AgentFold: Closed-Loop Agentic Search for Protein Folding Model Design

Mingquan Liu[1*] **Jiangyu Chen**[2*] **Hanqun Cao**[3*] **Xujun Zhang**[7] **Pengsen Ma**[1]
Xiangru Tang[5] **Shuting Jin**[4] **Annie Zheng**[6] **Zhuo Yang**[8] **Tianfan Fu**[2†]
Fang Wu[9†] **Xiangxiang Zeng**[1†]
[1]State Key Lab. of Chemo & Biosensing, Coll. of CS & Electronic Eng., Hunan Univ.
[2]State Key Lab. for Novel Software Technology, School of CS, Nanjing Univ.
[3]The Chinese University of Hong Kong [4]Wuhan University of Science and Technology
[5]Yale University [6]South China Normal University Affiliated High School
[7]Zhejiang University [8]Southeast University [9]Stanford University
xzeng@hnu.edu.cn

Abstract

Scientific LLM agents have shown promise in literature reasoning, tool use, and experiment planning, but it remains unclear whether they can autonomously improve large, tightly coupled scientific ML systems through executable code changes and expensive validation. We study this question in protein folding, where progress requires coordinated architectural edits, multi-objective evaluation, and domain-aware interpretation. We present **AgentFold**, a multi-agent framework that formulates folding-model development as closed-loop search over executable code variants. Starting from ESMFold, AgentFold proposes hypotheses, implements and debugs code-level modifications, evaluates variants, analyzes outcomes, and stores both successful and failed interventions in structured memory; an MCTS-style policy allocates compute across high-scoring branches. On an engineering-scale folding codebase (> **2,000** LOC), AgentFold explores ∼**80** variants using ∼**5,000** GPU-hours and ∼**170M** LLM tokens. At matched budget, AgentFold improves best lDDT by **7.5%** over independent Codex proposals and beats random control. Beyond model improvement, the intervention traces reveal recurring empirical design patterns: stable gains tend to arise from early soft learnable priors and gated refinement, whereas direct geometric perturbations and geometry-conditioned feedback often destabilize training. The code and experimental resources are publicly available at https://github.com/lmqfly/AgentFold.

1 Introduction

Scientific agents increasingly combine large language models (LLMs) with literature analysis, hypothesis generation, tool use, and experimental planning (Lu et al., 2024; Tang et al., 2025; Hsu et al., 2024; Qi et al., 2023; Fallahpour et al., 2025; Huang et al., 2025a; Hao et al., 2025; Huang et al., 2025b; Wang et al., 2026; Jin et al., 2026). Execution-grounded benchmarks separately show that iterative machine-learning experimentation and scientific-code development remain difficult even when outcomes can be checked automatically (Tian et al., 2024; Huang et al., 2023, 2024; Chan et al., 2024; Edwards et al., 2025). In scientific ML, a plausible proposal is insufficient: the system must implement the change in a coupled codebase, recover from failures, and compare expensive, noisy, multi-objective experiments.

We study this question in protein folding, where architectural changes are executable interventions in a tightly coupled scientific ML system. Folding models combine sequence and pair representations, geometric refinement, recycling, and structure losses, while evaluation spans both local and global structural metrics. This setting provides a suitable testbed for assessing whether LLM agents are capable of closed-loop scientific model development beyond code generation assistance.

We introduce **AgentFold**, a multi-agent framework for code-level search over folding-model variants. Starting from a compact ESMFold-derived substrate (Lin et al., 2022), AgentFold executes a propose–implement–evaluate loop: it retrieves evidence from a folding-model zoo and a structured memory, proposes architectural or algorithmic edits, applies and debugs code changes, evaluates executable variants, and records both successful and failed interventions. Failed or low-performing variants are retained as structured evidence, allowing later proposals to avoid repeated failure modes and supporting post-hoc comparison among related edits. We use the compact substrate to enable repeated training and evaluation while preserving the coupled structure-module setting that makes folding-model design nontrivial.

[*]Equal contribution.
[†]Corresponding authors.

