

ProtoCycle: Reflective Tool-Augmented Planning for Text-Guided Protein Design

Yutang Ge^{1,2*} Guojiang Zhao^{2*} Sihang Li³ Zheng Cheng⁴ Zifeng Zhao⁴
Hanchen Xia¹ Guolin Ke² Linfeng Zhang² Zhifeng Gao^{2†} Yuguang Wang^{1†}

¹Shanghai Jiao Tong University, School of Mathematical Sciences

²DP Technology

³University of Science and Technology of China

⁴AI for Science Institute

{lhx0627, yuguang.wang}@sjtu.edu.cn, {zhaogj, gaozf}@dp.tech

Abstract

Designing proteins that satisfy natural language functional requirements is a central goal in protein engineering. A straightforward baseline is to fine-tune generic instruction-tuned LLMs as direct text-to-sequence generators, but this is data- and compute-hungry. With limited supervision, LLMs can produce coherent plans in text yet fail to reliably realize them as sequences. This plan–execute gap motivates **ProtoCycle**, an agentic framework for protein design that uses LLMs primarily to drive a multi-round, feedback-driven decision cycle. ProtoCycle couples an LLM planner with a lightweight tool environment designed to emulate the iterative workflow of human protein engineers and uses LLM-driven reflection on tool feedback to revise plans. Trained with supervised trajectories and online reinforcement learning, ProtoCycle achieves strong language alignment while maintaining competitive foldability, and ablations show that reflection substantially improves sequence quality. Code and resources are available at <https://github.com/huggggooooo/ProtoCycle>.

1 Introduction

Designing proteins that achieve specified functional goals remains a central challenge with broad impact on enzyme engineering, therapeutics, and synthetic biology (Kortemme, 2024; Arnold, 2017; Ebrahimi and Samanta, 2023). Recent deep generative models have already shown encouraging

progress toward this goal. On one hand, protein language models such as ProGen (Madani et al., 2023) and the ESM family (Rives et al., 2021; Lin et al., 2023) are pretrained on large sequence corpora and can be adapted to design tasks. On the other hand, text-guided design frameworks such as ProteinDT (Liu et al., 2025b), PAAG (Yuan et al., 2024), ProDVa (Liu et al., 2025a), and Pinal (Dai et al., 2025) explicitly condition on natural language to generate protein sequences.

Because requirements are expressed in natural language while proteins are discrete amino-acid sequences over a 20-letter alphabet, a straightforward baseline is to adapt a generic instruction-tuned LLM to map requirement text directly to protein sequences, following instruction-tuned generation in other domains (Zhao et al., 2025; Wei et al., 2021). However, our analysis in Section 2 reveals a consistent pattern: compared to direct sequence generation, LLMs appear stronger at planning in text. Under lightweight adaptation, they achieve higher plan quality and exhibit lower estimated uncertainty when producing plans, yet translating those plans into residue-level decisions that yield satisfying sequences remains challenging.

This observation aligns with how protein design is carried out in practice: a protein engineer rarely goes from a specification to an ideal sequence in a single step. Instead, reaching a satisfying design typically requires an iterative design–evaluate–revise loop over multiple rounds. Concretely, practitioners typically start from an existing scaffold, make localized edits around functional sites, and

*Equal contribution.

†Corresponding authors.

evaluate candidates using structure- or function-related signals. When the outcome is unsatisfactory, they revisit earlier choices, adjust their strategy, and iterate (Wang et al., 2021; Jäckel et al., 2008).

Accordingly, we propose **ProtoCycle**, a text-guided protein generation framework that casts protein design as an iterative cycle of planning, tool calling, evaluation, and revision. In ProtoCycle, the LLM primarily serves as a high-level planner, responsible for requirement decomposition, tool selection, strategy updates, reflection, and termination decisions, while sequence generation and local editing are delegated to a lightweight set of specialized tools that can be called frequently and efficiently. We optimise the planner in this multi-step environment with a combination of supervised fine-tuning and reinforcement learning, so that it learns when to call which tools, how to interpret feedback, and how many cycles to run before terminating. Notably, ProtoCycle reaches competitive performance on our benchmarks using only $\sim 2,500$ training instances for planner optimization, whereas frontier end-to-end baselines such as Pinal are trained on $\sim 1.7\text{B}$ protein-text pairs (Dai et al., 2025).

This work makes the following contributions:

- We empirically characterise the behaviour and limitations of generic LLMs for text-guided protein generation.
- We introduce a new paradigm for text-guided protein design by formulating it as a multi-round decision-making problem.
- We propose **ProtoCycle**, an expert-workflow-inspired agent with a lightweight toolkit.
- We empirically validate ProtoCycle and show via ablations that reflective revision improves sequence quality.

2 Diagnosing Generic Instruction-tuned LLMs as Direct text-guided Protein Generators

This section studies a natural baseline: directly deploying generic instruction-tuned LLMs as zero-shot text-guided protein sequence generators. Our goal is to test whether their strong textual reasoning reliably translates into residue-level sequences. We probe this using a simple task and a token-level uncertainty estimator (Ma et al., 2025).

Table 1: Qualitative interpretation of high/low aleatoric (AU) and epistemic (EU) uncertainty.

	Low	High
AU	Unambiguous problem	Ambiguous problem
EU	Model is confident	Model lacks knowledge

2.1 Problem Setup

2.1.1 Task: text-guided Protein Generation

We use the protein-design subset of Mol-Instructions (Fang et al., 2024) as a benchmark, where each example pairs a natural-language description with a target sequence. We run a data-scaling sanity check on Qwen2.5-7B (Qwen et al., 2025) and adopt a chain-of-thought prompting format (Wei et al., 2023) that elicits a textual design rationale followed by an amino-acid sequence.

We study three base models (Qwen2.5-7B, Qwen2.5-72B (Qwen et al., 2025), and Llama-3.1-8B (Dubey et al., 2024)) under three usage modes: zero-shot, in-context learning (Brown et al., 2020), and supervised fine-tuning (SFT).

2.1.2 Token-level Uncertainty Estimation

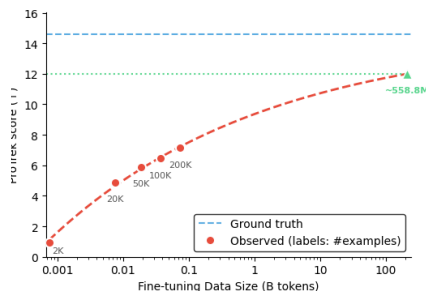
To probe model behaviour, we use Logits-induced Token Uncertainty (LogTokU) (Ma et al., 2025), which decomposes predictive uncertainty into: **aleatoric uncertainty (AU)**, capturing ambiguity of the next-token distribution, and **epistemic uncertainty (EU)**, capturing how weakly supported the prediction is by the model’s evidence. Table 1 summarises the qualitative regimes for high and low AU and EU. Appendix A provides a token-level illustration (Figure 7) and full formulas.

2.2 Behaviour of Generic LLMs on text-guided Protein Generation

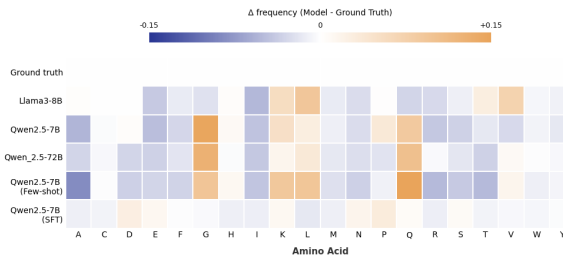
2.2.1 A data-scaling sanity check

We fine-tune Qwen2.5-7B on subsets of the Mol-Instructions protein-design split (about 200K protein-text pairs). For each run, we subsample 2K, 20K, 50K, 100K, or the full 200K examples and fine-tune for two epochs, then evaluate text-sequence alignment on a held-out test set using the ProTrek score (Su et al., 2025).

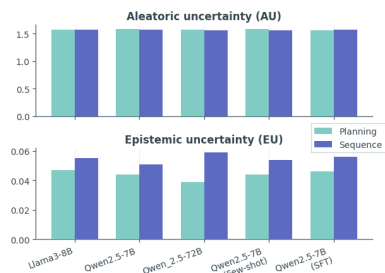
Figure 1a shows that more data monotonically improves ProTrek, but gains are modest: increasing the dataset from 2K to 200K only lifts the score from roughly 1 to 7, far below the ground-truth level of ≈ 14.6 . A simple power-law fit (Kaplan et al., 2020) suggests that reaching ProTrek around



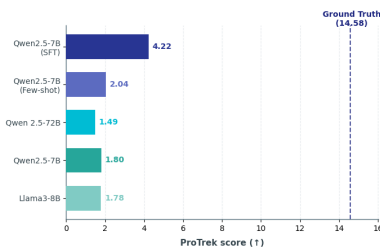
(a) Effect of fine-tuning data size.



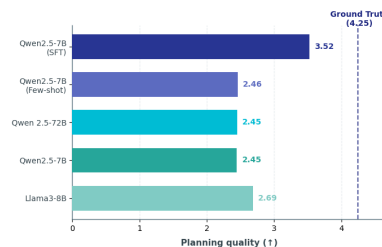
(b) Deviation of amino-acid composition of generated sequences from ground truth.



(c) Aleatoric and epistemic uncertainty on planning and sequence tokens.



(d) Text–sequence alignment score.



(e) Quality of textual design plans.

Figure 1: Behaviour of generic instruction-tuned LLMs when directly used as text-guided protein generators. (a) Achieving strong text–sequence alignment would require very large data regimes. (b) Generated amino-acid compositions exhibit clear collapse, with SFT reducing these distortions to some extent. (c) Token-level uncertainty indicates that, compared to planning tokens, sequence tokens lie in a more evidence-poor regime. (d) Generic LLMs lag far behind ground truth, even after SFT. (e) In contrast, SFT substantially improves the quality of textual design plans, highlighting a gap between planning competence and sequence-generation ability.

12 would require roughly 6×10^8 supervised examples for Qwen2.5-7B, and approaching ground-truth scores requires on the order of 10^9 pairs, similar in magnitude to Pinal (Dai et al., 2025), which trains a 16B structure-conditioned model on 1.7B protein–text pairs.

2.2.2 Sequence Collapse and Language Alignment Quality

We next analyse behaviour under the chain-of-thought protocol where the model first outputs a textual plan and then a sequence.

We begin with marginal amino-acid usage. For each model and usage mode, we aggregate generated sequences and compare residue frequencies against those of the ground-truth sequences (see Appendix B.1). Figure 1b visualises the resulting deviations. Generic LLMs exhibit pronounced distortions: some residues are heavily overused, while others rarely appear. SFT shifts these distributions toward the ground-truth profile, alleviating the most obvious collapse patterns.

However, composition alone does not guarantee good designs. Figure 1d reports ProTrek scores as language alignment metric. All generic LLMs

remain far below ground truth, even after SFT, indicating that better marginal statistics do not translate into strong language alignment.

2.2.3 Planning Quality

We also evaluate the natural-language design rationales produced before sequences. Plans are scored by DeepSeek-R1 (DeepSeek-AI et al., 2025) as an automatic judge; the rubric and its validation against expert ratings are given in Appendix B.2.

Figure 1e shows that SFT yields clear and consistent improvements in plan quality across models and usage modes. Yet the corresponding sequence-level gains are small, suggesting that the models learn to describe what should be designed much better than they learn to implement these plans at the sequence level.

2.2.4 Uncertainty Asymmetry

Figure 1c compares AU and EU on planning versus sequence tokens. AU is broadly similar in both regions, suggesting that, locally, predicting the next English word and the next residue are treated as similarly ambiguous. The asymmetry appears in EU: it is systematically higher on sequence tokens

and aligns with the poor sequence quality, indicating that models operate in an evidence-poor regime when mapping requirements to concrete amino-acid choices.

Under our text-only SFT recipe, this pattern largely persists. SFT improves planning quality, but sequence-level structure and EU show no consistent gains. This supports a view in which SFT mainly refines a text-rich planning subspace, while providing too little new evidence to close the knowledge gap in the description-to-sequence mapping, so sequences become more format-plausible but remain functionally weak.

3 Method

We now introduce **ProtoCycle**, which treats text-guided protein design as a multi-step decision process. In this view, LLM acts as a high-level planner that only decides how to design, while the actual sequence generation is delegated to a set of external tools (Yao et al., 2022; Schick et al., 2023). We first present an overview in Fig 2, then detail the planner interface and reflection mechanism.

3.1 Problem Formulation

Formally, given a design requirement r , the agent interacts with a tool environment T over multiple rounds. At round t , the planner outputs decision s_t and action a_t :

$$(a_t, s_t) = f(s_{t-1}, t_{t-1}, \dots, s_1, t_1, r), \quad (1)$$

where $t_i = T(a_i)$ denotes the tool feedback, which is a summary of tool results (see Section 3.3 for detail).

