, 2024.
- W Wallace Cleland. Determining the chemical mechanisms of enzyme-catalyzed reactions by kinetic studies. *Adv Enzymol Relat Areas Mol Biol*, 45:273–387, 1977.
- Gene Ontology Consortium. The gene ontology (go) database and informatics resource. *Nucleic acids research*, 32(suppl\_1):D258–D261, 2004.
- Robert A Copeland. *Enzymes: a practical introduction to structure, mechanism, and data analysis*. John Wiley & Sons, 2023.
- Gabriele Corso, Hannes Stärk, Bowen Jing, Regina Barzilay, and Tommi Jaakkola. Diffdock: Diffusion steps, twists, and turns for molecular docking. *arXiv preprint arXiv:2210.01776*, 2022.
- Justas Dauparas, Gyu Rie Lee, Robert Pecoraro, Linna An, Ivan Anishchenko, Cameron Glasscock, and David Baker. Atomic context-conditioned protein sequence design using ligandmpnn. *Biorxiv*, pp. 2023–12, 2023.
- Valerie de Crecy-Lagard, Raquel Dias, Iddo Friedberg, Yifeng Yuan, and Manal Swairjo. Limitations of current machine-learning models in predicting enzymatic functions for uncharacterized proteins. *bioRxiv*, pp. 2024–07, 2024.
- Mark A DePristo, Daniel M Weinreich, and Daniel L Hartl. Missense meanderings in sequence space: a biophysical view of protein evolution. *Nature Reviews Genetics*, 6(9):678–687, 2005.
- J-L Ferrer, MB Austin, C Stewart Jr, and JP Noel. Structure and function of enzymes involved in the biosynthesis of phenylpropanoids. *Plant Physiology and Biochemistry*, 46(3):356–370, 2008.
- Hazel M Girvan and Andrew W Munro. Applications of microbial cytochrome p450 enzymes in biotechnology and synthetic biology. *Current opinion in chemical biology*, 31:136–145, 2016.
- Margaret E Glasner, John A Gerlt, and Patricia C Babbitt. Evolution of enzyme superfamilies. *Current opinion in chemical biology*, 10(5):492–497, 2006.
- Vladimir Gligorijević, P Douglas Renfrew, Tomasz Kosciolk, Julia Koehler Leman, Daniel Berenberg, Tommi Vatanen, Chris Chandler, Bryn C Taylor, Ian M Fisk, Hera Vlamakis, et al. Structure-based protein function prediction using graph convolutional networks. *Nature communications*, 12(1):3168, 2021.
- Tomas Hayes, Roshan Rao, Halil Akin, Nicholas J Sofroniew, Deniz Oktay, Zeming Lin, Robert Verkuil, Vincent Q Tran, Jonathan Deaton, Marius Wiggert, et al. Simulating 500 million years of evolution with a language model. *bioRxiv*, pp. 2024–07, 2024.
- Esther Heid, Daniel Probst, William H Green, and Georg KH Madsen. Enzymemap: curation, validation and data-driven prediction of enzymatic reactions. *Chemical Science*, 14(48):14229–14242, 2023.
- Maarten L Hekkelman, Ida de Vries, Robbie P Joosten, and Anastassis Perrakis. Alphafill: enriching alphafold models with ligands and cofactors. *Nature Methods*, 20(2):205–213, 2023.
- C Eric Hodgman and Michael C Jewett. Cell-free synthetic biology: thinking outside the cell. *Metabolic engineering*, 14(3):261–269, 2012.
- Chenqing Hua, Sitao Luan, Qian Zhang, and Jie Fu. Graph neural networks intersect probabilistic graphical models: A survey. *arXiv preprint arXiv:2206.06089*, 2022a.



- Chenqing Hua, Guillaume Rabusseau, and Jian Tang. High-order pooling for graph neural networks with tensor decomposition. *Advances in Neural Information Processing Systems*, 35:6021–6033, 2022b.
- Chenqing Hua, Sitao Luan, Minkai Xu, Rex Ying, Jie Fu, Stefano Ermon, and Doina Precup. Mudiff: Unified diffusion for complete molecule generation. *arXiv preprint arXiv:2304.14621*, 2023.
- Chenqing Hua, Connor Coley, Guy Wolf, Doina Precup, and Shuangjia Zheng. Effective protein-protein interaction exploration with ppi retrieval. *arXiv preprint arXiv:2402.03675*, 2024a.
- Chenqing Hua, Bozita Zhong, Sitao Luan, Liang Hong, Guy Wolf, Doina Precup, and Shuangjia Zheng. Reactzyme: A benchmark for enzyme-reaction prediction. *arXiv preprint arXiv:2408.13659*, 2024b.
- Jaime Huerta-Cepas, Damian Szklarczyk, Davide Heller, Ana Hernández-Plaza, Sofia K Forslund, Helen Cook, Daniel R Mende, Ivica Letunic, Thomas Rattei, Lars J Jensen, et al. eggNOG 5.0: a hierarchical, functionally and phylogenetically annotated orthology resource based on 5090 organisms and 2502 viruses. *Nucleic acids research*, 47(D1):D309–D314, 2019.
- Clemens Isert, Kenneth Atz, and Gisbert Schneider. Structure-based drug design with geometric deep learning. *Current Opinion in Structural Biology*, 79:102548, 2023.
- Christian Jäckel, Peter Kast, and Donald Hilvert. Protein design by directed evolution. *Annu. Rev. Biophys.*, 37(1):153–173, 2008.
- Roy A Jensen. Enzyme recruitment in evolution of new function. *Annual review of microbiology*, 30(1):409–425, 1976.
- John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.
- Jay D Keasling. Manufacturing molecules through metabolic engineering. *Science*, 330(6009):1355–1358, 2010.
- George A Khoury, James Smadbeck, Chris A Kieslich, and Christodoulos A Floudas. Protein folding and de novo protein design for biotechnological applications. *Trends in biotechnology*, 32(2):99–109, 2014.
- Joseph Kraut. How do enzymes work? *Science*, 242(4878):533–540, 1988.
- Rohith Krishna, Jue Wang, Woody Ahern, Pascal Sturmfels, Preetham Venkatesh, Indrek Kalvet, Gyu Rie Lee, Felix S Morey-Burrows, Ivan Anishchenko, Ian R Humphreys, et al. Generalized biomolecular modeling and design with rosettafold all-atom. *Science*, 384(6693):ead12528, 2024.
- Alexander Kroll, Sahasra Ranjan, Martin KM Engqvist, and Martin J Lercher. A general model to predict small molecule substrates of enzymes based on machine and deep learning. *Nature communications*, 14(1):2787, 2023a.
- Alexander Kroll, Yvan Rousset, Xiao-Pan Hu, Nina A Liebrand, and Martin J Lercher. Turnover number predictions for kinetically uncharacterized enzymes using machine and deep learning. *Nature Communications*, 14(1):4139, 2023b.
- Maxat Kulmanov and Robert Hoehndorf. Deepgoplus: improved protein function prediction from sequence. *Bioinformatics*, 36(2):422–429, 2020.
- Haitao Lin, Yufei Huang, Meng Liu, Xuanjing Li, Shuiwang Ji, and Stan Z Li. Diffbp: Generative diffusion of 3d molecules for target protein binding. *arXiv preprint arXiv:2211.11214*, 2022.
- Yeqing Lin, Minji Lee, Zhao Zhang, and Mohammed AlQuraishi. Out of many, one: Designing and scaffolding proteins at the scale of the structural universe with genie 2. *arXiv preprint arXiv:2405.15489*, 2024.

- Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.
- Wenfang Liu and Ping Wang. Cofactor regeneration for sustainable enzymatic biosynthesis. *Biotechnology advances*, 25(4):369–384, 2007.
- Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow, 2022. URL <https://arxiv.org/abs/2209.03003>.
- Wei Lu, Jixian Zhang, Weifeng Huang, Ziqiao Zhang, Xiangyu Jia, Zhenyu Wang, Leilei Shi, Chengtao Li, Peter G Wolynes, and Shuangjia Zheng. Dynamicbind: Predicting ligand-specific protein-ligand complex structure with a deep equivariant generative model. *Nature Communications*, 15(1):1071, 2024.
- Sitao Luan, Mingde Zhao, Chenqing Hua, Xiao-Wen Chang, and Doina Precup. Complete the missing half: Augmenting aggregation filtering with diversification for graph convolutional networks. *arXiv preprint arXiv:2008.08844*, 2020.
- Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, and Doina Precup. Revisiting heterophily for graph neural networks. *Advances in neural information processing systems*, 35:1362–1375, 2022.
- Sitao Luan, Chenqing Hua, Qincheng Lu, Liheng Ma, Lirong Wu, Xinyu Wang, Minkai Xu, Xiao-Wen Chang, Doina Precup, Rex Ying, et al. The heterophilic graph learning handbook: Benchmarks, models, theoretical analysis, applications and challenges. *arXiv preprint arXiv:2407.09618*, 2024a.
- Sitao Luan, Chenqing Hua, Minkai Xu, Qincheng Lu, Jiaqi Zhu, Xiao-Wen Chang, Jie Fu, Jure Leskovec, and Doina Precup. When do graph neural networks help with node classification? investigating the homophily principle on node distinguishability. *Advances in Neural Information Processing Systems*, 36, 2024b.
- Xizeng Mao, Tao Cai, John G Olyarchuk, and Liping Wei. Automated genome annotation and pathway identification using the kegg orthology (ko) as a controlled vocabulary. *Bioinformatics*, 21(19):3787–3793, 2005.
- Andrew CR Martin, Christine A Orengo, E Gail Hutchinson, Susan Jones, Maria Karmirantzou, Roman A Laskowski, John BO Mitchell, Chiara Taroni, and Janet M Thornton. Protein folds and functions. *Structure*, 6(7):875–884, 1998.
- Lukasz Maziarka, Tomasz Danel, Slawomir Mucha, Krzysztof Rataj, Jacek Tabor, and Stanislaw Jastrzebski. Molecule attention transformer. *arXiv preprint arXiv:2002.08264*, 2020.
- Peter G Mikhael, Itamar Chinn, and Regina Barzilay. Clipzyme: Reaction-conditioned virtual screening of enzymes. *arXiv preprint arXiv:2402.06748*, 2024.
- Yukito Murakami, Jun-ichi Kikuchi, Yoshio Hisaeda, and Osamu Hayashida. Artificial enzymes. *Chemical reviews*, 96(2):721–758, 1996.
- Lianet Noda-Garcia, Wolfram Liebermeister, and Dan S Tawfik. Metabolite–enzyme coevolution: from single enzymes to metabolic pathways and networks. *Annual Review of Biochemistry*, 87(1):187–216, 2018.
- Noel M O’Boyle, Michael Banck, Craig A James, Chris Morley, Tim Vandermeersch, and Geoffrey R Hutchison. Open babel: An open chemical toolbox. *Journal of cheminformatics*, 3:1–14, 2011.
- Csaba Pál, Balázs Papp, and Martin J Lercher. An integrated view of protein evolution. *Nature reviews genetics*, 7(5):337–348, 2006.
- Marta Pelay-Gimeno, Adrian Glas, Oliver Koch, and Tom N Grossmann. Structure-based design of inhibitors of protein–protein interactions: mimicking peptide binding epitopes. *Angewandte Chemie International Edition*, 54(31):8896–8927, 2015.
- Gaspar P Pinto, Marina Corbella, Andrey O Demkiv, and Shina Caroline Lynn Kamerlin. Exploiting enzyme evolution for computational protein design. *Trends in Biochemical Sciences*, 47(5):375–389, 2022.

- Roshan M Rao, Jason Liu, Robert Verkuil, Joshua Meier, John Canny, Pieter Abbeel, Tom Sercu, and Alexander Rives. Msa transformer. In *International Conference on Machine Learning*, pp. 8844–8856. PMLR, 2021.
- Burkhard Rost. Enzyme function less conserved than anticipated. *Journal of molecular biology*, 318(2):595–608, 2002.
- Jae Yong Ryu, Hyun Uk Kim, and Sang Yup Lee. Deep learning enables high-quality and high-throughput prediction of enzyme commission numbers. *Proceedings of the National Academy of Sciences*, 116(28):13996–14001, 2019.
- Victor Garcia Satorras, Emiel Hoogetboom, and Max Welling. E(n) equivariant graph neural networks. In *International conference on machine learning*, pp. 9323–9332. PMLR, 2021.
- Ida Schomburg, Antje Chang, Oliver Hofmann, Christian Ebeling, Frank Ehrentreich, and Dietmar Schomburg. Brenda: a resource for enzyme data and metabolic information. *Trends in biochemical sciences*, 27(1):54–56, 2002.
- Philippe Schwaller, Benjamin Hoover, Jean-Louis Reymond, Hendrik Strobelt, and Teodoro Laino. Extraction of organic chemistry grammar from unsupervised learning of chemical reactions. *Science Advances*, 7(15):eabe4166, 2021.
- Tobias Sikosek and Hue Sun Chan. Biophysics of protein evolution and evolutionary protein biophysics. *Journal of The Royal Society Interface*, 11(100):20140419, 2014.
- Hannes Stärk, Bowen Jing, Regina Barzilay, and Tommi Jaakkola. Harmonic self-conditioned flow matching for multi-ligand docking and binding site design. *arXiv preprint arXiv:2310.05764*, 2023.
- Martin Steinegger and Johannes Söding. Mmseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature biotechnology*, 35(11):1026–1028, 2017.
- Janet M Thornton, Christine A Orengo, Annabel E Todd, and Frances MG Pearl. Protein folds, functions and evolution. *Journal of molecular biology*, 293(2):333–342, 1999.
- Oleg Trott and Arthur J Olson. Autodock vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *Journal of computational chemistry*, 31(2):455–461, 2010.
- Jérôme Tubiana, Dina Schneidman-Duhovny, and Haim J Wolfson. Scannet: an interpretable geometric deep learning model for structure-based protein binding site prediction. *Nature Methods*, 19(6):730–739, 2022.
- Bart van Beusekom, Wouter G Touw, Mahidhar Tatineni, Sandeep Somani, Gunaretnam Rajagopal, Jinquan Luo, Gary L Gilliland, Anastassis Perrakis, and Robbie P Joosten. Homology-based hydrogen bond information improves crystallographic structures in the pdb. *Protein Science*, 27(3):798–808, 2018.
- Michel van Kempen, Stephanie S Kim, Charlotte Tumescheit, Milot Mirdita, Cameron LM Gilchrist, Johannes Söding, and Martin Steinegger. Foldseek: fast and accurate protein structure search. *Biorxiv*, pp. 2022–02, 2022.
- Jue Wang, Sidney Lisanza, David Juergens, Doug Tischer, Ivan Anishchenko, Minkyung Baek, Joseph L Watson, Jung Ho Chun, Lukas F Milles, Justas Dauparas, et al. Deep learning methods for designing proteins scaffolding functional sites. *BioRxiv*, pp. 2021–11, 2021.
- Renxiao Wang, Xueliang Fang, Yipin Lu, and Shaomeng Wang. The pdbind database: Collection of binding affinities for protein- ligand complexes with known three-dimensional structures. *Journal of medicinal chemistry*, 47(12):2977–2980, 2004.
- Joseph L Watson, David Juergens, Nathaniel R Bennett, Brian L Trippe, Jason Yim, Helen E Eisenach, Woody Ahern, Andrew J Borst, Robert J Ragotte, Lukas F Milles, et al. De novo design of protein structure and function with rfdiffusion. *Nature*, 620(7976):1089–1100, 2023.

