Natural-Language-Guided Protein Generation for Ligand-Binding Design

Shumeng Li[1] **Xunkai Li**[1] **Sirui Zhang**[1] **Henan Sun**[2]
Yongfu Xiong[3] **Yi Liu**[3] **Rong-Hua Li**[1] **Guoren Wang**[1]
[1]Beijing Institute of Technology
[2]The Hong Kong University of Science and Technology
[3]Chongqing Ant Consumer Finance Co., Ltd.
Corresponding author: Rong-Hua Li (lironghuabit@126.com)

Abstract

Docking ultimately determines effective ligand binding, yet bridging global functional semantics and local physical constraints remains a central challenge for protein generation for ligand-binding design. We therefore use docking-oriented language describing docking processes, binding patterns, and local interaction constraints as generation conditions, rather than relying only on coarse-grained functional text. However, existing methods still insufficiently model docking constraints and show limited coordination among heterogeneous modalities. To address these limitations, we propose DockWizard, an encoder-router-decoder framework for multimodal protein generation. Concretely, (1) a **multimodal encoder** aligns function text, docking text, ligand descriptions, ligand SMILES, and ligand 3D information into shared conditional representations, improving semantic consistency across heterogeneous inputs; (2) a **router module** organizes heterogeneous conditions into role-specific memories by their scopes of action, preserving global functional guidance while strengthening local binding constraints; and (3) a **protein decoder** combines prefix guidance with full-memory injection to let different conditions act at different levels, improving controllability while preserving docking constraints during generation. We further curate a docking-language multimodal dataset with over 100,000 samples. Extensive experiments under a four-stage evaluation pipeline show that DockWizard outperforms strong baselines for this task, with average gains of 13% in docking confidence and 19% in complex stability.

1 Introduction

Natural-language-guided protein generation for ligand-binding design seeks to generate sequences that satisfy target functions and bind target ligands, with applications in enzyme engineering, molecular recognition, and therapeutic discovery [Quijano-Rubio et al., 2021, Lu et al., 2024]. Although natural-language-guided protein generation has progressed [Liu et al., 2025b, Song et al., 2025b], most methods still rely on coarse functional descriptions and underuse docking-related local constraints, motivating docking text as generation-time control [Song et al., 2025b].

As illustrated in Fig. 1, three limitations remain for ligand-binding design. **(1) Insufficient fine-grained language conditioning.** Existing language-guided methods mainly use coarse functional descriptions, so docking-related local constraints remain weakly represented in the generation conditions [Liu et al., 2025b, Song et al., 2025b]. **(2) No explicit role separation between global function and local constraints.** Even when some methods introduce functional, ligand, or structural information simultaneously, the different roles of these conditions are not clearly separated

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

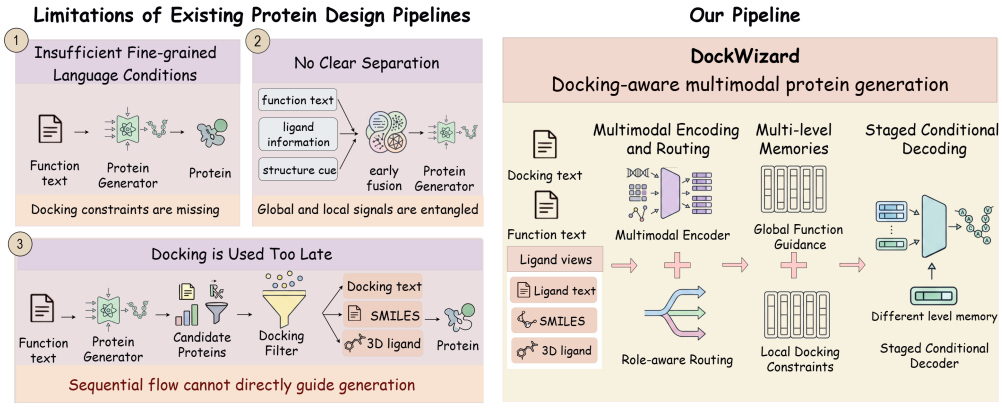


Figure 1: Comparison of representative conditional protein design methods and DockWizard.

according to their scopes of action. This easily leads to condition mixing and weakened constraints, which in turn compromises controllability and interpretability during generation [Luo et al., 2024, Madani et al., 2020]. **(3) Docking is disconnected from the generation process.** Current methods typically use docking for post hoc filtering, reranking, or external evaluation rather than as an active generation-time control signal, leaving binding-relevant information outside sequence generation [Song et al., 2025b]. Together, these limitations hinder ligand-binding protein generation from jointly modeling global semantics and local docking constraints.

To address the above problems, we propose **DockWizard**, a natural-language-guided, docking-aware multimodal protein generation framework for ligand-binding design. Our method contains three core innovations. **(1)** We design a multimodal encoder that aligns docking text with function text, ligand descriptions, ligand SMILES, and ligand 3D information, making binding patterns, local interactions, and key constraints explicit generation conditions. **(2)** We further introduce a routing-based role-separation mechanism that organizes global functional semantics and local docking constraints into distinct memory streams, with a learnable bridge enabling constrained coordination between them while preserving their functional boundaries. **(3)** We construct a staged condition-injection mechanism that keeps binding-related information active during decoding, turning docking from a post hoc evaluation signal into a generation-time control signal.

Our Contributions. (1) New Perspective: We introduce docking-oriented natural language as a generation-time control signal, enabling ligand-binding protein generation to jointly model functional semantics and fine-grained binding constraints. **(2) New Method:** We develop an encoder–routing–decoder framework that unifies multimodal representation, role-specific memory organization, and staged condition injection for controllable protein generation. **(3) New Benchmark:** We construct a large-scale docking-language benchmark covering protein sequences, function descriptions, ligand SMILES, ligand 3D structures, ligand descriptions, and protein–ligand docking descriptions. **(4) SOTA Performance:** Under a four-stage evaluation pipeline, DockWizard achieves state-of-the-art overall performance in designing high-quality proteins, improving protein design quality by more than 10% on average compared with other baselines.

2 Background and Related Work

2.1 Notations and Problem Formulation

We formulate docking-aware protein generation as autoregressive generation under multimodal conditional constraints. For each sample, the conditioning set is written as

$$c = \left(c_{\text{dock}}^{\text{text}},\, c_{\text{func}}^{\text{text}},\, c_{\text{lig}}^{\text{text}},\, c_{\text{smi}},\, c_{3d}\right). \tag{1}$$

where $c_{\text{dock}}^{\text{text}}$ and $c_{\text{func}}^{\text{text}}$ denote docking-oriented and function text conditions, while $c_{\text{lig}}^{\text{text}}$, c_{smi}, and c_{3d} denote ligand description, SMILES, and 3D structure.

To preserve both global functional guidance and local interaction constraints, we decompose the overall generation process into three stages: encoding, routing, and decoding. Let E_{enc} denote the multimodal condition encoder, R_{route} the routing module, and D_{dec} the conditioned decoder.

The overall process is written as

$$
\begin{aligned}
&(\mathbf{X}_c, \mathbf{m}_c) = E_{\mathrm{enc}}(c), \qquad (\mathbf{M}^{\mathrm{func}}, \mathbf{M}^{\mathrm{dl,full}}, \mathbf{M}^{\mathrm{dl,light}}) = R_{\mathrm{route}}(\mathbf{X}_c, \mathbf{m}_c), \\
&p_\theta(y \mid c) = D_{\mathrm{dec}}(y \mid c) = \prod_{t=1}^{T} p_\theta\big(y_t \mid y_{<t}, \mathbf{M}^{\mathrm{func}}, \mathbf{M}^{\mathrm{dl,light}}, \mathbf{M}^{\mathrm{dl,full}}\big)\,.
\end{aligned}
\tag{2}
$$

Here, $\mathbf{X}_c$ and $\mathbf{m}_c$ denote the packed multimodal context and its corresponding mask, respectively. The routing module further constructs three types of role-specific memories, where $\mathbf{M}^{\mathrm{func}}$ carries global functional semantics, $\mathbf{M}^{\mathrm{dl,full}}$ retains richer docking/ligand interaction details, and $\mathbf{M}^{\mathrm{dl,light}}$ provides a compressed form to support more efficient condition injection.

2.2 Docking Information in Protein Design

Binding-oriented protein design seeks to generate sequences that form geometrically compatible and stable ligand-binding interfaces. One line of work starts from target structures or interfaces, including target-structure-based binder design [Cao et al., 2022] and RFdiffusion binder design for flexible targets [Vázquez Torres et al., 2024], but these controls remain largely structural and provide limited access to fine-grained interaction constraints.

Early docking-guided design frameworks, such as RosettaAntibodyDesign [Adolf-Bryfogle et al., 2018], jointly sample sequence, structure, and binding space on antibody–antigen complexes. They are physically interpretable, but often depend on templates, energy functions, and post hoc search, leaving generation loosely coupled with the binding objective.

Docking information also enters end-to-end protein design models more directly. For example, DiffAb [Luo et al., 2022] jointly models the sequence and structure of antibody CDRs and explicitly conditions on specific antigen structures. HSRN [Jin et al., 2022] simultaneously folds and docks paratope sequences during generation. dyMEAN [Kong et al., 2023] bridges epitope–antibody connections through a shadow paratope and completes docking during iterative updates. ABDPO [Zhou et al., 2024] further fine-tunes a conditional diffusion model with residue-level energy preferences.

2.3 Conditional Protein Design under Multimodal Controls

Conditional protein design methods differ in their conditions, representations, injection mechanisms, and role separation. Figure 1 summarizes these differences against our method.

(1) Label-based and lightweight conditional control paradigms. Existing methods often represent conditions as discrete labels, control tokens, or lightweight parameterized modules. ProGen [Madani et al., 2020] organizes cross-family protein generation with control labels. PrefixProt [Luo et al., 2024] introduces prefix tuning into pretrained protein language models. ProCALM [Yang et al., 2024] further uses adapters to incorporate enzyme family, taxonomy, and natural-language descriptions. CtrlProt [Liu et al., 2025c] introduces multi-attribute controllability into a preference-optimization framework. These methods are efficient and controllable, but their conditioning space is often constrained by predefined labels or single-path inputs.

(2) Text- or annotation-driven paradigms. Another line of work directly introduces free text, database annotations, or textual functional descriptions to increase the semantic bandwidth of conditional expression. ProteinDT [Liu et al., 2025b] supports text-guided protein design through protein–text alignment. PAAG [Yuan et al., 2025] uses mulreti-level domain annotations to guide generation. ProtDAT [Guo et al., 2024] further strengthens the interaction between text and sequence through unified multimodal cross-attention. ProDVa [Liu et al., 2025a] combines textual functional descriptions with fragment-level vocabulary. Compared with label-based control, they better capture open-ended intent, but still emphasize function, property, or domain-level semantics rather than docking processes, binding patterns, or local interaction constraints.

(3) Condition organization paradigms. After conditions enter the generator, existing methods span flat fusion and structured conditioning. Prefix-Tuning [Li and Liang, 2021], Prompt Tuning [Lester

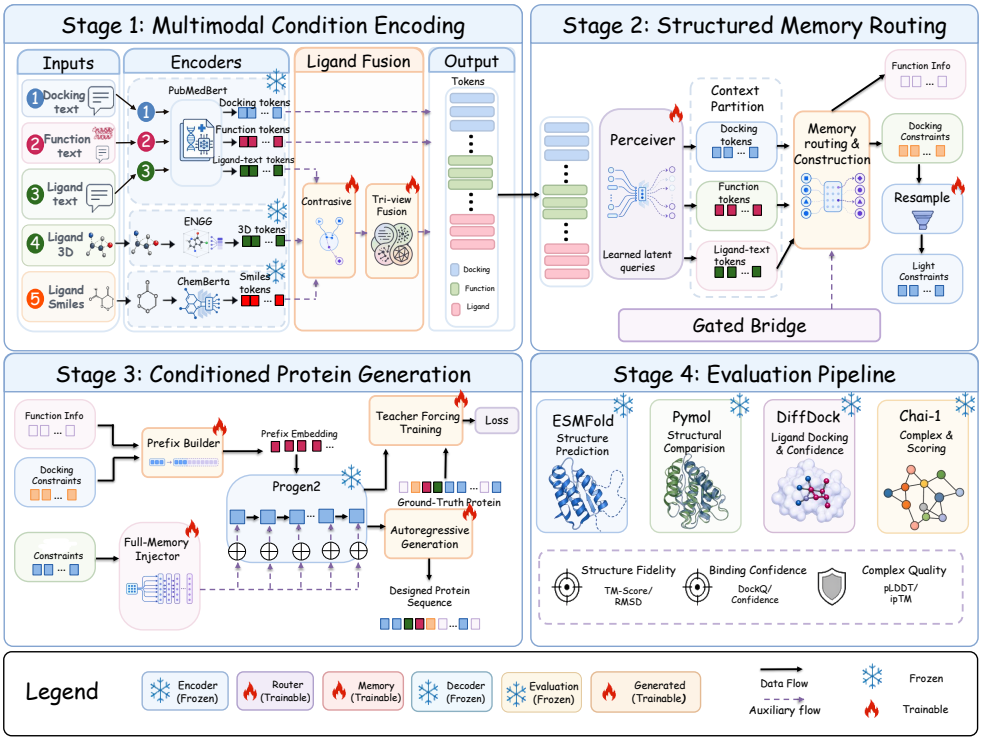


Figure 2: Overview of the proposed DockWizard framework.

et al., 2021], and P-Tuning v2 [Liu et al., 2022b] typically compress conditions into a single control stream. Perceiver [Jaegle et al., 2021b] and Perceiver IO [Jaegle et al., 2021a] organize heterogeneous inputs through latent bottlenecks. Flamingo [Alayrac et al., 2022] and BLIP-2 [Li et al., 2023] further connect different modalities through bridging / querying modules. Flat control streams are simple and parameter-efficient, whereas structured conditioning better preserves multi-source information. However, many methods still flatten heterogeneous conditions before decoding, weakening role separation between global functional guidance and local interaction constraints.

3 Method

DockWizard follows the framework in Fig. 2: multimodal condition encoding, Perceiver-based structured memory routing, and staged conditioned protein generation. The downstream evaluation pipeline in the figure is discussed in Section 4.

Motivation. We encode five heterogeneous conditions into a unified context space while preserving their modality-specific roles before the subsequent routing stage.

Text-stream Encoding. For the three textual conditions, namely docking text, function text, and ligand text, we use PubMedBERT [Gu et al., 2021] as the text backbone. For ligand SMILES, we use ChemBERTa [Chithrananda et al., 2020] as the symbolic-condition backbone. We then compress variable-length inputs into fixed-length token sequences through hierarchical query pooling, which is formally close to the query-based aggregation in Perceiver IO [Jaegle et al., 2021a]. Specifically, let $s \in \{D, F, T, S\}$ denote the four discrete condition streams, namely docking text, function text, ligand text, and SMILES. The encoding process is written as

$$\mathbf{H}_{s,j} = B_s(\mathbf{C}_{s,j}), \qquad \mathbf{X}^s = \mathrm{QPool}_s^{\mathrm{global}}\big(\{\mathrm{QPool}_s^{\mathrm{chunk}}(\mathbf{H}_{s,j})\}_{j=1}^{J_s}\big). \tag{3}$$

where $\mathbf{C}_{s,j}$ denotes the j-th local segment, B_s denotes the corresponding modality backbone, and $\mathrm{QPool}_s^{\mathrm{chunk}}$ and $\mathrm{QPool}_s^{\mathrm{global}}$ denote chunk-level and global-level query-based pooling, respectively. This yields length-controlled condition tokens for role-specific routing.

For ligand 3D, EGNN [Satorras et al., 2021] models atom types and spatial coordinates, and then produces a fixed-length sequence of 3D tokens aligned with the unified condition space.

Tri-view Ligand Fusion. We represent ligand text, SMILES, and ligand 3D structure as three complementary views, $\mathbf{X}^T$, $\mathbf{X}^S$, and $\mathbf{X}^G$, and fuse them into a unified ligand token sequence $\mathbf{L}$. The fusion weights are not a simple softmax, but include a lower-bound mixing term:

$$\begin{aligned} \mathbf{L} &= \mathrm{Fuse}(\mathbf{X}^T, \mathbf{X}^S, \mathbf{X}^G), \\ \mathbf{g} &= \mathrm{MLP}_{\mathrm{gate}}\big(\mathrm{Pool}(\mathbf{X}^T) \parallel \mathrm{Pool}(\mathbf{X}^S) \parallel \mathrm{Pool}(\mathbf{X}^G)\big), \\ \mathbf{w}^{\mathrm{lig}} &= (1-\alpha)\,\mathrm{softmax}(\mathbf{g}) + \alpha \cdot \frac{1}{3}\mathbf{1}. \end{aligned} \tag{4}$$

Here, $\mathbf{w}^{\mathrm{lig}}$ denotes the tri-view fusion weights, $\mathbf{g}$ denotes the gating scores produced by $\mathrm{MLP}_{\mathrm{gate}}$ from the pooled summaries of the three ligand views, $\mathrm{Pool}(\cdot)$ denotes sequence pooling, and α preserves a non-zero lower bound to avoid view collapse. We further align the three views with an auxiliary contrastive constraint and use $\mathbf{L}$ as the ligand-conditioning interface.