Each action a_i is factorised as

$$a_i = (a_i^{\text{type}}, a_i^{\text{arg}}),$$

where a_i^{type} specifies which tool to invoke, and a_i^{arg} specifies the input arguments for that tool.

3.2 Tool Environment

To emulate the expert design process under realistic compute constraints, we instantiate a lightweight tool environment T with three corresponding tools: scaffold generation, functional-site design, and evaluation. Additional implementation details, query templates, and back-end mappings are provided in Appendix C.

Scaffold generation Given a free-form requirement text, the scaffold generation tool retrieves and merges candidates from multiple protein knowledge bases (uni, 2025; Bansal et al., 2022; Blum et al., 2025; Binns et al., 2009), returning n scaffolds. Appendix C.1 shows more details.

Functional-site design. Given a selected scaffold and a local description text, the functional-site design tool generates site-level variants on top of ESM2-3B (Lin et al., 2023). Full procedure is shown in Appendix C.2.

Evaluation. We evaluate candidates along two axes: **Language Alignment** and **Foldability**. Language Alignment is measured by ProTrek-650M (Su et al., 2025). Foldability is assessed using Chai-1 (Chai Discovery, 2024) (no-MSA), reporting pTM, pLDDT, and PAE (Jumper et al., 2021), see Appendix F.2 for definitions.

3.3 Model Architecture

Given the output (s_t, a_t) at each step as defined in 1, the state s_t consists of two parts, <think> and <plan>, while the action a_t corresponds to a single <tool_call> token.

At $t = 1$ (first round).

- <think>: the planner decomposes the requirement r into sub-requirements $\{r_1, r_2, \dots, r_n\}$, aiming to expose finer-grained design goals.
- <plan>: the planner maps each r_j to a tool configuration $r_j \mapsto a_j = (\text{type}(r_j), \text{arg}(r_j))$, and proposes an execution order $a_{j_1} \rightarrow a_{j_2} \rightarrow \dots \rightarrow a_{j_n}$.

At $t \geq 2$ (subsequent rounds).

- <think>: the planner summarizes the performance of the current strategy and reflects on the previous trajectory $\{s_{t-1}, t_{t-1}, \dots, s_1, t_1\}$, deciding whether to (i) continue the current plan, (ii) modify it, or (iii) terminate.
- <plan>: the planner makes this high-level choice explicit, e.g., by executing the next scheduled sub-goal r_{j_t} or switching to an updated sub-goal $r_{j_t}^*$.

After generating s_t (i.e., <think> and <plan>), the planner produces the tool action $a_t = (a_t^{\text{type}}, a_t^{\text{arg}})$. In practice, we serialize a_t into a JSON-like tag such as <tool_call>{ "name":

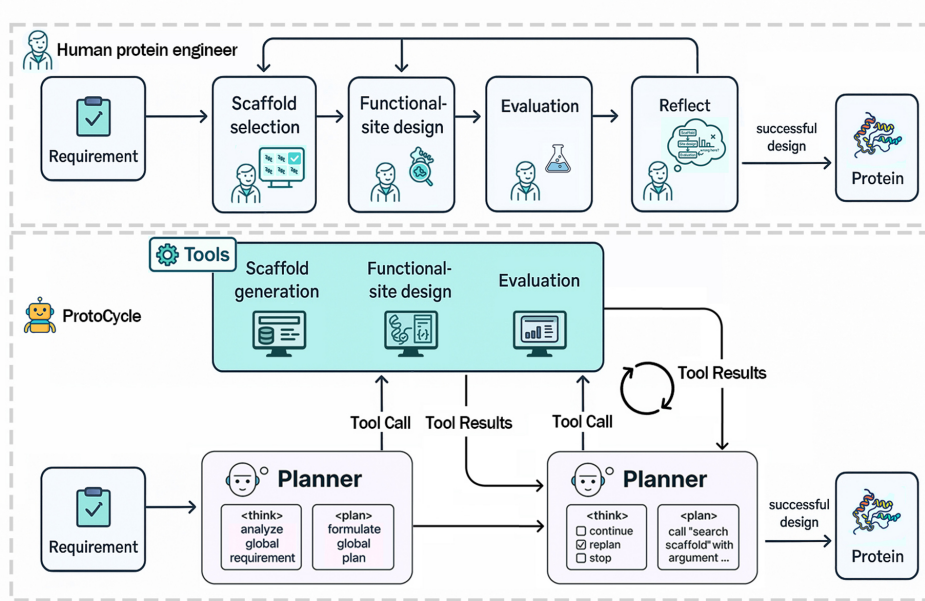


Figure 2: Overview of ProtoCycle compared to a human protein engineer. **Top:** Human workflow, which iterates between scaffold selection, functional-site design, evaluation, and reflection until a satisfactory protein is obtained. **Bottom:** ProtoCycle, where a planner interacts with three tools (scaffold generation, functional-site design, and evaluation) via `<think>`/`<plan>`/`<tool_call>` steps and receives summarized tool feedback to revise its strategy.

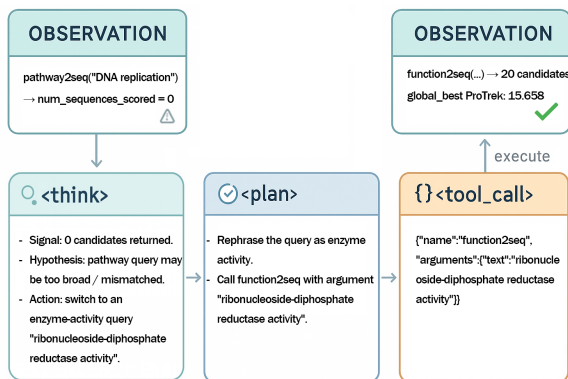


Figure 3: An example of **reflection** in ProtoCycle.

a_t^{type} , "argument": a_t^{arg} }/`</tool_call>`. A lightweight runtime then parses this tag and invokes the corresponding tool. Figure 3 illustrates a concrete example at $t \geq 2$. We provide a full example trajectory in Appendix E.1.2, and a complete case study in Appendix I.

Tool feedback summary. At each round of scaffold generation and functional-site design, the tools produce a set of sequences $P_i = \{p_{i1}, \dots, p_{im}\}$, which are scored by ProTrek-35M to obtain $U_i = \{u_{i1}, \dots, u_{im}\}$. We compute (i) the number of sequences m ; (ii) the current-round best score $w_i = \max(U_i)$; (iii) the global best score $b_{i-1} = \max(\bigcup_{j=1}^{i-1} U_j)$; and (iv) the improvement $\Delta_i =$

$$w_i - b_{i-1}.$$

The tool feedback summary t_i encapsulates these statistics and serves as a compact signal of the quality of P_i , which the planner then uses to decide how to proceed in the next round.

Termination mechanism. When the planner decides to stop further tool use and prepare a final sequence, it triggers the evaluation tool to re-score the current top- k candidates (we use $k=5$). The evaluation results are returned to the planner, which then either terminates and outputs the best-scoring sequence or resumes planning and invokes additional tool calls to further refine the design when improvements are still needed.

3.4 Training

Following recent cold-start + reinforcement learning (RL) recipes (Guo et al., 2025; Zhao et al., 2025; Li et al., 2025; Wei et al., 2025), we train ProtoCycle in two stages: supervised fine-tuning (SFT) to learn the basic `<think>`/`<plan>`/`<tool_call>` protocol and tool-usage conventions, followed by online RL in the tool environment of Section 3.2. Our training pipeline is implemented with the OpenAgentRL framework (Yu et al., 2025).

3.4.1 Supervised Fine-tuning

We first train the planner with supervised fine-tuning on trajectories collected in the tool environment. Each training example is a sequence

$$(r, s_1, t_1, s_2, t_2, \dots, s_n, t_n).$$

Formally, the SFT objective for one trajectory is

$$\mathcal{L}_{\text{SFT}}(r, s_{1:n}, t_{1:n}) = -\frac{1}{n} \sum_{i=1}^n \log p_{\theta}(s_i | r, s_{<i}, t_{<i}),$$

which corresponds to a standard cross-entropy loss applied only on the planner states s_1, \dots, s_n .

3.4.2 Online Reinforcement Learning

After SFT, we further optimise the planner with online reinforcement learning in the real tool environment. We treat the planner as a stochastic policy and update it with Group Relative Policy Optimization (GRPO) (Shao et al., 2024).

The reward $R(\tau)$ is a shaped signal that combines several components, encouraging (i) well-formed planner outputs, (ii) reasonable tool usage, (iii) reflection after poor intermediate feedback, and (iv) solving the task in a moderate number of rounds. The exact form and coefficients of these terms are given in Appendix D.

4 Experiments

4.1 Design from textual descriptions

Dataset We use the protein-design subset of Mol-Instructions (Fang et al., 2024). We evaluate on three disjoint held-out test splits of 100 instances each (Eval-A, Eval-B, Eval-C) sampled from the dataset. We report main results averaged over the three splits, and provide split-wise results in Appendix G. For SFT, we train on a disjoint set of 2,000 instances (see Appendix E.1 for details). For RL, we train for 5 epochs with 100 episodes per epoch, sampling requirements uniformly from the non-test pool in the real tool environment. Training details are in Appendix E.

Baselines We use Qwen2.5-7B as the base language model for our planner and refer to the full agent (SFT + RL) as ProtoCycle. As generic LLM baselines in the same tool environment (Section 3.2), we instantiate tool-interacting agents whose planners are Qwen2.5-7B, Qwen3-8B, and Qwen2.5-72B (Qwen et al., 2025), an example

of the full prompting format is provided in Appendix E.1.3. We further compare against two text-guided protein design methods: Pinal (Dai et al., 2025), a large two-stage structure-conditioned model trained on about 1.7B protein-text pairs, and ProDVa (Liu et al., 2025a), which couples a text encoder and a protein LM with fragment retrieval. To isolate the effect of RL, we report both the supervised-only planner (ProtoCycle-SFT) and the planner after SFT followed by online RL (ProtoCycle-RL).

Metrics We evaluate models using three groups of metrics (formal definitions in Appendix F), largely following prior work on text-guided protein design (Kuang et al., 2025; Liu et al., 2025a). **Sequence plausibility** is measured by perplexity (PPL) (Jelinek et al., 1977) under ESM2-3B (Lin et al., 2023) and the percentage of repeated residues (Repeat) following PDFBench (Kuang et al., 2025). **Foldability** is assessed using predicted TM-score (pTM), mean per-residue confidence (pLDDT), and mean predicted aligned error (PAE) (Jumper et al., 2021) from Chai-1 (Chai Discovery, 2024), summarizing how likely a sequence is to fold into a stable three-dimensional structure. **Language alignment** is evaluated with the ProTrek score (Su et al., 2025), the EvoLLaMA score (Liu et al., 2024), and retrieval accuracy (Kuang et al., 2025), which quantify how well the designed sequences match their textual requirements in a joint embedding space.

Results Table 2 compares generic LLM planners, prior text-guided protein design methods, and our ProtoCycle variants on sequence plausibility, foldability, and language alignment. Our key observations are as follows. (1) **ProtoCycle-RL is competitive with or better than specialized text-guided baselines.** It achieves the strongest language alignment overall, improving ProTrek by 3.66% over Pinal and by 21.97% over ProDVa, while maintaining competitive foldability: compared to Pinal, pTM/pLDDT drop by 2.15%/0.36% and PAE increases by 9.98%, while compared to ProDVa we improve all foldability metrics. (2) **Online RL yields substantial improvements over SFT, especially on language alignment.** Compared to ProtoCycle-SFT, ProtoCycle-RL improves ProTrek by about 17.43% and increases retrieval accuracy by 11.43%. Meanwhile, RL also improves plausibility, reducing PPL and Repeat by 6.85% and 12.16%, respectively.

Model	Sequence Plausibility		Foldability			Language Alignment		
	PPL↓	Repeat↓	pTM↑	pLDDT↑	PAE↓	ProTrek↑	EvoLLaMA↑	Retrieval↑
Natural	4.737	2.129	0.762	0.815	9.443	14.628	0.328	0.848
Qwen2.5-7B-Agent	8.235	5.153	0.542	0.699	15.299	6.926	0.261	0.523
Qwen2.5-72B-Agent	7.414	5.341	0.618	0.714	13.343	8.791	0.267	0.563
Qwen3-8B-Agent	7.227	3.795	0.650	0.723	13.493	8.705	0.277	0.573
ProDVa	5.265	1.580	0.765	0.800	8.761	12.037	0.317	0.730
Pinal	<u>3.990</u>	9.317	0.792	0.825	7.768	<u>14.162</u>	<u>0.318</u>	<u>0.807</u>
ProtoCycle-SFT (ours)	4.149	2.902	0.734	0.807	10.200	12.502	0.317	0.840
ProtoCycle-RL (ours)	3.865	<u>2.549</u>	<u>0.775</u>	<u>0.822</u>	<u>8.543</u>	14.681	0.323	0.936

Table 2: Mol-Instructions protein design results avg. over three test splits (**best**, second-best).