- David Whitford. *Proteins: structure and function*. John Wiley & Sons, 2013.
- Yu Xia and Michael Levitt. Simulating protein evolution in sequence and structure space. *Current Opinion in Structural Biology*, 14(2):202–207, 2004.
- Zhaoping Xiong, Dingyan Wang, Xiaohong Liu, Feisheng Zhong, Xiaozhe Wan, Xutong Li, Zhaojun Li, Xiaomin Luo, Kaixian Chen, Hualiang Jiang, et al. Pushing the boundaries of molecular representation for drug discovery with the graph attention mechanism. *Journal of medicinal chemistry*, 63(16):8749–8760, 2019.
- Jason Yang, Ariane Mora, Shengchao Liu, Bruce J Wittmann, Anima Anandkumar, Frances H Arnold, and Yisong Yue. Care: a benchmark suite for the classification and retrieval of enzymes. *arXiv preprint arXiv:2406.15669*, 2024.
- Kevin K Yang, Zachary Wu, and Frances H Arnold. Machine-learning-guided directed evolution for protein engineering. *Nature methods*, 16(8):687–694, 2019.
- Jason Yim, Andrew Campbell, Andrew YK Foong, Michael Gastegger, José Jiménez-Luna, Sarah Lewis, Victor Garcia Satorras, Bastiaan S Veeling, Regina Barzilay, Tommi Jaakkola, et al. Fast protein backbone generation with se (3) flow matching. *arXiv preprint arXiv:2310.05297*, 2023a.
- Jason Yim, Brian L Trippe, Valentin De Bortoli, Emile Mathieu, Arnaud Doucet, Regina Barzilay, and Tommi Jaakkola. Se (3) diffusion model with application to protein backbone generation. *arXiv preprint arXiv:2302.02277*, 2023b.
- Tianhao Yu, Haiyang Cui, Jianan Canal Li, Yunan Luo, Guangde Jiang, and Huimin Zhao. Enzyme function prediction using contrastive learning. *Science*, 379(6639):1358–1363, 2023.
- Odin Zhang, Yufei Huang, Shichen Cheng, Mengyao Yu, Xujun Zhang, Haitao Lin, Yundian Zeng, Mingyang Wang, Zhenxing Wu, Huifeng Zhao, et al. Deep geometry handling and fragment-wise molecular 3d graph generation. *arXiv preprint arXiv:2404.00014*, 2024a.
- Odin Zhang, Jieyu Jin, Haitao Lin, Jintu Zhang, Chenqing Hua, Yufei Huang, Huifeng Zhao, Chang-Yu Hsieh, and Tingjun Hou. Ecloudgen: Access to broader chemical space for structure-based molecule generation. *bioRxiv*, pp. 2024–06, 2024b.
- Xujun Zhang, Odin Zhang, Chao Shen, Wanglin Qu, Shicheng Chen, Hanqun Cao, Yu Kang, Zhe Wang, Ercheng Wang, Jintu Zhang, et al. Efficient and accurate large library ligand docking with karmadock. *Nature Computational Science*, 3(9):789–804, 2023a.
- Yangtian Zhang, Zuobai Zhang, Bozita Zhong, Sanchit Misra, and Jian Tang. Diffpack: A torsional diffusion model for autoregressive protein side-chain packing. *Advances in Neural Information Processing Systems*, 36, 2024c.
- Zaixi Zhang, Zepu Lu, Hao Zhongkai, Marinka Zitnik, and Qi Liu. Full-atom protein pocket design via iterative refinement. *Advances in Neural Information Processing Systems*, 36:16816–16836, 2023b.
- Zaixi Zhang, Wanxiang Shen, Qi Liu, and Marinka Zitnik. Generalized protein pocket generation with prior-informed flow matching. *Advances in neural information processing systems*, 2024d.
- Zaixi Zhang, Wanxiang Shen, Qi Liu, and Marinka Zitnik. Pocketgen: Generating full-atom ligand-binding protein pockets. *bioRxiv*, pp. 2024–02, 2024e.
- Zuobai Zhang, Minghao Xu, Arian Jamasb, Vijil Chenthamarakshan, Aurelie Lozano, Payel Das, and Jian Tang. Protein representation learning by geometric structure pretraining. *arXiv preprint arXiv:2203.06125*, 2022.
- Emile Zuckerkandl and Linus Pauling. Molecules as documents of evolutionary history. *Journal of theoretical biology*, 8(2):357–366, 1965.



## A FUTURE WORK IN PROGRESS: AI-DRIVEN ENZYME DESIGN PLATFORM

As discussed in Sec. 6, there are several limitations in the current version of EnzymeFlow. Here, we briefly outline the next steps and improvements we are actively working on for the upcoming version. Currently, EnzymeFlow generates only catalytic pocket residues rather than full enzyme structures. Ideally, the catalytic pocket should be designed first, followed by the reconstruction of the full enzyme structure based on the pocket. While we currently use ESM3 (Hayes et al., 2024) for this reconstruction, this approach is not ideal. Fine-tuning ESM3 or RFDiffAA (Krishna et al., 2024) would be preferable, but unfortunately, training scripts for these wonderful models are not provided, making it impossible to directly fine-tune them on our EnzymeFill dataset.

To address this, we are borrowing concepts from Wang et al. (2021) and Lin et al. (2024), which focuses on inpainting proteins and scaffolding functional motifs. We are working to integrate this concept into EnzymeFlow’s pipeline, as part of our primary design. Our goal is to develop an end-to-end automated AI-driven enzyme discovery system that works as follows:

- **1. Catalytic Pocket Design:** The system will first design enzyme catalytic pockets.
- **2. Scaffolding Functional Motifs:** Next, it will scaffold the functional motifs to generate full enzyme structures.
- **3. Substrate Docking:** Using methods like DiffDock (Corso et al., 2022), DynamicBind (Lu et al., 2024), or fine-tuned Chai (Chai, 2024) on EnzymeFill, the system will bind substrates to the catalytic pockets.
- **4. Inverse Folding:** The enzyme-substrate complex will undergo inverse folding using LigandMPNN (Dauparas et al., 2023).
- **5. Computational Screening:** Finally, the system will perform computational screening to select the best-generated enzymes.

This entire process is being developed into an integrated, end-to-end solution for AI-driven enzyme design. We are very excited about the potential of this project and look forward to achieving a fully automated enzyme design system in the near future.

## B OPEN DISCUSSION: WHY IS SUBSTRATE/REACTION-SPECIFIED ENZYME DESIGN NEEDED?

EnzymeFlow is unique in its leading approach to function-based *de novo* protein design. Currently, most protein design models, whether focused on backbone generation (Yim et al., 2023a;b; Bose et al., 2023; Campbell et al., 2024; Krishna et al., 2024) or pocket design (Zhang et al., 2023b;a; 2024e;d), are structure-based. These models aim to design or modify proteins to achieve a specific 3D structure, prioritizing stability, folding, and molecular interactions. The design process typically involves optimizing a protein structure to minimize energy and achieve a stable structural conformation (Khoury et al., 2014; Pelay-Gimeno et al., 2015).

In contrast, function-based protein design focuses on creating proteins that perform specific biochemical tasks, such as catalysis, signaling, or even binding (Martin et al., 1998; Thornton et al., 1999). These models are driven by the need for proteins to carry out particular functions rather than adopt a specific 3D structure. Function-based design often targets the active site or binding pockets, optimizing them for specific molecular interactions—in our case, the enzyme’s catalytic pockets.

Our philosophy is that protein function determines its structure, meaning that a protein folds into a specific 3D shape to achieve its intended function, and the resulting structure can then be translated into a proper sequence—essentially, *protein function*  $\rightarrow$  *protein structure*  $\rightarrow$  *protein sequence*. EnzymeFlow follows this philosophy. Specifically, the function of an enzyme is determined by its ability to catalyze a specific reaction or interact with a specific substrate. Therefore, our enzyme pocket design process begins with the reaction or substrate in mind, incorporating reaction/substrate specificity into the generation process. The reaction or substrate represents the functional target for the generated enzyme pockets.

In this approach, EnzymeFlow generates enzyme pocket structures specified for the desired protein function, which contrasts with current generative methods that prioritize structure first. These existing

methods operate on the idea that *protein structure*  $\rightarrow$  *protein function*  $\rightarrow$  *protein sequence*. However, proteins should be designed primarily for their functionality, not just their structures. EnzymeFlow’s focus on function-based design could serve as an inspiration for future advancements, leading the way toward more purposeful, function-driven protein design.

## C RELATED WORK

### C.1 PROTEIN REPRESENTATION LEARNING

Graph representation learning emerges as a potent strategy for representing and learning about proteins and molecules, focusing on structured, non-Euclidean data (Satorras et al., 2021; Luan et al., 2020; 2022; Hua et al., 2022a;b; Luan et al., 2024b;a). In this context, proteins and molecules can be effectively modeled as 2D graphs or 3D point clouds, where nodes correspond to individual atoms or residues, and edges represent interactions between them (Gligorićević et al., 2021; Zhang et al., 2022; Hua et al., 2023; Zhang et al., 2024a). Indeed, representing proteins and molecules as graphs or point clouds offers a valuable approach for gaining insights into and learning the fundamental geometric and chemical mechanisms governing protein-ligand interactions. This representation allows for a more comprehensive exploration of the intricate relationships and structural features within protein-ligand structures (Tubiana et al., 2022; Isert et al., 2023; Zhang et al., 2024b).

### C.2 PROTEIN FUNCTION ANNOTATION

Protein function prediction aims to determine the biological role of a protein based on its sequence, structure, or other features. It is a crucial task in bioinformatics, often leveraging databases such as Gene Ontology (GO), Enzyme Commission (EC) numbers, and KEGG Orthology (KO) annotations (Bairoch, 2000; Consortium, 2004; Mao et al., 2005). Traditional methods like BLAST, PSI-BLAST, and eggNOG infer function by comparing sequence alignments and similarities (Altschul et al., 1990; 1997; Huerta-Cepas et al., 2019). Recently, deep learning has introduced more advanced approaches for protein function prediction (Ryu et al., 2019; Kulmanov & Hoehndorf, 2020; Bonetta & Valentino, 2020). There are two major types of function prediction models, one uses only protein sequence as their input, while the other also uses experimentally-determined or predicted protein structure as input. Typically, these methods predict EC or GO annotations to approximate protein functions, rather than describing the exact catalyzed reaction, which is a limitation of these approaches.

### C.3 PROTEIN EVOLUTION

Protein evolution learns how proteins change over time through processes such as mutation, selection, and genetic drift (Pál et al., 2006; Bloom & Arnold, 2009), which influence protein functions. Studies on protein evolution focus on understanding the molecular mechanisms driving changes in protein sequences and structures. Zuckerkandl & Pauling (1965) introduce the concept of the molecular clock, which postulates that proteins evolve at a relatively constant rate over time, providing a framework for estimating divergence times between species. DePristo et al. (2005) show that evolutionary rates are influenced by functional constraints, with regions critical to protein function (e.g., active sites, binding interfaces) evolving more slowly due to purifying selection. This understanding leads to the development of methods for detecting functionally important residues based on evolutionary conservation. Understanding protein evolution has practical applications in protein engineering. By studying how natural proteins evolve to acquire new functions, researchers design synthetic proteins with desired properties (Xia & Levitt, 2004; Jäckel et al., 2008). Additionally, deep learning models increasingly integrate evolutionary principles to predict protein function and stability, design novel enzymes, and guide protein engineering (Yang et al., 2019; AlQuraishi, 2019; Jumper et al., 2021).

### C.4 GENERATIVE MODELS FOR PROTEIN AND POCKET DESIGN

Recent advancements in generative models have advanced the field of protein design and binding pocket design, enabling the creation of proteins or binding pockets with desired properties and functions (Yim et al., 2023a;b; Chu et al., 2024; Hua et al., 2024a; Abramson et al., 2024). For example, RFDiff (Watson et al., 2023) employs denoising diffusion in conjunction with RoseTTAFold (Baek et al., 2021) for *de novo* protein structure design, achieving wet-lab-level generated structures that can be extended to binding pocket design. RFDiffAA (Krishna et al., 2024) extends RFDiff for joint modeling of protein and ligand structures, generating ligand-binding proteins and further leveraging GNNs for sequence design. Additionally, FAIR (Zhang et al., 2023b) and PocketGen

(Zhang et al., 2024e) use a two-stage coarse-to-fine refinement approach to co-design pocket structures and sequences. Recent models leveraging flow matching frameworks have shown promising results in these tasks. For instance, FoldFlow (Bose et al., 2023) introduces a series of flow models for protein backbone design, improving training stability and efficiency. FrameFlow (Yim et al., 2023a) further enhances sampling efficiency and demonstrates success in motif-scaffolding tasks using flow matching, while MultiFlow (Campbell et al., 2024) advances to structure and sequence co-design. These flow models, initially applied to protein backbones, have been further generalized to binding pockets. For example, PocketFlow (Zhang et al., 2024d) combines flow matching with physical priors to explicitly learn protein-ligand interaction types in pocket design, achieving superior results compared to RFDiffAA. While these models excel in protein and binding pocket design, they primarily focus on static protein-(ligand) interactions and lack the ability to model the chemical transformations involved in enzyme-substrate interactions. This limitation may reduce their accuracy and generalizability in designing enzyme pockets for catalytic reactions.