Context Packing. We further perform group-level calibration and add type encodings, and then construct the packed context in a fixed order:

$$\mathbf{X}_c = [\mathbf{D} \parallel \mathbf{L} \parallel \mathbf{F}], \qquad \mathbf{m}_c = [\mathbf{m}^D \parallel \mathbf{m}^L \parallel \mathbf{m}^F]. \tag{5}$$

Here, $\mathbf{D}$, $\mathbf{L}$, and $\mathbf{F}$ denote the docking, ligand, and function token sequences, respectively, and $\mathbf{m}^D$, $\mathbf{m}^L$, and $\mathbf{m}^F$ denote the corresponding masks. The fixed ordering $[\mathbf{D} \parallel \mathbf{L} \parallel \mathbf{F}]$ defines group boundaries for function-oriented and docking/ligand-oriented memory construction.

3.1 Structured Memory Routing via Perceiver

Motivation. We use a Perceiver-based routing module to reorganize the packed context into role-specific memory streams for staged condition injection.

Role-specific Routing. We adopt a Perceiver-style latent bottleneck Jaegle et al. [2021b,a] for role-specific routing over the packed context. Based on the fixed group boundaries, we construct two branch-specific key-value sets:

$$\mathbf{KV}^{\mathrm{func}} = [\mathbf{F} \parallel \mathbf{L}], \quad \mathbf{m}^{\mathrm{func}} = [\mathbf{m}^F \parallel \mathbf{m}^L], \quad \mathbf{KV}^{\mathrm{dl}} = [\mathbf{D} \parallel \mathbf{L}], \quad \mathbf{m}^{\mathrm{dl}} = [\mathbf{m}^D \parallel \mathbf{m}^L]. \tag{6}$$

Here, $\mathbf{KV}^{\mathrm{func}}$ models global functional semantics, $\mathbf{KV}^{\mathrm{dl}}$ captures docking/ligand constraints, and shared ligand tokens connect the two branches.

For branch $b \in \{\mathrm{func}, \mathrm{dl}\}$, let $\mathbf{Z}_b^{(0)}$ denote its learnable latent queries and $\mathbf{KV}^b$ denote the corresponding key-value set. We first use cross-attention to read condition tokens:

$$\widetilde{\mathbf{Z}}_b^{(t)} = \mathbf{Z}_b^{(t)} + \mathrm{MHA}\big(\mathrm{LN}(\mathbf{Z}_b^{(t)}), \mathrm{LN}(\mathbf{KV}^b), \mathrm{LN}(\mathbf{KV}^b); \mathbf{m}^b\big). \tag{7}$$

Here, $\mathrm{MHA}(\cdot)$ denotes multi-head attention, $\mathrm{LN}(\cdot)$ denotes layer normalization, and the mask $\mathbf{m}^b$ sets invalid key positions to $-\infty$ before the attention softmax. The resulting latent states are then refined by standard self-attention and feed-forward layers, with the full update given in Appendix G.4. After several update layers, we obtain

$$\mathbf{M}^{\mathrm{func}} = \mathbf{Z}_{\mathrm{func}}^{(T_{\mathrm{func}})}, \qquad \mathbf{M}^{\mathrm{dl,full}} = \mathbf{Z}_{\mathrm{dl}}^{(T_{\mathrm{dl}})}. \tag{8}$$

Here, $\mathbf{M}^{\mathrm{func}}$ denotes the function-oriented memory, and $\mathbf{M}^{\mathrm{dl,full}}$ denotes the docking/ligand memory that preserves richer interaction details. Thus, the unified context is organized into role-specific memory representations before entering the decoder.

Gated Memory Bridging. To enable controlled interaction between the two memory branches, we use a bridge module that lets the function memory modulate the docking/ligand full memory:

$$\mathbf{U}_{\mathrm{br}} = \mathrm{BridgeBlock}\big(\mathbf{M}^{\mathrm{dl,full}}, \mathbf{M}^{\mathrm{func}}\big), \qquad \mathbf{G}_{\mathrm{br}} = \sigma\big(\mathrm{Gate}_{\mathrm{br}}(\mathbf{M}^{\mathrm{dl,full}})\big). \tag{9}$$

and then update the full interaction memory by gated modulation:

$$\widetilde{\mathbf{M}}^{\mathrm{dl,full}} = \mathbf{M}^{\mathrm{dl,full}} + \mathbf{G}_{\mathrm{br}} \odot \mathbf{U}_{\mathrm{br}}. \tag{10}$$

Here, $\mathrm{BridgeBlock}(\cdot)$ is a cross-attentive bridge from $\mathbf{M}^{\mathrm{func}}$ to $\mathbf{M}^{\mathrm{dl,full}}$, $\mathrm{Gate}_{\mathrm{br}}(\cdot)$ denotes the gating map, $\sigma(\cdot)$ denotes the element-wise sigmoid activation, $\sigma(x) = 1/(1+\exp(-x))$, and $\odot$ denotes token-wise modulation. This update lets global semantics influence the interaction memory without erasing its role.

Memory Resampling. We further compress $\widetilde{\mathbf{M}}^{\mathrm{dl,full}}$ into a lightweight docking/ligand memory using learnable latent queries rather than average pooling. Let $\mathbf{Z}^{(0)}_{\mathrm{light}}$ denote the resampler queries. Its readout process is written as

$$\mathbf{M}^{\mathrm{dl,light}} = \mathrm{Resampler}\Big(\widetilde{\mathbf{M}}^{\mathrm{dl,full}}\Big) = \mathrm{MHA}\big(\mathbf{Z}^{(0)}_{\mathrm{light}}, \widetilde{\mathbf{M}}^{\mathrm{dl,full}}, \widetilde{\mathbf{M}}^{\mathrm{dl,full}}\big). \tag{11}$$

where $\mathbf{M}^{\mathrm{dl,light}}$ denotes the compressed lightweight interaction memory. This memory provides a more compact conditional entry for the subsequent decoder.

3.2 Conditioned Protein Generation with Staged Injection

Motivation. We build a staged conditioning mechanism outside a pretrained causal protein language model Madani et al. [2020], Nijkamp et al. [2023], allowing different levels of conditions to act on the decoding process through different interfaces.

Prefix Construction. We project the function memory and lightweight docking/ligand memory into the decoder hidden space and assemble them as a conditional prefix, using them as persistent contexts during decoding. Let $\mathbf{W}_F$ and $\mathbf{W}_L$ denote the projection maps for the two memory types, and let $\mathbf{b}_F^{\mathrm{open}}$, $\mathbf{b}_F^{\mathrm{close}}$, $\mathbf{b}_L^{\mathrm{open}}$, and $\mathbf{b}_L^{\mathrm{close}}$ denote the corresponding boundary embeddings. The conditional prefix and the initial decoder input are jointly written as

$$\mathbf{H}^{(0)} = \big[\mathbf{b}_F^{\mathrm{open}} \parallel \mathbf{W}_F\mathbf{M}^{\mathrm{func}} \parallel \mathbf{b}_F^{\mathrm{close}} \parallel \mathbf{b}_L^{\mathrm{open}} \parallel \mathbf{W}_L\mathbf{M}^{\mathrm{dl,light}} \parallel \mathbf{b}_L^{\mathrm{close}} \parallel \mathbf{E}(y_{<1:T})\big]. \tag{12}$$

Here, $\mathbf{E}(y_{<1:T})$ denotes the shifted protein-token embeddings used for autoregressive prediction under teacher forcing; at inference, this term is replaced by the embeddings of the generated prefix. We denote the conditional prefix part as $\mathbf{\Pi}$.

Layer-wise Full-memory Injection. We use $\widetilde{\mathbf{M}}^{\mathrm{dl,full}}$ as a layer-wise correction signal for interaction-aware refinement on protein positions. Let $\widehat{\mathbf{H}}^p_\ell$, $\overline{\mathbf{M}}^{\mathrm{dl,full}}$, and $\boldsymbol{\mu}^{\mathrm{full}}$ denote the protein hidden states, projected full interaction memory, and its global summary, respectively, where the superscript p denotes protein-token positions. The injection update at layer ℓ is written as

$$\begin{aligned}
\mathbf{H}^p_\ell &= \widehat{\mathbf{H}}^p_\ell + \mathbf{g}_\ell \odot \mathbf{u}_\ell, \\
\mathbf{g}_\ell &= \sigma\Big(\mathrm{MLP}_\ell\Big(\big[\mathrm{LN}(\widehat{\mathbf{H}}^p_\ell) \parallel \mathbf{1}_{n_{\mathrm{prot}}}(\boldsymbol{\mu}^{\mathrm{full}})^\top\big]\Big)\Big), \\
\mathbf{u}_\ell &= \mathrm{MHA}_\ell\Big(\mathrm{LN}(\widehat{\mathbf{H}}^p_\ell), \overline{\mathbf{M}}^{\mathrm{dl,full}}, \overline{\mathbf{M}}^{\mathrm{dl,full}}\Big).
\end{aligned} \tag{13}$$

Here, $\mathrm{MHA}_\ell(\cdot)$, $\mathrm{LN}(\cdot)$, $\sigma(\cdot)$, and $\odot$ follow the definitions above, $\mathrm{MLP}_\ell(\cdot)$ denotes the gating map, and $\mathbf{1}_{n_{\mathrm{prot}}}$ denotes an all-one column vector over the n_{prot} protein-token positions, used to broadcast $\boldsymbol{\mu}^{\mathrm{full}}$ to all protein positions. This gated update allows the full interaction memory to participate in layer-wise refinement while controlling its perturbation to the backbone representation.

Staged Conditioning. The decoder therefore forms a two-level conditioned generation process with prefix-level conditioning and layer-wise full-memory injection:

$$p_\theta(y \mid c) = \prod_{t=1}^{T} p_\theta\Big(y_t \;\Big|\; y_{<t}, \mathbf{\Pi}, \{\overline{\mathbf{M}}^{\mathrm{dl,full}}\}_{\ell=1}^{L}\Big). \tag{14}$$

Here, $\mathbf{\Pi}$ denotes the persistent conditional prefix, and $\overline{\mathbf{M}}^{\mathrm{dl,full}}$ participates in token prediction through layer-wise injection, supporting stable conditional control and local-constraint expressiveness.

3.3 Training Objective and Inference

Motivation. Training and inference share the same staged-conditioning interface. We train with conditional language modeling supplemented by tri-view ligand alignment, and use the same decoder-side conditioning mechanism for autoregressive inference.

Table 1: Overall performance comparison on ligand-binding protein generation. DD denotes DiffDock confidence. Best results among generative methods are in **bold**, and second-best results are underlined. DD denotes DiffDock confidence, and Poses denotes the number of poses found. Arrows indicate whether higher (↑) or lower (↓) values are better.

Method	Gen.	Folded ↑	pLDDT ↑	RMSD ↓	DD Top-1 ↑	DD Top-5 ↑	Poses ↑	Chai-1 ↑
InstructPro	100	98	0.43	11.80	-1.19	-1.81	39	0.37
PrefixProt	100	19	0.58	18.58	-1.24	-1.75	56	0.25
ProCALM	100	100	**0.64**	13.47	-1.28	-1.86	37	0.24
ProGen2	100	100	0.41	8.39	-0.97	-1.66	49	0.12
ProteinDT	100	100	0.62	14.65	-1.40	-1.88	31	0.32
RFDiffusion3	100	100	0.57	15.47	-1.22	-1.53	**77**	0.36
Native / GT	100	100	0.79	—	-0.61	-0.74	85	0.59
DockWizard	100	100	0.58	**8.24**	**-0.84**	**-1.28**	67	**0.44**

Training Objective. During training, we use teacher forcing to perform conditional autoregressive modeling of the target protein sequence. The training objective combines conditional language modeling, tri-view ligand contrastive alignment, and bridge-gate regularization:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{LM}} + \lambda_{\text{lig}}\mathcal{L}_{\text{lig-ctr}} + \lambda_{\text{gate}}\mathcal{L}_{\text{gate}}. \tag{15}$$

Here, $\mathcal{L}_{\text{LM}}$ denotes the conditional language-modeling loss on protein positions, $\mathcal{L}_{\text{lig-ctr}}$ aligns ligand text, SMILES, and ligand 3D views, and $\mathcal{L}_{\text{gate}}$ denotes the bridge-gate regularization term.The gate regularization term is optional and is disabled in our reported experiments unless otherwise specified.

Inference Protocol. At inference time, DockWizard uses the same staged-conditioning interface for autoregressive sequence generation. The encoder and routing module build the conditional prefix $\mathbf{\Pi}$ and full docking/ligand memories once, and the decoder generates protein tokens conditioned on both components. Further implementation details are provided in Appendix G.7.

4 Experiments

We evaluate DockWizard on a unified multimodal dataset through a series of experiments. Our experiments aim to answer the following four questions: Q1 (Effectiveness): Does DockWizard outperform state-of-the-art baselines in protein generation? Q2 (Interpretability): What enables DockWizard to effectively generate proteins with specific functions and ligand-binding ability? Q3 (Robustness): Can DockWizard maintain stable performance under textual-condition noise and ligand perturbations? Q4 (Efficiency): In ligand-binding protein generation, does DockWizard achieve a reasonable trade-off in runtime, resource usage, scalability, and data efficiency?

4.1 Overall Performance (Q1)

To answer Q1, we compare DockWizard with baselines under a unified evaluation protocol.

Datasets, Baselines, and Evaluation Metrics. We compare DockWizard with six adapted baselines under the same generation budget and evaluation protocol, using a 30% sequence-identity homology-aware split to reduce leakage. Baselines are adapted through their closest compatible conditioning interfaces. Details are provided in Appendices E, F, and B.1.

Experimental Results. As shown in Table 1, DockWizard achieves the strongest binding-oriented performance among generative methods, as measured by DD and Chai-1. DockWizard obtains the best DiffDock Top-1 confidence, DiffDock Top-5 mean confidence, and Chai-1 Score, while also achieving the lowest RMSD. In comparison, DockWizard has a slightly lower pLDDT than ProCALM and fewer poses found than RFD3, indicating that its advantage does not simply come from monomer-level folding confidence or the number of docking poses, but is more concentrated in ligand-binding-related quality.

Overall Analysis. On the 100 matched test conditions, DockWizard improves DD top-1 confidence by 0.13 over ProGen2 and the Chai-1 score by 0.07 over InstructPro. Both differences remain significant after Holm correction.

Recovery of binding-site features. We next ask whether the predicted complexes recover interactions observed in experimentally resolved structures. Table 2 separates recovery of pocket residues from recovery of ligand atoms and contacts. DockWizard improves contact F1 by 6.6 percentage points over InstructPro. The gain extends to interaction types and fingerprint similarity, and all five paired differences remain significant after Holm correction. The experimental structures supply the reference contacts for this comparison.

Table 2: Recovery of interactions in 94 experimentally resolved complexes. Entries are paired differences between DockWizard and InstructPro. F1 differences are in percentage points (pp).

Metric	Paired difference [95% CI]
Pocket-residue F1	+7.6 pp [2.3, 12.9]
Ligand-atom F1	+6.5 pp [1.5, 11.6]
Interaction-type F1	+7.8 pp [2.1, 13.4]
Contact F1	+6.6 pp [1.0, 12.2]
Fingerprint similarity	+0.078 [0.026, 0.130]

Sequence recovery and diversity. Because training targets the reference sequence, we checked whether the binding-score advantage accompanies higher sequence recovery. Mean identity to the reference is 0.244 and mean identity to the nearest training sequence is 0.299, neither higher than for the main baselines. With five generations per condition, 97.6% of outputs are unique. On 61 test conditions selected by reference-to-training identity of at most 30%, the DD advantage over ProGen2 is 0.12 (95% CI 0.01 to 0.23). Filtering the training set for sequence, protein-family, ligand-scaffold, literature-source, and text overlap retains 71,867 of 108,436 examples. After retraining, DockWizard retains a DD advantage of 0.10 over ProGen2 (95% CI 0.01 to 0.19). The corresponding Chai-1 and contact F1 intervals against InstructPro include zero.

Common-budget comparison. Table 1 retains each method's native candidate selection, including ProCALM reranking. We therefore repeated the comparison for ProGen2, InstructPro, RFD3, RFD3 + LigandMPNN, and DockWizard on the same 100 conditions, using five candidates per condition, three matched decoding seeds, and no post-hoc reranking. Failures count toward the means. Redesigning the RFD3 backbone with LigandMPNN improves DD/Chai-1 from -1.21/0.36 to -1.04/0.39. DockWizard reaches -0.86/0.43 under the same protocol. Appendix B.6 reports the Vina scores and valid-interaction rates.

4.2 Interpretability Study (Q2)

After validating the overall advantage of DockWizard, we further answer Q2: where does this improvement come from? Specifically, we first examine the retention of candidate sequences along the four-stage evaluation pipeline, then analyze the contribution of internal designs through module ablation, and finally use condition perturbations to test whether docking text and ligand information truly participate in the generation process rather than merely serving as formal inputs.

Stage-wise Retention. We first decompose candidate results along the four-stage evaluation pipeline to observe at which stage effective candidates are mainly lost for different methods. Fig. 5 shows that DockWizard achieves a clearly higher docking pass rate than other baselines, indicating that its advantage does not mainly come from early folding-stage filtering, but from retaining more effective candidates after entering ligand-binding-related evaluation.