Model	Train	pLDDT↑	ProTrek↑	Kw. Rec.↑
Natural	–	0.79	10.02	1.0
Pinal	✓	0.75	11.78	0.39
ProDVa(Mol-Inst)	×	0.78	4.65	0.17
ProDVa(CAMEO)	✓	0.82	11.05	0.36
ProtoCycle-SFT	×	0.78	9.08	<u>0.48</u>
ProtoCycle-RL	×	<u>0.80</u>	<u>11.17</u>	0.59

Table 3: Generalization to CAMEO (**Best**, Second Best). Train indicates whether the model is trained on keyword-style data (✓) or not (×). Kw. Rec. denotes keyword recovery.

4.2 Generalization to CAMEO

Setup. We evaluate cross-dataset generalization on CAMEO (Haas et al., 2018), a keyword-style protein-design dataset (see an example in Table 11). All ProtoCycle variants use the same tool environment and evaluation protocol as in Section 4.1, and are trained only on the Mol-Instructions protein-design subset (i.e., without any keyword-style samples in the training data). We compare against Pinal, who is trained with keyword-style data (✓) and ProDVa with/without keyword-style training data (✓/×). In addition to the metrics in Section 4.1, we evaluate **keyword recovery** (Kw. Rec.), which measures how many of the reference function keywords in the input are recovered from the designed sequence via InterProScan (Jones et al., 2014). See Appendix F.3 for the full definition.

Results. Table 3 summarizes the main metrics. Although ProtoCycle is not trained to interpret compact keyword lists as requirements, it still transfers well to CAMEO. ProtoCycle-RL achieves strong language alignment and foldability, with performance comparable to Pinal, whose train dataset contains 800M keyword-protein pairs and ProDVa (CAMEO), who is trained directly on the CAMEO

Tool	ProTrek↑	PAE↓	pLDDT↑
Scaffold search	11.42	8.96	0.83
Functional-site design	12.87	10.73	0.80

Table 4: Tool output quality when each tool is invoked once with arguments proposed by the trained planner.

Component	Latency	Output
Scaffold search	4 s/round	N retrieved sequences
Functional-site design	20 s/seq	1 edited sequence
Eval (ProTrek-35M)	3 s/round	scores for N seqs
Eval (ProTrek-650M)	40 s/round	scores for N seqs

Table 5: Wall-clock latency of each component.

subset with 391M training examples, demonstrating robust cross-format generalization. Full metrics are reported in Table 10.

4.3 Tool Efficiency

Setup We quantify the effectiveness and runtime of our tool environment by invoking each tool in isolation with arguments produced by the trained ProtoCycle planner. We report standard foldability and alignment metrics on the tool outputs, and measure wall-clock latency per invocation on our implementation.

Results Table 4 shows that, under effective prompts, functional-site design improves language alignment relative to the raw scaffold-search output while keeping foldability metrics broadly comparable, suggesting that the tool can make meaningful local edits without catastrophically degrading structural confidence. Table 5 shows that scaffold search takes a few seconds per round (4 s/round), whereas functional-site design is the dominant per-sequence cost (20 s/seq); during interaction we therefore use ProTrek-35M for fast per-round feedback and reserve ProTrek-650M for final reporting.

4.4 Ablation: Reflection Mechanism

Setup To isolate the effect of reflection, we compare three planner variants under the same tool environment and evaluation protocol as in Section 4.1. (i) **Workflow**: we hard-code the tool order (SCAFFOLD→LOCAL-DESIGN→EVAL) in the prompt. (ii) **SFT (no reflection)**: the planner outputs `<think>` and `<plan>` but does not perform explicit reflection on previous plans; we fine-tune Qwen2.5-7B on non-reflective trajectories to learn the format and basic analysis. (iii) **ProtoCycle**: our reflective planner, reported as ProtoCycle-SFT and ProtoCycle-RL (SFT+RL).

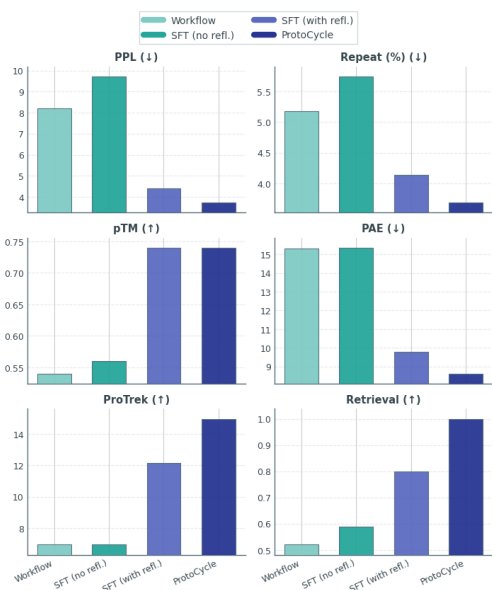
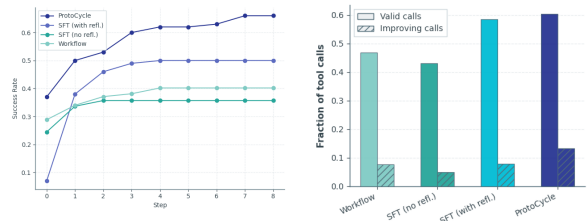


Figure 4: Reflection ablation on final plausibility, foldability, and alignment metrics.

Overall sequence quality Figure 4 shows that incorporating reflection yields substantially better designs. Reflective variants achieve nearly twice the language alignment score compared to their non-reflective counterparts, and also deliver a significant improvement in foldability metrics.

Notably, SFT (no reflection) behaves similarly to the fixed Workflow, suggesting that learning the `<think>/<plan>` format and performing one-shot analysis alone is insufficient; without the ability to reflect on tool feedback and revise strategy, the planner largely degenerates into executing a mechanical pipeline.

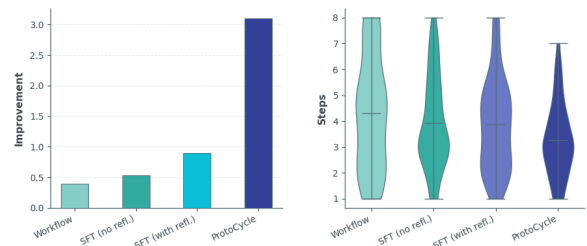
Effect of reflection across interaction steps To understand how reflection helps, we track success rate as a function of the interaction step. For instance n at step t , let $\text{succ}(t) = \frac{1}{m} \sum_{n=1}^m \text{succ}_n(t)$,



(a) Success rate vs. step.

(b) Valid / improving tool calls.

Figure 5: Reflection improves step-wise optimization and tool-use efficiency.



(a) Net improvement over the first step.

(b) Wasted steps after the best score.

Figure 6: Reflection improves decision quality and compute-aware stopping.

where $\text{succ}_n(t) = \mathbf{1}[P_n^*(t) \geq 12]$, $P_n^*(t) = \max\left(\bigcup_{i=1}^t P_n(i)\right)$, $P_n(i)$ is the set of ProTrek scores of all candidates produced at step i , and m is the number of test instances. As shown in Figure 5a, reflection increases success rates by roughly 25-40% over non-reflective baselines. Moreover, ProtoCycle (with RL) continues to improve with more steps, consistent with learning to explore and revise strategies based on intermediate feedback rather than repeatedly executing a static plan.

We further quantify tool-use quality by counting (i) **valid** tool calls that execute successfully and return non-empty outputs, and (ii) **improving** calls that strictly increase the current best score. Figure 5b shows that reflection increases both the valid-call rate (about +20% relative) and, more notably, the improving-call rate (about +40% relative), suggesting that reflection helps the planner not only invoke tools correctly but also choose arguments that lead to measurable progress.

Reflection induces more rational decisions. Finally, we measure whether the planner makes decisions that improve upon the initial tool outcome. For each instance n , we compute the net gain $P_n^* - P_n^*(1)$, where $P_n^* = \max_t P_n^*(t)$. Figure 6a shows that without reflection the net improvement

is small, whereas reflection—especially with RL—yields substantially larger gains (e.g., nearly a 6× improvement over the Workflow baseline in our results). This indicates that reflective planners are better at turning intermediate feedback into concrete strategy revisions.

We also examine whether the planner stops once further tool calls become unproductive. Let $t_{\text{best}}(n) = \arg \max_t P_n^*(t)$ and define $\text{waste}(n) = t_{\text{term}}(n) - t_{\text{best}}(n)$, where t_{term} is the terminal step. As shown in Figure 6b, reflection reduces wasted steps, indicating that the planner can recognize diminishing returns and terminate more promptly instead of continuing a fixed workflow.

5 Conclusion

Motivated by human protein-engineering workflows and our finding that generic LLMs are more reliable planners than residue-level generators, we propose ProtoCycle, which couples an LLM planner with a lightweight tool environment and an reflection mechanism for feedback-driven replanning. With only modest supervision and online interaction, ProtoCycle achieves foldability comparable to current state-of-the-art methods while outperforming them on language alignment.

Limitations

One limitation of ProtoCycle is that our current functional-site design tool is intentionally lightweight to fit realistic compute budgets, so it can improve the odds of finding plausible candidates but cannot guarantee an "ideal" sequence that realizes the intended binding/catalytic geometry. This is consistent with broader findings in protein design: while de novo design has advanced rapidly, reliably designing complex function (especially high-specificity binding/catalysis) remains challenging and often still relies on iterative optimization and/or experimental screening rather than strict guarantees.

Another limitation is the throughput–quality trade-off inherent to an agentic workflow. Because ProtoCycle invokes structure-informed tools during planning and evaluation, it incurs higher wall-clock time and compute than one-shot generators.

Ethics Statement

This work studies an agentic framework for text-guided protein sequence design. We do not use or

collect personal user data, and our experiments rely on publicly available protein resources and benchmarks (including Mol-Instructions and CAMEO). We follow the original dataset and database licenses/terms to ensure copyright- and license-compliant use, and we apply basic quality control (e.g., validity checks and filtering) to reduce malformed or misleading samples.

Human experts were involved in a limited way during dataset construction. Specifically, experts provided a small number of seed examples that were used to prompt an external language model to synthesize additional training instances, and they performed a brief manual spot-check (around 10 samples) to flag obvious errors or inconsistencies. We did not collect or report any personally identifying information about the experts, and we treat this involvement as a lightweight sanity check rather than a comprehensive human-subject study.

We acknowledge that synthetic data and benchmark datasets can contain noise and systematic biases (e.g., uneven coverage across organisms, protein families, or functional annotations), which may propagate into model behavior and affect the diversity and reliability of generated candidates. We therefore encourage careful downstream validation and transparent reporting of failure cases when applying the method to new domains.

Finally, protein design is potentially dual-use. Our work is intended for benign scientific research and does not provide wet-lab protocols or procedural instructions for harmful applications. Any real-world deployment should follow institutional oversight and applicable biosecurity policies, including appropriate screening and usage constraints.