## D CO-EVOLUTIONARY MSA TRANSFORMER

Co-evolution captures the dynamic relationship between an enzyme and its substrate during a catalytic reaction. AlphaFold2 (Jumper et al., 2021) has demonstrated the critical importance of leveraging protein evolution, specifically through multiple sequence alignments (MSA) across protein sequences, to enhance a model’s generalizability and expressive power. Previous works, such as MSA Transformer (Rao et al., 2021) and EvoFormer (Jumper et al., 2021), have focused on encoding and learning protein evolution from MSA results. Proper co-evolution encodings of enzymes and reactions are essential for capturing the dynamic changes that occur during catalytic processes, not only in our EnzymeFlow model but in other models as well.

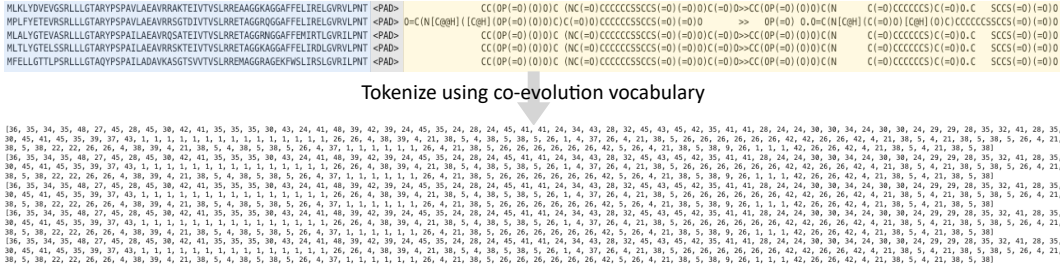


Figure 8: Enzyme-reaction co-evolution and tokenized representation.

### D.1 CO-EVOLUTION VOCABULARY

We provide our co-evolution dictionary for tokenization and encoding following:

```
{'pad': 0, ' ': 1, '#': 2, '%': 3, '(': 4, ')': 5, '*': 6, '+': 7, '-': 8, '.': 9,
'/': 10, '0': 11, '1': 12, '2': 13, '3': 14, '4': 15, '5': 16, '6': 17, '7': 18,
'8': 19, '9': 20, '=': 21, '>': 22, '@': 23, 'A': 24, 'B': 25, 'C': 26, 'D': 27,
'E': 28, 'F': 29, 'G': 30, 'H': 31, 'I': 32, 'J': 33, 'K': 34, 'L': 35, 'M': 36,
'N': 37, 'O': 38, 'P': 39, 'Q': 40, 'R': 41, 'S': 42, 'T': 43, 'U': 44, 'V': 45,
'W': 46, 'X': 47, 'Y': 48, 'Z': 49, '[': 50, ']'': 51, 'c': 52, 'e': 53, 's': 54,
'g': 55, 'i': 56, 'l': 57, 'n': 58, 'o': 59, 'r': 60, 's': 61, 'u': 62, '<unk>': 63}
```

Figure 9: EnzymeFlow co-evolution dictionary.

### D.2 COEVOFORMER IMPLEMENTATION

Here, we introduce a new co-evolutionary MSA transformer, coEvoFormer. The co-evolution of an enzyme-reaction pair is represented by a matrix  $U \in \mathbb{R}^{N_{\text{MSA}} \times N_{\text{token}}}$ , which combines the MSA results of enzyme sequences and reaction SMILES (illustrated in Fig. 2(3d)). In this matrix,  $N_{\text{MSA}}$  denotes the number of MSA sequences, and  $N_{\text{token}}$  denotes the length of the preserved MSA alignment. Each element  $u^{mn} \in \{1, \dots, 64\} \cup \{\mathbf{X}\}$  in  $U$  represents a tokenized character from our co-evolution vocabulary (provided in App. D.1), with  $\mathbf{X}$  indicating the *masking state*. The coEvoFormer takes the co-evolution matrix  $U$  as input and outputs an embedded co-evolution representation  $H_U \in \mathbb{R}^{N_{\text{MSA}} \times N_{\text{token}} \times D_{H_U}}$ , where  $D_{H_U}$  denotes the hidden dimension size.

The code for coEvoFormer follows directly:

```

1028 1 import math, copy
1029 2 import numpy as np
1030 3
1031 4 import torch
1032 5 import torch.nn as nn
1033 6 import torch.nn.functional as F
1034 7 from torch.autograd import Variable
1035 8
1036 9 ## Co-Evolution Transformer (coEvoFormer)
1037 10
1038 11 ## (12) Layer Norm
1039 12 class ResidualNorm(nn.Module):
1040 13     def __init__(self, size, dropout):
1041 14         super(ResidualNorm, self).__init__()
1042 15         self.norm = LayerNorm(size)
1043 16         self.dropout = nn.Dropout(dropout)
1044 17
1045 18     def forward(self, x, sublayer):
1046 19         return x + self.dropout(sublayer(self.norm(x)))
1047 20
1048 21
1049 22 ## (11) Residual Norm
1050 23 class LayerNorm(nn.Module):
1051 24     def __init__(self, features, eps=1e-6):
1052 25         super(LayerNorm, self).__init__()
1053 26         self.a_2 = nn.Parameter(torch.ones(features))
1054 27         self.b_2 = nn.Parameter(torch.zeros(features))
1055 28         self.eps = eps
1056 29
1057 30     def forward(self, x):
1058 31         mean = x.mean(-1, keepdim=True)
1059 32         std = x.std(-1, keepdim=True)
1060 33         x = self.a_2 * (x - mean) / (std + self.eps) + self.b_2
1061 34         return x
1062 35
1063 36
1064 37 ## (10) 2-layer MLP
1065 38 class MLP(nn.Module):
1066 39     def __init__(self, model_depth, ff_depth, dropout):
1067 40         super(MLP, self).__init__()
1068 41         self.w1 = nn.Linear(model_depth, ff_depth)
1069 42         self.w2 = nn.Linear(ff_depth, model_depth)
1070 43         self.dropout = nn.Dropout(dropout)
1071 44         self.silu = nn.SiLU()
1072 45
1073 46     def forward(self, x):
1074 47         return self.w2(self.dropout(self.silu(self.w1(x))))
1075 48
1076 49
1077 50 ## (9) Attention
1078 51 def attention(Q,K,V, mask=None):
1079 52     dk = Q.size(-1)
1080 53     T = (Q @ K.transpose(-2, -1))/math.sqrt(dk)
1081 54     if mask is not None:
1082 55         T = T.masked_fill_(mask.unsqueeze(1)==0, -1e9)
1083 56     T = F.softmax(T, dim=-1)
1084 57     return T @ V
1085 58
1086 59
1087 60 ## (8) Multi-Head Attention
1088 61 class MultiHeadAttention(nn.Module):
1089 62     def __init__(self,
1090 63                 num_heads,
1091 64                 embed_dim,
1092 65                 bias=False
1093 66                 ):
1094 67         super(MultiHeadAttention, self).__init__()
1095 68         self.num_heads = num_heads
1096 69         self.dk = embed_dim//num_heads
1097 70         self.WQ = nn.Linear(embed_dim, embed_dim, bias=bias)
1098 71         self.WK = nn.Linear(embed_dim, embed_dim, bias=bias)
1099 72         self.WV = nn.Linear(embed_dim, embed_dim, bias=bias)
1100 73         self.WO = nn.Linear(embed_dim, embed_dim, bias=bias)
1101 74
1102 75     def forward(self, x, kv, mask=None):
1103 76         batch_size = x.size(0)
1104 77         Q = self.WQ(x).view(batch_size, -1, self.num_heads, self.dk).transpose(1,2)
1105 78         K = self.WK(kv).view(batch_size, -1, self.num_heads, self.dk).transpose(1,2)
1106 79         V = self.WV(kv).view(batch_size, -1, self.num_heads, self.dk).transpose(1,2)

```



```

1080
1081     80
1082         if mask is not None:
1083             81
1084                 if len(mask.shape) == 2:
1085                     82
1086                         mask = torch.einsum('bi,bj->bij', mask, mask)
1087                     83
1088                         x = attention(Q, K, V, mask=mask)
1089                     84
1090                         x = x.transpose(1, 2).contiguous().view(batch_size, -1, self.num_heads*self.dk)
1091                     85
1092                         return self.WO(x)
1093                     86
1094             87
1095             88
1096             89
1097     ## (7) Positional Embedding
1098     90
1099     class PositionalEncoding(nn.Module):
1100     91
1101         def __init__(self, model_depth, max_len=5000):
1102     92
1103         super(PositionalEncoding, self).__init__()
1104     93
1105         pe = torch.zeros(max_len, model_depth)
1106     94
1107         position = torch.arange(0.0, max_len).unsqueeze(1)
1108     95
1109         div_term = torch.exp(torch.arange(0.0, model_depth, 2) *
1110     96
1111             -(math.log(10000.0) / model_depth))
1112     97
1113         pe[:, 0::2] = torch.sin(position * div_term)
1114     98
1115         pe[:, 1::2] = torch.cos(position * div_term)
1116     99
1117         pe = pe.unsqueeze(0)
1118     100
1119         self.register_buffer('pe', pe)
1120     101
1121
1122         def forward(self, x):
1123     102
1124         return x + Variable(self.pe[:, :x.size(1)], requires_grad=False)
1125     103
1126     104
1127     ## (6) Embedding
1128     105
1129     class Embedding(nn.Module):
1130     106
1131         def __init__(self, vocab_size, model_depth):
1132     107
1133         super(Embedding, self).__init__()
1134     108
1135         self.lut = nn.Embedding(vocab_size, model_depth)
1136     109
1137         self.model_depth = model_depth
1138     110
1139         self.positional = PositionalEncoding(model_depth)
1140     111
1141
1142         def forward(self, x):
1143     112
1144         emb = self.lut(x) * math.sqrt(self.model_depth)
1145     113
1146         return self.positional(emb)
1147     114
1148     115
1149     ## (5) Encoder Layer
1150     116
1151     class EncoderLayer(nn.Module):
1152     117
1153         def __init__(self,
1154     118
1155             n_heads,
1156     119
1157             model_depth,
1158     120
1159             ff_depth,
1160     121
1161             dropout=0.0
1162     122
1163             ):
1164     123
1165         super(EncoderLayer, self).__init__()
1166     124
1167         self.self_attn = MultiHeadAttention(embed_dim=model_depth, num_heads=n_heads)
1168     125
1169         self.resnorm1 = ResidualNorm(model_depth, dropout)
1170     126
1171         self.ff = MLP(model_depth, ff_depth, dropout)
1172     127
1173         self.resnorm2 = ResidualNorm(model_depth, dropout)
1174     128
1175
1176         def forward(self, x, mask):
1177     129
1178         x = self.resnorm1(x, lambda arg: self.self_attn(arg, arg, mask))
1179     130
1180         x = self.resnorm2(x, self.ff)
1181     131
1182         return x
1183     132
1184     133
1185     ## (4) Encoder
1186     134
1187     class Encoder(nn.Module):
1188     135
1189         def __init__(self,
1190     136
1191             n_layers,
1192     137
1193             n_heads,
1194     138
1195             model_depth,
1196     139
1197             ff_depth,
1198     140
1199             dropout
1200     141
1201             ):
1202     142
1203         super(Encoder, self).__init__()
1204     143
1205         self.layers = nn.ModuleList([EncoderLayer(n_heads, model_depth, ff_depth, dropout) for
1206     144
1207             i in range(n_layers)])
1208     145
1209         self.lnorm = LayerNorm(model_depth)
1210     146
1211
1212         def forward(self, x, mask):
1213     147
1214         for layer in self.layers:
1215     148
1216             x = layer(x, mask)
1217     149
1218         return self.lnorm(x)
1219     150
1220     151
1221     152
1222     153
1223     154
1224     155
1225     156
1226     157
1227     158
1228     159

```

```

1134 160 ## (3)Generator
1135 161 class Generator(nn.Module):
1136 162     def __init__(self,
1137 163                 model_depth,
1138 164                 vocab_size
1139 165             ):
1140 166         super(Generator, self).__init__()
1141 167         self.ff = nn.Linear(model_depth, vocab_size)
1142 168
1143 169     def forward(self, x):
1144 170         return F.log_softmax(self.ff(x), dim=-1)
1145 171
1146 172 ## (2)CoEvoEmbedder
1147 173 class CoEvoEmbedder(nn.Module):
1148 174     def __init__(self,
1149 175                 vocab_size,
1150 176                 n_layers=2,
1151 177                 n_heads=4,
1152 178                 model_depth=64,
1153 179                 ff_depth=64,
1154 180                 dropout=0.0,
1155 181             ):
1156 182         super(CoEvoFormer, self).__init__()
1157 183
1158 184         self.model_depth = model_depth
1159 185         self.encoder = Encoder(n_layers=n_layers,
1160 186                               n_heads=n_heads,
1161 187                               model_depth=model_depth,
1162 188                               ff_depth=ff_depth,
1163 189                               dropout=dropout,
1164 190                               )
1165 191
1166 192         if vocab_size is not None:
1167 193             if isinstance(vocab_size, int):
1168 194                 self.set_vocab_size(vocab_size)
1169 195
1170 196             else:
1171 197                 self.set_vocab_size(vocab_size[0], vocab_size[1])
1172 198
1173 199     def set_vocab_size(self, src_vocab_size):
1174 200         self.src_embedder = Embedding(src_vocab_size, self.model_depth)
1175 201         self.generator = Generator(self.model_depth, src_vocab_size)
1176 202
1177 203         for p in self.parameters():
1178 204             if p.dim() > 1:
1179 205                 nn.init.xavier_uniform_(p)
1180 206
1181 207     def forward(self, src, src_mask=None):
1182 208         enc_out = self.encoder(self.src_embedder(src), src_mask)
1183 209
1184 210         return enc_out
1185 211
1186 212 ## (1)CoEvoFormer
1187 213 class CoEvoFormer(nn.Module):
1188 214     def __init__(self, model_conf):
1189 215         super(CoEvoFormer, self).__init__()
1190 216         torch.set_default_dtype(torch.float32)
1191 217         self._model_conf = model_conf
1192 218         self._msa_conf = model_conf.msa
1193 219
1194 220         self.msa_encoder = CoEvoEmbedder(
1195 221             vocab_size=self._msa_conf.num_msa_vocab,
1196 222             n_layers=self._msa_conf.msa_layers,
1197 223             n_heads=self._msa_conf.msa_heads,
1198 224             model_depth=self._msa_conf.msa_embed_size,
1199 225             ff_depth=self._msa_conf.msa_hidden_size,
1200 226             dropout=self._model_conf.dropout,
1201 227         )
1202 228
1203 229         self.col_attn = MultiHeadAttention(
1204 230             num_heads=self._msa_conf.msa_heads,
1205 231             embed_dim=self._msa_conf.msa_embed_size,
1206 232         )
1207 233
1208 234         self.row_attn = MultiHeadAttention(
1209 235             num_heads=self._msa_conf.msa_heads,
1210 236             embed_dim=self._msa_conf.msa_embed_size,
1211 237         )
1212 238
1213 239
1214 240

```

```

1188
1189 241 def forward(
1190 242     self,
1191 243     msa_feature,
1192 244     msa_mask=None,
1193 245 ):
1194 246     bs, n_msa, n_token = msa_feature.size()
1195 247     msa_feature = msa_feature.reshape(bs*n_msa, n_token)
1196 248     msa_embed = self.msa_encoder(msa_feature).reshape(bs, n_msa, n_token, -1)
1197 249     msa_embed = msa_embed.transpose(1, 2).reshape(bs*n_token, n_msa, -1)
1198 250
1199 251     if msa_mask is not None:
1200 252         msa_mask = msa_mask.transpose(1, 2).reshape(bs*n_token, n_msa)
1201 253
1202 254     msa_embed = self.col_attn(msa_embed, msa_embed, mask=msa_mask).reshape(bs, n_token,
1203 255     n_msa, -1).transpose(1, 2)
1204 256     msa_embed = msa_embed.reshape(bs*n_msa, n_token, -1)
1205 257
1206 258     if msa_mask is not None:
1207 259         msa_mask = msa_mask.reshape(bs, n_token, n_msa)
1208 260         msa_mask = msa_mask.transpose(1, 2).reshape(bs*n_msa, n_token)
1209 261
1210 262     msa_embed = self.row_attn(msa_embed, msa_embed, mask=msa_mask).reshape(bs, n_msa,
1211 263     n_token, -1)
1212
1213     return msa_embed

```

Listing 1: Pytorch Implementation of coEvoFormer.