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing, pages 3802–3830
October 24-29, 2026 ©2026 Association for Computational Linguistics

To allocate compute over long-horizon exploration, AgentFold uses an MCTS-style tree controller over concrete code snapshots. Each node represents an executable implementation, while expansions are prioritized using standard folding metrics and a normalized search utility. On an engineering-scale codebase (>2,000 LOC), AgentFold explores roughly 80 variants using approximately 5,000 GPU-hours and 170M LLM tokens. At a matched evaluation budget on the CAMEO2022 (Haas et al., 2017) development benchmark, AgentFold achieves 7.5% higher best lDDT than an independent Codex-proposal baseline and also outperforms a random-search controller. The strongest variants obtain these improvements with only modest parameter overhead. Analysis of both successful and failed interventions further reveals descriptive regularities: stable improvements frequently co-occur with early soft learnable priors and gated refinement, whereas direct geometric perturbations and geometry-conditioned feedback are often associated with training instability.

Contributions. Our contributions are:

- **Closed-loop folding model search.** We present AgentFold, a multi-agent framework that formulates folding-model development as propose–implement–debug–evaluate cycles over executable code variants rather than limiting the search to textual hypotheses.
- **Engineering-scale matched-budget evaluation.** Starting from a compact ESMFold-derived substrate, AgentFold evaluates roughly 80 code variants under expensive structural validation. At a matched evaluation budget, it achieves 7.5% higher best lDDT than an independent Codex-proposal baseline and also outperforms a random-search controller.
- **Trace-based design evidence.** We analyze the resulting intervention traces to identify recurring post-hoc empirical patterns: stable gains are associated with early soft learnable priors and gated refinement, whereas direct geometric perturbations and geometry-conditioned feedback often destabilize training.

2 Related Work

2.1 Autonomous AI Research

LLM-based systems support literature synthesis and hypothesis generation, end-to-end scientific workflows, and biomedical research planning (Lu et al., 2024; Boiko et al., 2023; Swanson et al., 2025). Execution-grounded benchmarks further evaluate agents on iterative machine-learning experimentation, scientific or data-science code generation, and research-code extensions (Tian et al., 2024; Huang et al., 2023; Chan et al., 2024; Huang et al., 2024; Edwards et al., 2025). These studies expose the difficulty of long-horizon implementation and validation, but they do not specialize the loop to protein-model development.

A complementary line couples LLM-generated programs or designs with executable feedback. FunSearch and AlphaEvolve evolve programs, MCTS-AHD applies tree search to heuristic design, and RZ-NAS and ASI-ARCH search model architectures (Romera-Paredes et al., 2023; Novikov et al., 2025; Zheng et al., 2025; Ji et al., 2025; Liu et al., 2025). AgentFold provides a domain-specific instantiation of these general components in a tightly coupled protein-folding codebase, where each proposed variant must be implemented, debugged, trained, and evaluated against multiple structural metrics.

2.2 Protein Folding

Protein structure prediction has progressed from MSA-based systems such as AlphaFold2 and RoseTTAFold (Jumper et al., 2021; Baek et al., 2023) to unified complex predictors such as AlphaFold3 and RoseTTAFold All-Atom (Abramson et al., 2024; Krishna et al., 2023). Open, trainable platforms including OpenFold and UniFold support method development and reproducible engineering (Ahdritz et al., 2024; Li et al., 2022). Other work explores MSA-free language-model-based prediction with OmegaFold and ESMFold (Wu et al., 2022; Lin et al., 2022), training efficiency with FastFold and MiniFold (Cheng et al., 2022; Wohlwend et al., 2025), and generative or flow-based formulations such as EigenFold, AlphaFlow/ESMFlow, and SimpleFold (Jing et al., 2023, 2024; Wang et al., 2025).

3 Method

We view autonomous folding-model development as a *search over code-level interventions* and their measured outcomes. AgentFold is designed to produce two coupled artifacts: (*i*) improved model variants and (*ii*) accumulated design evidence distilled from intervention–outcome traces. We use an

MCTS-style tree controller over executable code variants, enabling compute-efficient exploration and controlled comparisons among competing design choices. A self-evolving multi-agent loop proposes and implements edits, evaluates variants, and recovers from failures. Finally, an attribution-and-retrieval stage writes structured intervention artifacts to a database-backed memory, while periodic re-scoring updates node values and refines the search policy. Prompts and templates are provided separately (see Appendix C).