References

- 2025. Uniprot: the universal protein knowledgebase in 2025. *Nucleic acids research*, 53(D1):D609–D617.
- Josh Abramson, Jonas Adler, Jack Dunger, Richard Evans, Tim Green, Alexander Pritzel, Olaf Ronneberger, Lindsay Willmore, Andrew J. Ballard, Joshua Bambrick, Sebastian W. Bodenstein, and 1 others. 2024. [Accurate structure prediction of biomolecular interactions with AlphaFold 3](#). *Nature*, 630(8016):493–500.
- Frances H Arnold. 2017. Directed evolution: bringing new chemistry to life. *Angewandte Chemie (International Ed. in English)*, 57(16):4143.
- Parit Bansal, Anne Morgat, Kristian B Axelsen, Venkatesh Muthukrishnan, Elisabeth Coudert, Lucila

- Aimo, Nevila Hyka-Nouspikel, Elisabeth Gasteiger, Arnaud Kerhornou, Teresa Batista Neto, and 1 others. 2022. Rhea, the reaction knowledgebase in 2022. *Nucleic acids research*, 50(D1):D693–D700.
- David Binns, Emily Dimmer, Rachael Huntley, Daniel Barrell, Claire O’donovan, and Rolf Apweiler. 2009. Quickgo: a web-based tool for gene ontology searching. *Bioinformatics*, 25(22):3045–3046.
- Matthias Blum, Antonina Andreeva, Laise Cavalcanti Florentino, Sara Rocio Chuguransky, Tiago Grego, Emma Hobbs, Beatriz Lazaro Pinto, Ailsa Orr, Typhaine Paysan-Lafosse, Irina Ponamareva, and 1 others. 2025. Interpro: the protein sequence classification resource in 2025<? mode longmeta?>. *Nucleic acids research*, 53(D1):D444–D456.
- Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. [Language Models are Few-Shot Learners](#). *arXiv preprint*. ArXiv:2005.14165.
- Chai Discovery. 2024. [Chai-1: Decoding the molecular interactions of life](#). *bioRxiv*.
- Fengyuan Dai, Shiyang You, Yudian Zhu, Yuan Gao, Lihao Fu, Xibin Zhou, Jin Su, Chentong Wang, Yuliang Fan, Xiaoxiao Ma, Xianjun Deng, Letong Yu, Hui Qian, Yan He, Yitao Ke, Chenchen Han, Xing Chang, Liangzhen Zheng, Sheng Wang, and 7 others. 2025. [Toward de novo protein design from natural language](#). *bioRxiv*.
- DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. [DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning](#). *arXiv preprint*. ArXiv:2501.12948.
- Abhimanyu Dubey and 1 others. 2024. [The llama 3 herd of models](#). *Preprint*, arXiv:2407.21783.
- Sasha B Ebrahimi and Devleena Samanta. 2023. Engineering protein-based therapeutics through structural and chemical design. *Nature communications*, 14(1):2411.
- Yin Fang, Xiaozhuan Liang, Ningyu Zhang, Kangwei Liu, Rui Huang, Zhuo Chen, Xiaohui Fan, and Hua-jun Chen. 2024. [Mol-Instructions: A Large-Scale Biomolecular Instruction Dataset for Large Language Models](#). *arXiv preprint*. ArXiv:2306.08018.
- Yu Gu, Robert Tinn, Hao Cheng, Michael Lucas, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. 2021. [Domain-specific language model pretraining for biomedical natural language processing](#). *ACM Transactions on Computing for Healthcare*, 3(1):1–23.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.
- Jürgen Haas, Alessandro Barbato, Dario Behringer, Gabriel Studer, Steven Roth, Martino Bertoni, Khaled Mostaguir, Rafal Gumienny, and Torsten Schwede. 2018. Continuous automated model evaluation (cameo) complementing the critical assessment of structure prediction in casp12. *Proteins: Structure, Function, and Bioinformatics*, 86:387–398.
- Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. 1977. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1):S63–S63.
- Philip Jones, David Binns, Hsin-Yu Chang, Matthew Fraser, Weizhong Li, Craig McAnulla, Hamish McWilliam, John Maslen, Alex Mitchell, Gift Nuka, and 1 others. 2014. Interproscan 5: genome-scale protein function classification. *Bioinformatics*, 30(9):1236–1240.
- John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, and 15 others. 2021. [Highly accurate protein structure prediction with AlphaFold](#). *Nature*, 596(7873):583–589.
- Christian Jäckel, Peter Kast, and Donald Hilvert. 2008. [Protein Design by Directed Evolution](#). *Annual Review of Biophysics*, 37(1):153–173.
- Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. [Scaling Laws for Neural Language Models](#). *arXiv preprint*. ArXiv:2001.08361.
- Tanja Kortemme. 2024. De novo protein design—from new structures to programmable functions. *Cell*, 187(3):526–544.
- Jiahao Kuang, Nuowei Liu, Jie Wang, Changzhi Sun, Tao Ji, and Yuanbin Wu. 2025. Pdfbench: A benchmark for de novo protein design from function. *arXiv preprint arXiv:2505.20346*.
- Jiatong Li, Weida Wang, Qinggang Zhang, Junxian Li, Di Zhang, Changmeng Zheng, Shufei Zhang, Xiaoyong Wei, and Qing Li. 2025. Mol-r1: Towards explicit long-cot reasoning in molecule discovery. *arXiv preprint arXiv:2508.08401*.

- Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Robert Verkuil, Ori Kabeli, Yaniv Shmueli, Allan Dos Santos Costa, Maryam Fazel-Zarandi, Tom Sercu, Salvatore Candido, and Alexander Rives. 2023. [Evolutionary-scale prediction of atomic-level protein structure with a language model](#). *Science*, 379(6637):1123–1130.
- Nuowei Liu, Jiahao Kuang, Yanting Liu, Tao Ji, Changzhi Sun, Man Lan, and Yuanbin Wu. 2025a. [Protein Design with Dynamic Protein Vocabulary](#). *arXiv preprint*. ArXiv:2505.18966.
- Nuowei Liu, Changzhi Sun, Tao Ji, Junfeng Tian, Jianxin Tang, Yuanbin Wu, and Man Lan. 2024. [Evolama: Enhancing llms’ understanding of proteins via multimodal structure and sequence representations](#). *arXiv preprint arXiv:2412.11618*.
- Shengchao Liu, Yanjing Li, Zhuoxinran Li, Anthony Gitter, Yutao Zhu, Jiarui Lu, Zhao Xu, Weili Nie, Arvind Ramanathan, Chaowei Xiao, Jian Tang, Hongyu Guo, and Anima Anandkumar. 2025b. [A text-guided protein design framework](#). *Nature Machine Intelligence*, 7(4):580–591.
- Huan Ma, Jingdong Chen, Joey Tianyi Zhou, Guangyu Wang, and Changqing Zhang. 2025. [Estimating LLM Uncertainty with Evidence](#). *arXiv preprint*. ArXiv:2502.00290.
- Ali Madani, Ben Krause, Eric R. Greene, Subu Subramanian, Benjamin P. Mohr, James M. Holton, Jose Luis Olmos, Caiming Xiong, Zachary Z. Sun, Richard Socher, James S. Fraser, and Nikhil Naik. 2023. [Large language models generate functional protein sequences across diverse families](#). *Nature Biotechnology*, 41(8):1099–1106.
- Valerio Mariani, Marco Biasini, Alessandro Barbato, and Torsten Schwede. 2013. [Iddt: a local superposition-free score for comparing protein structures and models using distance difference tests](#). *Bioinformatics*, 29(21):2722–2728.
- Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiayi Yang, Jingren Zhou, Junyang Lin, and 24 others. 2025. [Qwen2.5 Technical Report](#). *arXiv preprint*. ArXiv:2412.15115.
- Alexander Rives, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, Jason Liu, Demi Guo, Myle Ott, C. Lawrence Zitnick, Jerry Ma, and Rob Fergus. 2021. [Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences](#). *Proceedings of the National Academy of Sciences*, 118(15):e2016239118.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. [Toolformer: Language models can teach themselves to use tools](#). *Advances in Neural Information Processing Systems*, 36:68539–68551.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. [DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models](#). *arXiv preprint*. ArXiv:2402.03300.
- Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. [Hybridflow: A flexible and efficient rlhf framework](#). *arXiv preprint arXiv:2409.19256*.
- Jin Su, Yan He, Shiyang You, Shiyu Jiang, Xibin Zhou, Xuting Zhang, Yuxuan Wang, Xining Su, Igor Tolstoy, Xing Chang, Hongyuan Lu, and Fajie Yuan. 2025. [A trimodal protein language model enables advanced protein searches](#). *Nature Biotechnology*.
- Yajie Wang, Pu Xue, Mingfeng Cao, Tianhao Yu, Stephan T. Lane, and Huimin Zhao. 2021. [Directed Evolution: Methodologies and Applications](#). *Chemical Reviews*, 121(20):12384–12444.
- Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. [Finetuned language models are zero-shot learners](#). *arXiv preprint arXiv:2109.01652*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. [Chain-of-Thought Prompting Elicits Reasoning in Large Language Models](#). *arXiv preprint*. ArXiv:2201.11903.
- Lai Wei, Yuting Li, Kaipeng Zheng, Chen Wang, Yue Wang, Linghe Kong, Lichao Sun, and Weiran Huang. 2025. [Advancing multimodal reasoning via reinforcement learning with cold start](#). *arXiv preprint arXiv:2505.22334*.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. [React: Synergizing reasoning and acting in language models](#). In *The eleventh international conference on learning representations*.
- Zhaochen Yu, Ling Yang, Jiaru Zou, Shuicheng Yan, and Mengdi Wang. 2025. [Demystifying reinforcement learning in agentic reasoning](#). *arXiv preprint arXiv:2510.11701*.
- Chaohao Yuan, Songyou Li, Geyan Ye, Yikun Zhang, Long-Kai Huang, Wenbing Huang, Wei Liu, Jianhua Yao, and Yu Rong. 2024. [Annotation-guided Protein Design with Multi-Level Domain Alignment](#). *arXiv preprint*. ArXiv:2404.16866.
- Yang Zhang and Jeffrey Skolnick. 2004. [Scoring function for automated assessment of protein structure template quality](#). *Proteins: Structure, Function, and Bioinformatics*, 57(4):702–710.

Guojiang Zhao, Sihang Li, Zixiang Lu, Zheng Cheng, Haitao Lin, Lirong Wu, Hanchen Xia, Hengxing Cai, Wentao Guo, Hongshuai Wang, and 1 others. 2025. Molreasoner: Toward effective and interpretable reasoning for molecular llms. *arXiv preprint arXiv:2508.02066*.

A Token-level Uncertainty Estimation with LogTokU

We follow the Logits-induced Token Uncertainty (LogTokU) framework (Ma et al., 2025) to obtain a token-level uncertainty score for each generated token. At decoding step t , the language model produces logits $\mathbf{z}_t \in \mathbb{R}^V$ over the vocabulary. At each decoding step the top- K logits are selected,

$$\mathcal{S}_t = \{k_1, \dots, k_K\}, \quad z_{t,k} = (\mathbf{z}_t)_k, \quad k \in \mathcal{S}_t,$$

and treated as *evidence* over the small candidate set \mathcal{S}_t ; we adopt the same procedure in our experiments.

Aleatoric uncertainty The evidence values are mapped to non-negative scalars $e_{t,k} \geq 0$ by applying a rectified linear transform to the top- K logits,

$$e_{t,k} = \max(0, z_{t,k}),$$

and then used to define Dirichlet parameters

$$\alpha_{t,k} = e_{t,k} + 1, \quad \alpha_{t,0} = \sum_{k \in \mathcal{S}_t} \alpha_{t,k}.$$

This induces a Dirichlet distribution over the categorical probabilities of the K candidates. The **aleatoric uncertainty** (AU) at step t is defined as the expected entropy of this data distribution:

$$\text{AU}_t = - \sum_{k \in \mathcal{S}_t} \frac{\alpha_{t,k}}{\alpha_{t,0}} \left[\psi(\alpha_{t,k} + 1) - \psi(\alpha_{t,0} + 1) \right],$$

where $\psi(\cdot)$ is the digamma function. A larger AU_t indicates that the next-token distribution is more diffuse or multi-modal (the problem itself is more ambiguous), whereas a small AU_t corresponds to an essentially unambiguous next token.

Epistemic uncertainty The total amount of evidence,

$$E_t = \sum_{k \in \mathcal{S}_t} e_{t,k},$$

is used to quantify how much experience the model has accumulated for the current context. The **epistemic uncertainty** (EU) at step t is defined as

$$\text{EU}_t = \frac{K}{E_t + K},$$

so that EU_t is large when the total evidence is small (the model has little knowledge about this context), and decreases as the model accumulates more evidence.

Token-level uncertainty. The token-level predictive uncertainty is defined as:

$$u_t = \text{EU}_t \text{AU}_t, \quad (2)$$

Larger values of u_t correspond to tokens that are both weakly supported by the model (high epistemic uncertainty) and not strongly constrained by the data (non-negligible aleatoric uncertainty), and are therefore treated as less reliable.

B Additional Details for LLM Experiments

B.1 Amino-acid Frequency Estimation

Let the model generate sequences s_1, \dots, s_N , where $s_k = (a_{k1}, \dots, a_{kL_k})$ and each token a_{kj} is one of the 20 standard amino acids in the alphabet \mathcal{A} . For each residue type $r \in \mathcal{A}$, we estimate its empirical frequency under the model as

$$f_{\text{model}}(r) = \frac{\sum_{k=1}^N \sum_{j=1}^{L_k} \mathbf{1}[a_{kj} = r]}{\sum_{k=1}^N L_k}, \quad (3)$$

and compute the corresponding ground-truth frequency $f_{\text{gt}}(r)$ from the reference sequences in the dataset. In Figure 1b, we visualise the deviation $\Delta f(r) = f_{\text{model}}(r) - f_{\text{gt}}(r)$ for each residue type and model configuration.

B.2 Plan-quality Scoring with DeepSeek-R1

We use DeepSeek-R1 (DeepSeek-AI et al., 2025) as an automatic judge to evaluate the quality of the textual design plans produced by the models. Each plan is scored along three dimensions:

- **Framework:** whether the overall design strategy is coherent and logically organised;
- **Bio:** biological plausibility and correctness of the proposed operations;
- **Alignment:** how well the plan addresses the given textual requirement.