## E MOLECULE GNN

### E.1 3D MOLECULE GNN

The 3D molecule GNN plays a crucial role in EnzymeFlow. During the structure-based hierarchical pre-training, it encodes ligand molecule representations, learning the constrained geometry between protein binding pockets and ligand molecules. This pre-training process makes the 3D molecule GNN transferable. When the flow model is fine-tuned, the 3D molecule GNN is also fine-tuned, transferring its prior knowledge about ligand molecules to substrate molecules in enzyme-catalyzed reactions. This allows for substrate-specific encodings while leveraging the knowledge learned from protein-ligand interactions.

Consider a molecule  $l_s$  with  $N_{l_s}$  atoms; this could be a ligand conformation in a protein-ligand pair or a substrate conformation in an enzyme-substrate pair. The molecule  $l_s$  can be viewed as a set of atomic point clouds in 3D Euclidean space, where each atom is characterized by its atomic type. There is a distance relationship between each atom pair in the point cloud, which can be processed as bonding features. In our 3D molecule GNN, we use a radial basis function to process these pairwise atomic distances, a technique commonly employed to ensure equivariance and invariance in model design (Hua et al., 2023; Zhang et al., 2024a;b). The 3D molecule GNN takes a molecule conformation  $l_s$  as input and outputs an embedded molecule representation  $H_{l_s} \in \mathbb{R}^{N_{l_s} \times D_{H_{l_s}}}$ , where  $D_{H_{l_s}}$  denotes the hidden dimension size.

The code for 3D Molecule GNN follows directly:

```

1228
1229 1 import math
1230 2 import numpy as np
1231 3
1232 4 import torch
1233 5 import torch.nn as nn
1234 6 from torch.nn import functional as F
1235 7
1236 8 ## (1) 3D Molecule GNN
1237 9 class MolEmbedder3D(nn.Module):
1238 10     def __init__(self, model_conf):
1239 11         super(MolEmbedder3D, self).__init__()
1240 12         torch.set_default_dtype(torch.float32)
1241 13         self._model_conf = model_conf
1242 14         self._embed_conf = model_conf.embed
1243 15
1244 16         node_embed_dims = self._model_conf.num_atom_type
1245 17         node_embed_size = self._model_conf.node_embed_size
1246 18         self.node_embedder = nn.Sequential(
1247 19             nn.Embedding(node_embed_dims, node_embed_size, padding_idx=0),
1248 20             nn.SiLU(),
1249 21             nn.Linear(node_embed_size, node_embed_size),

```

```

1242         nn.LayerNorm(node_embed_size),
1243     )
1244
1245     self.node_aggregator = nn.Sequential(
1246         nn.Linear(node_embed_size + self._model_conf.edge_embed_size, node_embed_size),
1247         nn.SiLU(),
1248         nn.Linear(node_embed_size, node_embed_size),
1249         nn.SiLU(),
1250         nn.Linear(node_embed_size, node_embed_size),
1251         nn.LayerNorm(node_embed_size),
1252     )
1253
1254     self.dist_min = self._model_conf.ligand_rbf_d_min
1255     self.dist_max = self._model_conf.ligand_rbf_d_max
1256     self.num_rbf_size = self._model_conf.num_rbf_size
1257     self.edge_embed_size = self._model_conf.edge_embed_size
1258
1259     self.edge_embedder = nn.Sequential(
1260         nn.Linear(self.num_rbf_size + node_embed_size + node_embed_size, self.
1261         edge_embed_size),
1262         nn.SiLU(),
1263         nn.Linear(self._model_conf.edge_embed_size, self._model_conf.edge_embed_size),
1264         nn.SiLU(),
1265         nn.Linear(self._model_conf.edge_embed_size, self._model_conf.edge_embed_size),
1266         nn.LayerNorm(self._model_conf.edge_embed_size),
1267     )
1268
1269     mu = torch.linspace(self.dist_min, self.dist_max, self.num_rbf_size)
1270     self.mu = mu.reshape([1, 1, 1, -1])
1271
1272     self.sigma = (self.dist_max - self.dist_min) / self.num_rbf_size
1273
1274     # Distance function -- pair-wise distance computation
1275     def coord2dist(self, coord, edge_mask):
1276         n_batch, n_atom = coord.size(0), coord.size(1)
1277         radial = torch.sum((coord.unsqueeze(1) - coord.unsqueeze(2)) ** 2, dim=-1)
1278         dist = torch.sqrt(
1279             radial + 1e-10
1280         ) * edge_mask
1281
1282         radial = radial * edge_mask
1283         return radial, dist
1284
1285     # RBF function -- distance encoding
1286     def rbf(self, dist):
1287         dist_expand = torch.unsqueeze(dist, -1)
1288         _mu = self.mu.to(dist.device)
1289         rbf = torch.exp(-(((dist_expand - _mu) / self.sigma) ** 2))
1290         return rbf
1291
1292     def forward(
1293         self,
1294         ligand_atom,
1295         ligand_pos,
1296         edge_mask,
1297     ):
1298         num_batch, num_atom = ligand_atom.shape
1299
1300         # Atom Embedding
1301         node_embed = self.node_embedder(ligand_atom)
1302
1303         # Edge Feature Computation
1304         radial, dist = self.coord2dist(
1305             coord=ligand_pos,
1306             edge_mask=edge_mask,
1307         )
1308
1309         edge_embed = self.rbf(dist) * edge_mask[..., None]
1310         src_node_embed = node_embed.unsqueeze(1).repeat(1, num_atom, 1, 1)
1311         tar_node_embed = node_embed.unsqueeze(2).repeat(1, 1, num_atom, 1)
1312         edge_embed = torch.cat([src_node_embed, tar_node_embed, edge_embed], dim=-1)
1313
1314         # Edge Embedding
1315         edge_embed = self.edge_embedder(edge_embed.to(torch.float))
1316
1317         # Message-Passing
1318         src_node_agg = (edge_embed.sum(dim=1) / (edge_mask[..., None].sum(dim=1)+1e-10)) *
1319         ligand_atom.clamp(max=1.)[..., None]
1320         src_node_agg = torch.cat([node_embed, src_node_agg], dim=-1)
1321
1322         # Residue Connection
1323         node_embed = node_embed + self.node_aggregator(src_node_agg)

```

```

1296
1297 101
1298 102
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350

```

```

return node_embed, edge_embed

```

Listing 2: Pytorch Implementation of 3D Molecule GNN.

## E.2 2D MOLECULE GNN

Like the 3D molecule GNN, the 2D molecule GNN is also important in our EnzymeFlow implementation. In an enzyme-catalyzed reaction, the substrate molecule is transformed into a product molecule, with enzyme-substrate interactions driving this chemical transformation. The 2D molecule GNN plays a key role in modeling and encoding this transformation during the catalytic process, making it equally important as our use of co-evolutionary dynamics. While the 3D molecule GNN encodes the substrate, the 2D molecule GNN encodes the product, guiding the design of the enzyme catalytic pocket.

Consider a product molecule  $l_p$  with  $N_{l_p}$  atoms in a catalytic reaction. This molecule can be represented as a graph, where nodes correspond to atoms and edges represent bonds. In our 2D molecule GNN, we use fingerprints with attention mechanisms (Xiong et al., 2019) to facilitate message passing between atoms, enabling effective communication across the molecule. The 2D molecule GNN takes this molecular graph  $l_p$  as input and outputs an embedded molecule representation  $H_{l_p} \in \mathbb{R}^{N_{l_p} \times D_{H_{l_p}}}$ , where  $D_{H_{l_p}}$  denotes the hidden dimension size.

The code for 2D Molecule GNN follows directly:

```

1 import torch
2 import torch.nn as nn
3 from torch_geometric.nn.models import AttentiveFP
4
5 ## (1) 2D Molecule GNN
6 class MolEmbedder2D(nn.Module):
7     def __init__(self, model_conf):
8         super(MolEmbedder2D, self).__init__()
9         torch.set_default_dtype(torch.float32)
10        self._model_conf = model_conf
11
12        self.node_embed_dims = self._model_conf.mpnn.mpnn_node_embed_size
13        self.edge_embed_dims = self._model_conf.mpnn.mpnn_edge_embed_size
14
15        self.node_embedder = nn.Sequential(
16            nn.Embedding(self._model_conf.num_atom_type, self.node_embed_dims),
17            nn.SiLU(),
18            nn.Linear(self.node_embed_dims, self.node_embed_dims),
19            nn.LayerNorm(self.node_embed_dims),
20        )
21
22        self.edge_embedder = nn.Sequential(
23            nn.Embedding(self._model_conf.mpnn.num_edge_type, self.edge_embed_dims),
24            nn.SiLU(),
25            nn.Linear(self.edge_embed_dims, self.edge_embed_dims),
26            nn.LayerNorm(self.edge_embed_dims),
27        )
28
29        # Message Passing with Attention and Fingerprint
30        self.mpnn = AttentiveFP(
31            in_channels=self.node_embed_dims,
32            hidden_channels=self.node_embed_dims,
33            out_channels=self.node_embed_dims,
34            edge_dim=self.edge_embed_dims,
35            num_layers=self._model_conf.mpnn.mpnn_layers,
36            num_timesteps=self._model_conf.mpnn.n_timesteps,
37            dropout=self._model_conf.mpnn.dropout,
38        )
39
40        # Dense Edge Matrix to Sparse Edge Matrix
41        def dense_to_sparse(
42            self,
43            mol_atom,
44            mol_edge,
45            mol_edge_feat,
46            mol_atom_mask,
47            mol_edge_mask,
48        ):
49            mol_atom_list = mol_atom[mol_atom_mask]
50            mol_edge_feat_list = mol_edge_feat[mol_edge_mask]

```

```

1350 51
1351 52     if mol_edge.size(dim=1) == 2:
1352 53         mol_edge = mol_edge.transpose(1,2)
1353 54     mol_edge_list = [edge[mask] for edge, mask in zip(mol_edge, mol_edge_mask)]
1354 55
1355 56     n_nodes = mol_atom_mask.sum(dim=1, keepdim=True)
1356 57     cum_n_nodes = torch.cumsum(n_nodes, dim=0)
1357 58     new_mol_edge_list = [mol_edge_list[0]]
1358 59     for edge, size in zip(mol_edge_list[1:], cum_n_nodes[:-1]):
1359 60         new_mol_edge = edge + size
1360 61         new_mol_edge_list.append(new_mol_edge)
1361 62
1362 63     new_mol_edge_list = torch.cat(new_mol_edge_list, dim=0)
1363 64
1364 65     if new_mol_edge_list.size(dim=1) == 2:
1365 66         new_mol_edge_list = new_mol_edge_list.transpose(1,0)
1366 67
1367 68     idx = 0
1368 69     batch_mask = []
1369 70     for size in n_nodes:
1370 71         batch_mask.append(torch.zeros(size, dtype=torch.long) + idx)
1371 72         idx += 1
1372 73     batch_mask = torch.cat(batch_mask).to(mol_atom.device)
1373 74
1374 75     return mol_atom_list, new_mol_edge_list, mol_edge_feat_list, batch_mask
1375 76
1376 77 def forward(
1377 78     self,
1378 79     mol_atom,
1379 80     mol_edge,
1380 81     mol_edge_feat,
1381 82     mol_atom_mask,
1382 83     mol_edge_mask,
1383 84 ):
1384 85     n_batch = mol_atom.size(0)
1385 86
1386 87     mol_atom_mask = mol_atom_mask.bool()
1387 88     mol_edge_mask = mol_edge_mask.bool()
1388 89     mol_atom, mol_edge, mol_edge_feat, batch_mask = self.dense_to_sparse(mol_atom,
1389 90     mol_edge, mol_edge_feat, mol_atom_mask, mol_edge_mask)
1390 91     assert mol_edge.size(1) == mol_edge_feat.size(0)
1391 92
1392 93     # Atom Embedding
1393 94     mol_atom = self.node_embedder(mol_atom)
1394 95
1395 96     # Edge Embedding
1396 97     mol_edge_feat = self.edge_embedder(mol_edge_feat)
1397 98
1398 99     # Message-Passing
1399 100     mol_rep = self.mpnn(mol_atom, mol_edge, mol_edge_feat, batch_mask)
1400 101
1401 102     return mol_rep

```

Listing 3: Pytorch Implementation of 2D Molecule GNN.

## F VECTOR FIELD COMPUTATION AND SAMPLING

Here, we describe how to compute vectors fields and perform sampling for catalytic pocket residues frames, EC-class, as well as the enzyme-reaction co-evolution.