3.1 Problem Formulation & Overview

Given a base folding model $\mathcal{M}_0$ (ESMFold (Lin et al., 2022)), we aim to discover variants $\{\mathcal{M}_t\}_{t=1}^{T}$ that improve target evaluation metrics and, in parallel, to summarize recurring empirical design patterns $\mathcal{P} = \{P_k\}_{k=1}^{K}$ from repeated intervention evidence. Each iteration logs a structured *intervention trace* that records the parent variant, the typed edit (e.g., priors, refinement control, geometry operations), the code diff, stability signals, and metric deltas, which supports cross-variant attribution and empirical pattern mining.

To address the complexity of the ESMFold codebase, we propose AgentFold, an LLM-based multi-agent framework with an MCTS-style tree controller. As illustrated in Figure 1, AgentFold operates via a dual-loop mechanism:

- **Inner Exploration Loop:** A continuous cycle of Sampling, Evolution, Experiment, and Analysis that iteratively generates and verifies new model variants.
- **Outer Periodic Update:** A batched update mechanism (e.g., every 10 iterations) that refines the search tree and candidate sets using a composite scoring function.

A central Database & Metadata module serves as an experiment memory: it stores executable code snapshots, code diffs, configurations, logs, and structured attributions, linking them to retrieved literature so that future edits can be proposed and evaluated using accumulated evidence.

3.2 MCTS-based Dynamic Sampling

The search process begins with the Experience Pool (Search Tree), which structurally organizes model variants.

Top-k Sampling Strategy. Sampling multiple siblings from the same parent node creates near-controlled comparisons (holding most code constant), supporting attribution of gains or losses to specific intervention types and consolidation of recurring design patterns. At the start of each inner loop, the sampler selects high-scoring nodes together with diverse reference nodes, approximating an exploration–exploitation trade-off.

Context Summarization. The Summarizer prioritizes evidence that is comparable to the current parent node (e.g., similar edit types or failure modes), producing a compact brief that highlights successful outcomes, failed interventions, and empirical patterns currently supported by the accumulated traces.

3.3 Self-Evolving Agentic Workflow

The Evolution phase transforms the summarized context into executable code through a specialized agent chain:

1. **Deduplication.** First, a Deduplicator Agent screens the proposed optimization direction against historical data to prevent redundant experiments.
2. **Unified Planning & Coding.** Valid proposals are passed to the Unified Planner. Unlike decoupled approaches, this agent is solely responsible for both architectural design and code implementation, reducing interface mismatches between design and implementation.
3. **Interactive Debugging.** The generated code enters the Training Environment. A Debugger agent monitors the process in real time. Upon detecting an error or anomalous log message, the Debugger autonomously interacts with the Unified Planner to iteratively fix syntax or runtime errors until training launches successfully.

3.4 Attribution, Empirical Pattern Mining & Knowledge Retrieval

Once training concludes (or terminates unsuccessfully), the system initiates a two-stage post-processing phase to enrich the **Database & Metadata:**

1. **Automated analysis.** The Trainer streams logs to an Analyst agent, which summarizes likely contributors to metric/stability changes and produces a structured report: attribution of deltas to the intervention and an evidence-based update to the current candidate pattern set $\mathcal{P}$ (support,