Ablation Study. To disentangle the contribution of different modules to the final results, we conduct module-level ablations: (i) removing full-memory injection, which prevents the decoder from accessing the complete conditional memory during decoding; (ii) removing gated fusion, which weakens the information interaction between functional semantics and docking-ligand semantics; (iii) removing ligand-view contrastive learning, which no longer explicitly constrains the consistency of multi-view ligand representations; (iv) removing docking text, which directly tests the role of docking language; and (v) removing Perceiver routing, which no longer compresses and separates heterogeneous conditions. Fig. 3 shows that the full model achieves the best overall performance. In particular, removing full-memory injection, docking text, or Perceiver routing leads to clear degradation in docking confidence and complex stability, indicating that DockWizard benefits from the effective organization and utilization of multi-source conditional information.

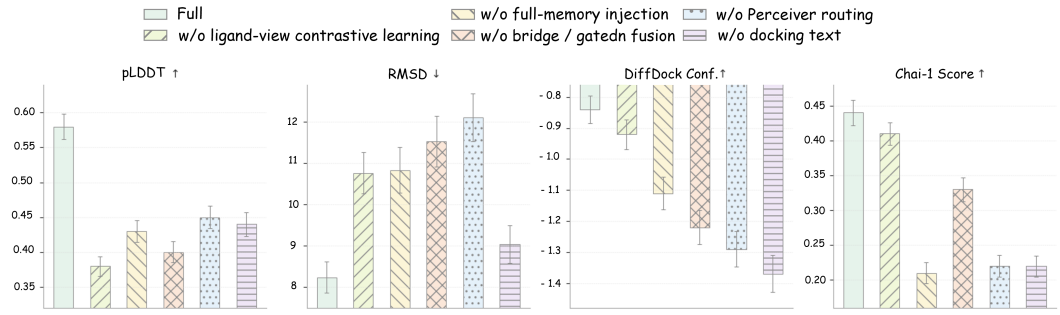


Figure 3: The comparison evaluates the effects of removing full-memory injection, gated fusion, ligand-view contrastive learning, docking text, and Perceiver routing.

Table 3: Conditional controllability analysis of DockWizard.

Condition	pLDDT ↑	RMSD ↓	DD Top-1 ↑	Chai-1 ↑
Full input	**0.58**	**8.24**	**-0.84**	**0.44**
w/o docking text	0.41	13.98	<u>-1.04</u>	0.21
Shuffled docking text	<u>0.51</u>	13.08	-1.34	0.19
Mismatched docking text	0.49	<u>11.95</u>	-1.06	0.24
Mismatched ligand	0.41	13.20	-1.50	<u>0.26</u>
Randomized SMILES	0.48	12.77	-1.30	0.21

Conditional Controllability. To determine whether conditional information truly participates in generation, we construct counterfactual inputs: (i) removing docking text; (ii) shuffling the sentence order of docking text; (iii) replacing docking text with that from another sample; (iv) replacing the ligand SMILES with that from another sample; and (v) using a randomized SMILES representation of the same molecule. Table 3 shows that when key conditions are removed, shuffled, or mismatched, ligand-binding-related metrics decrease clearly.

What does the text contribute? Replacing a description can change both the named entities and the relation between them. To separate these effects, we keep the protein and ligand names in a fixed template (identity-only), or mask entity cues while preserving relational clauses (semantics-only). A generic control matches the full description in token and clause counts. As Table 4 shows, full descriptions improve DD and Chai-1 by 0.11 and 0.09 over entity names alone (both significant after Holm correction). Masked relational descriptions also outperform generic text on all four measures. Across 42 pairs matched for protein family, ligand scaffold, and text length, changing only the text increases realization of the target interaction by 12.1 percentage points (95% CI 7.3 to 16.8).

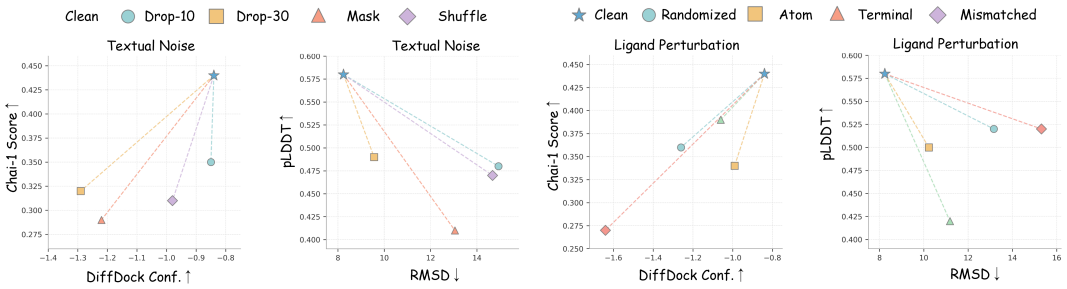
Table 4: Effect of relational text on generation. All rows use the same checkpoint, test conditions, decoding procedure, and evaluators. DD denotes DiffDock top-1 confidence.

Text condition	DD ↑	Chai-1 ↑	Vina ↓	Valid interactions ↑
Full text	-0.84	0.44	-7.62	72%
Identity-only	-0.95	0.35	-7.27	62%
Semantics-only	-0.90	0.39	-7.44	68%
Length-matched generic	-1.00	0.29	-7.16	56%
No text	-1.04	0.21	-6.99	49%
Mismatched text	-1.06	0.24	-7.07	53%

4.3 Condition Sensitivity (Q3)

After confirming that DockWizard can effectively use docking text and ligand conditions, we further answer Q3: can the model still maintain stable generation when these conditions contain noise, missing information, or perturbations? Specifically, we systematically evaluate the robustness of DockWizard to condition changes from two aspects: docking-text noise and ligand perturbation.

Textual Condition Noise. We first apply different forms of noise to docking text, including sentence deletion, keyword masking, and sentence shuffling, to evaluate whether the model remains



(a) Textual condition noise.

(b) SMILES / ligand perturbation.

Figure 4: Sensitivity of DockWizard under condition perturbations.

stable under incomplete or locally distorted docking descriptions. Fig. 4a shows that under mild textual perturbations, DockWizard still maintains ligand-binding-related performance higher than all baselines. Under stronger perturbations, it also remains better than most baselines, demonstrating its robustness to fluctuations in text quality.

Ligand Perturbation. We further perturb ligand conditions by randomizing SMILES representations, replacing local atoms, deleting terminal atoms, and mismatching ligands from other samples. Fig. 4b shows that representation-level perturbations and mild chemical perturbations lead to some performance degradation, but the model still retains part of its ligand-binding-related performance. In contrast, mismatched ligands cause simultaneous degradation in RMSD, docking confidence, and Chai-1 Score, indicating that DockWizard is highly sensitive to ligand identity changes. The perturbations differ in severity, which limits comparisons across modalities.

4.4 Efficiency Study (Q4)

Finally, we answer Q4 by systematically quantifying the computational cost of DockWizard, evaluating whether it achieves a reasonable trade-off between binding-oriented performance gains and generation efficiency. Specifically, we analyze its practical usability from two aspects: runtime and resource usage, and data efficiency.

Runtime and Resource Usage. We measure the runtime and peak GPU memory usage of DockWizard during training and inference. As shown in Table 5, DockWizard achieves competitive efficiency, with training cost comparable to other trainable baselines, faster generation than most baselines, and moderate peak memory usage.

Data Efficiency. DockWizard improves task-specific data efficiency by reusing pretrained sequence and condition representations, while training mainly the alignment, routing, prefix, and memory-injection modules. Under the local ERM assumptions in the appendix, the analysis gives an $O(1/\epsilon)$ sample requirement for these task-specific adaptation modules to reach ϵ excess risk. The detailed analysis is provided in Appendix B.5.

5 Conclusion

In this paper, we propose DockWizard, a docking-aware multimodal framework for de novo ligand-binding protein generation under functional and ligand constraints. Through multimodal condition encoding, role-specific memory routing, and staged condition injection, DockWizard alleviates the gap between docking constraints and the sequence generation process. It also reduces role confusion caused by flat fusion of heterogeneous conditions, enabling joint modeling of global functional semantics and local ligand-binding constraints. Experiments demonstrate that DockWizard achieves strong generation quality and conditional controllability in ligand-binding protein generation. Future work will explore more robust docking-language representations, finer-grained ligand interaction modeling, and integration with experimental validation workflows.

Acknowledgments and Disclosure of Funding

This work is supported by the New Generation Artificial Intelligence-National Science and Technology Major Project (2025ZD0123403), NSFC Grants 625B2021, 62427808, U24A20255, U2241211, and Chongqing Ant Consumer Finance Co., Ltd.

References

Hadi Abdine, Michail Chatzianastasis, Costas Bouyioukos, and Michalis Vazirgiannis. Prot2text: Multimodal protein's function generation with gnns and transformers. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 10757–10765, 2024.

Jared Adolf-Bryfogle, Oleks Kalyuzhniy, Michael Kubitz, Brian D Weitzner, Xiaozhen Hu, Yumiko Adachi, William R Schief, and Roland L Dunbrack Jr. Rosettaantibodydesign (rabd): A general framework for computational antibody design. *PLoS computational biology*, 14(4):e1006112, 2018.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736, 2022.

Longxing Cao, Brian Coventry, Inna Goreshnik, Buwei Huang, William Sheffler, Joon Sung Park, Kevin M Jude, Iva Marković, Rameshwar U Kadam, Koen HG Verschueren, et al. Design of protein-binding proteins from the target structure alone. *Nature*, 605(7910):551–560, 2022.

Seyone Chithrananda, Gabriel Grand, and Bharath Ramsundar. Chemberta: large-scale self-supervised pretraining for molecular property prediction. *arXiv preprint arXiv:2010.09885*, 2020.

Gene Ontology Consortium. The gene ontology resource: 20 years and still going strong. *Nucleic acids research*, 47(D1):D330–D338, 2019.

Justas Dauparas, Ivan Anishchenko, Nathaniel Bennett, Hua Bai, Robert J Ragotte, Lukas F Milles, Basile IM Wicky, Alexis Courbet, Rob J de Haas, Neville Bethel, et al. Robust deep learning–based protein sequence design using proteinmpnn. *Science*, 378(6615):49–56, 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. *Nature Methods*, 22(4):717–723, 2025.

Xiao Fei, Michail Chatzianastasis, Sarah Almeida Carneiro, Hadi Abdine, Lawrence P Petalidis, and Michalis Vazirgiannis. Prot2text-v2: Protein function prediction with multimodal contrastive alignment. *arXiv preprint arXiv:2505.11194*, 2025.

Yu Gu, Robert Tinn, Hao Cheng, Michael Lucas, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. Domain-specific language model pretraining for biomedical natural language processing. *ACM Transactions on Computing for Healthcare (HEALTH)*, 3(1): 1–23, 2021.

Xiao-Yu Guo, Yi-Fan Li, Yuan Liu, Xiaoyong Pan, and Hong-Bin Shen. Protdat: A unified framework for protein sequence design from any protein text description. *arXiv preprint arXiv:2412.04069*, 2024.

Thomas 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. *Science*, 387(6736):850–858, 2025.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR, 2019.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3, 2022.

John B Ingraham, Max Baranov, Zak Costello, Karl W Barber, Wujie Wang, Ahmed Ismail, Vincent Frappier, Dana M Lord, Christopher Ng-Thow-Hing, Erik R Van Vlack, et al. Illuminating protein space with a programmable generative model. *Nature*, 623(7989):1070–1078, 2023.

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al. Perceiver io: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795*, 2021a.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pages 4651–4664. PMLR, 2021b.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Antibody-antigen docking and design via hierarchical equivariant refinement. *arXiv preprint arXiv:2207.06616*, 2022.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in neural information processing systems*, 34:1022–1035, 2021.

Xiangzhe Kong, Wenbing Huang, and Yang Liu. End-to-end full-atom antibody design. *arXiv preprint arXiv:2302.00203*, 2023.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pages 3045–3059, 2021.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pages 19730–19742. PMLR, 2023.

Mingchen Li, Yang Tan, Xinzhu Ma, Bozitao Zhong, Huiqun Yu, Ziyi Zhou, Wanli Ouyang, Bingxin Zhou, Pan Tan, and Liang Hong. Prosst: Protein language modeling with quantized structure and disentangled attention. *Advances in Neural Information Processing Systems*, 37:35700–35726, 2024.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, 2021.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965, 2022a.

Nuowei Liu, Jiahao Kuang, Yanting Liu, Tao Ji, Changzhi Sun, Man Lan, and Yuanbin Wu. Protein design with dynamic protein vocabulary. *arXiv preprint arXiv:2505.18966*, 2025a.

Shengchao Liu, Yanjing Li, Zhuoxinran Li, Anthony Gitter, Yutao Zhu, Jiarui Lu, Zhao Xu, Weili Nie, Arvind Ramanathan, Chaowei Xiao, et al. A text-guided protein design framework. *Nature Machine Intelligence*, 7(4):580–591, 2025b.

Xiangyu Liu, Yi Liu, Silei Chen, and Wei Hu. Controllable protein sequence generation with llm preference optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 505–513, 2025c.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 61–68, 2022b.

Yi Liu and Brian Kuhlman. Rosettadesign server for protein design. *Nucleic acids research*, 34 (suppl_2):W235–W238, 2006.

Lei Lu, Xuxu Gou, Sophia K Tan, Samuel I Mann, Hyunjun Yang, Xiaofang Zhong, Dimitrios Gazgalis, Jesús Valdiviezo, Hyunil Jo, Yibing Wu, et al. De novo design of drug-binding proteins with predictable binding energy and specificity. *Science*, 384(6691):106–112, 2024.

Jiawei Luo, Jiahao Li, Xianliang Liu, Yang Zhang, Qingcai Chen, and Junjie Chen. Controllable protein design by prefix-tuning protein language models. *Journal of Chemical Information and Modeling*, 2024.

Shitong Luo, Yufeng Su, Xingang Peng, Sheng Wang, Jian Peng, and Jianzhu Ma. Antigen-specific antibody design and optimization with diffusion-based generative models for protein structures. *Advances in Neural Information Processing Systems*, 35:9754–9767, 2022.

Sergey Lyskov and Jeffrey J Gray. The rosettadock server for local protein–protein docking. *Nucleic acids research*, 36(suppl_2):W233–W238, 2008.

Ali Madani, Bryan McCann, Nikhil Naik, Nitish Shirish Keskar, Namrata Anand, Raphael R Eguchi, Po-Ssu Huang, and Richard Socher. Progen: Language modeling for protein generation. *arXiv preprint arXiv:2004.03497*, 2020.

Ali Madani, Ben Krause, Eric R Greene, Subu Subramanian, Benjamin P Mohr, James M Holton, Jose Luis Olmos Jr, Caiming Xiong, Zachary Z Sun, Richard Socher, et al. Large language models generate functional protein sequences across diverse families. *Nature biotechnology*, 41 (8):1099–1106, 2023.

Erik Nijkamp, Jeffrey A Ruffolo, Eli N Weinstein, Nikhil Naik, and Ali Madani. Progen2: exploring the boundaries of protein language models. *Cell systems*, 14(11):968–978, 2023.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. Adapterfusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th conference of the European chapter of the association for computational linguistics: main volume*, pages 487–503, 2021.

Alfredo Quijano-Rubio, Hsien-Wei Yeh, Jooyoung Park, Hansol Lee, Robert A Langan, Scott E Boyken, Marc J Lajoie, Longxing Cao, Cameron M Chow, Marcos C Miranda, et al. De novo design of modular and tunable protein biosensors. *Nature*, 591(7850):482–487, 2021.

Vıctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E (n) equivariant graph neural networks. In *International conference on machine learning*, pages 9323–9332. PMLR, 2021.

Zhenqiao Song, Ramith Hettiarachchi, Chuan Li, Jianwen Xie, and Lei Li. Instructpro: Natural language guided ligand-binding protein design. *arXiv preprint arXiv:2506.09332*, 2025a.

Zhenqiao Song, Ramith Hettiarachchi, Chuan Li, Jianwen Xie, and Lei Li. Natural language guided ligand-binding protein design. *arXiv e-prints*, pages arXiv–2506, 2025b.

Susana Vázquez Torres, Philip JY Leung, Preetham Venkatesh, Isaac D Lutz, Fabian Hink, Huu-Hien Huynh, Jessica Becker, Andy Hsien-Wei Yeh, David Juergens, Nathaniel R Bennett, et al. De novo design of high-affinity binders of bioactive helical peptides. *Nature*, 626(7998):435–442, 2024.

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 Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.

K Weissenow, J Gomez Sanchez, A Henkel, M Mirdita, M Steinegger, and R Prostt. Bilingual language model for protein sequence and structure. *bioRxiv*, 2024.

Kevin E Wu, Howard Chang, and James Zou. Proteinclip: enhancing protein language models with natural language. *bioRxiv*, pages 2024–05, 2024.

Minghao Xu, Xinyu Yuan, Santiago Miret, and Jian Tang. Protst: Multi-modality learning of protein sequences and biomedical texts. In *International conference on machine learning*, pages 38749–38767. PMLR, 2023.

Jason Yang, Aadyot Bhatnagar, Jeffrey A Ruffolo, and Ali Madani. Function-guided conditional generation using protein language models with adapters. *arXiv preprint arXiv:2410.03634*, 2024.