### F.1 BACKGROUND

**Catalytic Pocket Frame.** We refer to the protein structure as the backbone atomic coordinates of each residue. A pocket of length  $N_r$  can be parameterized into SE(3) residue frames  $\{(x^i, r^i, c^i)\}_{i=1}^{N_r}$ , where  $x^i \in \mathbb{R}^3$  represents the position (translation) of the  $C_\alpha$  atom of the  $i$ -th residue,  $r^i \in \text{SO}(3)$  is a rotation matrix defining the local frame relative to a global reference frame, and  $c^i \in \{1, \dots, 20\} \cup \{\text{X}\}$  denotes the amino acid type, with additional  $\text{X}$  indicating a *masking state* of the amino acid type. We refer to the residue block as  $T^i = (x^i, r^i, c^i)$ , and the entire pocket is described by a set of residues  $\mathbf{T} = \{T^i\}_{i=1}^{N_r}$ . Additionally, we denote the graph representations of substrate and product molecules in the catalytic reaction as  $l_s$  and  $l_p$ , respectively. An enzyme-reaction pair can therefore be described as  $(\mathbf{T}, l_s, l_p)$ . For simplicity, we omit  $i$ .

**EC-Class.** An EC-class is denoted as  $y_{\text{ec}} \in \{1, \dots, 7\} \cup \{\text{X}\}$ , with  $\text{X}$  indicating the *masking state*.



**Co-evolution.** The co-evolution of an enzyme-reaction pair is represented by a matrix  $U \in \mathbb{R}^{N_{\text{MSA}} \times N_{\text{token}}}$ , which combines the MSA results of enzyme sequences and reaction SMILES, where  $N_{\text{MSA}}$  denotes the number of MSA sequences and  $N_{\text{token}}$  denotes the length of the MSA alignment preserved. And each element  $u^{mn} \in \{1, \dots, 64\} \cup \{\times\}$  in  $U$  denotes a tokenized character from our co-evolution vocabulary, with additional  $\times$  indicating the *masking state*.

**Vector Field.** flow matching describes a process where a flow transforms a simple distribution  $p_0$  into the target data distribution  $p_1$  (Lipman et al., 2022). The goal in flow matching is to train a neural network  $v_\theta(\epsilon_t, t)$  that approximates the vector field  $u_t(\epsilon)$ , which measures the transformation of the distribution  $p_t(\epsilon_t)$  as it evolves toward  $p_1(\epsilon_1)$  over time  $t \in [0, 1]$ . The process is optimized using a regression loss defined as  $\mathcal{L}_{\text{FM}} = \mathbb{E}_{t \sim \mathcal{U}[0,1], p_t(\epsilon_t)} \|v_\theta(\epsilon_t, t) - u_t(\epsilon)\|^2$ . However, directly computing  $u_t(\epsilon)$  is often intractable in practice. Instead, a conditional vector field  $u_t(\epsilon|\epsilon_1)$  is defined, and the conditional flow matching objective is computed as  $\mathcal{L}_{\text{CFM}} = \mathbb{E}_{t \sim \mathcal{U}[0,1], p_t(\epsilon_t)} \|v_\theta(\epsilon_t, t) - u_t(\epsilon|\epsilon_1)\|^2$ . Notably,  $\nabla_\theta \mathcal{L}_{\text{FM}} = \nabla_\theta \mathcal{L}_{\text{CFM}}$ .

During inference or sampling, an ODEsolver, e.g., Euler method, is typically used to solve the ODE governing the flow, expressed as  $\epsilon_1 = \text{ODEsolver}(\epsilon_0, v_\theta, 0, 1)$ , where  $\epsilon_0$  is the initial data and  $\epsilon_1$  is the generated data. In actual training, rather than directly predicting the vector fields, it is more common to use the neural network to predict the final state at  $t = 1$ , then interpolates to calculate the vector fields. This approach has been shown to be more efficient and effective for network optimization (Yim et al., 2023a; Bose et al., 2023; Campbell et al., 2024).

## F.2 CONTINUOUS VARIABLE TRAJECTORY

Given the predictions for translation  $\hat{x}_1$  and rotation  $\hat{r}_1$  at  $t = 1$ , we interpolate and their corresponding vector fields are computed as follows:

$$v_\theta(x_t, t) = \frac{\hat{x}_1 - x_t}{1 - t}, \quad v_\theta(r_t, t) = \frac{\log_{r_t} \hat{r}_1}{1 - t}. \quad (9)$$

The sampling or trajectory can then be computed using Euler steps with a step size  $\Delta t$ , as follows:

$$x_{t+\Delta t} = x_t + v_\theta(x_t, t) \cdot \Delta t, \quad r_{t+\Delta t} = r_t + v_\theta(r_t, t) \cdot \Delta t, \quad (10)$$

where the prior of  $x_0, r_0$  are chosen as the uniform distribution on  $\mathbb{R}^3$  and  $\text{SO}(3)$ , respectively.

## F.3 DISCRETE VARIABLE TRAJECTORY

For the discrete variables, including amino acid types, EC-class, and co-evolution, we follow Campbell et al. (2024) to use continuous time Markov chains (CTMC).

**Continuous Time Markov Chain.** A sequence trajectory  $\epsilon_t$  over time  $t \in [0, 1]$  that follows a CTMC alternates between resting in its current state and periodically jumping to another randomly chosen state. The frequency and destination of the jumps are determined by the rate matrix  $R_t \in \mathbb{R}^{N \times N}$  with the constraint its off-diagonal elements are non-negative. The probability of  $\epsilon_t$  jumping to a different state  $s$  follows  $R_t(\epsilon_t, s)dt$  for the next infinitesimal time step  $dt$ . We can express the transition probability as

$$p_{t+dt}(s|\epsilon_t) = \delta\{\epsilon_t, s\} + R_t(\epsilon_t, s)dt, \quad (11)$$

where  $\delta(a, b)$  is the Kronecker delta, equal to 1 if  $a = b$  and 0 if  $a \neq b$ , and  $R_t(\epsilon_t, \epsilon_t) = -\sum_{\gamma \neq \epsilon} R_t(\epsilon_t, \gamma)$  (Campbell et al., 2024). Therefore,  $p_{t+dt}$  is a Categorical distribution with probabilities  $\delta(\epsilon_t, \cdot) + R_t(\epsilon_t, \cdot)dt$  with notation  $s \sim \text{Cat}(\delta(\epsilon_t, s) + R_t(\epsilon_t, s)dt)$ .

For finite time intervals  $\Delta t$ , a sequence trajectory can be simulated with Euler steps following:

$$\epsilon_{t+\Delta t} \sim \text{Cat}(\delta(\epsilon_t, \epsilon_{t+\Delta t}) + R_t(\epsilon_t, \epsilon_{t+\Delta t})\Delta t). \quad (12)$$

The rate matrix  $R_t$  along with an initial distribution  $p_0$  define CTMC. Furthermore, the probability flow  $p_t$  is the marginal distribution of  $\epsilon_t$  at every time  $t$ , and we say the rate matrix  $R_t$  generates  $p_t$  if  $\partial_t p_t = R_t^T p_t, \forall t \in [0, 1]$ .

In the actual training, Campbell et al. (2024) show that we can train a neural network to approximate the true denoising distribution using the standard cross-entropy:

$$\mathcal{L}_{\text{CE}} = \mathbb{E}_{t \sim \mathcal{U}[0,1], p_t(\epsilon_t)} [\log p_\theta(\epsilon_1|\epsilon_t)], \quad (13)$$

which leads to our neural network objectives for amino acid types, EC-class, and co-evolution as:

$$\begin{aligned}\mathcal{L}_{\text{aa}} &= \mathbb{E}_{t \sim \mathcal{U}[0,1], p_t(c_t)} [\log p_\theta(c_1|c_t)], \mathcal{L}_{\text{ec}} = \mathbb{E}_{t \sim \mathcal{U}[0,1], p_t(y_{\text{ec}_t})} [\log p_\theta(y_{\text{ec}_1}|y_{\text{ec}_t})], \\ \mathcal{L}_{\text{coevo}} &= \mathbb{E}_{t \sim \mathcal{U}[0,1], p_t(u_t)} [\log p_\theta(u_1|u_t)].\end{aligned}\quad (14)$$

**Rate Matrix for Inference.** The conditional rate matrix  $R_t(\epsilon_t, s|s_1)$  generates the conditional flow  $p_t(\epsilon_t|\epsilon_1)$ . And  $R_t(\epsilon_t, s) = \mathbb{E}_{p_1(\epsilon_1|\epsilon_t)} [R_t(\epsilon_t, s|\epsilon_1)]$ , for which the expectation is taken over  $p_1(\epsilon_1|\epsilon_t) = \frac{p_t(\epsilon_t|\epsilon_1)p_1(\epsilon_1)}{p_t(\epsilon_t)}$ . With the conditional rate matrix, the sampling can be performed:

$$\begin{aligned}R_t(\epsilon_t, \cdot) &\leftarrow \mathbb{E}_{p_1(\epsilon_1|\epsilon_t)} [R_t(\epsilon_t, \cdot|\epsilon_1)], \\ \epsilon_{t+\Delta t} &\sim \text{Cat}(\delta(\epsilon_t, \epsilon_{t+\Delta t}) + R_t(\epsilon_t, \epsilon_{t+\Delta t})\Delta t).\end{aligned}\quad (15)$$

The rate matrix generates the probability flow for discrete variables.

Campbell et al. (2024) define the conditional rate matrix starting with

$$R_t(\epsilon_t, s|\epsilon_1) = \frac{\text{ReLU}(\partial_t p_t(s|\epsilon_1) - \partial_t p_t(\epsilon_t|\epsilon_1))}{N \cdot p_t(\epsilon_t|\epsilon_1)}.\quad (16)$$

In practice, the closed-form of conditional rate matrix with *masking state*  $\times$  is defined as:

$$R_t(\epsilon_t, s|\epsilon_1) = \frac{\delta(\epsilon_1, s)}{1-t} \delta(\epsilon_t, \times).\quad (17)$$

With the definition of the conditional rate matrix  $R_t(\epsilon_t, s|\epsilon_1)$ , we can perform sampling and inference for amino acid types, EC-class, and co-evolution following:

$$\begin{aligned}c_{t+\Delta t} &\sim \text{Cat}(\delta(c_t, c_{t+\Delta t}) + R_t(c_t, c_{t+\Delta t}|v_\theta(c_t, t)) \cdot \Delta t), \\ y_{\text{ec}_{t+\Delta t}} &\sim \text{Cat}(\delta(y_{\text{ec}_t}, y_{\text{ec}_{t+\Delta t}}) + R_t(y_{\text{ec}_t}, y_{\text{ec}_{t+\Delta t}}|v_\theta(y_{\text{ec}_t}, t)) \cdot \Delta t), \\ u_{t+\Delta t} &\sim \text{Cat}(\delta(u_t, u_{t+\Delta t}) + R_t(u_t, u_{t+\Delta t}|v_\theta(u_t, t)) \cdot \Delta t).\end{aligned}\quad (18)$$

## G ENZYMEFLOW SE(3)-EQUIVARIANCE

**Theorem.** Let  $\phi$  denote an SE(3) transformation. The catalytic pocket design in EnzymeFlow, represented as  $p_\theta(\mathbf{T}|l_s)$ , is SE(3)-equivariant, meaning that  $p_\theta(\phi(\mathbf{T})|\phi(l_s)) = p_\theta(\mathbf{T}|l_s)$ , where  $\mathbf{T}$  represents the generated catalytic pocket, and  $l_s$  denotes the substrate conformation.

*Proof.* Given an SE(3)-invariant prior, such that  $p(\mathbf{T}_0, l_s) = p(\phi(\mathbf{T}_0), \phi(l_s))$ , and an SE(3)-equivariant transition state for each time step  $t$  via an SE(3)-equivariant neural network, such that  $p_\theta(\mathbf{T}_{t+\Delta t}, l_s) = p_\theta(\phi(\mathbf{T}_{t+\Delta t}), \phi(l_s))$ , it follows that for the total time steps  $T$ , we have:

$$\begin{aligned}p_\theta(\phi(\mathbf{T}_1)|\phi(l_s)) &= \int p_\theta(\phi(\mathbf{T}_0, l_s)) \prod_{n=0}^{T-1} p_\theta(\phi(\mathbf{T}_{n\Delta t+\Delta t}, l_s)|\phi(\mathbf{T}_{n\Delta t}, l_s)) \\ &= \int p_\theta(\mathbf{T}_0, l_s) \prod_{n=0}^{T-1} p_\theta(\phi(\mathbf{T}_{n\Delta t+\Delta t}, l_s)|\phi(\mathbf{T}_{n\Delta t}, l_s)) \\ &= \int p_\theta(\mathbf{T}_0, l_s) \prod_{n=0}^{T-1} p_\theta(\mathbf{T}_{n\Delta t+\Delta t}, l_s|\mathbf{T}_{n\Delta t}, l_s) \\ &= p_\theta(\mathbf{T}_1|l_s).\end{aligned}\quad (19)$$

□

## H ENZYMEFLOW DATASET STATISTICS

**Data Source.** We construct a curated and validated dataset of enzyme-reaction pairs by collecting data from the Rhea (Bansal et al., 2022), MetaCyc (Caspi et al., 2020), and Brenda (Schomburg et al., 2002) databases. For enzymes in these databases, we exclude entries missing UniProt IDs or protein sequences. For reactions, we apply the following procedures: (1) remove cofactors, small

Data	Reaction	Enzyme	Substrate		Product		Enzyme Commission Class						
	#reaction	#enzyme	#substrate	#avg atom	#product	#avg atom	EC1	EC2	EC3	EC4	EC5	EC6	EC7
Rawdata	232520	97912	7259	30.81	7664	30.34	44881 (19.30%)	75944 (32.66%)	37728 (16.23%)	47242 (20.32%)	8315 (3.58%)	18281 (7.86%)	129 (0.06%)
40% Homo	19379	6922	4798	31.06	4897	30.24	4754 (24.53%)	5857 (30.22%)	4839 (24.97%)	1764 (9.10%)	759 (3.92%)	1379 (7.12%)	27 (0.14%)
50% Homo	34750	13442	5675	31.45	5871	30.75	8184 (23.55%)	11174 (32.16%)	8050 (23.17%)	3203 (9.22%)	1357 (3.91%)	2752 (7.92%)	30 (0.09%)
60% Homo	53483	22350	6112	30.95	6331	30.34	11674 (21.83%)	18419 (34.44%)	11394 (21.30%)	5555 (10.39%)	2194 (4.10%)	4200 (7.85%)	47 (0.09%)
80% Homo	100925	43458	6619	30.46	6943	29.95	21308 (21.11%)	34344 (34.03%)	18925 (18.75%)	14010 (13.88%)	3901 (3.87%)	8371 (8.29%)	66 (0.07%)
90% Homo	132047	55697	6928	30.32	7298	29.81	28833 (21.84%)	43287 (32.78%)	23989 (18.17%)	20070 (15.20%)	5015 (3.80%)	10766 (8.15%)	87 (0.07%)

Table 3: Enzyme Catalytic Pocket Dataset Statistics.