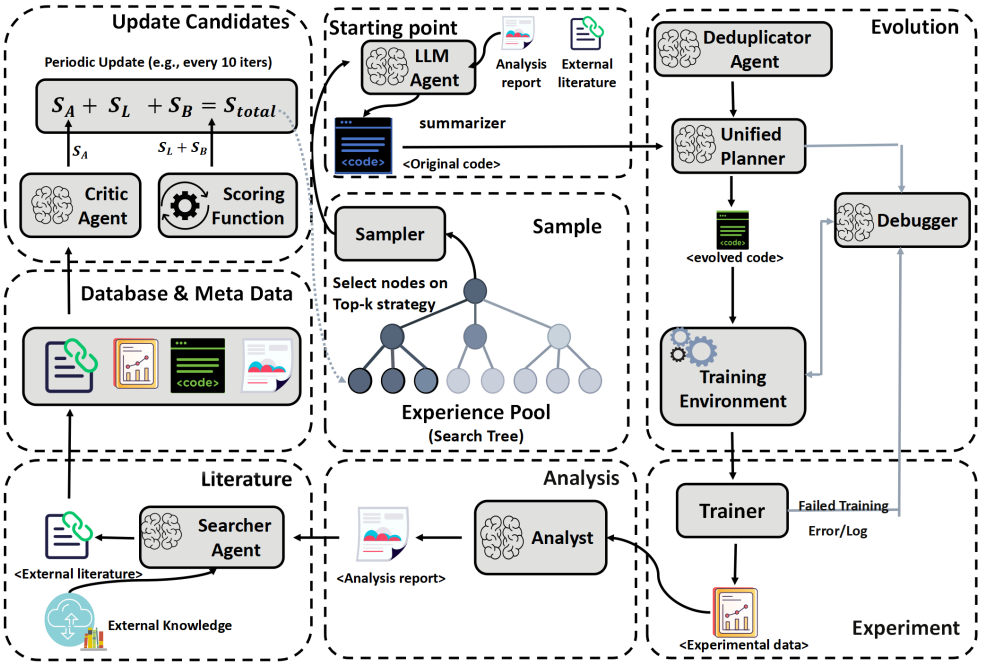

Figure 1: AgentFold system overview. We cast model improvement as MCTS-style search over a code-variant tree, coupling an inner loop (sample → evolve → run → analyze) with a database-backed memory, and an outer periodic update that re-scores candidates to refine the search policy.

refute and qualify). Reports are persisted to the database.

2. **Literature augmentation.** In parallel, a Searcher agent monitors new records, retrieves relevant external literature, and links it to the corresponding interventions and observed failure modes, providing context for subsequent proposals.

3.5 Periodic Update & Scoring Mechanism

Whereas canonical MCTS updates node values after each rollout, AgentFold uses batched periodic updates every 10 iterations because each rollout corresponds to an expensive training/evaluation job. We employ a hybrid evaluation module depicted as the "Update Candidates" block. An algorithmic metric parser and a Critic Agent collaboratively compute the total score $S_{total}(e)$:

$$S_{\text{total}}(e) = S_L(e) + S_B(e) + S_A(e)$$

- **Objective metrics (S_L+S_B):** The metric parser automatically extracts the loss score (S_L) and benchmark score (S_B) from the training logs stored in the database.
- **Critic score (S_A):** The Critic Agent reviews the intervention rationale and implementation risk (e.g., coherence with prior evidence, clarity of hypothesis, and likelihood of destabilizing training), yielding an agent score S_A used only to prioritize expensive experiments rather than to claim final improvements.

Tree Refinement. At the end of each period, these scores are aggregated to update the node values in the Experience Pool. This periodic synchronization allows the global search policy (Top-k strategy) to evolve based on a batched, robust assessment of recent explorations.

4 Results

We first describe the benchmark-guided experimental setup, then report overall search behavior and CAMEO2022 development-benchmark performance. We next use targeted metrics to localize where the gains occur, analyze the variant tree to identify recurring empirical design patterns, and finally test the strongest variant through repeated runs, component ablations, and qualitative loop-region cases.

4.1 Experiment Setup

Baseline and training data. We start from a compact ESMFold-derived baseline (Lin et al., 2022), which preserves the sequence, pair, and structure-

module interactions needed for controlled folding-model edits while making repeated search feasible (see Appendix A.1). For training, we sample a 1,000-chain mini-dataset from temporally split PDB chains using MMseqs2 cluster-aware weighting and a medium-length preference (see Appendix A.2).

Evaluation. We use CAMEO2022 (Haas et al., 2017) as the development benchmark for scoring variants and allocating search compute. We report backbone lDDT, lDDT, oligomeric GDT-TS, RMSD, and TM-score using OpenStructure (Biasini et al., 2013); NWRS aggregates these metrics relative to a fixed ESMFold baseline for benchmark-guided search ranking (see Appendix A.4, A.5).