Chaohao Yuan, Songyou Li, Geyan Ye, Yikun Zhang, Long-Kai Huang, Wenbing Huang, Wei Liu, Jianhua Yao, and Yu Rong. Annotation-guided protein design with multi-level domain alignment. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pages 1855–1866, 2025.

Ningyu Zhang, Zhen Bi, Xiaozhuan Liang, Siyuan Cheng, Haosen Hong, Shumin Deng, Jiazhang Lian, Qiang Zhang, and Huajun Chen. Ontoprotein: Protein pretraining with gene ontology embedding. *arXiv preprint arXiv:2201.11147*, 2022.

Hanjing Zhou, Mingze Yin, Wei Wu, Mingyang Li, Kun Fu, Jintai Chen, Jian Wu, and Zheng Wang. Protclip: Function-informed protein multi-modal learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 22937–22945, 2025.

Xiangxin Zhou, Dongyu Xue, Ruizhe Chen, Zaixiang Zheng, Liang Wang, and Quanquan Gu. Antigen-specific antibody design via direct energy-based preference optimization. *Advances in Neural Information Processing Systems*, 37:120861–120891, 2024.

Appendix

A Additional Related Work

A.1 Docking-guided and binding-oriented protein design

Protein design with target-structure or ligand information has long relied on structural modeling, energy optimization, and interface search. Early docking-guided design methods were typically built around physical energy functions and structural sampling, jointly optimizing sequence, conformation, and binding interfaces through frameworks such as RosettaDock, RosettaDesign, and RosettaAntibodyDesign [Lyskov and Gray, 2008, Liu and Kuhlman, 2006, Adolf-Bryfogle et al., 2018]. These methods are physically interpretable and can explicitly represent interface complementarity, hydrogen bonding, hydrophobic interactions, electrostatics, and steric exclusion. They have therefore been influential in antibody design, enzyme engineering, and protein-complex modeling. However, their performance often depends on structural templates, hand-designed energy terms, predefined binding regions, or post-processing search procedures, leaving a partial decoupling between the generative model itself and the ligand-binding objective.

With the development of deep learning methods, structure-conditioned protein design has gradually shifted from explicit energy search toward data-driven generation. ProteinMPNN showed that neural networks can generate structure-compatible protein sequences from given three-dimensional backbones [Dauparas et al., 2022]. LigandMPNN further incorporates small molecules, nucleic acids, metal ions, and other non-protein components as atom-level context, allowing sequence design to sense the local chemical environment around ligands [Dauparas et al., 2025]. RFdiffusion introduced diffusion models into *de novo* protein design and has been widely used for binder design, motif scaffolding, and structure-constrained generation [Watson et al., 2023]. Chroma formulates protein generation as structure and sequence sampling under programmable constraints, allowing geometric, structural, and functional constraints to jointly participate in generation [Ingraham et al., 2023]. These methods substantially improve controllability at the structural level, but their conditions are usually expressed as three-dimensional structures, geometric constraints, or programmable objectives, leaving limited modeling of docking processes and ligand-binding interaction patterns in natural language form.

In antibody and binding-protein design, DiffAb, HSRN, dyMEAN, and ABDPO further introduce binding-interface information into generation or optimization. DiffAb jointly generates antibody CDR sequences and structures with a diffusion model and explicitly conditions on antigen structures [Luo et al., 2022]. HSRN emphasizes the coupling between paratope folding and paratope–epitope docking, allowing local binding conformations to be progressively refined during generation [Jin et al., 2022]. dyMEAN builds an information bridge between antibody and epitope through a shadow paratope and performs full-atom docking through iterative updates [Kong et al., 2023]. ABDPO uses residue-level energy preference optimization to tune conditional diffusion models so that generated results better match binding-energy preferences [Zhou et al., 2024]. These studies indicate that docking and binding information is important for protein generation.

However, most existing binding-oriented protein design methods still depend on explicit structural inputs, epitope locations, antibody frameworks, residue-level energy terms, or geometric priors. Compared with structural or energy signals, docking-oriented natural language can describe binding pockets, interaction patterns, ligand recognition processes, and local constraints in a more flexible form, but such information has not yet been fully used as a generation-time condition. For ligand-binding protein generation, the central issue is not only how to introduce binding signals, but also how to organize global functional semantics and local ligand-binding constraints during generation.

A.2 Text-guided protein generation

Natural language provides a more open conditional representation for protein generation than discrete labels. Early controllable protein language models typically relied on family labels, species labels, functional labels, or control tokens to guide generation, allowing models to generate sequences with particular backgrounds or functional tendencies within predefined categories [Madani et al., 2020, 2023, Luo et al., 2024]. These methods expanded the controllability of protein language

models, but their condition spaces are limited by label sets and are less able to express open-ended functional descriptions, complex design intents, or finer-grained biological constraints.

Cross-modal alignment between protein sequences and natural language has further advanced text-guided protein modeling. ProteinDT aligns protein sequences and textual descriptions in a shared representation space, allowing text to serve as a semantic condition for protein generation and editing [Liu et al., 2025b]. ProtST constructs a multimodal pretraining framework for protein sequences and biomedical text, enhancing protein language models with textual descriptions of protein functions and properties [Xu et al., 2023]. OntoProtein introduces Gene Ontology from the perspective of external knowledge and jointly incorporates structured functional knowledge and protein sequences into pretraining [Zhang et al., 2022, Consortium, 2019]. ProteinCLIP and ProtCLIP further use contrastive learning to establish semantic correspondence between protein sequences and functional text, enabling models to learn protein representations more oriented toward functional interpretation [Wu et al., 2024, Zhou et al., 2025]. These works show that natural language and structured functional knowledge can provide richer semantic supervision for protein representation learning than sequence modeling alone.

Corresponding to text-to-protein generation, protein-to-text generation also indicates an emerging trend toward bidirectional modeling between proteins and natural language. Prot2Text reformulates protein function prediction from conventional label classification into free-text generation, producing protein functional descriptions by combining protein structure, sequence representations, and language models [Abdine et al., 2024]. Prot2Text-V2 further uses a protein language model as the sequence encoder and connects it to a large language model through a lightweight projector, enabling free-form functional descriptions to be generated from amino-acid sequences [Fei et al., 2025]. Although these methods do not directly perform text-to-protein generation, they indicate that natural language can carry more detailed and open-ended protein function information, rather than only serving as a substitute for discrete annotation labels.

In generation tasks, PAAG, ProtDAT, ProDVa, ProCALM, and CtrlProt further explore how natural language or functional annotations can enter the protein design process. PAAG uses multi-level domain annotations to guide protein generation [Yuan et al., 2025]. ProtDAT strengthens the interaction between textual descriptions and protein sequences through multimodal cross-attention [Guo et al., 2024]. ProDVa retrieves natural protein fragments or constructs dynamic protein vocabularies from textual functional descriptions to improve the foldability and functional consistency of generated proteins [Liu et al., 2025a]. ProCALM introduces conditional adapters into pretrained protein language models, enabling them to accept enzyme family, taxonomy, and natural-language functional descriptions [Yang et al., 2024]. CtrlProt uses preference optimization to improve multi-attribute controllability [Liu et al., 2025c]. These methods move natural language from external annotation toward a conditioning signal inside the generation process.

Although natural language substantially improves the semantic expressiveness of protein generation, existing text-guided protein generation still mainly focuses on function-level semantics, such as protein function, property, family, domain, catalytic type, or database annotation. For ligand-binding protein generation, a global functional description alone is usually insufficient to specify the full design objective. Ligand binding depends not only on the overall function of a protein, but also on binding-pocket shape, local residue arrangement, hydrogen-bond networks, hydrophobic contacts, aromatic stacking, electrostatic complementarity, and spatial constraints implied by the docking pose. Docking-oriented text can therefore serve as an important complement to function text, extending natural-language conditions from global functional descriptions to local binding-process descriptions.

A.3 Multimodal controls for protein design

Conditional information in protein design is expanding from a single modality to multiple modalities. Traditional protein generation methods usually take sequences, structures, or discrete labels as their main conditions, whereas recent models begin to integrate multiple sources of information, including sequence, structure, function, annotation, ligand, and text. ESM3 represents protein sequence, structure, and function as unified token tracks and jointly models different levels of protein information within a single generative model [Hayes et al., 2025]. ProSST and ProstT5 further explore joint modeling between sequence and structure, allowing structural information to enter protein language models through discrete tokens, translation tasks, or shared representations [Li et al.,

2024, Weissenow et al., 2024]. These advances suggest that multimodal conditions can improve the representational capacity and task adaptability of protein generation models.

In ligand-binding design, multimodal conditions are particularly necessary. Ligand binding is determined not only by protein sequence, but also by ligand chemical identity, three-dimensional shape, conformational state, and local interaction patterns. LigandMPNN uses ligand atoms, nucleic acids, metal ions, and other non-protein components as contextual inputs, allowing the model to sense local chemical environments during sequence design [Dauparas et al., 2025]. Structural generative methods such as RFdiffusion and Chroma start from structural motifs, geometric constraints, or programmable conditions, enabling protein generation to be constrained by target structures and functional goals [Watson et al., 2023, Ingraham et al., 2023]. Different modalities cover different aspects of ligand-binding design, so a single conditional form is often insufficient to express the complete design objective.

For ligand-binding protein generation, different conditional modalities have clear but distinct roles. Function text mainly provides the global functional intent and determines the biological task and functional direction that the generated protein should serve. Docking text describes local binding patterns and interaction relationships, providing language-based constraints close to the binding process. Ligand descriptions provide the natural-language identity and functional background of ligands. SMILES encodes ligand chemical topology [Weininger, 1988]. Ligand three-dimensional representation provides spatial shape, geometric distribution, and potential conformational information. These modalities are not redundant inputs, but complementary descriptions of the same ligand-binding design objective from semantic, chemical, and spatial levels.

Therefore, the key question for multimodal protein generation is not only how to introduce more conditions, but also how to organize them. If function text, docking text, ligand description, SMILES, and ligand three-dimensional representation are simply concatenated into a single input, the model may struggle to distinguish the scopes of different modalities. Global functional semantics should mainly affect the overall sequence distribution and function-related motifs, whereas local docking and ligand conditions should more directly constrain binding-related residues and their interaction patterns. An effective multimodal condition-organization mechanism needs to preserve heterogeneous information in a unified representation space while avoiding interference among conditions with different roles during decoding.

A.4 Condition organization and role-specific conditioning

Condition-organization mechanisms determine how generative models receive, preserve, and use external information. Prefix-Tuning, Prompt Tuning, and P-Tuning v2 implement efficient conditional control through continuous prompt or prefix parameters, often with frozen or partially frozen pretrained language models [Li and Liang, 2021, Lester et al., 2021, Liu et al., 2022b]. Parameter-efficient fine-tuning methods such as adapters, LoRA, IA3, AdapterFusion, and Compacter use lightweight inserted modules, low-rank updates, or combinations of multiple adapters to adapt large pretrained models to new tasks at relatively low training cost [Houlsby et al., 2019, Hu et al., 2022, Liu et al., 2022a, Pfeiffer et al., 2021, Karimi Mahabadi et al., 2021]. These methods are parameter-efficient, inexpensive to transfer, and simple to implement, making them suitable for conditional adaptation on large pretrained models.

However, when input conditions come from multiple modalities with different semantic roles, single-path flat conditioning can compress heterogeneous information into the same control stream and weaken structural differences among conditions. For ligand-binding protein generation, function text, docking text, SMILES, and ligand three-dimensional representation do not have the same scope of action. If these conditions are compressed into a single control vector, the model may struggle to distinguish which information should dominate global generation and which information should constrain local binding-related decisions.

In complex multimodal settings, structured condition-organization methods can better preserve multi-source information. Perceiver and Perceiver IO process inputs of different lengths and types through latent bottlenecks, allowing heterogeneous information to be aggregated in a unified latent space [Jaegle et al., 2021b,a]. Flamingo uses gated cross-attention to connect visual information to a language model, allowing external modalities to influence output layer by layer during language generation [Alayrac et al., 2022]. BLIP-2 uses a querying transformer to connect a frozen visual en-

coder and a language model, providing a parameter-efficient approach to cross-modal bridging [Li et al., 2023]. These methods show that for tasks with heterogeneous inputs, inconsistent modality lengths, and different information densities, structured condition access is often more favorable for information preservation and cross-modal interaction than simple concatenation.

Ligand-binding protein generation imposes finer-grained requirements on condition organization. Function text is better suited as a global semantic condition that affects overall functional direction and sequence distribution. Docking text is closer to local interaction description, emphasizing binding pockets, residue contacts, and ligand-recognition patterns. SMILES and ligand three-dimensional representation respectively provide chemical topology and spatial geometric constraints [Weininger, 1988]. The core idea of role-specific conditioning is to organize heterogeneous conditions into memories with explicit semantic division of labor, rather than compressing all modalities into the same condition stream. By distinguishing global functional guidance from local docking constraints, a generative model can preserve functional consistency while strengthening the effect of ligand-binding information on decoding.

B More Experiments

B.1 Evaluation protocol and metric definitions

This section describes the evaluation protocol and metric definitions used in our experiments. Ligand-binding-oriented protein generation requires generated sequences not only to have basic foldability, but also to approach the target protein at the structural level and further form a reasonable docking pose and stable complex with the given ligand. We therefore adopt a stage-wise evaluation protocol. The overall pipeline first generates candidate protein sequences, then predicts their structures, compares them against the GT structure, evaluates protein–ligand docking quality with DiffDock, and finally evaluates the protein–ligand complex with Chai-1.

Generated. Generated denotes the number of candidate proteins generated by each method on the test set. We mainly use 100 test samples for unified evaluation, so this metric is used to ensure that different methods are compared under the same generation budget. Generated does not itself indicate generation quality; it is the starting point for subsequent folding, structure comparison, docking, and complex evaluation.

Fold Pass. Fold Pass denotes the proportion or number of samples that pass the folding-quality filter. We use pLDDT > 0.5 as the fold-pass threshold. This metric measures whether a generated sequence can at least form a predicted structure with basic confidence. A higher Fold Pass means that more candidate sequences enter subsequent structural evaluation, but it does not directly indicate ligand-binding performance.

pLDDT. pLDDT is the average confidence of the predicted monomer structure and is used to measure whether a generated sequence can form a credible folded structure. A higher pLDDT usually indicates that the structure predictor is more confident about the local conformation of the generated protein. However, pLDDT mainly reflects monomer-level folding quality and cannot by itself show whether the generated protein can effectively bind the given ligand.

RMSD. RMSD denotes the root-mean-square deviation between the predicted structure of the generated protein and the GT structure. It measures structural proximity, and lower values are better in our evaluation. Unlike pLDDT, RMSD focuses on geometric deviation from the reference structure and therefore reflects whether a model approaches the target protein or target conformation at the structural level.

DiffDock Top-1 Confidence. DiffDock Top-1 Confidence denotes the confidence score of the highest-confidence docking pose produced by DiffDock. Since this metric is usually negative in our evaluation, larger values closer to zero indicate better docking confidence. This metric measures whether the generated protein and target ligand can form a high-confidence top-ranked docking pose.

DiffDock Top-5 Mean Confidence. DiffDock Top-5 Mean Confidence denotes the average confidence score of the top five candidate docking poses produced by DiffDock. Compared with consid-

ering only the Top-1 pose, this metric better reflects the stability of docking results. A higher Top-5 mean confidence indicates that the model does not merely produce one favorable pose by chance, but maintains relatively stable docking quality across multiple candidate poses.

Poses Found. Poses Found denotes the number of samples, among the 100 test samples, for which DiffDock successfully finds a valid pose. This metric reflects whether the docking pipeline can identify usable binding conformations for generated proteins and given ligands. A higher Poses Found suggests that generated proteins are more easily recognized by the docking tool as having feasible binding conformations, but it should still be interpreted together with DiffDock confidence and Chai-1 Score.

Docking Pass. Docking Pass denotes the proportion of samples that pass the docking-confidence threshold. We use DiffDock Top-1 Confidence > -1.0 as the docking-pass criterion. This metric is used in stage-wise retention analysis to measure how many candidate proteins are retained at the docking stage. Docking Pass more directly reflects whether generated results satisfy local binding requirements in ligand-binding-oriented evaluation.

Chai-1 Score. Chai-1 Score uses the mean-best aggregate score from Chai-1 to evaluate protein–ligand complex-level structural stability and complex confidence. Unlike pLDDT and RMSD, Chai-1 Score focuses on the credibility of the protein–ligand complex as a whole, and therefore complements DiffDock pose-level evaluation.

These metrics cover four complementary levels. pLDDT and Fold Pass reflect the basic foldability of generated sequences, RMSD reflects structural proximity, DiffDock confidence and Poses Found reflect ligand docking quality, and Chai-1 Score provides protein–ligand complex-level evaluation.

It is important to distinguish folding quality from ligand-binding quality. A method may generate sequences with high pLDDT, but these sequences may not form reasonable docking poses with the given ligand. Conversely, a ligand-binding-oriented design method should improve docking- and complex-related metrics while maintaining basic foldability. We therefore jointly evaluate folding, structure comparison, docking, and complex quality as a protocol more aligned with ligand-binding-oriented protein generation.