The final planning quality score is computed as a weighted combination of these components:

$$\begin{aligned} \text{PlanScore} &= 0.4 \cdot \text{framework} \\ &\quad + 0.3 \cdot \text{bio} \\ &\quad + 0.3 \cdot \text{alignment}. \end{aligned}$$

To validate this automatic evaluation, we randomly select 20 descriptions and obtain human

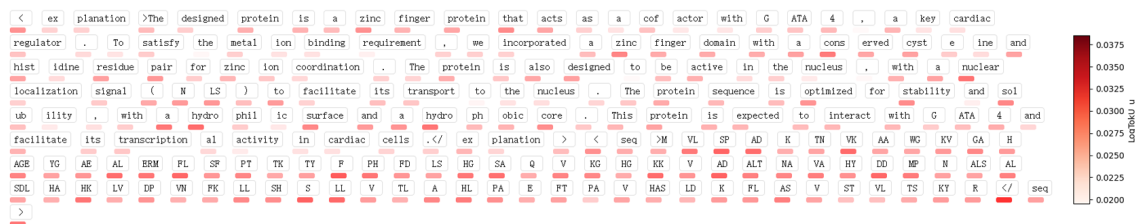


Figure 7: Token-level uncertainty visualization for a representative example. The top panel shows the generated explanation and sequence tokens, and the bottom colour bar encodes the scalar uncertainty u_t (Eq. 2) for each token. Regions with darker colours correspond to higher predictive uncertainty.

expert ratings for both the framework and alignment of the generated plans. The overall DeepSeek score shows a strong correlation with human judgments (Pearson $r = 0.73$), and the alignment subscore correlates with the ProTrek text–text similarity between the plan and the requirement (Pearson $r = 0.65$). These results suggest that DeepSeek provides a reasonably faithful proxy for expert assessment of planning quality.

C Tool Environment Details

C.1 Scaffold Generation

The scaffold-generation tool $T(a^{\text{SCAF}}, (\text{text}, \text{num_sequences}))$ wraps several database-specific queries behind a unified interface. The argument `text` is first lightly normalized (lowercasing and simple keyword matching) and then routed to one or more back-ends depending on its content:

- general functional descriptions (e.g., enzyme names, cofactor phrases) \rightarrow UniProtKB (uni, 2025);
- pathway- or reaction-related cues \rightarrow Rhea (Bansal et al., 2022);
- GO terms or GO-style phrases \rightarrow QuickGO (Binns et al., 2009);
- explicit domain / motif / structural keywords \rightarrow InterPro (Blum et al., 2025).

Each back-end returns a small set of candidate proteins together with basic annotations; we extract their amino-acid sequences as candidate scaffolds and merge them into a single list. When the union exceeds `num_sequences`, we subsample using a simple ranking heuristic that favors proteins with richer annotations and lengths within a task-specific window. The mapping between query

types and back-end databases is summarized in Table 6.

Table 6: Query types and back-end resources used by the scaffold-generation tool.

Query type	Back-end database
General functional description	UniProtKB
Pathway / reaction information	Rhea
GO terms or GO-style phrases	QuickGO
Domain / structural motif cues	InterPro

C.2 Implementation Details of the Functional-site Design Tool

Identifying and locking annotated functional residues Given a scaffold sequence $s = (s_1, \dots, s_L)$, we query UniProt for residue-level annotations across five feature types: `binding_site`, `active_site`, `motif`, `domain`, and `region`. Each annotation returns an interval $[i, j]$ on the sequence. We define the set of *locked* positions as

$$\mathcal{L} = \bigcup_{(i,j) \in \mathcal{A}} \{i, \dots, j\},$$

where \mathcal{A} is the union over all returned intervals. Residues in \mathcal{L} are treated as immutable: subsequent motif insertions and local edits are restricted to indices $t \notin \mathcal{L}$. This preserves known functional or structurally critical regions while allowing the tool to operate on less constrained parts of the scaffold.

Motif retrieval and ESM-based likelihood scoring The textual specification `text` describes the desired local function (e.g., "binds Zn^{2+} with a Cys_2His_2 coordination motif"). We use `text` as a query to UniProt (uni, 2025) (optionally restricted by organism or keyword filters) and collect a small set of documented motif sequences $\mathcal{M} = \{\mathbf{m}^{(1)}, \dots, \mathbf{m}^{(K)}\}$, each with length $|\mathbf{m}^{(k)}| = \ell_k$.

For a candidate motif \mathbf{m} and an admissible insertion window $[p, p + \ell - 1]$ that does not intersect \mathcal{L} , we construct a hypothetical edited sequence

$$\mathbf{x}^{(\mathbf{m},p)} = (x_1^{(\mathbf{m},p)}, \dots, x_L^{(\mathbf{m},p)}),$$

by replacing positions $p, \dots, p + \ell - 1$ of \mathbf{s} with \mathbf{m} (or inserting \mathbf{m} into a loop if allowed by the backbone representation). We then score this candidate using ESM2-3B (Lin et al., 2023).

Because ESM2-3B is a masked language model, we estimate a *pseudo* negative log-likelihood (NLL) over a local region $\Omega^{(\mathbf{m},p)}$ around the graft (e.g., the motif plus a small flanking window):

$$\text{NLL}(\mathbf{x}^{(\mathbf{m},p)}) = - \sum_{t \in \Omega^{(\mathbf{m},p)}} \log p_{\theta}(x_t^{(\mathbf{m},p)} | \mathbf{x}_{\setminus t}^{(\mathbf{m},p)}),$$

where $p_{\theta}(\cdot | \mathbf{x}_{\setminus t})$ is the ESM2-3B conditional distribution obtained by masking position t and running a single forward pass. Intuitively, this score penalizes candidates that are implausible under the protein language model, while leaving the rest of the scaffold unchanged.

Across all motifs $\mathbf{m} \in \mathcal{M}$ and all admissible insertion windows p , the tool selects the configuration

$$(\mathbf{m}^*, p^*) = \arg \min_{\mathbf{m}, p} \text{NLL}(\mathbf{x}^{(\mathbf{m},p)}),$$

and returns $\mathbf{x}^{(\mathbf{m}^*, p^*)}$ as the proposed functional-site design (subject to additional filtering below).

D Reward Shaping Details

Given a trajectory τ with T rounds, the final reward is

$$R(\tau) = \begin{cases} R_{\text{soft}}(\tau), & \text{valid } \langle \text{answer} \rangle \text{ in } \tau, \\ -0.5, & \text{otherwise.} \end{cases}$$

The shaped component is

$$R_{\text{soft}}(\tau) = 0.5 R_{\text{format}}(\tau) + 1.0 R_{\text{tools}}(\tau) + 0.5 R_{\text{ProTrek}}(\tau) + 1.5 R_{\text{eff}}(T) + R_{\text{refl}}(\tau)$$

Format reward Let $F_{\text{tp}}(\tau)$ be 1 if τ contains at least one pair of $\langle \text{think} \rangle$ and $\langle \text{plan} \rangle$ tags, and 0 otherwise. Let $F_{\text{ans}}(\tau)$ be 1 if τ contains a valid $\langle \text{answer} \rangle$ tag. We set

$$R_{\text{format}}(\tau) = 0.5 F_{\text{tp}}(\tau) + 0.5 F_{\text{ans}}(\tau).$$

Sequence-quality reward (ProTrek) Let $P^*(\tau)$ be the best ProTrek score over all sequences in τ , and let $S(\tau) \in \{0, 1\}$ indicate whether at least one sequence was generated. Then

$$R_{\text{ProTrek}}(\tau) = \begin{cases} \text{clip}(P^*(\tau)/20, 0, 1), & S(\tau) = 1, \\ -0.2, & S(\tau) = 0. \end{cases}$$

Efficiency reward Let T be the total number of rounds. We first separate the two regimes

$$R_{\text{eff}}(T) = \begin{cases} 0.1(4 - T), & T \leq 4, \\ r_{\text{mid}}(T), & T > 4, \end{cases}$$

and then define

$$r_{\text{mid}}(T) = -0.25(T - 4) - 0.5 \mathbf{1}[T > 8] - 1.0 \mathbf{1}[T > 12].$$

Tool-usage reward For each round i , we define a local tool reward $r_{\text{tool}}(a_i, t_i)$ and sum over the episode:

$$R_{\text{tools}}(\tau) = \sum_{i=1}^T r_{\text{tool}}(a_i, t_i).$$

In our implementation, $r_{\text{tool}}(a_i, t_i)$ is (i) a small positive constant $\alpha_{\text{gen}} = 0.3\text{--}0.6$ for the first few successful calls to generative tools, (ii) an extra bonus $\alpha_{\text{score}} = 0.3$ for successful `get_score` calls, and (iii) a penalty of -0.1 for failed tool calls.

Reflection reward Let $\text{bad}(t_i) \in \{0, 1\}$ indicate that the feedback at round i is below a threshold, and let $\text{refl}(s_{i+1}) \in \{0, 1\}$ indicate that the next state contains an explicit reflection step. We define

$$R_{\text{refl}}(\tau) = \sum_{i=1}^{T-1} r_{\text{refl}}(t_i, s_{i+1}),$$

$$r_{\text{refl}}(t_i, s_{i+1}) = \alpha_{\text{good}} \text{bad}(t_i) \text{refl}(s_{i+1}) + \alpha_{\text{noop}} \text{bad}(t_i) (1 - \text{refl}(s_{i+1})),$$

with $\alpha_{\text{good}} > 0$ and $\alpha_{\text{noop}} < 0$ (e.g. $\alpha_{\text{good}} = 0.2$ and $\alpha_{\text{noop}} = -0.2$).

E Training Details

This appendix provides the detailed training setup for ProtoCycle, including data construction, SFT settings, RL settings, tool-environment interface, and compute resources.

E.1 SFT Trajectory Synthesis

E.1.1 Overview

We construct 2,000 multi-round tool-interaction trajectories for SFT by seeding GPT-4O with 5 expert-written trajectories as few-shot demonstrations. To mimic the reflection mechanism, we additionally prompt GPT-4O to condition its next-step decision on whether the previous tool call succeeded and on the returned ProTrek score: (i) if the tool call fails, it either retries with revised arguments or switches to an alternative tool; (ii) if the tool call succeeds but the ProTrek score is below 12, it reflects on potential issues in the current workflow and replans; (iii) if the tool call succeeds and the ProTrek score is above 12, it continues with the current plan.

Each trajectory follows an interaction pattern with `<think>`, `<plan>`, and `<tool_call>` segments, interleaved with tool observations. We additionally score the initial global plan of each synthesized trajectory using DEEPSEEK-R1 following the rubric in Appendix B.2; plans that do not meet the quality threshold are discarded and the trajectory is regenerated.

E.1.2 Example Tool-Interaction Trajectories

```
[USER REQUIREMENT]
1. The protein should have a specific function in
intein-mediated protein splicing, cell-cell signaling and
should be localized to plasma membrane, endoplasmic
reticulum membrane, Golgi membrane in order to carry out
this function efficiently.
2. The designed protein have Hint domains with specific
sequence motifs that are crucial for its activity.

[ASSISTANT]
<think>
[Requirement decomposition]
- general function: "specific function in intein-mediated
protein splicing, cell-cell signaling".
- pathway: "intein-mediated protein splicing", "cell-cell
signaling"
- domain: "Hint domains".
- go: "plasma membrane", "endoplasmic reticulum membrane",
"Golgi membrane".
- motif: "specific sequence motifs that are crucial for its
activity"
- signal: localization to "plasma membrane, endoplasmic
reticulum membrane, Golgi membrane
</think>
<plan>
- First, explore functional scaffolds using pathway2seq
with "intein-mediated protein splicing" as the pathway
keyword, since it directly relates to the protein's
function.
- Once initial scaffolds are generated, introduce
motif2constraints to enforce the presence of specific
sequence motifs and signal2constraints to ensure proper
localization.
- After constraints are applied, call get_score to rank
sequences and stop when the global best score is high and
stable across multiple rounds.
</plan>
<tool_call>
{"name": "pathway2seq", "arguments": {"text":
"intein-mediated protein splicing"}}
</tool_call>

[TOOL OBSERVATION]
Round: 1
- tool: pathway2seq
```

```
- num_sequences_scored: 0
- delta_best: +0.000
- argument_pathway: intein-mediated protein splicing
```

```
[ASSISTANT]
<think>
The first attempt to generate scaffolds using the
pathway2seq tool with the pathway "intein-mediated protein
splicing" did not yield any candidate sequences. This
indicates that the argument may have been too specific or
not aligned with the tool's expectations.
To proceed, I will try a different approach by using the
function2seq tool with a more general description of the
protein's function. The phrase "intein-mediated protein
splicing, cell-cell signaling" can be simplified to
"intein-mediated splicing and cell signaling" to see if it
retrieves any candidate sequences.
This should help in generating a broader range of scaffolds
that align with the protein's functional requirements.
</think>
<plan>
Call the function2seq tool with the argument
"intein-mediated splicing and cell signaling" to generate
candidate protein sequences.
</plan>
<tool_call>
{"name": "function2seq", "arguments": {"text":
"intein-mediated splicing and cell signaling"}}
</tool_call>
```

```
[TOOL OBSERVATION]
Round: 2
- tool: function2seq
- num_sequences_scored: 50
- best_ProTrek_this_round: 13.858
- global_best_ProTrek: 13.858
- delta_best: +13.858
```

```
[ASSISTANT]
<think>
The function2seq tool successfully generated 20 candidate
sequences, with the best ProTrek score reaching 13.858.
This is a strong score, indicating that the scaffold aligns
well with the functional requirements. However, the design
still needs to incorporate specific constraints related to
localization and sequence motifs.
To ensure proper localization to the plasma membrane,
endoplasmic reticulum membrane, and Golgi membrane, I will
introduce motif2constraints to enforce the presence of
specific sequence motifs crucial for the protein's
activity.
</think>
<plan>
Call the motif2constraints tool to enforce the presence of
specific sequence motifs that are crucial for the protein's
activity.
</plan>
<tool_call>
{"name": "motif2constraints", "arguments": {"motif":
"specific sequence motifs that are crucial for its
activity"}}
</tool_call>
```

```
[TOOL OBSERVATION]
Round: 3
- tool: motif2constraints
- num_sequences_scored: 5
- global_best_ProTrek: 14.476
- prev_global_best_ProTrek: 13.858
- delta_best: +0.618
- argument_motif: specific sequence motifs that are crucial
for its activity
```

```
[ASSISTANT]
<think>
The motif constraint has improved the global ProTrek score,
suggesting the current top candidate is already aligned
with the requirement.
Before allocating additional tool calls, I will run the
evaluator once to obtain calibrated scores for language
alignment and foldability of the current best candidate.
</think>
<plan>
Call get_score to evaluate the current top candidates and
verify requirement satisfaction.
</plan>
<tool_call>
{"name": "get_score", "arguments": {"top_k": 5}}
</tool_call>
```

```
[TOOL OBSERVATION]
Round: 4
- tool: get_score
- num_sequences_scored: 5
- ProTrek_best: 14.421
- pTM_best: 0.78
- pLDDT_best: 0.81
- PAE_best: 9.4
```

```
[ASSISTANT]
<think>
Given that the score is strong and stable, I will stop and
return the current best candidate as the final design.
</think>
<answer>
MDE..KSS
</answer>
```