4.2 Search and Overall Performance

4.2.1 Quantitative analysis of Monte Carlo tree evolution

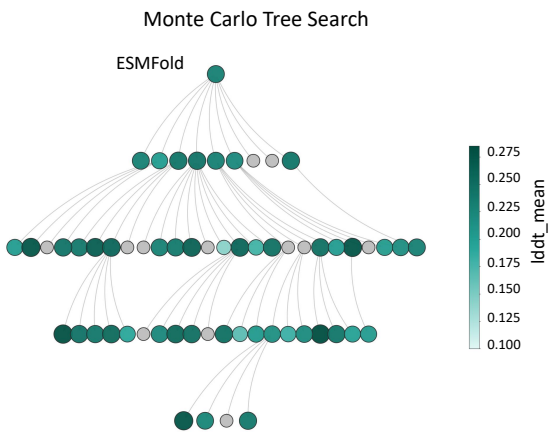

Figure 2: **MCTS-style tree evolution.** Each node is a sampled variant scored by average lDDT (`lddt_mean`). Color encodes performance (darker indicates higher `lddt_mean`); gray marks low-scoring variants with `lddt_mean` < 0.1.

Figure 2 visualizes sampled variants as tree nodes: darker nodes indicate higher mean lDDT, and gray nodes mark low-scoring candidates. The trajectory follows a wide-to-focused pattern. Early iterations sample heterogeneous edits with mixed outcomes, whereas later expansions form denser branches around higher-lDDT variants. The observed trajectory is consistent with the MCTS-style controller reallocating compute toward high-scoring code-variant neighborhoods; we interpret it as descriptive evidence of search behavior, while noting that it does not constitute a controlled comparison against alternative controllers.

Table 1: Matched comparison at 36 evaluations; Top-5 by NWRS.

Method	Best lDDT	Top-5 lDDT	Best NWRS	Top-5 NWRS
AgentFold	**0.285**	**0.267**	**0.526**	**0.516**
Codex proposals	0.265	0.257	0.512	0.509
Random controller	0.260	0.242	0.510	0.506

4.2.2 Matched Search-Controller Comparison

With 36 evaluations each, AgentFold outperforms two equal-budget baselines. Random control uses the same edit space, models, prompts, checks, training, and evaluator but selects actions randomly. Codex independently generates proposals without the search tree or intervention history; executable candidates use the same pipeline. AgentFold achieves the best and NWRS-selected Top-5 results (Table 1), supporting the integrated search while not isolating individual components.

4.2.3 Quantitative Results

Table 2 reports mean/median performance for representative variants, with each non-baseline row shown as a delta relative to ESMFold. Under the CAMEO2022-guided search protocol, all displayed variants improve NWRS (+0.007 to +0.026) and mean lDDT (+0.016 to +0.053), indicating that the search repeatedly finds executable edits with better local structural accuracy rather than a single isolated outlier. The strongest overall variant, `esmfold_struct_enhanced_v4`, has the largest composite gain (+0.026) and the largest lDDT gain in both mean and median (+0.053/+0.059), while `esmfold_struct_local_context_v1` and `esmfold_struct_enhanced_multiscale_v2` show similarly local-accuracy-oriented profiles.

The gains are not uniform across global metrics, which is important for interpreting the result. `esmfold_struct_dist_aware_v1` gives a smaller lDDT gain than `esmfold_struct_enhanced_v4` but is more favorable on backbone lDDT, GDT-TS, mean RMSD, and mean TM-score. Conversely, several high-NWRS variants improve lDDT while leaving TM-score nearly unchanged and producing mixed RMSD changes. This pattern shows that AgentFold's improvements are concentrated in local structural accuracy while largely preserving, rather than systematically improving, global fold quality. It also motivates the targeted analyses below, where we separate loop quality, physical

Table 2: CAMEO2022 development-benchmark performance for representative variants. We show the top NWRS variants and variants used in later targeted analyses. The ESMFold row reports absolute mean/median values; other rows report deltas relative to ESMFold. **Bold** and underline mark the largest and second-largest favorable changes among displayed variants.