B.2 Detailed explanation of overall performance

The main table in the paper compares DockWizardEnd-to-End with multiple baselines under a unified evaluation protocol. This table should not be read as a ranking by a single metric, but as a joint comparison across multiple evaluation stages. Different metrics correspond to different failure modes in the generation task: low pLDDT usually means that generated sequences are difficult to fold into credible structures; high RMSD means that predicted structures deviate substantially from the GT structure; poor DiffDock confidence or fewer Poses Found means that generated proteins have difficulty forming valid docking poses with target ligands; and low Chai-1 Score indicates insufficient complex-level stability or confidence.

Overall, some baselines show advantages on individual metrics, but these advantages do not necessarily translate into ligand-binding-oriented performance. For example, ProCALM has a relatively high average pLDDT, suggesting that it benefits from the sequence prior of a pretrained protein language model and can generate sequences that are easier to fold. However, its docking confidence and Chai-1 Score are weaker, indicating that foldability does not automatically guarantee ligand-binding ability. RFD3 performs well on Poses Found, suggesting that its structure generation or docking pipeline more easily finds candidate poses, but its RMSD and Chai-1 Score still limit the final complex quality. ProteinDT and InstructPro can use textual conditions for a degree of semantic control, but they lack an explicit organization mechanism for docking text and ligand representations, leaving gaps on ligand-binding-related metrics.

DockWizard's advantage mainly appears in more balanced multi-stage performance. It does not only pursue the highest folding confidence. Instead, through multimodal condition encoding, role-specific memory routing, and staged condition injection, it allows function text, docking text, and ligand information to play different roles during generation. As a result, DockWizard performs better on metrics more closely related to structural proximity, ligand docking, and complex stability, including RMSD, DiffDock Top-1 Confidence, and Chai-1 Score. This suggests that DockWizard's

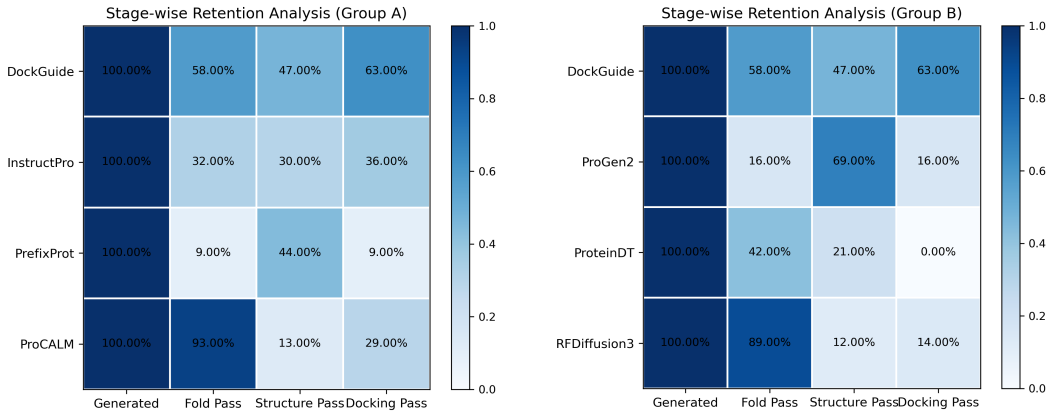


Figure 5: Stage-wise retention heatmaps under the unified evaluation protocol.

improvement is closer to the core objective of ligand-binding design, rather than merely improving the foldability of general protein sequence generation.

The GT row provides a natural reference rather than a trainable baseline. It helps readers understand the remaining gap between automatically generated results and true protein–ligand pairs. DockWizard's improvement over the baselines indicates that explicitly introducing docking-oriented language and ligand multimodal constraints can reduce this gap, but computational evaluation cannot replace subsequent experimental validation.

B.3 Stage-wise retention analysis

To further analyze how different methods are retained along the full evaluation pipeline, we perform stage-wise retention analysis. This analysis uses the same generated results as the overall performance evaluation and does not regenerate samples. Instead, it progressively counts the retention ratio of each method at each stage of the unified evaluation protocol. Specifically, Generated denotes the initial generated-sample ratio for each method and is set to 1.00. Fold Pass uses pLDDT > 0.5 as the criterion. Structure Pass uses RMSD < 10 as the criterion. Docking Pass uses DiffDock Top-1 Confidence > -1.0 as the criterion.

For clearer comparison, we divide the methods into two groups and visualize the stage-wise retention ratios as two 4×4 heatmaps in Fig. 5. In both heatmaps, columns denote evaluation stages, rows denote methods, and color intensity denotes retention ratios. DockWizard is included in both heatmaps as a common reference.

These results more clearly explain the failure modes of different methods. ProCALM and RFD3 have high retention rates on Fold Pass, indicating that they generate many foldable candidates, but both drop substantially at the Structure Pass or Docking Pass stages. This suggests that their foldability advantage does not transfer stably to later stages of ligand-binding-oriented evaluation. ProGen2 has a relatively high Structure Pass, indicating that some generated structures are close to the GT structure, but its Docking Pass is not strong, suggesting that structural proximity does not necessarily lead to better ligand docking. ProteinDT nearly fails on Docking Pass, indicating that relying only on general textual conditions is insufficient to ensure effective use of ligand-binding constraints.

DockWizard does not have the highest Fold Pass among all methods, but it maintains a more balanced retention pattern across Structure Pass and Docking Pass, and is especially stronger than other methods on Docking Pass. This indicates that DockWizard's advantage is not simply generating more foldable proteins. Rather, it retains more effective candidates at the structure-comparison and ligand-docking stages, which are closer to the task objective. This result supports the core view of the paper: protein generation for ligand-binding design should jointly consider global functional semantics and local docking/ligand constraints, instead of optimizing only monomer folding quality.

Table 5: Training time, generation time, and peak GPU memory for different methods. Lower values are better.

Method / Variant	Training Time ↓	Generation Time ↓	Peak GPU Memory ↓
ProGen2	4.7 h	0.8 h	18.5 GB
ProteinDT	6.5 h	1.1 h	21.0 GB
PrefixProt	2.8 h	1.6 h	19.4 GB
ProCALM	5.2 h	1.4 h	24.6 GB
InstructPro	4.6 h	1.3 h	23.2 GB
RFD3	0 h	2.6 h	31.8 GB
DockWizard	5.1 h	1.0 h	19.7 GB

B.4 Efficiency and resource usage

We further compare the resource cost of different methods in terms of training time, generation time, and peak GPU memory. Here, Generation Time only counts the runtime cost of generating candidate proteins with the model. It does not include the runtime of downstream evaluation tools such as ESMFold, DiffDock, and Chai-1. This separation distinguishes the operating efficiency of the generation model itself from the computational cost of the external evaluation pipeline, and avoids attributing evaluation-tool cost to model inference.

In terms of training time, DockWizard has the same order of cost as the main trainable baselines. Its training time is lower than ProteinDT and ProCALM. Compared with InstructPro and ProGen2, DockWizard requires slightly more training time, but remains within an acceptable range. This indicates that introducing multimodal condition encoding, role-specific routing, and staged condition injection does not cause a substantial increase in training cost. The reason is that DockWizard does not learn the full protein generation distribution from scratch. Instead, it builds on a pretrained protein language model and condition encoders, and mainly trains modules related to condition alignment, routing, and injection.

In terms of generation time, DockWizard takes 1.0 h, which is lower than ProteinDT, PrefixProt, ProCALM, InstructPro, and RFD3, and only slightly higher than ProGen2. Considering that DockWizard simultaneously processes function text, docking text, and ligand-related conditions during generation, this result suggests that the additional inference cost introduced by its condition-organization mechanism is controlled. Compared with structural generation methods such as RFD3, DockWizard has substantially lower generation cost, suggesting that sequence generation guided by multimodal conditions can have an efficiency advantage.

In terms of peak memory, DockWizard uses 19.7 GB, close to ProGen2 and PrefixProt, and lower than ProteinDT, ProCALM, InstructPro, and RFD3. This shows that although DockWizard introduces role-specific memories and staged condition injection, it does not substantially increase memory usage. Compared with directly expanding large model parameters or performing structure-level generation, DockWizard uses relatively compact condition-organization modules to support ligand-binding-oriented controllability, giving a reasonable balance between resource use and conditional modeling capacity.

B.5 Complexity analysis

This section provides additional analysis of DockWizard's data efficiency and inference complexity. Overall, DockWizard's computational cost mainly comes from three components: multimodal condition encoding, role-specific memory routing, and prefix guidance with full-memory injection during decoding. Compared with methods that directly generate sequences using only a protein language model, DockWizard introduces additional condition-organization cost. Compared with structural diffusion or structural sampling methods, however, DockWizard still follows a computational path centered on autoregressive sequence generation. Its overall complexity is therefore mainly controlled by the generation length, the number of decoder layers, the memory length, and the hidden dimension.

DockWizard's data efficiency comes from reusing pretrained representations and decomposing the task-specific adaptation modules. Specifically, the protein decoder backbone uses pretrained Pro-

Gen2, and the text encoder and SMILES encoder on the condition side are also provided by pretrained models. The training stage mainly updates condition-alignment projectors, query-pooling modules, ligand-view fusion and contrastive heads, Perceiver-style routing modules, prefix projections, boundary embeddings, and full-memory injection layers. This design allows task supervision to focus on learning how multimodal conditions are aligned, routed, and injected into the generation process, rather than requiring the model to relearn the full protein sequence distribution from limited task data.

Sample complexity of task-specific adaptation. Assume that the pretrained encoder and decoder representations are fixed, the task-specific features are bounded, and the empirical objective with respect to trainable adaptation parameters is locally strongly convex and smooth near the population minimizer. Let the effective dimension of the trainable adaptation class be d_{eff}. Standard regularized empirical risk minimization gives

$$\mathcal{E}(\hat{\theta}) - \mathcal{E}(\theta^\star) = O\left(\frac{d_{\text{eff}}}{n}\right). \tag{16}$$

Consequently, the number of samples required for the task-specific adaptation modules to achieve excess risk at most ϵ is

$$n = O\left(\frac{d_{\text{eff}}}{\epsilon}\right). \tag{17}$$

When d_{eff} is treated as fixed, this sample requirement simplifies to $O(1/\epsilon)$.

With fixed pretrained representations, the trainable part of DockWizard can be viewed as a task-specific empirical risk minimization problem with finite effective dimension. Since the input features are bounded, gradients and Hessians can be controlled in a local neighborhood. Local strong convexity further converts parameter-estimation error into an excess-risk bound. By standard stability or uniform-convergence analyses for regularized strongly convex ERM, the estimation error decreases at rate $O(d_{\text{eff}}/n)$, as in (16). Setting $d_{\text{eff}}/n \le \epsilon$ gives (17). When the effective dimension is treated as fixed, the result can be written as $O(1/\epsilon)$.

The proposition indicates that DockWizard's data efficiency mainly comes from the decomposition into pretrained representations and task-specific adaptation modules. Compared with methods that require large updates to the main generative model, DockWizard concentrates training in a smaller condition-adaptation space, thereby reducing the effective complexity of task-specific learning. This statement is not an unconditional global complexity guarantee for all modules. It only explains that, under fixed pretrained representations and finite effective dimension, the task-specific adaptation part admits a standard sample-complexity interpretation.

We next analyze inference complexity. Let T be the generated sequence length, L the number of decoder layers, d the hidden dimension, M_p the prefix-memory length, M_f the full-memory length, and M_c the condition-token length. DockWizard inference first encodes function text, docking text, and ligand representations into condition tokens. It then organizes these conditions into function memory, light docking-ligand memory, and full docking-ligand memory through Perceiver-style routing. Finally, it performs autoregressive decoding under prefix guidance and full-memory injection.

Multimodal condition processing is a one-time cost. It mainly includes the forward computation of text, SMILES, and ligand three-dimensional encoders, as well as the computation of alignment projectors, query pooling, ligand-view fusion, and routing modules. We denote the total cost of this part as

$$C_{\text{cond}}(M_c, d). \tag{18}$$

This cost mainly depends on condition length, the number of modalities, and hidden dimension, and is not repeated at every decoding step. It is the main upfront cost added by DockWizard compared with a standard protein language model baseline.

Under no-KV-cache generation, the t-th generation step recomputes the prefix and the generated sequence so far. The main cost of decoder self-attention is therefore

$$\sum_{t=1}^{T} O\left(L(M_p + t)^2 d + L(M_p + t)d^2\right). \tag{19}$$

In addition, full-memory injection interacts the generated positions with M_f full-memory tokens in each decoder layer. Its main additional cost is

$$\sum_{t=1}^{T} O\left(LtM_f d + L(t + M_f)d^2\right). \tag{20}$$

Thus, DockWizard's total inference complexity can be summarized as

$$\begin{aligned} C_{\mathrm{DockWizard}} =& C_{\mathrm{cond}}(M_c, d) + \sum_{t=1}^{T} O\left(L(M_p + t)^2 d + L(M_p + t)d^2\right) \\ &+ \sum_{t=1}^{T} O\left(LtM_f d + L(t + M_f)d^2\right). \end{aligned} \tag{21}$$

In (21), the first term comes from multimodal condition encoding and routing, the second term comes from autoregressive decoding with prefix tokens, and the third term comes from full-memory injection. Compared with standard autoregressive decoding, DockWizard's additional cost mainly comes from the longer context induced by prefix memory and the memory cross-attention introduced by full-memory injection. This additional cost keeps function-level guidance and docking/ligand-related constraints active during generation.

The sources of complexity differ substantially across baselines. ProGen2 mainly relies on a pre-trained protein language model for autoregressive sequence generation, with a direct inference path whose main cost comes from the decoder itself. It therefore has high generation efficiency, but limited conditional controllability because it lacks explicit organization of docking text and ligand representations. PrefixProt adds prefix control to a generation path similar to ProGen2, with extra cost mainly from the longer context caused by prefix tokens. This approach is parameter-efficient, but different conditions are often compressed into a single prefix stream, making it difficult to distinguish global functional semantics from local ligand-binding constraints.

ProteinDT's complexity mainly comes from protein–text alignment and text-guided generation. It needs to process textual conditions and map textual semantics into the protein generation space, so it introduces text encoding or cross-modal alignment cost compared with a pure protein language model. However, its conditions are mainly function-level text and do not explicitly model docking processes or ligand structural information. Its complexity is therefore concentrated in text–protein semantic alignment rather than multimodal ligand-binding constraint organization.

ProCALM and InstructPro are stronger conditional generation baselines. ProCALM introduces functional, textual, or ligand-related conditions through conditional adapters, so its complexity mainly comes from adapter-based condition injection and protein language model decoding. InstructPro relies on instruction-formatted input and compresses multiple conditions into a unified instruction input, so its complexity mainly comes from instruction processing and subsequent conditional generation. Compared with simple protein language model baselines, both methods have stronger conditional expressiveness, but their condition organization remains closer to flat conditioning and usually does not explicitly distinguish function-level guidance from docking/ligand-level constraints.

RFDiffusion3 has a different source of complexity from the sequence-generation baselines above. It is a structure-generation or structure-constrained generation method, and inference usually involves structure-level sampling, iterative updates, or diffusion-style generation. Its computational cost therefore depends more on structure length, sampling steps, and three-dimensional geometric representations. Compared with autoregressive sequence generation methods, structural generation methods often have higher generation time and peak memory, but they can model structural constraints more directly. This explains why RFDiffusion3 has advantages on some structure- or pose-related metrics, while usually requiring more resources.

Overall, DockWizard lies between simple sequence-generation baselines and structural generation baselines. It introduces more condition-processing cost than methods such as ProGen2 and PrefixProt, but this cost is mainly concentrated in compact routing and memory-injection modules. At the same time, it avoids the heavier structure-level sampling process used by methods such as RFDiffusion3. In other words, DockWizard trades a moderate condition-organization cost for stronger ligand-binding-oriented controllability, giving a relative balance between efficiency and conditional expressiveness. Combined with the runtime and memory results in Table 5, DockWizard's additional conditional modeling cost remains within an acceptable range in the current experimental setting.

B.6 Controls for relational language and sequence design

The comparisons below use paired test conditions. We estimate uncertainty by resampling conditions and apply Holm correction to families of paired tests. Three matched decoding seeds assess variation in generation with the trained models held fixed.

Relational text. We normalized protein and ligand entities in Europe PMC records, filtered for binding-related triggers, and removed duplicate text. Of 150 independently audited descriptions, 94.7% matched the intended pair and 82.7% contained specific interaction information (inter-annotator $\kappa = 0.78$). Identity-only text names the entities in a fixed template. Semantics-only text masks names, identifiers, organisms, and residue indices or identities while preserving relational clauses. Generic text matches token and clause counts using pair-independent sentences. All variants use the same checkpoint, samples, and decoding settings.

See Table 4 in the main text.

Correct descriptions yield interaction macro-F1 of 0.66 against experimental annotations on 96 examples, a gain of 0.16 (95% CI 0.11 to 0.21) over matched incorrect text. Across 42 counterfactual pairs matched on family, scaffold, and text length, changing only the text increases target realization by 12.1 percentage points (95% CI 7.3 to 16.8). The gain is 9.4 points (2.1 to 16.7) on the 24 low-overlap pairs.