E.1.3 Prompt for ProtoCycle

In our implementation, tool-related instructions (i.e., the tool list, schemas, and calling formats) are automatically prepended to the model prompt by VERL (Sheng et al., 2024) based on the provided tool specifications.

Below we show the resulting trajectory prompt template used in our experiments.

You are an agentic protein-design assistant. For each conversation, the user gives a natural-language design requirement for a protein, and you must explore tools step by step and finally output one amino-acid sequence.

Conversation structure:

- The user sends ONE main requirement description at the beginning.
- Then the conversation alternates between:
 - assistant messages (you think/plan/call tools or give the final answer),
 - tool messages (OBSERVATION from the tools you called).

Your message format:

At every assistant turn you must choose EXACTLY ONE of the following patterns:

- (1) FIRST STEP (the very first assistant message after the user requirement),
- (2) INTERMEDIATE STEP (later steps that still call tools),
- (3) FINAL STEP (no more tool calls, only output the sequence).

```
=====
(1) FIRST ASSISTANT STEP
=====
```

The first assistant message in the conversation MUST have this structure:

```
<think>
[Requirement decomposition]
- general function: present/not mentioned/not specified -
quote key phrases if present.
- pathway: ...
- co-factor: ...
- reaction: ...
- domain: ...
- dna-binding: ...
- go: ...
- motif: ...
- signal: ...
</think>
```

```
<plan>
- A high-level multi-step plan:
  * how you will explore scaffolds with Stage-1 tools,
  * when and why you will introduce constraints (Stage-2),
  * how you will use refinement/scoring tools (Stage-3),
  * under what conditions you will stop and output the
  final sequence.
</plan>
```

```
<tool_call>
{"name": "ONE_STAGE1_TOOL_NAME", "arguments": { ... }}
</tool_call>
```

Rules for the first step:

- You MUST call exactly ONE Stage-1 tool in the first step.
- You MUST NOT call Stage-2 or Stage-3 tools in the first step.
- You MUST NOT output <answer> in the first step.

```
=====
(2) INTERMEDIATE STEPS (LATER)
=====
```

Any later assistant step that still calls a tool MUST follow this structure:

```
<think>
- Summarize what has happened so far, especially the latest
tool OBSERVATION
(scores, whether sequences were found, whether
constraints worked, etc.).
- Decide whether to continue exploring scaffolds, add
constraints, refine, or replan.
- Choose exactly ONE tool to call next and explain briefly
why it is appropriate now.
- Explain how you choose its key arguments (e.g., simplify
terms if previous calls failed).
</think>
```

```
<plan>
- A concise description of the NEXT action:
  - which SINGLE tool you will call,
  - what main arguments you will pass,
  - and what you expect to learn or improve.
</plan>
```

```
<tool_call>
{"name": "TOOL_NAME", "arguments": { ... }}
</tool_call>
```

Rules for intermediate steps:

- You MUST include exactly one <think>, one <plan>, and one <tool_call>.
- You MUST call exactly ONE tool per intermediate step.
- You MUST NOT include <answer> in an intermediate step.

```
=====
(3) FINAL STEP (STOP AND OUTPUT SEQUENCE)
=====
```

Before the final step:

- You MUST have called the scoring tool get_score at least once in this conversation.
- You MUST read its OBSERVATION to know the current best design and its sequence.

The FINAL assistant message MUST contain ONLY:

```
<answer>
AA_SEQUENCE
</answer>
```

Rules for the final answer:

- The content inside <answer> must be exactly ONE continuous amino-acid sequence (letters from ACDEFGHIKLMNPQRSTVWY), typically copied from the best sequence reported by the most recent get_score OBSERVATION.
- Do NOT add any extra commentary, explanation, or text inside <answer>.
- If the evaluation indicates that the current best candidate is not yet satisfactory (e.g., low alignment or poor foldability), you MUST NOT output <answer>. Instead, revise the plan and continue tool interaction.

```
=====
Heuristics and stopping criteria
=====
```

- Use several intermediate tool steps to:
 - * generate scaffolds with Stage-1 tools,
 - * optionally refine with Stage-2 constraints,
 - * optionally refine with Stage-3 esm_inpaint,
 - * and monitor the scores reported in OBSERVATIONS.
- If the global ProTrek score is high, OR appears to have

plateaued around a reasonable level, you SHOULD:
 1) call `get_score` once to aggregate and re-score all known sequences, and then
 2) in the NEXT turn, produce a FINAL STEP containing only `<answer>`.

- If tools return OBSERVATIONS like "no new sequences" or `num_sequences_scored=0`, treat that call as FAILED:
 * in `<think>`, diagnose why (argument too long, wrong type, not a real motif/cofactor, etc.),
 * then adjust arguments (simplify or clean them) or switch to a more robust Stage-1 tool.

Hard constraints:

- NEVER call more than one tool in a single assistant message.
 - NEVER mix `<answer>` with `<tool_call>`.
 - ALWAYS use the FIRST STEP structure for the first assistant message, the INTERMEDIATE STEP structure for later tool-calling steps, and the FINAL STEP structure when you are ready to output the sequence.

The following text is the design requirement you must satisfy for this conversation.
`{requirement}`

E.2 Supervised Fine-Tuning (SFT)

We fine-tune QWEN2.5-7B-INSTRUCT (Qwen et al., 2025). The maximum training context length to 32,768 tokens; in practice, the multi-round trajectories typically occupy around 20K tokens. We use a total batch size of 32 and train for 5 epochs with full-parameter fine-tuning.

Unless specified above, we follow the default SFT recipe in the OpenAgentRL framework (Yu et al., 2025) (e.g., optimizer and learning-rate schedule).

E.3 Reinforcement Learning with GRPO

Episode and rollout settings We enable multi-turn rollouts with a maximum of 8 user turns and 8 assistant turns. We set the maximum prompt length to 8,192 tokens and the maximum response length to 20,480 tokens. We sample $n=4$ rollouts per requirement during training and use $n=1$ for validation. For validation generation, we use top- p sampling with $p=0.6$ and temperature 1.0.

Optimization settings We use an actor learning rate of 1×10^{-6} , a training batch size of 64, and a PPO mini-batch size of 16. We aggregate the token-level loss by token-mean. We disable KL regularization (both in-reward KL and KL loss) and apply clipped updates with clip ratios in $[0.20, 0.28]$. Gradient clipping is set to 1.0. For training data, each epoch contains 100 instances (disjoint from the test set), and we train for 5 epochs.

E.4 Tool Environment Interface

The tool environment and tool definitions are described in Section 3.2. Here we clarify the *summary*

returned to the planner after each tool call. Each tool returns a compact observation that includes: (i) the tool name; (ii) the number of sequences scored in this round; (iii) the global best ProTrek score so far; and (iv) the improvement over the previous global best. Concretely, we format the observation as:

```
Round: t
- tool: {tool_name}
- num_sequences_scored: {m}
- global_best_ProTrek: {best}
- prev_global_best_ProTrek: {prev_best}
- delta_best: {best-prev_best}
- argument_{type}: {arg}
```

E.5 Compute Resources

All experiments are run on $8 \times A100$ GPUs on a single node with CUDA 12.2, PyTorch 2.6.0, and VERL 0.5.0.dev0.

F Evaluation Metrics

F.1 Sequence Plausibility

Pseudo-perplexity (PPL) We measure sequence plausibility using the pseudo-perplexity under a masked protein language model (ESM2-3B). Let L be the number of amino-acids. For each position i , we mask x_i and compute the log-probability of the original token under the masked LM.

$$\text{PPL}(x) = \exp\left(-\frac{1}{L} \sum_{i=1}^L \log p_{\theta}(x_i | x_{\setminus i})\right),$$

where $x_{\setminus i}$ denotes the sequence with position i masked.

Repeat percentage (Repeat) We quantify local degeneration using the contiguous-repeat criterion, following PDFBench (Kuang et al., 2025). We scan window sizes $w \in \{1, \dots, W\}$ where $W = \min(20, \lfloor n/2 \rfloor)$. For each start position i , if the substring $x_{i:i+w}$ repeats consecutively at least three times, we mark the whole repeated span as a repeated region. Let $\mathcal{R}(x)$ be the union of all such repeated index intervals (merged if overlapping). We define

$$\text{Repeat}(x) = 100 \cdot \frac{|\mathcal{R}(x)|}{n}.$$

F.2 Foldability

We evaluate foldability using the Chai-1 (Chai Discovery, 2024) structure predictor. For each sequence, Chai-1 predicts a 3D structure and outputs three confidence/error signals (pLDDT, PAE, pTM) following the standard AlphaFold-style definitions (Jumper et al., 2021).

pLDDT (predicted IDDT-C α) The local distance difference test (IDDT) is a superposition-free local quality metric (Mariani et al., 2013). Let d_{ij} and d_{ij}^* be the C α -C α distances between residues (i, j) in the predicted structure and the (unknown) true structure, respectively. For residue i , the IDDT-C α is

$$\text{IDDT}_i = \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \frac{1}{4} \sum_{\delta \in \Delta} \mathbf{1}\{|d_{ij} - d_{ij}^*| < \delta\},$$

where $\Delta = \{0.5, 1, 2, 4\}$, $\mathcal{N}(i)$ is a local neighborhood of residue pairs used by IDDT. Chai-1 outputs $\text{pLDDT}_i \approx \mathbb{E}[\text{IDDT}_i]$ as a learned per-residue confidence score.

PAE (predicted aligned error) Predicted aligned error (PAE) measures confidence in the *relative placement* of two residues/domains. Following AlphaFold, define the alignment-frame error

$$e_{ij} = \|T_i^{-1} \circ \mathbf{x}_j - (T_i^*)^{-1} \circ \mathbf{x}_j^*\|,$$

i.e., the positional error of residue j after aligning predicted and true structures on residue i . Chai-1 outputs

$$\text{PAE}_{ij} \approx \mathbb{E}[e_{ij}],$$

which is typically reported in Å and is not symmetric in general.

pTM (predicted TM-score) TM-score is a global superposition metric (Zhang and Skolnick, 2004). AlphaFold derives a computable predictor using the aligned-error distribution:

$$\text{pTM} = \max_i \frac{1}{N} \sum_{j=1}^N \mathbb{E} \left[\frac{1}{1 + \left(\frac{e_{ij}}{d_0(N)}\right)^2} \right],$$

$$d_0(N) = 1.24(\max(N, 19) - 15)^{1/3} - 1.8,$$

where N is the number of residues. Chai-1 reports pTM using the same underlying definition.