Data	Reaction	Enzyme	Substrate		Product		Enzyme Commission						
	#reaction	#enzyme	#substrate	#avg atom	#product	#avg atom	EC1	EC2	EC3	EC4	EC5	EC6	EC7
Rawdata	232520	97912	7259	30.81	7664	30.34	44881 (19.30)	75944 (32.66)	37728 (16.23)	47242 (20.32)	8315 (3.58)	18281 (7.86)	129 (0.06)
Train Data	53483	22350	6112	30.95	6331	30.34	11674 (21.83)	18419 (34.44)	11394 (21.30)	5555 (10.39)	2194 (4.10)	4200 (7.85)	47 (0.09)
Eval Data	100	100	100	30.7	94	28.84	17 (17.00)	17 (17.00)	17 (17.00)	17 (17.00)	16 (16.00)	16 (16.00)	0 (0.00)

Table 4: EnzymeFlow Evaluation Data Statistics.

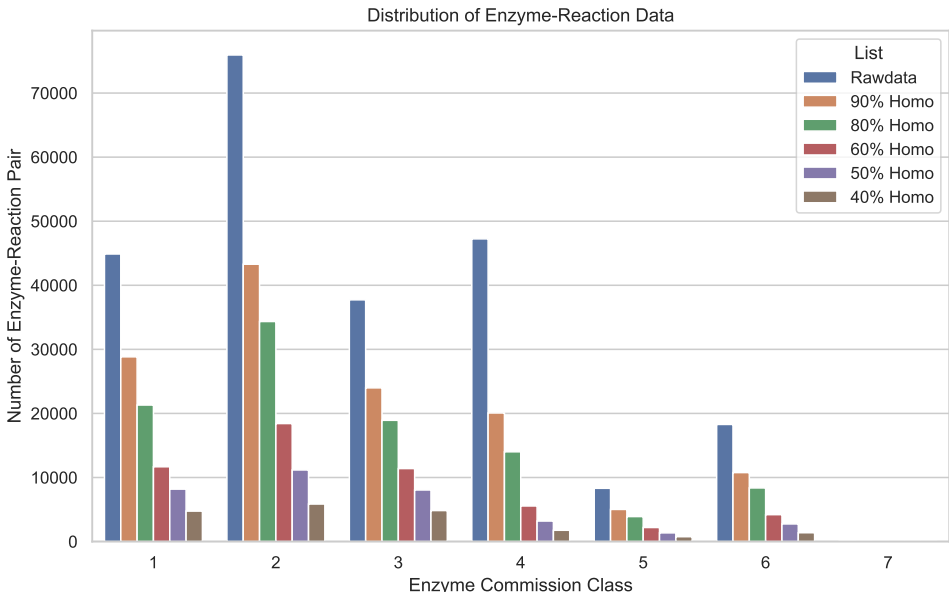


Figure 10: Distribution of enzyme-reaction pairs over EC-class.

ion groups, and molecules that appear in both substrates and products within a single reaction; (2) exclude reactions with more than five substrates or products; and (3) apply OpenBabel (O’Boyle et al., 2011) to standardize molecular SMILES. Ultimately, we obtain a total of 328,192 enzyme-reaction pairs, comprising 145,782 unique enzymes and 17,868 unique reactions.

**Debiasing.** To ensure the quality of catalytic pocket data, we exclude pockets with fewer than 32 residues, resulting in 232,520 enzyme-reaction pairs. Additionally, enzymes and their catalytic pockets can exhibit significant sequence similarity. When enzymes that are highly similar in sequence appear too frequently in the dataset, they tend to belong to the same cluster or homologous group, which can introduce substantial biases during model training. To mitigate this issue and ensure a more balanced dataset, it is important to reduce the number of homologous enzymes by clustering and selectively removing enzymes from the same clusters. This helps to debias the data and improve the model’s generalizability. We perform sequence alignment to cluster enzymes and identify homologous ones (Steinegger & Söding, 2017). We then revise the dataset into five major categories based on enzyme sequence similarity, resulting in: (1) 19,379 pairs with at most 40% homology, (2) 34,750 pairs with at most 50% homology, (3) 53,483 pairs with at most 60% homology, (4) 100,925 pairs with at most 80% homology, and (5) 132,047 pairs with at most 90% homology. We provide data statistics, including the EC-class distribution, in Table 3, and visualize the distribution in Figure 10.

From the data, we observe that EC1, EC2, EC3, and EC4 contribute the most enzyme-reaction pairs to our dataset. Specifically, EC1 refers to oxidation/reduction reactions, involving the transfer of hydrogen, oxygen atoms, or electrons from one substance to another. EC2 involves the transfer of a functional group (such as methyl, acyl, amino, or phosphate) from one substance to another. EC3

is associated with the formation of two products from a substrate through hydrolysis, while EC4 involves the non-hydrolytic addition or removal of groups from substrates, potentially cleaving C-C, C-N, C-O, or C-S bonds. Our dataset distribution closely follows the natural enzyme-reaction enzyme commission class distribution, with Transferases (EC2) being the most dominant.

## I WORK IN PROGRESS: ENZYME POCKET-REACTION RECRUITMENT WITH ENZYME CLIP MODEL

In addition to evaluating the catalytic pockets generated from the functional and structural perspectives, we may raise a key question of how we *quantitatively* determine whether the generated pockets can catalyze a specific reaction. To answer it, we are working to train an enzyme-reaction CLIP model using enzyme-reaction pairs (with pocket-specific information) from the 60%-clustered data, excluding the 100 evaluation samples from training. All enzymes not annotated to catalyze a specific reaction are treated as negative samples, following the approach in Yang et al. (2024); Mikhael et al. (2024). For the 100 generated catalytic pockets of each reaction, we select the Top-1 pocket with the highest TM-score for evaluation using the enzyme CLIP model.

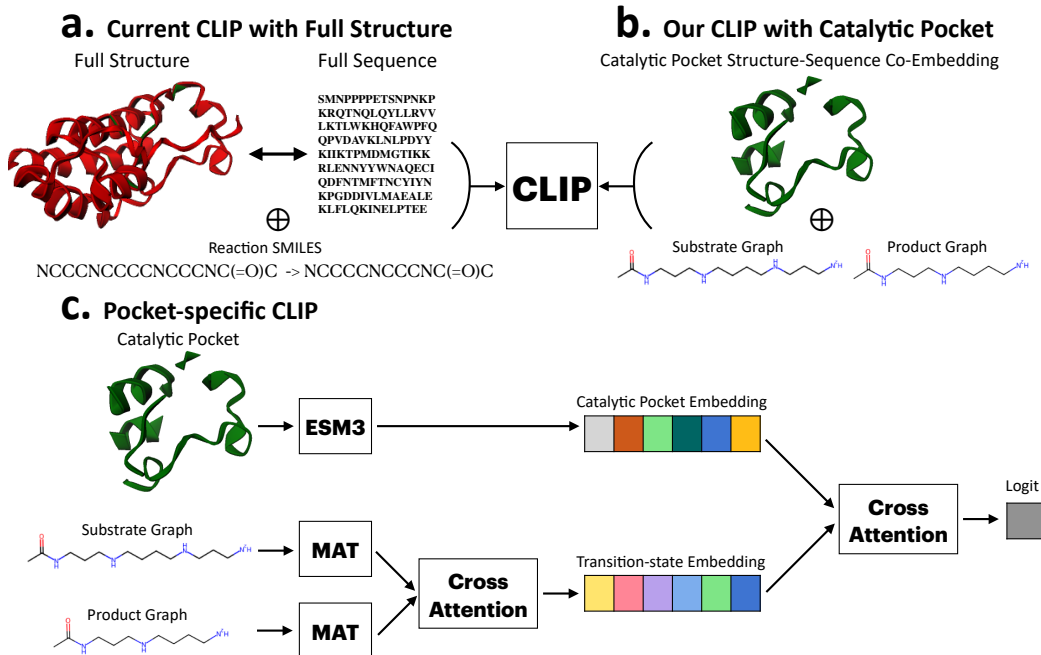


Figure 11: Enzyme-Reaction CLIP model comparison. (a) Existing CLIP models use the full enzyme structure or full enzyme sequence, paired with reaction SMILES as input. (b) Our pocket-specific CLIP model focuses on catalytic pockets, using both their structures and sequences paired with molecular graphs of reactions. The pocket-specific CLIP approach learns from enzyme active sites, which exhibit higher functional concentration. (c) Overview of Pocket-specific CLIP model.

**Pocket-specific CLIP.** Unlike existing methods that typically train on full enzyme structures or sequences (Yu et al., 2023; Mikhael et al., 2024), our pocket-specific CLIP approach is designed to focus specifically on catalytic pockets, including both their structures and sequences, paired with molecular graphs of catalytic reactions (illustrated in Fig. 11). As shown in Fig. 4(b), catalytic pockets are usually the regions that exhibit high functional concentration, while the remaining parts tend to be less functionally important. Therefore, focusing on catalytic pockets is more applicable and effective for enzyme CLIP models. The advantage of the pocket-specific CLIP is that it learns from active sites that are highly meaningful both structurally and sequentially.

We illustrate our pocket-specific enzyme CLIP approach in Fig. 11. In our pocket-specific CLIP model, we encode the pocket structure and sequence using ESM3 (Hayes et al., 2024), and the substrate and product molecular graphs using MAT (Maziarka et al., 2020). Cross-attention is applied to compute the transition state of the reaction, capturing the transformation of the substrate into the

product, as proposed in Hua et al. (2024b). This is followed by another cross-attention mechanism to learn the interactions between the catalytic pocket and the reaction. The model is trained by enforcing high logits for positive enzyme-reaction pairs and low logits for negative enzyme-reaction pairs.

**Metrics.** To evaluate the catalytic ability of the designed pockets for a given reaction, we employ retrieval-based ranking as proposed in Hua et al. (2024b). This ranking-based evaluation ensures fairness and minimizes biases. The metrics include:  $\text{Top-k Acc}$ , which quantifies the proportion of instances in which the catalytic pocket is ranked within the CLIP’s top-k predictions;  $\text{Mean Rank}$ , which calculates the average position of the pocket in the retrieval list;  $\text{Mean Reciprocal Rank (MRR)}$ , which measures how quickly the pocket is retrieved by averaging the reciprocal ranks of the first correct pocket across all reactions. These metrics help assess whether a catalytic pocket designed for a specific reaction ranks highly in the recruitment list, indicating its potential to catalyze the reaction.

### I.1 INPAINTING CATALYTIC POCKET WITH ESM3 FOR FULL ENZYME RECRUITMENT

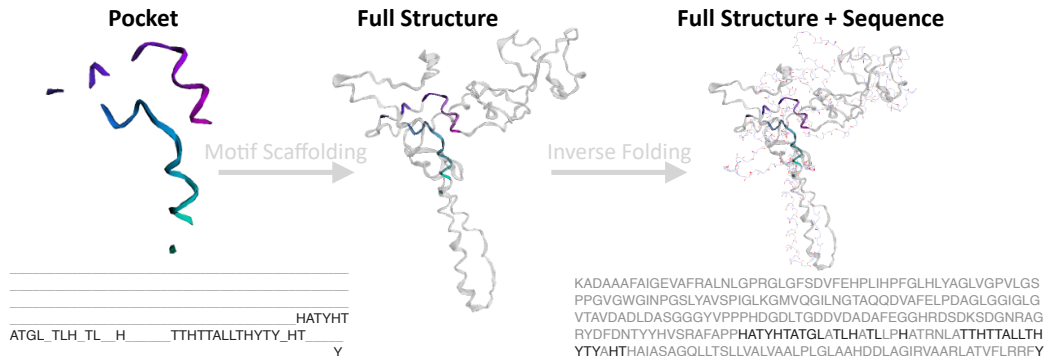


Figure 12: Inpainting catalytic pocket using ESM3.

ESM3 (Hayes et al., 2024) can inpaint missing structures and sequences with functional motifs. In this context, we train a separate full enzyme CLIP model for the enzyme recruitment task. This model is trained using the same 60%-clustered data but incorporates full enzyme structures and sequences. For generated catalytic pockets and those in the evaluation set, we use ESM3 to inpaint them, completing the structures and sequences predicted by ESM3. These ESM3-inpainted enzymes are then evaluated using the full enzyme CLIP model, applying the same retrieval-based ranking metrics as before. We illustrate the catalytic pocket inpainting pipeline in Fig. 12.

In conclusion, we are developing a pocket-specific enzyme CLIP model for pocket-based enzyme recruitment tasks and a full-enzyme CLIP model using ESM3 for inpainting and pocket scaffolding in full enzyme recruitment tasks. However, we recognize that directly using ESM3 for catalytic pocket inpainting lacks domain-specific knowledge, making fine-tuning necessary. To address this, we are working on a fine-tuning open-source large biological model, *e.g.*, Genie2 (Lin et al., 2024), on our EnzymeFill dataset. Genie2, pre-trained on FoldSeek-clustered AlphaFold- and Protein-DataBank proteins for *de novo* protein design and (multi-)motif scaffolding, aligns well with our catalytic pocket scaffolding task. Fine-tuning Genie2 on EnzymeFill will enhance its performance in catalytic pocket inpainting. The development of EnzymeFlow, aimed at achieving an AI-driven automated enzyme design platform, is discussed in App. A.

## J RFDIFFAA-DESIGNED POCKETS

In Fig. 13, we visualize some RFDiffAA-designed pockets (Krishna et al., 2024) (before superimposition) with LigandMPNN-predicted sequences (Dauparas et al., 2023) and CLEAN-predicted EC-class (Yu et al., 2023). In Fig. 14, we visualize some EnzymeFlow-designed pockets (before superimposition) with predicted sequences and EC-class. In real design, we do not have the structure or sequence information of ground-truth catalytic pockets, so we choose to visualize examples before the superimposition of the ground-truth ones and generated ones.