Sequence recovery. Generated sequences have mean identity 0.244 to the reference and 0.299 to the nearest training sequence. With five generations per condition, 97.6% of outputs are unique and within-condition diversity is 0.697. Among 61 conditions whose reference has at most 30% identity to its nearest training sequence, the DD advantage over ProGen2 is 0.12 (95% CI 0.01 to 0.23).

Training and test overlap. Removing examples above the sequence identity threshold, from shared families, with scaffold or fingerprint overlap, with shared literature sources, or with near-duplicate text leaves 71,867 of 108,436 records. We retrained DockWizard and the comparison model for each primary metric (ProGen2 for DD, InstructPro for Chai-1 and Vina). Paired advantages are 0.10 (95% CI 0.01 to 0.19) for DD, 0.05 (-0.01 to 0.11) for Chai-1, and 0.21 kcal/mol (0.03 to 0.39) in the favorable direction for Vina. Valid interactions improve by 6.1 percentage points (0.7 to 11.5) and contact F1 by 5.2 points (-2.0 to 12.4). Separate test-subset analyses use the original models. Binding-mode overlap is assessed against 1,006 structure-mapped, PLIP-valid training complexes using fingerprints that omit residue indices and entity names. Intervals on the joint-strict and binding-mode-clean subsets include zero.

Table 6: Differences on overlap-controlled test subsets. Positive Vina differences denote lower energy for DockWizard.

Subset	N	ΔDD ↑	ΔChai-1 ↑	ΔVina ↑
Full	100	+0.13	+0.07	+0.28
Family-clean	72	+0.10	+0.06	+0.22
Scaffold-clean	58	+0.09	+0.04	+0.18
Text-deduplicated	80	+0.12	+0.05	+0.25
Joint strict	34	+0.07	+0.04	+0.15
Binding-mode-clean <0.50	31	+0.08	+0.04	+0.16

Structural evaluation. Vina scoring and PLIP interaction analysis complement the learned evaluators. DockWizard has a Vina score of -7.62 kcal/mol versus -7.23 for ProGen2 (Holm-adjusted $p = 0.010$). Its valid-interaction rate is 72% versus 64% for InstructPro. The expanded evaluation below uses 94 experimentally resolved reference structures. All five paired comparisons survive Holm correction.

See Table 2 in the main text.

Candidate selection. The original comparison preserves each method's selection procedure, including ProCALM reranking. In a separate run we evaluated ProGen2, InstructPro, RFD3, RFD3 + LigandMPNN, and DockWizard on 100 common IDs, with five candidates, three matched decoding

seeds, no post-hoc reranking, and failures included in the means. LigandMPNN redesigns non-fixed residues on an RFD3-generated backbone with ligand coordinates fixed. Redesigned sequences undergo independent refolding or cofolding. In order of DD, Chai-1, Vina, and valid interactions, RFD3 achieves -1.21, 0.36, -7.28, and 61.8%. Adding LigandMPNN yields -1.04, 0.39, -7.42, and 66.4%. DockWizard achieves -0.86, 0.43, -7.57, and 71.1%, with a 6.9% failure rate.

Ligand representation and conditioning pathways. For each ligand we generated three additional ETKDG conformers from SMILES at fixed decoding seeds. Mean sequence identity between alternative- and default-conformer outputs was 0.86. DD/Chai-1 shifted from -0.84/0.44 to -0.88/0.42. The median symmetry-corrected heavy-atom RMSD between isolated and experimentally bound conformers was 1.46 Å. We also removed each ligand view separately. In the pathway ablations, removing the bridge disables cross-memory communication, whereas setting its gate regularization weight to zero only removes the penalty. Removing the ligand contrastive loss leaves the bridge in place.

Table 7: Ablation of the three ligand views.

Ligand setting	DD ↑	Chai-1 ↑	Vina ↓
Full tri-view	-0.84	0.44	-7.62
w/o ligand text	-0.93	0.40	-7.45
w/o SMILES	-1.07	0.33	-7.18
w/o ligand 3D	-0.98	0.37	-7.31

Table 8: Separate ablations of the bridge pathway and the ligand contrastive objective.

Variant	DD ↑	Chai-1 ↑	Vina ↓	Valid interactions ↑
Full DockWizard	-0.84	0.44	-7.62	72%
w/o bridge	-0.91	0.40	-7.43	66%
w/o ligand contrastive	-0.93	0.39	-7.39	65%
w/o both	-0.98	0.35	-7.25	60%

C Theoretical Proofs

C.1 Notation

Let D, L, and F denote the conditional tokens from docking text, ligand representations, and function text, respectively. The multimodal conditional context is written as

$$\mathbf{X}_c = [D \,\|\, L \,\|\, F]. \tag{22}$$

Role-specific routing maps $\mathbf{X}_c$ into three types of memory:

$$\mathbf{M}^{\text{func}}, \qquad \mathbf{M}^{\text{dl,light}}, \qquad \mathbf{M}^{\text{dl,full}}, \tag{23}$$

where $\mathbf{M}^{\text{func}}$ denotes global functional memory, $\mathbf{M}^{\text{dl,light}}$ is used to construct the prefix, and $\mathbf{M}^{\text{dl,full}}$ is used for full-memory injection inside decoder layers. The conditional prefix is denoted by

$$\mathbf{\Pi} = \text{Prefix}(\mathbf{M}^{\text{func}}, \mathbf{M}^{\text{dl,light}}). \tag{24}$$

The full memory after projection into the decoder hidden space is denoted by

$$\overline{M}_{\text{dl}}^{\text{full}}. \tag{25}$$

Let $h_t^{(\ell)}$ denote the hidden state at decoder layer ℓ and generation position t.

C.2 Role-specific routing and condition separation

Proposition 1 (Role-specific routing reduces direct condition competition)**.** *Under a simplified attention setting, flat fusion makes conditions with different semantic roles share the same attention normalization. Role-specific routing assigns different conditions to different memory spaces, thereby reducing direct attention competition between function conditions and docking/ligand conditions.*

Proof sketch. In flat fusion, all conditions are concatenated as

$$M_{\text{flat}} = [D \,\|\, L \,\|\, F]. \tag{26}$$

Given a decoder query q, its attention weight is

$$\alpha_i = \frac{\exp(q^\top k_i)}{\sum_{j \in M_{\text{flat}}} \exp(q^\top k_j)}. \tag{27}$$

Let

$$s_j = q^\top k_j. \tag{28}$$

Then, for $i \neq j$,

$$\frac{\partial \alpha_i}{\partial s_j} = -\alpha_i \alpha_j. \tag{29}$$

Therefore, when the score s_j of one type of condition token increases, the normalized attention weights of other tokens decrease. In flat fusion, function tokens, docking tokens, and ligand tokens share the same attention budget.

In role-specific routing, conditions are decomposed into different role-specific memories:

$$\mathbf{M}^{\text{func}} = R_{\text{func}}(F, L), \tag{30}$$

$$M_{\text{dl}} = R_{\text{dl}}(D, L). \tag{31}$$

The corresponding attention reads are

$$z_{\text{func}} = \text{Attn}(q, \mathbf{M}^{\text{func}}, \mathbf{M}^{\text{func}}), \tag{32}$$

$$z_{\text{dl}} = \text{Attn}(q, M_{\text{dl}}, M_{\text{dl}}). \tag{33}$$

The two types of conditions are then normalized in their respective memory spaces:

$$\alpha_i^{\text{func}} = \frac{\exp(q^\top k_i^{\text{func}})}{\sum_{j \in \mathbf{M}^{\text{func}}} \exp(q^\top k_j^{\text{func}})}, \tag{34}$$

$$\alpha_i^{\mathrm{dl}} = \frac{\exp(q^\top k_i^{\mathrm{dl}})}{\sum_{j \in M_{\mathrm{dl}}} \exp(q^\top k_j^{\mathrm{dl}})}. \tag{35}$$

The final conditional representation can be written as

$$z = \lambda_{\mathrm{func}} z_{\mathrm{func}} + \lambda_{\mathrm{dl}} z_{\mathrm{dl}}. \tag{36}$$

Thus, role-specific routing decomposes a shared attention problem into multiple role-specific attention problems and reduces direct normalization competition between global functional semantics and local docking/ligand constraints. □

C.3 Staged condition injection and conditional influence

Proposition 2 (Staged condition injection preserves conditional influence)**.** *Under a linearized decoder update, staged condition injection makes the hidden state contain both a prefix-conditioned global term and a full-memory-injected local constraint term, providing two explicit influence paths for conditions with different roles.*

Proof sketch. Prefix guidance concatenates the conditional prefix with the generated tokens as the decoder input:

$$H^{(0)} = [\mathbf{\Pi} \,\|\, E(y_{<t})]. \tag{37}$$

Therefore, the hidden state at layer ℓ can be affected by the prefix:

$$\Delta h_{t,\mathrm{prefix}}^{(\ell)} = G_\ell(h_t^{(\ell)}, \mathbf{\Pi}). \tag{38}$$

Full-memory injection introduces an additional residual path inside decoder layers. Let $\widehat{h}_t^{(\ell)}$ denote the output of the original decoder block. The injection can be abstracted as

$$h_t^{(\ell+1)} = \widehat{h}_t^{(\ell)} + g_\ell \odot \mathrm{Attn}\left(\widehat{h}_t^{(\ell)}, \overline{M}_{\mathrm{dl}}^{\mathrm{full}}, \overline{M}_{\mathrm{dl}}^{\mathrm{full}}\right). \tag{39}$$

Denote the influence of full-memory injection by

$$\Delta h_{t,\mathrm{full}}^{(\ell)} = g_\ell \odot \mathrm{Attn}\left(\widehat{h}_t^{(\ell)}, \overline{M}_{\mathrm{dl}}^{\mathrm{full}}, \overline{M}_{\mathrm{dl}}^{\mathrm{full}}\right). \tag{40}$$

Under a linearized approximation, the decoder hidden state can be decomposed as

$$h_t^{(\ell)} \approx h_{t,\mathrm{base}}^{(\ell)} + \Delta h_{t,\mathrm{prefix}}^{(\ell)} + \Delta h_{t,\mathrm{full}}^{(\ell)}. \tag{41}$$

Here, $h_{t,\mathrm{base}}^{(\ell)}$ denotes the autoregressive generation term of the original protein language model, $\Delta h_{t,\mathrm{prefix}}^{(\ell)}$ denotes the global conditional influence introduced by prefix guidance, and $\Delta h_{t,\mathrm{full}}^{(\ell)}$ denotes the docking/ligand conditional influence introduced by full-memory injection.

Therefore, staged condition injection separates conditional influence into two paths:

$$\mathbf{\Pi} \longrightarrow \Delta h_{t,\mathrm{prefix}}^{(\ell)} \tag{42}$$

and

$$\overline{M}_{\mathrm{dl}}^{\mathrm{full}} \longrightarrow \Delta h_{t,\mathrm{full}}^{(\ell)}. \tag{43}$$

Prefix guidance mainly provides global functional context, while full-memory injection continuously provides docking/ligand-related constraints inside decoder layers. This analysis gives a structural interpretation of how staged injection preserves influence paths from different condition sources, but it does not constitute a strict performance guarantee for downstream metrics. □

D Pseudocode

This appendix provides the training and inference pseudocode for DockWizard. It covers multimodal condition encoding, ligand-side representation alignment, role-specific memory routing, staged condition injection, and no-KV-cache autoregressive inference.

Algorithm 1: Training procedure of DockWizard

Input: Training set $\mathcal{D} = \{(c_{\text{func}}, c_{\text{dock}}, c_{\text{lig}}, y)\}$; pretrained condition encoders and protein decoder backbone; trainable parameters θ

Output: Trained DockWizard parameters θ

Initialize DockWizard with pretrained backbones and trainable condition alignment, routing, and injection modules;

Initialize AdamW optimizer over trainable parameters θ;

foreach *minibatch* $\mathcal{B} \subset \mathcal{D}$ **do**

- Encode docking text c_{dock} into docking tokens D;
- Encode function text c_{func} into function tokens F;
- Encode ligand condition c_{lig} from ligand text, SMILES, and ligand 3D inputs into ligand-view tokens (T, S, G);
- Fuse ligand views (T, S, G) to obtain ligand tokens L;
- **if** *ligand-side alignment is enabled* **then**
  - Compute auxiliary ligand alignment loss $\mathcal{L}_{\text{lig-ctr}}$;
- Pack multimodal condition tokens as $\mathbf{X}_c = [D \| L \| F]$;
- Route $\mathbf{X}_c$ with the role-specific routing module to obtain $\mathbf{M}^{\text{func}}$, $\mathbf{M}^{\text{dl,light}}$, and $\mathbf{M}^{\text{dl,full}}$;
- Construct conditional prefix $\mathbf{\Pi}$ from $\mathbf{M}^{\text{func}}$ and $\mathbf{M}^{\text{dl,light}}$;
- Run the protein decoder under teacher forcing on target sequence y, conditioned on $\mathbf{\Pi}$ and full-memory injection from $\mathbf{M}^{\text{dl,full}}$;
- Compute conditional language-modeling loss $\mathcal{L}_{\text{LM}}$;
- $\mathcal{L} \leftarrow \mathcal{L}_{\text{LM}}$;
- **if** *ligand-side alignment is enabled* **then**
  - $\mathcal{L} \leftarrow \mathcal{L} + \lambda_{\text{lig}} \mathcal{L}_{\text{lig-ctr}}$;
- **if** *bridge-gate regularization is enabled* **then**
  - Compute gate regularization $\mathcal{L}_{\text{gate}}$;
  - $\mathcal{L} \leftarrow \mathcal{L} + \lambda_{\text{gate}} \mathcal{L}_{\text{gate}}$;
- Backpropagate $\mathcal{L}$;
- Clip gradients and update trainable parameters θ with AdamW;

return θ;

DockWizard training. The training inputs of DockWizard include the target protein sequence y, function text c_{func}, docking text c_{dock}, and ligand condition c_{lig}. The ligand condition c_{lig} may include a ligand description, a SMILES representation, and a ligand 3D representation. During training, the model first encodes different modalities and then organizes docking, ligand, and function conditions into a unified sequence of condition tokens. The role-specific routing module then routes the condition tokens into global functional memory, light docking-ligand memory, and full docking-ligand memory. The decoder receives $\mathbf{M}^{\text{func}}$ and $\mathbf{M}^{\text{dl,light}}$ through prefix guidance, and receives $\mathbf{M}^{\text{dl,full}}$ through full-memory injection. The training objective is mainly the conditional language-modeling loss, with optional ligand-side alignment loss and optional bridge-gate regularization.

DockWizard inference. The inference inputs of DockWizard include target function text c_{func}, docking text c_{dock}, and ligand condition c_{lig}. During inference, the model first performs multimodal condition encoding and role-specific memory routing once, obtaining $\mathbf{M}^{\text{func}}$, $\mathbf{M}^{\text{dl,light}}$, and $\mathbf{M}^{\text{dl,full}}$. Then $\mathbf{M}^{\text{func}}$ and $\mathbf{M}^{\text{dl,light}}$ are used to construct the conditional prefix $\mathbf{\Pi}$, while $\mathbf{M}^{\text{dl,full}}$ is used for full-memory injection inside decoder layers. Generation uses no-KV-cache autoregressive inference: at each step, the model reconstructs $[\mathbf{\Pi} \| \hat{y}_{<t}]$ and recomputes the decoder forward pass with KV cache disabled, so that staged condition injection remains active throughout generation.

E Dataset Details

E.1 Overview of dataset construction

We construct a multimodal conditional dataset for ligand-binding protein generation. The goal of the dataset is not only to collect protein sequences, but also to provide global functional descriptions,

Algorithm 2: Autoregressive inference of DockWizard

Input: Function text c_{func}, docking text c_{dock}, ligand condition c_{lig}, maximum generation length T
Output: Generated protein sequence $\hat{y}$
Encode c_{dock}, c_{func}, and c_{lig} into condition tokens D, F, and L;
Pack condition tokens as $\mathbf{X}_c = [D \| L \| F]$;
Route $\mathbf{X}_c$ to obtain $\mathbf{M}^{\mathrm{func}}$, $\mathbf{M}^{\mathrm{dl,light}}$, and $\mathbf{M}^{\mathrm{dl,full}}$;
Construct conditional prefix $\mathbf{\Pi}$ from $\mathbf{M}^{\mathrm{func}}$ and $\mathbf{M}^{\mathrm{dl,light}}$;
Enable full-memory injection using $\mathbf{M}^{\mathrm{dl,full}}$;
Initialize generated history $\hat{y}_{<1}$ with the beginning-of-sequence token;
for $t = 1$ **to** T **do**
    Form decoder input $[\mathbf{\Pi} \| \hat{y}_{<t}]$;
    Recompute the decoder forward pass over $[\mathbf{\Pi} \| \hat{y}_{<t}]$ with KV cache disabled;
    Inject $\mathbf{M}^{\mathrm{dl,full}}$ through the decoder full-memory injection modules;
    Obtain next-token distribution $p(\hat{y}_t \mid \hat{y}_{<t}, \mathbf{\Pi}, \mathbf{M}^{\mathrm{dl,full}})$ from the final position;
    Select or sample the next token $\hat{y}_t$;
    Append $\hat{y}_t$ to the generated history;
    if $\hat{y}_t$ *is an end-of-sequence token* **then**
        break;
Decode generated tokens into protein sequence $\hat{y}$;
return $\hat{y}$;

local docking-oriented language, ligand chemical representations, and ligand three-dimensional representations for each protein–ligand pair. This supports DockWizard in receiving both function-level guidance and ligand-binding-related constraints during generation.