F.3 Language Alignment

ProTrek score We compute a text-sequence alignment score using ProTrek-650M (Su et al., 2025). In our implementation, we load the released ProTrek-650M checkpoint together with its accompanying text/protein encoders (PubMedBERT (Gu et al., 2021) and ESM2-650M, respectively). Given a textual description t and a protein sequence x , ProTrek encodes them into embeddings $\mathbf{e}_t = \tau_t(t)$

and $\mathbf{e}_x = \tau_p(x)$, and returns the temperature-scaled inner product:

$$\text{ProTrek}(t, x) = \frac{\mathbf{e}_x^\top \mathbf{e}_t}{\tau},$$

where τ is the model temperature parameter.

EvoLLaMA score Following (Kuang et al., 2025), we adopt a generative alignment metric based on EvoLLaMA (Liu et al., 2024). Given a sequence x , EvoLLaMA is prompted to generate a predicted function description t' . We then embed the ground-truth text t and the generated text t' using PubMedBERT (Gu et al., 2021), average token embeddings, and compute cosine similarity:

$$\text{EvoLLaMA}(t, x) = \text{sim} \left(\frac{1}{|t|} \sum_{i=1}^{|t|} \text{Embed}(t_i), \frac{1}{|t'|} \sum_{j=1}^{|t'|} \text{Embed}(t'_j) \right).$$

Retrieval accuracy Retrieval accuracy evaluates whether a description retrieves its matched sequence among randomly sampled negatives. For each test description t_k , we form a candidate set \mathcal{C}_k containing its matched sequence x_k and $K - 1$ randomly sampled negative sequences (we use $K = 32$ in all experiments), compute similarities in the same embedding space, and count a hit if the matched sequence ranks top-1:

$$\text{RA} = \frac{1}{N} \sum_{k=1}^N \mathbf{1} \left[\arg \max_{x \in \mathcal{C}_k} s(t_k, x) = x_k \right].$$

Keyword recovery Following PDF-Bench (Kuang et al., 2025), we define keyword recovery as a protein-level metric based on InterProScan annotations. For each test instance, let K_{ref} denote the reference set of function keywords provided in the input, and let K_{pred} denote the set of function keywords identified from the designed sequence using InterProScan (Jones et al., 2014). Keyword recovery is then defined as

$$\text{Kw. Rec.} = \frac{|K_{\text{pred}} \cap K_{\text{ref}}|}{|K_{\text{ref}}|}.$$

G Robustness Across Disjoint Test Splits

To assess robustness to test-set sampling, we report split-wise results on three disjoint held-out splits (Eval-A/B/C). Tables 7,8,9 show that the relative ranking and overall trends are consistent across splits.

Model	Sequence Plausibility		Foldability			Language Alignment		
	PPL↓	Repeat↓	pTM↑	pLDDT↑	PAE↓	ProTrek↑	EvoLLaMA↑	Retrieval↑
Natural	4.583	2.649	0.757	0.818	9.109	14.583	0.327	0.875
Qwen2.5-7B-Agent	8.204	5.180	0.542	0.699	15.296	6.938	0.261	0.520
Qwen2.5-72B-Agent	7.371	5.302	0.618	0.714	13.249	8.803	0.267	0.560
Qwen3-8B-Agent	7.200	3.846	0.650	0.723	13.490	8.704	0.277	0.570
ProDVa	5.138	1.658	0.751	0.798	8.899	12.394	0.316	0.720
Pinal	<u>4.068</u>	9.106	0.790	0.822	7.797	<u>14.194</u>	<u>0.317</u>	<u>0.830</u>
ProtoCycle-SFT (ours)	4.391	4.140	0.743	0.804	9.780	12.139	0.322	0.800
ProtoCycle-RL (ours)	3.732	<u>3.687</u>	<u>0.774</u>	<u>0.816</u>	<u>8.529</u>	14.961	0.322	1.000

Table 7: Mol-Instructions protein design results on Eval-A (**best**, second-best).

Model	Sequence Plausibility		Foldability			Language Alignment		
	PPL↓	Repeat↓	pTM↑	pLDDT↑	PAE↓	ProTrek↑	EvoLLaMA↑	Retrieval↑
Natural	4.888	2.051	0.769	0.803	9.975	14.735	0.325	0.870
Qwen2.5-7B-Agent	7.520	5.960	0.520	0.682	16.420	6.320	0.248	0.460
Qwen2.5-72B-Agent	7.050	5.740	0.606	0.701	13.880	8.420	0.254	0.520
Qwen3-8B-Agent	6.620	4.420	0.628	0.705	14.280	8.150	0.262	0.510
ProDVa	6.061	1.744	<u>0.778</u>	0.799	<u>8.661</u>	11.577	0.326	0.710
Pinal	3.916	9.945	0.785	0.821	8.132	<u>14.071</u>	<u>0.324</u>	0.820
ProtoCycle-SFT (ours)	4.455	<u>2.194</u>	0.717	0.798	10.752	12.784	0.323	<u>0.838</u>
ProtoCycle-RL (ours)	<u>3.917</u>	2.424	0.766	<u>0.812</u>	8.951	14.395	0.326	0.880

Table 8: Mol-Instructions protein design results on Eval-B (**best**, second-best).

Model	Sequence Plausibility		Foldability			Language Alignment		
	PPL↓	Repeat↓	pTM↑	pLDDT↑	PAE↓	ProTrek↑	EvoLLaMA↑	Retrieval↑
Natural	4.740	1.686	0.760	0.823	9.245	14.566	0.330	0.800
Qwen2.5-7B-Agent	8.980	4.320	0.565	0.715	14.180	7.520	0.274	0.590
Qwen2.5-72B-Agent	7.820	4.980	0.631	0.726	12.900	9.150	0.279	0.610
Qwen3-8B-Agent	7.860	3.120	0.672	0.741	12.710	9.260	0.292	0.640
ProDVa	4.595	1.338	0.767	0.802	8.725	12.139	0.310	0.760
Pinal	3.984	8.900	0.800	<u>0.833</u>	7.373	<u>14.219</u>	<u>0.313</u>	0.770
ProtoCycle-SFT (ours)	3.602	2.373	0.741	0.818	10.067	12.582	0.306	<u>0.880</u>
ProtoCycle-RL (ours)	<u>3.946</u>	<u>1.536</u>	<u>0.785</u>	0.837	<u>8.148</u>	14.687	0.320	0.929

Table 9: Mol-Instructions protein design results on Eval-C (**best**, second-best).

Model	Train	Sequence Plausibility		Foldability			Language Alignment		
		PPL↓	Repeat↓	pTM↑	pLDDT↑	PAE↓	ProTrek↑	K.w. Rec.↑	Retrieval↑
Natural	–	4.358	2.416	0.756	0.788	8.631	10.020	1.000	0.690
Pinal	✓	5.695	9.497	0.683	0.752	9.974	11.784	0.394	0.550
ProDVa	×	6.362	<u>1.147</u>	<u>0.756</u>	0.776	<u>8.663</u>	4.646	0.172	0.120
ProDVa	✓	5.653	0.827	0.794	0.817	6.975	11.054	0.361	0.360
ProtoCycle-SFT	×	<u>5.593</u>	2.144	0.706	0.779	12.129	9.078	<u>0.480</u>	<u>0.684</u>
ProtoCycle-RL	×	4.061	2.128	<u>0.756</u>	<u>0.805</u>	10.493	<u>11.171</u>	0.592	0.833

Table 10: Full CAMEO evaluation metrics (**Best, Second Best**). Train indicates whether the model is trained on CAMEO (✓) or not (×).

H Additional Generalization Results on CAMEO

H.1 Input style comparison: Mol-Instructions vs. CAMEO

Dataset	Example input
Mol-Instructions (protein design) (Fang et al., 2024)	Create a protein sequence that satisfies the following specifications: 1. The protein must exhibit the following characteristics: Also acts as a cofactor with GATA4, a key cardiac regulator. 2. The protein should have metal ion binding and be active in nucleus.
CAMEO (Haas et al., 2018)	Generate a protein sequence for a novel protein that integrates the following function keywords: Polyphenol_oxidase_C, Tyrosinase_Cu-bd,TAT_signal

Table 11: Representative conditioning-text examples from Mol-Instructions (protein-design subset) and CAMEO.

To contextualize the cross-dataset generalization setting, Table 11 provides representative examples of the conditioning-text formats in Mol-Instructions (protein design) (Fang et al., 2024) and CAMEO (Haas et al., 2018). Mol-Instructions typically uses instruction-style natural language requirements, whereas CAMEO uses compact keyword-like annotations. The examples are lightly paraphrased for readability.

H.2 Full results

We provide full evaluation metrics for CAMEO, including sequence plausibility, foldability, and language alignment. The evaluation protocol matches Section 4.1. ProtoCycle variants are trained on the

Mol-Instructions protein-design subset only (i.e., no CAMEO supervision), and are tested directly on CAMEO. We report both ProDVa with/without keyword-style training (✓/×), and Pinal with keyword-style training (✓).

Table 10 shows that ProtoCycle-RL substantially improves generalization to keyword-style inputs over its SFT-only variant. Compared to PINAL (trained on keyword-style data), ProtoCycle-RL achieves much higher keyword recovery (+50.3%) and retrieval (+51.5%), with markedly less repetition, while its ProTrek is slightly lower (−5.2%). Against PRODVA trained only on Mol-Instructions, PROTOCYCLE-RL yields large gains in alignment; relative to PRODVA-CAMEO, PROTOCYCLE-RL attains higher keyword recovery and retrieval accuracy, albeit with weaker foldability.

I Case Study

We select a representative example (full trajectory in I.2) to illustrate ProtoCycle’s multi-round decision process (Fig. 8–9). In addition to language alignment and foldability metrics, we fold intermediate sequences with AlphaFold3 (Abramson et al., 2024) and use InterProScan (Jones et al., 2014) to identify key functional regions, which are then annotated on the folded structure.

The case highlights two behaviors. (i) **Replanning under tool failure**: ProtoCycle initially follows a GO-term scaffold search plus functional-site refinement, but when the *motif2constraints* call fails in Round 3, it immediately switches to an alternative plan based on general-function scaffold search and continues refinement. (ii) **Plateau awareness and proper termination**: after obtaining a high semantic-similarity scaffold (Round 5), further functional-site design does not improve the score, and ProtoCycle detects diminishing returns and stops with the best candidate.

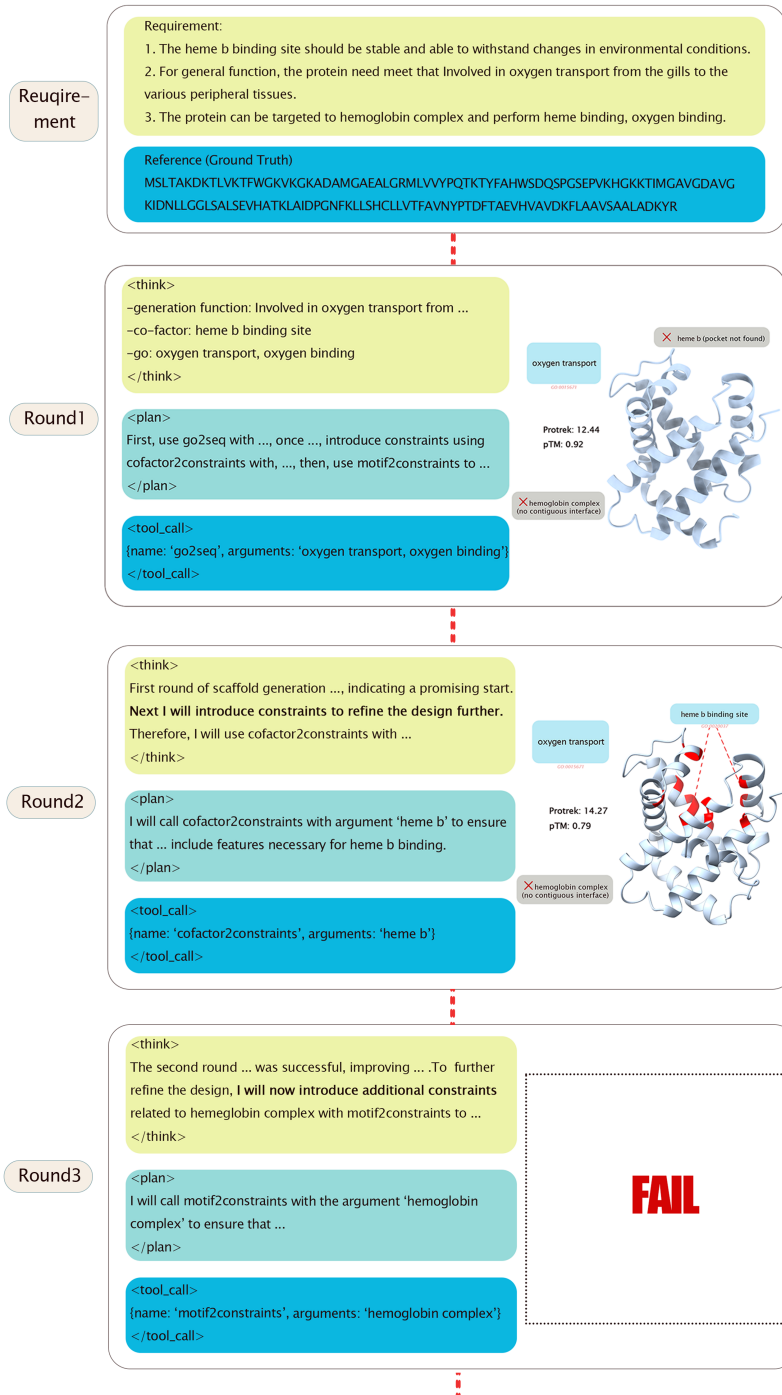


Figure 8: Case study (Part I)