**RFDiffAA-designed Samples (Non-Superimposed Samples 1-12)**

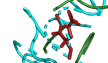
UniProt	Sequence	EC-Class	Substrate	Product	EC-Class
Q9WYS1	ADPFAAAAAAAAAAAAAAAAAAAAAA	4	OCC1O(O)C@@H]([C@H])([C@H]O)O(C)O	OCC1O(O)C@@H]([C@H])([C@H]O)O(C)O	5
	AAAAAALAAVDPFAAAAAAAAAAAAA	3			
	GAFAAAAAAAAAAAAAAAAAAAAAAA	2			
	SAFAAAAAAAAAAAAAAAAAAAAAAA	2			
	GGGAIAAAAAAAAAAAAAAAAAAAAA	3			
	SLDLAALAAFDRAIRDRDARAAELRUA	1			
	SVPSVPYVDLDAIGTVDCTPTTDCDVNFK	5			
	SAFAIAAAAAAAAAAAAAAAAAAAAA	4			
	SAFAAAAVNRLLASGGDRKEETARUAAAR	1			
	MTLEIAEAVKAAVDPNKAPEALELXNG	1			
	SAFAIAAAAAAAAAAAAAAAAAAAAA	2			
	ALLAVIAAVLFDVLVLGALAAALLFG	1			

Figure 13: Some RFDiffAA-designed pockets for Uniprot Q9WYS1 (Samples 1-12).


**EnzymeFlow-designed Samples (Non-Superimposed Samples 1-12)**




**Catalytic Pocket (Q9WYS1)**




Sequence: ATGGQADRYTRDTHLHDIYEYTTTLHGHFH  
EC-Class: 4



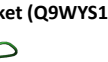
Sequence: CGATGTSLTGGATLCAALGATGGTHHNTAT  
EC-Class: 2




Sequence: TGTGTGTHNSATTTTAGTGTAAFGTGAACQ  
EC-Class: 2




Sequence: LCGPTTGLLHHHTATZALNALQDLTHL  
EC-Class: 3




UniProt: Q9WYS1  
Sequence: GTGQDSVREADHTDAWKGSLHSGSDVKTAGT  
Substrate: OCC1(O)(O)[C@@H]([C@H])([C@H](O)(O)[C@H](O)C)C(=O)O  
Product: OCC1(O)(O)[C@@H]([C@H]([C@H](O)(O)C)C(=O)O)C(=O)O  
EC-Class: 5




Sequence: ATLAAGHDTQLATLHATAHHHQQLTGLQ  
EC-Class: 1




Sequence: GTATAAHNYGTGTATNTATAGGSGHAL  
EC-Class: 5



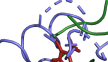
Sequence: TTGGGAATCCCTGTATGTGTGGATTGAT  
EC-Class: 4




Sequence: QHTLAGSQLFHHHRHHTAHNALLHQDHT  
EC-Class: 6




Sequence: ATHTYHLIHTLAAHTLALNLTWHST  
EC-Class: 3



Sequence: TTAGGGAATTTTGTGAQQQYTHNTTAAAT  
EC-Class: 5



Sequence: HLHTLAHYLLHAAHLHAHHTATLAHLH  
EC-Class: 2



Sequence: TSGTGTGATTTATGTGTGGGHTGATGTTT  
EC-Class: 5

Figure 14: Some EnzymeFlow-designed pockets for Uniprot Q9WYS1 (Samples 1-12).

## K ENZYMEFLOW NEURAL NETWORK IMPLEMENTATION

The equivariant neural network is based on the Invariant Point Attention (IPA) implemented in AlphaFold2 (Jumper et al., 2021). In the following, we detail how enzyme catalytic pockets, substrate molecules, product molecules, EC-class, and co-evolution interact within our network.

The code for EnzymeFlow main network follows directly:

```
1 import functools as fn
2 import math
3
4 import torch
5 import torch.nn as nn
```

```

1728 6 from torch.nn import functional as F
1729 7
1730 8 from ofold.utils.rigid_utils import Rigid
1731 9
1732 10 from model import ipa_pytorch
1733 11 from flowmatch.data import all_atom
1734 12 from flowmatch.data import utils as du
1735 13
1736 14 ## EnzymeFlow Main Network
1737 15
1738 16 ## (8) Distogram
1739 17 def calc_distogram(pos, min_bin, max_bin, num_bins):
1740 18     dists_2d = torch.linalg.norm(pos[:, :, None, :] - pos[:, None, :, :], axis=-1)[
1741 19         ..., None
1742 20     ]
1743 21     lower = torch.linspace(min_bin, max_bin, num_bins, device=pos.device)
1744 22     upper = torch.cat([lower[1:], lower.new_tensor([1e8])], dim=-1)
1745 23     dgram = ((dists_2d > lower) * (dists_2d < upper)).type(pos.dtype)
1746 24     return dgram
1747 25
1748 26
1749 27 ## (7) Index Embedding
1750 28 def get_index_embedding(indices, embed_size, max_len=2056):
1751 29     K = torch.arange(embed_size // 2, device=indices.device)
1752 30     pos_embedding_sin = torch.sin(
1753 31         indices[..., None] * math.pi / (max_len ** (2 * K[None] / embed_size))
1754 32     ).to(indices.device)
1755 33     pos_embedding_cos = torch.cos(
1756 34         indices[..., None] * math.pi / (max_len ** (2 * K[None] / embed_size))
1757 35     ).to(indices.device)
1758 36     pos_embedding = torch.cat([pos_embedding_sin, pos_embedding_cos], axis=-1)
1759 37     return pos_embedding
1760 38
1761 39
1762 40 ## (6) Time Embedding
1763 41 def get_timestep_embedding(timesteps, embedding_dim, max_positions=10000):
1764 42     assert len(timesteps.shape) == 1
1765 43     timesteps = timesteps * max_positions
1766 44     half_dim = embedding_dim // 2
1767 45     emb = math.log(max_positions) / (half_dim - 1)
1768 46     emb = torch.exp(
1769 47         torch.arange(half_dim, dtype=torch.float32, device=timesteps.device) * -emb
1770 48     )
1771 49     emb = timesteps.float()[:, None] * emb[None, :]
1772 50     emb = torch.cat([torch.sin(emb), torch.cos(emb)], dim=1)
1773 51     if embedding_dim % 2 == 1: # zero pad
1774 52         emb = F.pad(emb, (0, 1), mode="constant")
1775 53     assert emb.shape == (timesteps.shape[0], embedding_dim)
1776 54     return emb
1777 55
1778 56
1779 57 ## (5) Edge Feature Network
1780 58 class EdgeFeatureNet(nn.Module):
1781 59     def __init__(self, module_cfg):
1782 60         super(EdgeFeatureNet, self).__init__()
1783 61         self._cfg = module_cfg
1784 62
1785 63         self.c_s = self._cfg.embed.c_s
1786 64         self.c_z = self._cfg.embed.c_z
1787 65         self.feats_dim = self._cfg.embed.feats_dim
1788 66
1789 67         self.linear_s_p = nn.Linear(self.c_s, self.feats_dim)
1790 68         self.linear_relpos = nn.Linear(self.feats_dim, self.feats_dim)
1791 69
1792 70         total_edge_feats = self.feats_dim * 3 + self._cfg.embed.num_bins * 2 + 2
1793 71
1794 72         self.edge_embedder = nn.Sequential(
1795 73             nn.Linear(total_edge_feats, self.c_z),
1796 74             nn.ReLU(),
1797 75             nn.Linear(self.c_z, self.c_z),
1798 76             nn.ReLU(),
1799 77             nn.Linear(self.c_z, self.c_z),
1800 78             nn.LayerNorm(self.c_z),
1801 79         )
1802 80
1803 81     def embed_relpos(self, r):
1804 82         d = r[:, :, None, :] - r[:, None, :, :]
1805 83         pos_emb = get_index_embedding(d, self.feats_dim, max_len=2056)
1806 84         return self.linear_relpos(pos_emb)
1807 85
1808 86     def _cross_concat(self, feats_ld, num_batch, num_res):

```

```

1782     return torch.cat([
1783         torch.tile(feats_ld[:, :, None, :], (1, 1, num_res, 1)),
1784         torch.tile(feats_ld[:, None, :, :], (1, num_res, 1, 1)),
1785     ], dim=-1).float().reshape([num_batch, num_res, num_res, -1])
1786
1787 def forward(self, s, t, sc_t, edge_mask, flow_mask):
1788     # Input: [b, n_res, c_s]
1789     num_batch, num_res, _ = s.shape
1790
1791     # [b, n_res, c_z]
1792     p_i = self.linear_s_p(s)
1793     cross_node_feats = self._cross_concat(p_i, num_batch, num_res)
1794
1795     # [b, n_res]
1796     r = torch.arange(
1797         num_res, device=s.device).unsqueeze(0).repeat(num_batch, 1)
1798     relpos_feats = self.embed_relpos(r)
1799
1800     dist_feats = calc_distogram(
1801         t, min_bin=1e-3, max_bin=20.0, num_bins=self._cfg.embed.num_bins)
1802     sc_feats = calc_distogram(
1803         sc_t, min_bin=1e-3, max_bin=20.0, num_bins=self._cfg.embed.num_bins)
1804
1805     all_edge_feats = [cross_node_feats, relpos_feats, dist_feats, sc_feats]
1806
1807     diff_feat = self._cross_concat(flow_mask[..., None], num_batch, num_res)
1808     all_edge_feats.append(diff_feat)
1809
1810     edge_feats = self.edge_embedder(torch.concat(all_edge_feats, dim=-1).to(torch.float))
1811     edge_feats *= edge_mask.unsqueeze(-1)
1812     return edge_feats
1813
1814
1815 ## (4) Node Feature Network
1816 class NodeFeatureNet(nn.Module):
1817     def __init__(self, module_cfg):
1818         super(NodeFeatureNet, self).__init__()
1819         self._cfg = module_cfg
1820         self.c_s = self._cfg.embed.c_s
1821         self.c_pos_emb = self._cfg.embed.c_pos_emb
1822         self.c_timestep_emb = self._cfg.embed.c_timestep_emb
1823         embed_size = self.c_pos_emb + self.c_timestep_emb * 2 + 1
1824
1825         self.aatype_embedding = nn.Embedding(21, self.c_s) # Always 21 because of 20 amino
1826         # acids + 1 for unk
1827         embed_size += self.c_s + self.c_timestep_emb + self._cfg.num_aa_type
1828
1829         self.linear = nn.Sequential(
1830             nn.Linear(embed_size, self.c_s),
1831             nn.ReLU(),
1832             nn.Linear(self.c_s, self.c_s),
1833             nn.ReLU(),
1834             nn.Linear(self.c_s, self.c_s),
1835             nn.LayerNorm(self.c_s),
1836         )
1837
1838     def embed_t(self, timesteps, mask):
1839         timestep_emb = get_timestep_embedding(
1840             timesteps,
1841             self.c_timestep_emb,
1842             max_positions=2056
1843         )[:, None, :].repeat(1, mask.shape[1], 1)
1844         return timestep_emb * mask.unsqueeze(-1)
1845
1846     def forward(
1847         self,
1848         *,
1849         t,
1850         res_mask,
1851         flow_mask,
1852         pos,
1853         aatypes,
1854         aatypes_sc,
1855     ):
1856         # [b, n_res, c_pos_emb]
1857         pos_emb = get_index_embedding(pos, self.c_pos_emb, max_len=2056)
1858         pos_emb = pos_emb * res_mask.unsqueeze(-1)
1859
1860         # [b, n_res, c_timestep_emb]
1861         input_feats = [
1862             pos_emb,

```

```

1836         flow_mask[..., None],
1837         self.embed_t(t, res_mask),
1838         self.embed_t(t, res_mask)
1839     ]
1840     input_feats.append(self.aatype_embedding(aatypes))
1841     input_feats.append(self.embed_t(t, res_mask))
1842     input_feats.append(aatypes_sc)
1843     return self.linear(torch.cat(input_feats, dim=-1))
1844
1845 ## (3) Distance Embedder
1846 class DistEmbedder(nn.Module):
1847     def __init__(self, model_conf):
1848         super(DistEmbedder, self).__init__()
1849         torch.set_default_dtype(torch.float32)
1850         self._model_conf = model_conf
1851         self._embed_conf = model_conf.embed
1852
1853         edge_embed_size = self._model_conf.edge_embed_size
1854
1855         self.dist_min = self._model_conf.bb_ligand_rbf_d_min
1856         self.dist_max = self._model_conf.bb_ligand_rbf_d_max
1857         self.num_rbf_size = self._model_conf.num_rbf_size
1858         self.edge_embedder = nn.Sequential(
1859             nn.Linear(self.num_rbf_size, edge_embed_size),
1860             nn.ReLU(),
1861             nn.Linear(edge_embed_size, edge_embed_size),
1862             nn.ReLU(),
1863             nn.Linear(edge_embed_size, edge_embed_size),
1864             nn.LayerNorm(edge_embed_size),
1865         )
1866
1867         mu = torch.linspace(self.dist_min, self.dist_max, self.num_rbf_size)
1868         self.mu = mu.reshape([1, 1, 1, -1])
1869         self.sigma = (self.dist_max - self.dist_min) / self.num_rbf_size
1870
1871     def coord2dist(self, coord, edge_mask):
1872         n_batch, n_atom = coord.size(0), coord.size(1)
1873         radial = torch.sum((coord.unsqueeze(1) - coord.unsqueeze(2)) ** 2, dim=-1)
1874         dist = torch.sqrt(
1875             radial + 1e-10
1876         ) * edge_mask
1877
1878         radial = radial * edge_mask
1879         return radial, dist
1880
1881     def rbf(self, dist):
1882         dist_expand = torch.unsqueeze(dist, -1)
1883         _mu = self.mu.to(dist.device)
1884         rbf = torch.exp(-((dist_expand - _mu) / self.sigma) ** 2)
1885         return rbf
1886
1887     def forward(
1888         self,
1889         rigid,
1890         ligand_pos,
1891         bb_ligand_mask,
1892     ):
1893         curr_bb_pos = all_atom.to_atom37(Rigid.from_tensor_7(torch.clone(rigid)))[-1][:, :,
1894         1].to(ligand_pos.device)
1895
1896         curr_bb_lig_pos = torch.cat([curr_bb_pos, ligand_pos], dim=1)
1897         edge_mask = bb_ligand_mask.unsqueeze(dim=1) * bb_ligand_mask.unsqueeze(dim=2)
1898
1899         radial, dist = self.coord2dist(
1900             coord=curr_bb_lig_pos,
1901             edge_mask=edge_mask,
1902         )
1903
1904         edge_embed = self.rbf(dist) * edge_mask[..., None]
1905         edge_embed = self.edge_embedder(edge_embed.to(torch.float))
1906
1907         return edge_embed
1908
1909 ## (2) Cross-Attention
1910 class CrossAttention(nn.Module):
1911     def __init__(self, query_input_dim, key_input_dim, output_dim):
1912         super(CrossAttention, self).__init__()
1913         self.out_dim = output_dim