The raw protein information is derived from reviewed UniProtKB entries, i.e., the UniProtKB/Swiss-Prot source. We first select reviewed entries with explicit functional annotations and binding-site ligand annotations. Protein sequences are taken from the UniProt `Sequence` field, protein function text is taken from the `Function [CC]` field, and ligand information is mainly obtained from ligand names and ChEBI IDs in binding-site annotations. Each binding-site ligand annotation is expanded into a candidate protein–ligand pair, so the final dataset uses pair-level samples as its basic unit.

After obtaining the initial protein–ligand pairs, the construction pipeline further completes ligand-side information. Specifically, ligand names or ChEBI IDs are used to query PubChem and ChEBI for candidate SMILES strings. All SMILES strings are then parsed, validated, and canonicalized with RDKit. Ligands that cannot be correctly parsed or canonicalized by RDKit are removed. For valid ligands, the pipeline further generates ligand text, canonical SMILES, InChIKey, and ligand three-dimensional representations. The ligand three-dimensional representation is generated by RDKit from canonical SMILES and includes three-dimensional conformations, atom types, and coordinates, which are stored as indexable three-dimensional features.

Overall, the dataset construction process can be summarized as follows. We extract high-confidence proteins and their binding-site ligand annotations from reviewed UniProt entries, expand binding-site annotations into protein–ligand pairs, complete and validate ligand SMILES, textual descriptions, and three-dimensional conformations, retrieve and extract literature sentences related to protein–ligand binding, and finally assemble protein sequences, function text, docking-oriented text, ligand text, SMILES, and ligand three-dimensional representations into unified multimodal conditional samples.

E.2 Docking-language construction

Docking-language construction is a key step in this dataset. The `docking_text` field is not generated from the numerical output of direct docking simulations. Instead, it is literature-derived docking-oriented natural language obtained through literature retrieval and rule-based extraction. This design provides the model with protein–ligand interaction context in a form closer to human

scientific descriptions, including descriptions related to binding pockets, ligand recognition, interaction patterns, affinity, docking, and binding modes.

For each protein–ligand pair, the data pipeline first constructs protein aliases from UniProt metadata and ligand aliases from PubChem and ChEBI information. It then uses protein aliases, ligand aliases, and pair-level search terms to retrieve relevant literature from Europe PMC and OpenAlex. When Europe PMC provides `fullTextXML`, full text is preferred; when full text is unavailable, abstracts are used as candidate text sources.

Candidate sentences are extracted with rule-based co-occurrence and relevance scoring. A sentence usually needs to contain both a protein alias and a ligand alias, and also include binding- or docking-related trigger words such as binding, docking, interaction, pocket, or affinity, before it is treated as a candidate docking-oriented sentence. The candidate sentences are then deduplicated, length-filtered, relevance-ranked, and aggregated at the pair level. Each protein–ligand pair finally receives an aggregated `docking_text`, together with the docking sentence count and docking paper count supporting that text.

During final dataset assembly, we retain only protein-ligand pairs with non-empty docking-oriented text. In other words, every protein–ligand pair in the final training and evaluation data contains at least one segment of docking-related natural language that can be used as a generation condition. This filtering ensures that DockWizard can consistently receive docking-oriented text during training and inference, rather than relying only on global function text or ligand SMILES.

E.3 Multimodal condition fields

Each final sample contains several complementary modalities. The protein side mainly includes `protein_sequence` and protein function text. The protein sequence serves as the generation target or reference sequence, while protein function text provides global functional semantics. The ligand side includes ligand text, canonical SMILES, InChIKey, and ligand three-dimensional representation. Ligand text provides a natural-language description of ligand identity, SMILES provides chemical topology, InChIKey supports ligand identity tracking, and ligand three-dimensional representation provides spatial shape and geometric information. The pair-level side includes `docking_text`, `docking_sentence_count`, `docking_paper_count`, and `pair_key`, which describe binding-related language evidence and sample indexing for the protein–ligand pair.

In downstream training, these fields are organized as multimodal conditional inputs. Function text serves as global semantic guidance, `docking_text` serves as a ligand-binding-oriented local constraint, and ligand text, SMILES, and ligand three-dimensional representation jointly describe the semantic, chemical, and spatial properties of the ligand. For downstream settings that require instruction-formatted input, function text, docking text, and ligand information can be further converted into an instruction field. However, instruction fields are better viewed as downstream format-conversion outputs rather than core fields in the raw dataset construction process.

E.4 Data filtering and quality control

The dataset construction process applies several layers of quality control. First, at the UniProt entry level, we retain only reviewed entries with valid protein sequences, function annotations, and binding-site ligand annotations. Records missing protein sequences, missing functional annotations, or lacking expandable ligand annotations are not included in subsequent pair-level construction.

Second, at the protein sequence level, the pipeline cleans abnormal sequences and restricts sequences to standard amino-acid characters. Overly long sequences are filtered to avoid excessive computational cost during training and structural evaluation. Duplicate samples are also removed, especially redundant records with the same protein, ligand SMILES, and cleaned sequence.

At the ligand level, all candidate SMILES strings must pass RDKit validation. Ligands that cannot be parsed, canonicalized, or converted into valid chemical structures are filtered out. For retained ligands, the pipeline generates canonical SMILES, InChIKey, and three-dimensional conformations. The ligand three-dimensional representation also relies on RDKit conformer generation and geometry optimization, providing the model with ligand three-dimensional inputs in a unified format.

At the docking-language level, empty text, duplicate sentences, overly short sentences, and candidate text without protein–ligand co-occurrence are filtered. Samples without valid `docking_text` are not included in the final dataset. This rule makes the dataset more suitable for the task studied in this paper, because DockWizard is designed to use docking-oriented natural language to connect global functional semantics with local ligand-binding constraints.

F Baseline Details

This section describes how each baseline is adapted in this paper. Existing protein generation methods differ in their original task settings and input interfaces. Some methods mainly target natural-language-conditioned generation, some target structure generation, and others rely on discrete labels or prefix control. For fair comparison, we adapt these methods to the unified ligand-binding-oriented protein generation task while preserving their original design principles as much as possible.

All baselines use the same test samples and the same generation budget. The generated protein sequences are passed through the same downstream evaluation pipeline, including structure prediction, RMSD comparison with the GT structure, DiffDock docking evaluation, and Chai-1 complex-level scoring. For methods that do not natively support certain modalities, we convert function text, docking text, ligand SMILES, or ligand structure into the closest input form accepted by that method, without introducing DockWizard-specific role-specific routing or staged memory injection into the baselines.

ProCALM. Yang et al. [2024] is adapted as a strong conditional protein language model baseline. The adapted ProCALM preserves its original ProGen2-style backbone, conditional adapters, and autoregressive sequence generation framework, but replaces the original condition interface with the ligand-binding conditions required by our task. Specifically, function text and docking-oriented text are used as textual condition inputs, while ligand SMILES is used as the ligand-side chemical condition input. The ProGen2 backbone serves as the main protein sequence generator, and training mainly updates condition projectors and adapter modules. This baseline tests whether adapter-based conditional generation can effectively use function, docking, and ligand information, and also helps distinguish standard conditional adaptation from DockWizard's role-specific routing and staged injection design.

RFDiffusion3. RFDiffusion3 [Watson et al., 2023] is used as a structure-generation-oriented baseline. Since RFDiffusion3 cannot directly accept free-form natural language as a generation condition, we adapt it through constraint translation: the design intent in the instruction, function text, docking text, and ligand information is converted into coarse structural or binding-related constraints, and ligand SMILES is converted into a three-dimensional ligand structure to match the input format of RFDiffusion3. After the model generates protein structures, protein sequences are extracted from the generated structures and passed into the same downstream evaluation pipeline as other methods. This baseline represents a ligand-aware design path centered on structure generation and structural constraints, in contrast to DockWizard's language-conditioned sequence generation path.

ProteinDT. ProteinDT [Liu et al., 2025b] is used as a text-guided protein generation baseline. The original method maps natural-language descriptions into the protein generation space through representation alignment between proteins and text. To adapt it to our task, we organize the instruction, function text, docking text, and ligand SMILES as a unified textual condition, allowing ProteinDT to receive task-relevant information under its original protein–text generation framework. This baseline tests the transfer ability of general protein–text alignment methods in ligand-binding-oriented generation. Compared with DockWizard, ProteinDT mainly relies on text-level conditional expression and does not explicitly distinguish the different roles of global functional semantics, docking-oriented language, and ligand representation.

ProGen2. ProGen2 [Nijkamp et al., 2023] is used as a strong pretrained protein language model baseline. Since native ProGen2 lacks an explicit conditioning interface for our task, we adapt it as a conditional-prefix baseline. Function text, docking-related task text, and ligand SMILES are encoded as continuous prefix embeddings and concatenated before the token embeddings of ProGen2, so that the frozen ProGen2 backbone is guided by conditional information during autoregressive

generation. This baseline evaluates whether combining a strong protein sequence prior with simple prefix conditioning is sufficient for ligand-binding-oriented protein generation. Its main difference from DockWizard is that ProGen2-condPrefix compresses heterogeneous conditions into a single prefix path, whereas DockWizard further organizes conditions into memories with different semantic roles.

PrefixProt. PrefixProt [Luo et al., 2024] is used as a discrete-control controllable generation baseline. The original strength of PrefixProt is parameter-efficient label control through virtual prefix tokens, rather than direct processing of open natural language and multimodal ligand information. Therefore, we map function text, docking text, and ligand information into discrete control labels, including functional type, binding-related pattern, and ligand category, and combine these labels into a control code. During inference, the corresponding prefix-based generation module is selected according to the control code of the input sample. This baseline tests whether compressing complex natural-language and ligand conditions into symbolic control codes can still preserve sufficient ligand-binding controllability.

InstructPro. InstructPro [Song et al., 2025a] is used as an instruction-based conditional generation baseline. We organize function text, docking text, and ligand SMILES into instruction-style input, allowing the model to receive task conditions in a unified instruction format. The generation backbone remains a ProGen2-style protein language model, and training mainly updates condition-related modules so that instruction and ligand-related information can be mapped into the protein generation process. Compared with DockWizard, InstructPro represents a flat instruction-conditioning strategy: conditions from different sources are placed into the same instruction stream, rather than being explicitly routed into global functional memory and docking/ligand-related memory.

Fairness protocol. Natural-language-guided ligand-binding protein generation with docking-oriented text and multimodal ligand conditions is still an emerging task, and most existing protein generation methods do not provide a native interface for all required conditions. To ensure fair comparison, all methods are evaluated on the same test set, with the same number of generated samples and the same downstream evaluation pipeline. For baselines that cannot natively process certain input modalities, we only convert these modalities into the closest forms accepted by the original method, such as textual prompts, discrete control codes, continuous prefixes, or structure-compatible ligand inputs. We do not add DockWizard-specific role-specific memory routing, staged condition injection, or docking-aware decoder structures to the baselines. This preserves the original methodological characteristics of each baseline while ensuring that all methods are evaluated under the same ligand-binding-oriented generation setting.

G Model and Implementation Details

This section provides additional model details and formulas that are compressed in the main Method section. Unlike the pseudocode in Appendix D, this section focuses on the concrete representation forms, condition organization, training objectives, and inference mechanism of DockWizard.

G.1 Multimodal condition encoding

DockWizard receives five types of conditional input: function text, docking-oriented text, ligand description, ligand SMILES, and ligand three-dimensional representation. To use these heterogeneous signals for protein sequence generation, the model first encodes each modality separately, and then maps them into a unified condition hidden space through projection layers and query pooling.

Let c_{func}, c_{dock}, c_{ltxt}, and c_{smi} denote function text, docking text, ligand text, and ligand SMILES, respectively. For any textual or sequence modality s, the encoding process is written as

$$\begin{aligned} H_s &= B_s(c_s), \\ \widetilde{H}_s &= P_s(H_s), \\ Z_s &= \mathrm{QPool}_s(\widetilde{H}_s), \end{aligned} \tag{44}$$

where B_s denotes the corresponding pretrained encoder, P_s denotes a modality-specific projection, and QPool_s denotes learnable query pooling. Function text, docking text, and ligand text use a biomedical text encoder; SMILES uses a pretrained SMILES language model. By default, these backbones are kept frozen to reduce task-specific training cost.

For the ligand three-dimensional representation, the input consists of atom types and three-dimensional coordinates. The model uses a 3D encoder to encode ligand geometry, followed by projection and query pooling to obtain fixed-length 3D ligand tokens:

$$Z_{3D} = \mathrm{QPool}_{3D}\left(P_{3D}(B_{3D}(A, R))\right), \tag{45}$$

where A denotes atom types and R denotes atomic coordinates. After obtaining docking tokens D, function tokens F, and ligand tokens L, the model organizes them into a unified condition context:

$$\mathbf{X}_c = [D \| L \| F]. \tag{46}$$

Here, D denotes docking-oriented tokens, L denotes fused ligand tokens, and F denotes function tokens. This ordering provides structured input for the subsequent role-specific memory routing.

G.2 Ligand-side representation alignment

The ligand condition contains three views: ligand text, SMILES, and 3D geometry. To keep different ligand views consistent in a unified representation space, DockWizard introduces ligand-side representation alignment. Let T, S, and G denote ligand text tokens, SMILES tokens, and 3D geometry tokens, respectively. Each view is first pooled and projected into a sample-level representation:

$$\begin{aligned} e_v &= \mathrm{Pool}(V), \\ z_v &= \mathrm{Normalize}(W_v e_v), \\ v &\in \{T, S, G\}. \end{aligned} \tag{47}$$

For any two ligand views a, b, the model uses a symmetric InfoNCE loss:

$$\mathcal{L}_{a,b} = \frac{1}{2}\left[\mathrm{CE}\left(\frac{z_a z_b^\top}{\tau}, I\right) + \mathrm{CE}\left(\frac{z_b z_a^\top}{\tau}, I\right)\right], \tag{48}$$

where τ is the temperature and I denotes the positive matching relation within the same minibatch. Two ligand views from the same sample form a positive pair, while views from other samples form in-batch negatives. The three-view ligand alignment loss is

$$\mathcal{L}_{\text{lig-ctr}} = \frac{1}{3}\left(\mathcal{L}_{S,G} + \mathcal{L}_{S,T} + \mathcal{L}_{G,T}\right). \tag{49}$$

The three ligand views are then fused into unified ligand tokens through gated weighted fusion. Let the weights of the three views be

$$w = (1-\alpha)\mathrm{softmax}\left(\mathrm{MLP}([e_S, e_G, e_T])\right) + \alpha \cdot \frac{1}{3}\mathbf{1}. \tag{50}$$

The fused ligand representation is written as

$$L = \mathrm{Adapter}\left(w_S S + w_G G + w_T T\right). \tag{51}$$

This design aligns semantic, chemical, and spatial information from ligand text, SMILES, and 3D geometry before they enter the routing module, providing a more stable ligand representation for subsequent ligand-aware generation.

G.3 Role-specific memory routing

Given the unified condition context

$$\mathbf{X}_c = [D \| L \| F], \tag{52}$$

DockWizard uses role-specific routing to organize heterogeneous conditions into memories with different semantic roles. The key-value set of the function branch is defined as

$$KV_{\text{func}} = [F \| L], \tag{53}$$

and the key-value set of the docking-ligand branch is defined as

$$KV_{\mathrm{dl}} = [D \| L]. \tag{54}$$

The function branch mainly preserves global functional semantics, while the docking-ligand branch mainly preserves docking-oriented language and ligand-related constraints. Each branch uses learnable latent queries to read information from its corresponding key-value set:

$$Z_b^{(0)} = \mathrm{Latent}_b, \tag{55}$$

$$Z_b^{(1)} = Z_b^{(0)} + \mathrm{CrossAttn}\left(\mathrm{LN}(Z_b^{(0)}), \mathrm{LN}(KV_b)\right), \qquad b \in \{\mathrm{func}, \mathrm{dl}\}. \tag{56}$$

Here, $\mathrm{CrossAttn}$ is implemented as multi-head attention and corresponds to the MHA operation used in the main text. After latent refinement, the memories are obtained as

$$\begin{aligned} \mathbf{M}^{\mathrm{func}} &= Z_{\mathrm{func}}^{(T_{\mathrm{func}})}, \\ \mathbf{M}^{\mathrm{dl,full}} &= Z_{\mathrm{dl}}^{(T_{\mathrm{dl}})}. \end{aligned} \tag{57}$$

In addition, the model uses a gated bridge to let function memory modulate docking-ligand memory in a controlled manner:

$$\widetilde{M}_{\mathrm{dl}}^{\mathrm{full}} = \mathbf{M}^{\mathrm{dl,full}} + \sigma(g_{\mathrm{br}}(\mathbf{M}^{\mathrm{dl,full}})) \odot \mathrm{Bridge}(\mathbf{M}^{\mathrm{dl,full}}, \mathbf{M}^{\mathrm{func}}). \tag{58}$$

The full docking-ligand memory is then compressed into lightweight memory:

$$\mathbf{M}^{\mathrm{dl,light}} = \mathrm{Resampler}(\widetilde{M}_{\mathrm{dl}}^{\mathrm{full}}). \tag{59}$$

Thus, DockWizard obtains global functional memory, full docking-ligand memory, and light docking-ligand memory. Compared with directly concatenating all conditions into a single control stream, role-specific routing assigns global functional semantics and local docking/ligand constraints to different memory spaces, reducing role confusion among heterogeneous conditions.