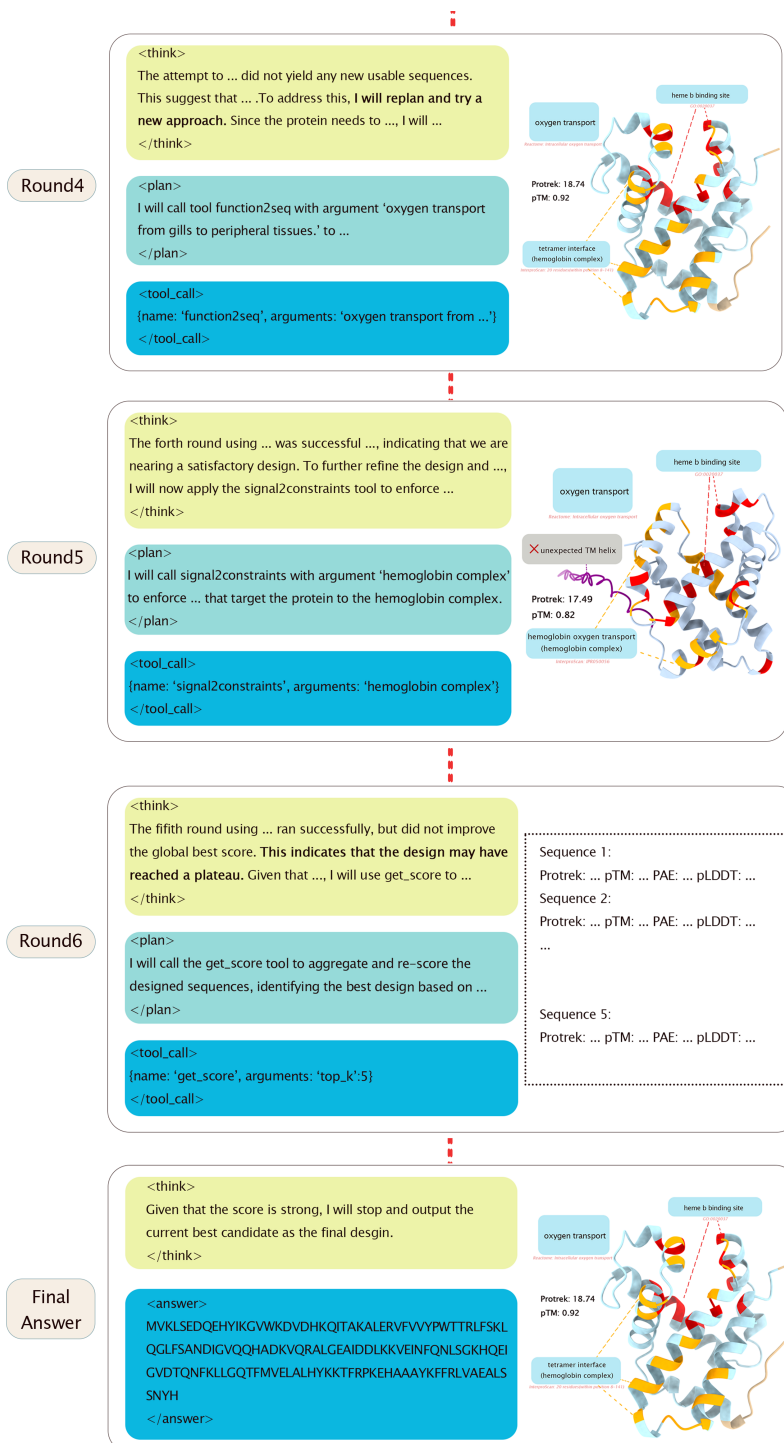


Figure 9: Case study (Part II)

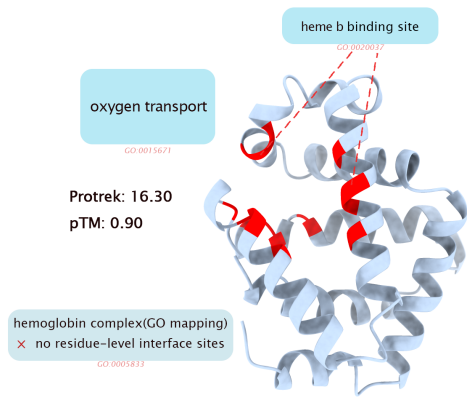


Figure 10: Pinal on the case-study requirement.

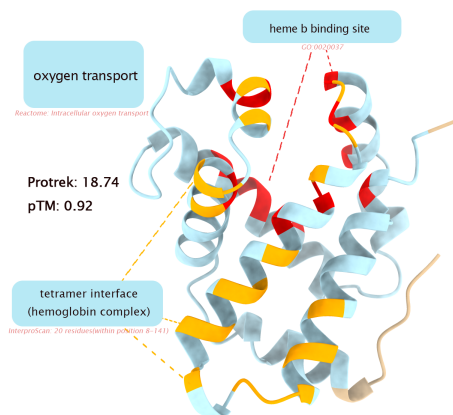


Figure 11: ProtoCycle (final) on the same requirement.

I.1 Comparison with baselines

ProtoCycle and a strong baseline (Pinal) both achieve good language alignment and foldability on this case (Figs. 10 and 11). The requirement consists of three key aspects: (i) a stable *heme b* binding site, (ii) oxygen transport/oxygen-binding function, and (iii) association with the hemoglobin complex. In both designs, we observe evidence consistent with (ii) via oxygen-transport related pathway/function annotations, and with (i) via *heme* binding annotations around the predicted pocket. However, for (iii), Pinal mainly provides homology-level support (from GO/InterPro mappings) without residue-level interface site annotations, whereas ProtoCycle yields a design with clearer structure-level support for complex/interface-related regions highlighted by our annotation pipeline (Fig. 11).

I.2 Full trajectory of Fig. 8–9

[USER REQUIREMENT]
1. The heme b binding site should be stable and able to withstand changes in environmental conditions.

2. For general function, the protein need meet that Involved in oxygen transport from the gills to the various peripheral tissues.
3. The protein can be targeted to hemoglobin complex and perform heme binding, oxygen binding.

```
[ASSISTANT]
<think>
[Requirement decomposition]
- general function: "Involved in oxygen transport from the gills to the various peripheral tissues."
- pathway: "oxygen transport pathway."
- co-factor: "heme b binding site."
- go: "oxygen transport," "heme binding," "oxygen binding."
</think>
<plan>
- First, explore functional scaffolds using go2seq with the GO-like terms "oxygen transport," "heme binding," and "oxygen binding."
- Once promising scaffolds are identified, introduce constraints using motif2constraints and cofactor2constraints to ensure heme b binding and stability.
- Finally, use get_score to rank and select the best-performing sequence, stopping when the global best score stabilizes across iterations.
</plan>
```

```
<tool_call>
{"name": "go2seq", "arguments": {"go_term": "oxygen transport, heme binding, oxygen binding"}}
</tool_call>
```

```
[TOOL_OBSERVATION]
Round: 1
- tool: go2seq
- num_sequences_scored: 17
- best_ProTrek_this_round: 12.442
- global_best_ProTrek: 12.442
- delta_best: +12.442
```

```
[ASSISTANT]
<think>
The first round of scaffold generation using the go2seq tool was successful, yielding 17 new sequences with a global best ProTrek score of 12.442. This score indicates a promising start.
```

```
Next, I will introduce constraints to refine the design further. The protein needs to bind heme b and be stable under varying environmental conditions. Therefore, I will use the cofactor2constraints tool to enforce the requirement for heme b binding.
</think>
```

```
<plan>
- I will call the cofactor2constraints tool with the argument {"cofactor": "heme b"} to ensure that the generated sequences include features necessary for heme b binding.
</plan>
<tool_call>
{"name": "cofactor2constraints", "arguments": {"cofactor": "heme b"}}
</tool_call>
```

```
[TOOL OBSERVATION]
Round: 2
- tool: cofactor2constraints
- num_sequences_scored: 5
- best_ProTrek_this_round: 14.274
- global_best_ProTrek: 14.274
- prev_global_best_ProTrek: 12.442
- delta_best: +1.832
```

```
[ASSISTANT]
<think>
The second round using the cofactor2constraints tool was successful, yielding 5 new sequences and improving the global best ProTrek score to 14.274 (+1.832). This indicates progress, to further refine the design, I will now introduce additional constraints related to the hemoglobin complex and the oxygen transport function. Since no specific motif was mentioned in the requirements, I will use the motif2constraints tool to enforce a general hemoglobin complex motif that could enhance stability and functionality.
</think>
<plan>
```

I will call the motif2constraints tool with the argument {"motif": "hemoglobin complex"} to ensure that the generated sequences include features necessary for stable heme b binding.

```
</plan>
<tool_call>
{"name": "motif2constraints", "arguments": {"motif": "hemoglobin complex"}}
</tool_call>
```

[TOOL OBSERVATION]
Round: 3

- tool: motif2constraints
- num_sequences_scored: 0
- global_best_ProTrek: 14.274
- prev_global_best_ProTrek: 14.274
- delta_best: +0.000

[ASSISTANT]
<think>

The attempt to introduce a hemoglobin complex motif using the motif2constraints tool did not yield any new usable sequences. This suggests that the motif argument may have been too vague or not specific enough to produce results. To address this, I will replan and try a different approach. Since the protein needs to be stable and function in oxygen transport, I will use the function2seq tool with a more focused description of the general function, specifically targeting "oxygen transport from gills to peripheral tissues." This may help generate new scaffolds that align better with the requirement.

```
</think>
<plan>
I will call the function2seq tool with the argument {"text": "oxygen transport from gills to peripheral tissues"} to generate new scaffolds based on the general function of the protein.
</plan>
<tool_call>
{"name": "function2seq", "arguments": {"text": "oxygen transport from gills to peripheral tissues"}}
</tool_call>
```

[TOOL OBSERVATION]
Round: 4

- tool: function2seq
- num_sequences_scored: 50
- best_ProTrek_this_round: 18.744
- global_best_ProTrek: 18.744
- prev_global_best_ProTrek: 14.274
- delta_best: +4.469

[ASSISTANT]
<think>

The fourth round using the function2seq tool was successful, yielding 20 new sequences and significantly improving the global best ProTrek score to 18.744 (+4.469), indicating that we are nearing a satisfactory design. To further refine the design and ensure stability and functionality, I will now apply the signal2constraints tool to introduce a signal peptide that targets the protein to the hemoglobin complex, which is essential for its function in oxygen transport.

```
</think>
<plan>
I will call the signal2constraints tool with argument "hemoglobin complex" to enforce the requirement for a signal peptide that targets the protein to the hemoglobin complex.
</plan>
<tool_call>
{"name": "signal2constraints", "arguments": {"signal": "hemoglobin complex"}}
</tool_call>
```

[TOOL OBSERVATION]
Round: 5

- tool: signal2constraints
- num_sequences_scored: 5
- best_ProTrek_this_round: 17.494
- global_best_ProTrek: 18.744
- prev_global_best_ProTrek: 18.744
- delta_best: -1.250

[ASSISTANT]
<think>

The fifth round using the signal2constraints tool successfully generated 5 new sequences, but the global best ProTrek score remains unchanged at 18.744. This indicates that while we have made progress, the design may have reached a plateau.

Given that the global best score has stabilized, I will proceed to aggregate and re-score all known sequences using the get_score tool. This will allow us to identify the best design based on the current set of sequences and their scores.

```
</think>
<plan>
I will call the get_score tool to aggregate and re-score all known sequences, identifying the best design based on the current scores.
</plan>
<tool_call>
{"name": "get_score", "arguments": {"top_k": 5}}
</tool_call>
```

[TOOL OBSERVATION]
Round: 6

- tool: get_score
- num_sequences_scored: 5
- ProTrek_best: 18.744
- pTM_best: 0.92
- pLDDT_best: 0.91
- PAE_best: 3.4

[ASSISTANT]
<think>

Given that the score is strong and stable, I will stop and return the current best candidate as the final design.

```
</think>
<answer>
MVK...NYH
</answer>
```