```

```

1890         self.W_Q = nn.Linear(query_input_dim, output_dim)
1891         self.W_K = nn.Linear(key_input_dim, output_dim)
1892         self.W_V = nn.Linear(key_input_dim, output_dim)
1893         self.scale_val = self.out_dim ** 0.5
1894         self.softmax = nn.Softmax(dim=-1)
1895
1896     def forward(self, query_input, key_input, value_input, query_input_mask=None,
1897                 key_input_mask=None):
1898         query = self.W_Q(query_input)
1899         key = self.W_K(key_input)
1900         value = self.W_V(value_input)
1901
1902         attn_weights = torch.matmul(query, key.transpose(1, 2)) / self.scale_val
1903         attn_mask = query_input_mask.unsqueeze(-1) * key_input_mask.unsqueeze(-1).transpose(1,
1904         2)
1905         attn_weights = attn_weights.masked_fill(attn_mask == False, -1e9)
1906         attn_weights = self.softmax(attn_weights)
1907         output = torch.matmul(attn_weights, value)
1908
1909         return output, attn_weights
1910
1911     ## (1) Protein-Ligand Network
1912     class ProteinLigandNetwork(nn.Module):
1913     def __init__(self, model_conf):
1914         super(ProteinLigandNetwork, self).__init__()
1915         torch.set_default_dtype(torch.float32)
1916         self._model_conf = model_conf
1917
1918         # Input Node Embedder
1919         self.node_feature_net = NodeFeatureNet(model_conf)
1920
1921         # Input Edge Embedder
1922         self.edge_feature_net = EdgeFeatureNet(model_conf)
1923
1924         # 3D Molecule GNN
1925         self.mol_embedding_layer = MolEmbedder(model_conf)
1926
1927         # Invariant Point Attention (IPA) Network
1928         self.ipanet = ipa_pytorch.IpaNetwork(model_conf)
1929
1930         # Node Fusion
1931         self.node_embed_size = self._model_conf.node_embed_size
1932         self.node_embedder = nn.Sequential(
1933             nn.Embedding(self._model_conf.num_aa_type, self.node_embed_size),
1934             nn.ReLU(),
1935             nn.Linear(self.node_embed_size, self.node_embed_size),
1936             nn.LayerNorm(self.node_embed_size),
1937         )
1938         self.node_fusion = nn.Sequential(
1939             nn.Linear(self.node_embed_size + self.node_embed_size, self.node_embed_size),
1940             nn.ReLU(),
1941             nn.Linear(self.node_embed_size, self.node_embed_size),
1942             nn.LayerNorm(self.node_embed_size),
1943         )
1944
1945         # Backbone-Substrate Fusion
1946         self.bb_lig_fusion = CrossAttention(
1947             query_input_dim=self.node_embed_size,
1948             key_input_dim=self.node_embed_size,
1949             output_dim=self.node_embed_size,
1950         )
1951
1952         # Edge Fusion
1953         self.edge_embed_size = self._model_conf.edge_embed_size
1954         self.edge_dist_embedder = DistEmbedder(model_conf)
1955
1956         # Amino Acid Prediction Network
1957         self.aatype_pred_net = nn.Sequential(
1958             nn.Linear(self.node_embed_size, self.node_embed_size),
1959             nn.ReLU(),
1960             nn.Linear(self.node_embed_size, self.node_embed_size),
1961             nn.ReLU(),
1962             nn.Linear(self.node_embed_size, model_conf.num_aa_type),
1963         )
1964
1965         if self._model_conf.flow_msa:
1966             # Co-Evolution Embedder
1967             self.msa_embedding_layer = CoEvoFormer(model_conf)
1968
1969         # Coevo-Backbone-Substrate Fusion

```



```

1944         self.msa_bb_lig_fusion = CrossAttention(
1945             query_input_dim=model_conf.msa.msa_embed_size,
1946             key_input_dim=self.node_embed_size,
1947             output_dim=self.node_embed_size,
1948         )
1949         # Coevo Prediction Network
1950         self.msa_pred = nn.Sequential(
1951             nn.Linear(self.node_embed_size, self.node_embed_size),
1952             nn.SiLU(),
1953             nn.Linear(self.node_embed_size, self.node_embed_size),
1954             nn.SiLU(),
1955             nn.Linear(self.node_embed_size, model_conf.msa.num_msa_vocab),
1956         )
1957         if self._model_conf.ec:
1958             # EC Embedder
1959             self.ec_embedding_layer = nn.Sequential(
1960                 nn.Embedding(model_conf.ec.num_ec_class, model_conf.ec.ec_embed_size),
1961                 nn.SiLU(),
1962                 nn.Linear(model_conf.ec.ec_embed_size, model_conf.ec.ec_embed_size),
1963                 nn.LayerNorm(model_conf.ec.ec_embed_size),
1964             )
1965             # EC-Backbone-Substrate Fusion
1966             self.ec_bb_lig_fusion = CrossAttention(
1967                 query_input_dim=model_conf.ec.ec_embed_size,
1968                 key_input_dim=self.node_embed_size,
1969                 output_dim=self.node_embed_size,
1970             )
1971             # EC Prediction Network
1972             self.ec_pred = nn.Sequential(
1973                 nn.Linear(self.node_embed_size, self.node_embed_size),
1974                 nn.SiLU(),
1975                 nn.Linear(self.node_embed_size, self.node_embed_size),
1976                 nn.SiLU(),
1977                 nn.Linear(self.node_embed_size, model_conf.ec.num_ec_class),
1978             )
1979         self.condition_generation = self._model_conf.guide_by_condition
1980         if self.condition_generation:
1981             # 2D Molecule GNN
1982             self.guide_ligand_mpnn = MolEmbedder2D(model_conf)
1983             # Backbone-Product Fusion
1984             self.guide_bb_lig_fusion = CrossAttention(
1985                 query_input_dim=self.node_embed_size,
1986                 key_input_dim=self.node_embed_size,
1987                 output_dim=self.node_embed_size,
1988             )
1989         def forward(self, input_feats, use_context=False):
1990             # Frames as [batch, res, 7] tensors.
1991             bb_mask = input_feats["res_mask"].type(torch.float32) # [B, N]
1992             flow_mask = input_feats["flow_mask"].type(torch.float32)
1993             edge_mask = bb_mask[..., None] * bb_mask[..., None, :]
1994             n_batch, n_res = bb_mask.shape
1995             # Encode Backbone Nodes with Input Node Embedder
1996             init_bb_node_embed = self.node_feature_net(
1997                 t=input_feats["t"],
1998                 res_mask=bb_mask,
1999                 flow_mask=flow_mask,
2000                 pos=input_feats["seq_idx"],
2001                 aatypes=input_feats["aatype_t"],
2002                 aatypes_sc=input_feats["sc_aa_t"],
2003             )
2004             # Encode Backbone Edges with Input Edge Embedder
2005             init_bb_edge_embed = self.edge_feature_net(
2006                 s=init_bb_node_embed,
2007                 t=input_feats["trans_t"],
2008                 sc_t=input_feats["sc_ca_t"],
2009                 edge_mask=edge_mask,
2010                 flow_mask=flow_mask,
2011             )
2012             # Masking Padded Residues
2013             bb_node_embed = init_bb_node_embed * bb_mask[..., None]

```

```

1998 407 bb_edge_embed = init_bb_edge_embed * edge_mask[..., None]
1999 408
2000 409 # AminoAcid embedding
2001 410 bb_aa_embed = self.node_embedder(input_feats["aatype_t"]) * bb_mask[..., None]
2002 411 bb_aa_embed = torch.cat([bb_aa_embed, bb_node_embed], dim=-1)
2003 412 # Backbone-AminoAcid Fusion
2004 413 bb_node_embed = self.node_fusion(bb_aa_embed)
2005 414 bb_node_embed = bb_node_embed * bb_mask[..., None]
2006 415
2007 416 # Initialize Substrate Masking
2008 417 lig_mask = input_feats["ligand_mask"]
2009 418 lig_edge_mask = lig_mask[..., None] * lig_mask[..., None, :]
2010 419 # Encode Substrate with 3D Molecule GNN
2011 420 lig_init_node_embed, _ = self.mol_embedding_layer(
2012 421     ligand_atom=input_feats["ligand_atom"],
2013 422     ligand_pos=input_feats["ligand_pos"],
2014 423     edge_mask=lig_edge_mask,
2015 424 )
2016 425 lig_node_embed = lig_init_node_embed * lig_mask[..., None]
2017 426
2018 427 # Backbone-Substrate Fusion
2019 428 bb_lig_rep, _ = self.bb_lig_fusion(
2020 429     query_input=bb_node_embed,
2021 430     key_input=lig_node_embed,
2022 431     value_input=lig_node_embed,
2023 432     query_input_mask=bb_mask,
2024 433     key_input_mask=lig_mask,
2025 434 )
2026 435
2027 436 # Residue Connection
2028 437 bb_node_embed = bb_node_embed + bb_lig_rep
2029 438
2030 439 # Conditioning on Product Molecule
2031 440 if self.condition_generation:
2032 441     # Encode Product with 2D Molecule GNN
2033 442     guide_ligand_rep = self.guide_ligand_mpnn(
2034 443         mol_atom=input_feats["guide_ligand_atom"],
2035 444         mol_edge=input_feats["guide_ligand_edge_index"],
2036 445         mol_edge_feat=input_feats["guide_ligand_edge"],
2037 446         mol_atom_mask=input_feats["guide_ligand_atom_mask"],
2038 447         mol_edge_mask=input_feats["guide_ligand_edge_mask"],
2039 448     ).unsqueeze(1)
2040 449
2041 450 # Initialize Product Masking
2042 451 guide_ligand_mask = input_feats["guide_ligand_atom_mask"][:, 0:1]
2043 452 # Backbone-Product Fusion
2044 453 bb_guide_lig_rep, _ = self.guide_bb_lig_fusion(
2045 454     query_input=bb_node_embed,
2046 455     key_input=guide_ligand_rep,
2047 456     value_input=guide_ligand_rep,
2048 457     query_input_mask=bb_mask,
2049 458     key_input_mask=guide_ligand_mask,
2050 459 )
2051 460
2052 461 # Residue Connection
2053 462 bb_node_embed = bb_node_embed + bb_guide_lig_rep
2054 463
2055 464 # Initialize Backbone-Substrate Masking
2056 465 bb_ligand_mask = torch.cat([bb_mask, lig_mask], dim=-1)
2057 466 # Backbone-Substrate Distance Embedding
2058 467 bb_lig_edge = self.edge_dist_embedder(
2059 468     rigid=input_feats["rigids_t"],
2060 469     ligand_pos=input_feats["ligand_pos"],
2061 470     bb_ligand_mask=bb_ligand_mask,
2062 471 )
2063 472
2064 473 # Backbone-Backbone-Product Edge Fusion
2065 474 bb_edge_embed = bb_edge_embed + bb_lig_edge[:, :n_res, :n_res, :]
2066 475
2067 476 # Masking Padded Residues
2068 477 bb_node_embed = bb_node_embed[:, :n_res, :] * bb_mask[..., None]
2069 478 bb_edge_embed = bb_edge_embed[:, :n_res, :n_res, :] * edge_mask[..., None]
2070 479
2071 480 # Run IPA Network
2072 481 model_out = self.ipanet(bb_node_embed, bb_edge_embed, input_feats)
2073 482 node_embed = model_out["node_embed"] * bb_mask[..., None]
2074 483
2075 484 # Amino Acid Prediction with Amino Acid Prediction Network
2076 485 aa_pred = self.aatype_pred_net(node_embed) * bb_mask[..., None]
2077 486
2078 487 if self._model_conf.flow_msa:

```

```

2052 488         # Encode Coevo with Co-Evolution Embedder
2053 489         msa_mask = input_feats["msa_mask"]
2054 490         msa_embed = self.msa_embedding_layer(input_feats["msa_t"], msa_mask=msa_mask) *
2055 491         msa_mask[..., None] #[B, N_msa, N_token, D]
2056         msa_rep = msa_embed.sum(dim=1) / (msa_mask[..., None].sum(dim=1) + 1e-10) #[B, 1,
2057         D]
2058         _msa_mask = msa_mask[:, 0] #torch.ones_like(msa_rep[..., 0]).to(msa_embed.device)
2059
2060         # Coevo-Backbone Fusion
2061         msa_rep, _ = self.msa_bb_lig_fusion(
2062             query_input=msa_rep,
2063             key_input=node_embed,
2064             value_input=node_embed,
2065             query_input_mask=_msa_mask,
2066             key_input_mask=bb_mask,
2067         )
2068
2069         # Coevo Prediction with Coevo Prediction Network
2070         msa_pred = self.msa_pred(msa_rep)
2071
2072         if self._model_conf.flow_ec:
2073             # Encode EC with EC Embedder
2074             ec_embed = self.ec_embedding_layer(input_feats["ec_t"])
2075             ec_mask = torch.ones_like(ec_embed[..., 0]).to(ec_embed.device)
2076
2077             # EC-Backbone Fusion
2078             ec_rep, _ = self.ec_bb_lig_fusion(
2079                 query_input=ec_embed,
2080                 key_input=node_embed,
2081                 value_input=node_embed,
2082                 query_input_mask=ec_mask,
2083                 key_input_mask=bb_mask,
2084             )
2085
2086             # EC Prediction with EC Prediction Network
2087             ec_rep = ec_rep.reshape(n_batch, -1)
2088             ec_pred = self.ec_pred(ec_rep)
2089
2090         # Main Network Ouput
2091         pred_out = {
2092             "amino_acid": aa_pred,
2093             "rigids_tensor": model_out["rigids"],
2094         }
2095
2096         if self._model_conf.flow_msa:
2097             pred_out["msa"] = msa_pred * _msa_mask[..., None]
2098
2099         if self._model_conf.flow_ec:
2100             pred_out["ec"] = ec_pred
2101
2102         pred_out["rigids"] = model_out["rigids"].to_tensor_7()
2103         return pred_out

```

Listing 4: Pytorch Implementation of EnzymeFlow Main Network.

*Fun Fact:* While implementing enzyme-substrate and enzyme-product interactions by cross-attention fusion networks, we experimented with using PairFormer (with only 3-4 layers) as implemented in AlphaFold3 (Abramson et al., 2024). However, the computational load was immense—it would take years to run on our A40 GPU. Our fusion network turns to be a more efficient approach. It makes me wonder who has the resources to re-train AlphaFold3, given the heavy computational demands!