G.4 Self-attention latent refinement

In role-specific routing, latent memory tokens not only read information from external conditions through cross-attention, but also perform self-attention refinement within the latent space. For latent tokens Z from any branch, one refinement block is written as

$$\begin{aligned} Z &\leftarrow Z + \mathrm{Attn}(\mathrm{LN}(Z), \mathrm{LN}(Z)), \\ Z &\leftarrow Z + \mathrm{FFN}(\mathrm{LN}(Z)). \end{aligned} \tag{60}$$

This operation allows latent tokens within the same memory to exchange information, improving the ability of the memory to organize multi-source conditions. Unlike global fusion directly over raw condition tokens, latent refinement operates on memory tokens after they have been read by role-specific branches, and therefore can strengthen information integration within each memory while preserving semantic role separation.

G.5 Staged condition injection

DockWizard's decoder conditioning contains two stages: prefix guidance and full-memory injection. First, the model constructs a conditional prefix from $\mathbf{M}^{\mathrm{func}}$ and $\mathbf{M}^{\mathrm{dl,light}}$:

$$\Pi = [b_F^{\mathrm{open}} \| W_F \mathbf{M}^{\mathrm{func}} \| b_F^{\mathrm{close}} \| b_{DL}^{\mathrm{open}} \| W_L \mathbf{M}^{\mathrm{dl,light}} \| b_{DL}^{\mathrm{close}}]. \tag{61}$$

Under teacher forcing, the decoder input is

$$[\Pi \| E(y_{<t})], \tag{62}$$

where labels at prefix positions are masked and the language-modeling loss is computed only at protein token positions. Prefix guidance provides the decoder with global functional semantics and lightweight docking-ligand guidance.

Second, full-memory injection uses $\mathbf{M}^{\mathrm{dl,full}}$. The model first projects it into the decoder hidden dimension:

$$\overline{M}_{\mathrm{dl}}^{\mathrm{full}} = W_{\mathrm{full}}\mathbf{M}^{\mathrm{dl,full}}. \tag{63}$$

In the ℓ-th decoder block, let H_ℓ^p denote the hidden states at protein token positions. Full-memory injection is written as

$$\begin{aligned} U_\ell &= \mathrm{CrossAttn}\left(\mathrm{LN}(H_\ell^p), \overline{M}_{\mathrm{dl}}^{\mathrm{full}}, \overline{M}_{\mathrm{dl}}^{\mathrm{full}}\right), \\ g_\ell &= \sigma\left(\mathrm{MLP}_\ell\left([\mathrm{LN}(H_\ell^p), \mu_{\mathrm{full}}]\right)\right), \\ \mu_{\mathrm{full}} &= \mathrm{Mean}(\overline{M}_{\mathrm{dl}}^{\mathrm{full}}), \\ H_\ell^p &\leftarrow H_\ell^p + g_\ell \odot U_\ell. \end{aligned} \tag{64}$$

Prefix guidance and full-memory injection have different scopes of action. The former provides a relatively stable global conditional context at the decoder input, while the latter continuously injects more complete docking/ligand-related constraints inside decoder layers. Compared with simply concatenating all conditions at the input side, staged injection assigns conditions with different roles to different injection paths, making the distinction between global functional semantics and local ligand-binding constraints clearer.

G.6 Training objectives

Training uses conditional language modeling under teacher forcing. Given the target protein sequence

$$y = (y_1, \ldots, y_T), \tag{65}$$

the model defines

$$p_\theta(y \mid c_{\mathrm{func}}, c_{\mathrm{dock}}, c_{\mathrm{lig}}) = \prod_{t=1}^{T} p_\theta(y_t \mid y_{<t}, \Pi, \mathbf{M}^{\mathrm{dl,full}}). \tag{66}$$

The language-modeling loss is

$$\mathcal{L}_{\mathrm{LM}} = -\sum_{t=1}^{T} \log p_\theta(y_t \mid y_{<t}, \Pi, \mathbf{M}^{\mathrm{dl,full}}). \tag{67}$$

The total training objective is

$$\mathcal{L}_{\mathrm{total}} = \mathcal{L}_{\mathrm{LM}} + \lambda_{\mathrm{lig}}\mathcal{L}_{\text{lig-ctr}} + \mathbb{I}_{\mathrm{gate}}\lambda_{\mathrm{gate}}\mathcal{L}_{\mathrm{gate}}. \tag{68}$$

Here, $\mathcal{L}_{\text{lig-ctr}}$ is the ligand-side contrastive alignment loss and λ_{lig} denotes its effective weight. $\mathcal{L}_{\mathrm{gate}}$ is optional bridge-gate regularization and is included in the training objective only when the corresponding configuration is enabled. Repetition penalty is not written as an independent training loss; it is mainly used as decoding control during generation.

G.7 Autoregressive inference without KV cache

At inference time, the condition encoder and routing module run only once to obtain $\mathbf{M}^{\mathrm{func}}$, $\mathbf{M}^{\mathrm{dl,light}}$, and $\mathbf{M}^{\mathrm{dl,full}}$, and to construct the conditional prefix Π. The model then uses no-KV-cache autoregressive inference. At generation step t, the decoder input is

$$[\Pi \| E(\hat{y}_{<t})]. \tag{69}$$

The model recomputes the full forward pass and injects $\mathbf{M}^{\mathrm{dl,full}}$ into decoder layers through full-memory injection. The next-token generation distribution is

$$\hat{y}_t \sim p_\theta\left(y_t \mid \hat{y}_{<t}, \Pi, \mathbf{M}^{\mathrm{dl,full}}\right), \qquad t = 1, \ldots, T. \tag{70}$$

If layer-wise full-memory injection is written explicitly, this can also be expressed as

$$\hat{y}_t \sim p_\theta\left(y_t \mid \hat{y}_{<t}, \Pi, \{\overline{M}_{\mathrm{dl}}^{\mathrm{full}}\}_{\ell=1}^{L}\right). \tag{71}$$

No-KV-cache inference has a slightly higher inference-time cost because the prefix and generated sequence are recomputed at every step. However, it ensures that prefix guidance and layer-wise full-memory injection act consistently at each generation step, avoiding possible position-dependent injection inconsistency caused by cached one-token decoding.

H Hyperparameters and Compute Resources

This section reports the main training and generation hyperparameters of DockWizard. The baseline adaptation protocol is provided in Appendix F, and the evaluation protocol and metric definitions are provided in Appendix B.

H.1 DockWizard training hyperparameters

Table 9: DockWizard training hyperparameters.

Hyperparameter	Value	Description
Context hidden dimension d_c	1024	Shared hidden dimension for representations.
Docking condition tokens	64	Fixed-length docking text tokens after pooling.
Function condition tokens	32	Fixed-length function text tokens query pooling.
Ligand condition tokens	16	Fused ligand tokens.
Function memory length M_{func}	64	Number of global functional memory tokens.
Full docking-ligand memory length	128	Number of full docking-ligand memory tokens.
Light docking-ligand length	16	Number of lightweight docking-ligand memory tokens used for prefix construction.
Perceiver depth	4	Number of Perceiver routing blocks per branch.
Self-attention refinement layers	2	Number of latent self-attention layers.
Perceiver attention heads	8	Number of attention heads in the routing modules.
Perceiver dropout	0.1	Dropout rate in Perceiver routing.
Training epochs	50	Number of full training epochs.
Batch size	4	Per-device training batch size.
Validation batch size	4	Batch size used during validation.
Train / validation / test split	90 / 5 / 5	Homology-aware split using 30% sequence-identity clusters.
Random seed	42	Random seed for reproducibility.
Initial learning rate	5×10^{-5}	Initial learning rate.
Weight decay	0.01	Weight decay used in training.
Gradient clipping	1.0	Global gradient norm clipping threshold.
Warmup steps	500	Number of linear warmup steps.
Maximum protein length	1024	Maximum target protein sequence length.
Maximum docking text length	512	Maximum tokenized length for docking text.
Maximum function text length	256	Maximum tokenized length for function.
Maximum ligand text length	256	Maximum length for ligand description.
Maximum SMILES length	256	Maximum tokenized length for ligand SMILES.
Maximum ligand atoms	128	Maximum number of ligand atoms loaded.
Ligand contrastive temperature	0.07	Temperature used in ligand-view InfoNCE.
Ligand contrastive effective weight	0.0025	Effective weight for ligand-side alignment.
Bridge-gate regularization weight	0.0	Weight for optional bridge-gate regularization.
EOS-token loss multiplier	2.0	Loss multiplier for the end-of-sequence token.
Repeat unlikelihood loss weight	0.0	Repeat unlikelihood loss is not enabled during training.

H.2 DockWizard generation hyperparameters

Table 10: DockWizard generation hyperparameters.

Hyperparameter	Value	Description
Number of input cases generated	100	Number of input rows generated in this setting.
Disable full injection after length	-1	Length threshold for disabling full-memory injection; negative value means no disabling.
Maximum total length	256	Maximum total decoding length.
Minimum generated length	80	EOS is masked before this generated length.
Typical decoding mass	0.95	Probability mass retained by typical decoding.
Typical minimum tokens kept	2	Minimum number of tokens kept after typical filtering.
Temperature	0.82	Sampling temperature.
Top-k	30	Top-k filtering threshold.
Top-p	0.92	Nucleus sampling threshold when nucleus decoding is used.
Repetition penalty	1.08	Global penalty applied to previously generated tokens.
Recent-window size	24	Window length for recent-token repetition control.
Recent-window frequency threshold	0.35	Threshold for recent-token frequency penalty.
Frequency penalty strength	2.5	Logit penalty strength for frequent recent tokens.
Dominance penalty threshold	0.42	Threshold for dominant-token suppression.
Dominance penalty strength	4.0	Penalty strength for dominant token patterns.
Dominance margin threshold	0.80	Margin threshold for applying dominant-token suppression.
Dominance logit suppression	2.0	Extra logit subtraction for dominant-token suppression.
Tail repeat block length	5	Length threshold for blocking severe tail repetition.
Tail repeat EOS trigger length	8	Length threshold for EOS encouragement under severe tail repetition.
Tail EOS bonus	3.0	EOS logit bonus under severe tail repetition.
EOS bias start length	96	Length at which soft EOS bias begins.
EOS bias full length	160	Length at which EOS bias reaches its maximum.
Maximum EOS bias	1.2	Maximum length-based EOS logit bias.

H.3 Compute resources

This section reports the main compute environment used for DockWizard training and generation experiments. The table lists the hardware and software versions relevant to experimental reproducibility.

Table 11: Compute resources used for DockWizard training and generation.

Resource	Value
GPU model	NVIDIA RTX PRO 6000 Blackwell Server Edition
Number of GPUs	2
GPU memory	95.6 GiB
NVIDIA driver version	590.44.01
System CUDA version	13.1
Python version	3.10.19
PyTorch version	2.10.0+cu128
PyTorch CUDA version	12.8
cuDNN version	91002
CPU model	Intel(R) Xeon(R) Platinum 8470Q
CPU cores	208
RAM	1.0 TiB
Operating system	Ubuntu 22.04.5 LTS
Kernel version	5.15.0-78-generic

I Limitations, Broader Impact

I.1 Limitations and future work

DockWizard is designed for ligand-binding-oriented protein generation and uses function text, docking-oriented text, and ligand multimodal representations to guide protein sequence generation. Although the experimental results show favorable performance on multiple automatic evaluation metrics, several limitations remain.

First, this paper mainly relies on computational evaluation, including folding confidence, structure comparison, DiffDock docking confidence, and Chai-1 complex-level score. These metrics provide automated assessment from the perspectives of structural plausibility, ligand docking tendency, and complex confidence, but they cannot replace wet-lab validation. Real ligand binding affinity, protein stability, solubility, expression level, and biological activity still require experimental verification. The results in this paper should therefore be understood as evidence for computational screening and candidate generation, rather than final confirmation of real biological function.

Second, DockWizard's performance depends on the quality of docking-oriented text. If docking text is noisy, overly generic, missing key interaction patterns, or mismatched with the actual protein–ligand pair, the model may not receive accurate local binding constraints. The current docking-language construction mainly relies on literature retrieval, rule-based extraction, and text cleaning. Future work may introduce finer-grained residue–ligand interaction annotations, binding-pocket descriptions, and experimentally validated binding evidence to improve the reliability of docking-language representations.

Third, the current ligand representation still has limited expressiveness. SMILES can describe ligand chemical topology, and ligand three-dimensional representation can provide spatial geometric information, but a single static conformation may not fully represent ligand conformational flexibility, solvent effects, or induced fit. Protein–ligand binding often involves protein conformational changes, ligand flexibility, and dynamic interaction networks, which are only limitedly modeled by the current model. Future work may incorporate molecular dynamics simulation, ensemble docking, conformer-aware ligand representations, or residue-level interaction graphs to better represent dynamic constraints during binding.

Fourth, although we use a homology-aware protein-level split based on 30% sequence-identity clusters to reduce homologous-sequence leakage, the model may still be affected by ligand-scaffold imbalance and binding-mode bias in the training data. Future work can further evaluate general-

ization through stricter ligand-scaffold splits, lower-identity cross-family splits, and broader out-of-distribution evaluation.

Finally, DockWizard uses no-KV-cache autoregressive inference during generation to keep prefix guidance and layer-wise full-memory injection consistently active at every generation step. This design helps maintain consistency in condition injection, but it also increases inference cost. Future work may explore more efficient cached decoding strategies that reduce generation time and memory usage while preserving the effect of staged condition injection.

I.2 Broader impact

DockWizard aims to provide a natural-language-guided multimodal generation framework for ligand-binding protein design. The method can help researchers generate candidate protein sequences that satisfy functional descriptions and ligand constraints at the early design stage, thereby reducing the cost of manual screening and trial-and-error exploration. Potential applications include enzyme engineering, molecular recognition, biosensing, therapeutic discovery, and protein–ligand interaction design.

On the positive side, DockWizard allows protein generation models to make fuller use of functional text, docking-oriented language, and ligand information, helping connect natural-language design intent with local ligand-binding constraints. This framework can serve as part of a computer-aided protein design workflow for proposing candidate sequences, narrowing the experimental screening space, and supporting more efficient hypothesis generation.

At the same time, such techniques should be used carefully. Protein generation models could be misused to design biological molecules with potential risks, especially without biosafety review and experimental validation. The use of DockWizard should therefore follow relevant biosafety norms, institutional review processes, and experimental validation standards. This paper does not advocate directly using computationally generated results in clinical settings, environmental release, or other high-risk scenarios. Any practical application should involve expert review, experimental validation, and risk assessment.

Responsible use. The released code and dataset are intended for in silico research use only. Generated protein sequences should not be used for clinical, environmental, or biological deployment without expert review, biosafety assessment, and wet-lab validation. Users should follow applicable institutional, legal, and biosafety regulations when adapting the method or analyzing generated candidates.

Generative AI usage. Generative AI tools were used only for language polishing during manuscript preparation. All scientific claims, methodological designs, experimental results, and conclusions were reviewed and verified by the authors.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the scope as computational ligand-binding protein generation, and the experiments and limitations are consistent with this scope.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations and future work are discussed in Appendix I, including computational-only validation, docking-text quality, ligand representation limits, data-distribution bias, and inference cost.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Appendix C provides theorem-style statements and proof sketches under simplified attention and linearized decoder assumptions.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper provides the evaluation protocol, pseudocode, dataset construction, baseline adaptation, model details, hyperparameters, and compute resources in Appendices B–H.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The supplementary material includes anonymized code, data-construction scripts, and instructions for reproducing the main experiments. The processed dataset will be released after curation, while the original protein records are derived from public UniProt entries.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Dataset splits, metric definitions, baseline adaptation, model details, generation settings, and hyperparameters are provided in Appendices B, E, F, G, and H.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We report sample-level paired uncertainty estimates and Holm-adjusted comparisons in Section 4 and Appendix B.6.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Runtime and peak-memory usage are reported in Appendix B, and the hardware/software environment is reported in Appendix H.

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The authors confirm compliance with the NeurIPS Code of Ethics; Appendix I discusses limitations, broader impact, biosafety considerations, and responsible use.

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Appendix I discusses potential positive applications in protein design and potential biological misuse risks.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [Yes]

    Justification: The supplementary material includes a responsible-use note stating that the released code and data-construction pipeline are intended for in silico research use only and that generated sequences require expert review, biosafety assessment, and wet-lab validation before any practical deployment.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The datasets, pretrained models, and software tools used in this work are publicly available or open-source, and the supplementary documentation lists their sources and usage terms.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The supplementary material includes the data-construction pipeline and documentation describing data sources, construction steps, fields, filtering rules, and intended research use. The curated processed dataset will be released after additional organization.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The study is computational and uses database/literature-derived data; no crowdsourcing or human-subject experiments are involved.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: No human-subject research or crowdsourcing is involved, so IRB approval is not applicable.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: Appendix I discloses that generative AI tools were used only for language polishing, with all scientific claims, methodological designs, experimental results, and conclusions reviewed and verified by the authors.