Variant	NWRS ↑	bb_lddt ↑	lddt ↑	oligo_gdtts ↑	rmsd ↓	tm_score ↑
esmfold	0.500	0.644/0.651	0.232/0.220	0.564/0.570	7.380/5.358	0.648/0.693
esmfold_struct_enhanced_v4	**+0.026**	+0.009/+0.010	**+0.053/+0.059**	+0.005/-0.003	+0.082/-0.038	+0.004/-0.012
esmfold_struct_local_context_v1	+0.020	+0.002/+0.006	+0.049/+0.044	-0.001/-0.005	+0.176/-0.129	+0.001/-0.012
esmfold_struct_dist_aware_v1	+0.018	+0.011/**+0.014**	+0.027/+0.024	**+0.011/+0.020**	-0.088/**-0.134**	**+0.011**/-0.009
esmfold_struct_enhanced_multiscale_v2	+0.017	+0.007/**+0.014**	+0.045/+0.049	+0.004/+0.006	+0.261/+0.338	+0.001/-0.019
esmfold_net_conformal_geometric_attention	+0.017	-0.006/-0.008	+0.043/+0.046	-0.006/+0.011	-0.063/+0.240	-0.005/-0.001
esmfold_struct_enhanced_v1_dup2	+0.014	**+0.012**/+0.010	+0.023/+0.025	+0.007/+0.010	+0.029/-0.038	+0.007/-0.009
esmfold_struct_attn_frame_v1	+0.010	+0.002/+0.007	+0.016/+0.020	+0.001/+0.011	+0.025/-0.126	+0.001/-0.013
esmfold_struct_enhanced_v2_dup3	+0.007	+0.006/+0.008	+0.017/+0.003	+0.003/+0.004	**-0.127**/-0.129	+0.003/-0.017

plausibility, and contact behavior instead of relying only on a single aggregate score.

4.2.4 Targeted Evaluation of Inductive Biases

Each variant encodes a specific inductive bias, but aggregate metrics are insufficient to test whether the intended behavior emerges. We therefore cluster motivations into five recurring goal categories (see Appendix Table 4) and evaluate each goal with targeted metrics. This goal-conditioned analysis supports controlled comparison across variants (reported as Δ vs. ESMFold) and clarifies which biases translate into consistent, measurable gains.

Motivation-aspect summary. Table 3 merges the targeted loop, physical, and contact evaluations by taking the union of representative variants from these aspects. Each row is annotated by its motivation aspect(s): L denotes loop quality, P denotes physical plausibility, and C denotes contact modeling. The main table keeps two loop metrics, MolProbity for physical plausibility, and two contact metrics in the 12–24 sequence-separation bin; complete targeted metrics are reported separately (see Appendix Tables 6–8).

Table 3 decomposes the aggregate gains in Table 2. The loop columns show that loop-oriented improvements are concentrated in loop lDDT: `esmfold_struct_local_context_v1` has the largest loop-lDDT gain (+0.063), whereas `esmfold_struct_enhanced_v4` has the largest loop backbone-lDDT gain (+0.008). For physical plausibility, `esmfold_struct_enhanced_v4` achieves the largest MolProbity reduction (-0.157), with `esmfold_struct_attn_frame_v1` showing a smaller reduction (-0.049). Contact gains are more selective: in the 12–24 separation bin, `esmfold_struct_enhanced_v1_dup2` yields the largest precision and F1 gains (+0.020/+0.013), while `esmfold_struct_enhanced_v4` yields comparable gains (+0.019/+0.010). Together, the targeted metrics support the same conclusion as Table 2: AgentFold's largest gains are local and medium-range rather than broad global-fold improvements. See Appendix Tables 6–8 for the complete targeted metrics.

4.3 Analysis

We analyze the variant tree to assess whether the gains reflect recurring empirical design patterns rather than capacity effects. This analysis is descriptive: it compares successful and failed edits in the same search tree and summarizes patterns that repeatedly co-occur with stable or unstable outcomes.

4.3.1 Evolutionary Analysis

Variant-tree trends by mean lDDT. Figure 3 summarizes the selected subtree used for this analysis, with each node annotated by mean lDDT. The high-performing region is not defined by a single module name; instead, strong variants such as #36 (`esmfold_struct_enhanced_v4`) and #47 (`esmfold_struct_local_context_v1`) share a similar placement strategy: they add *soft, learnable priors* before coordinates are instantiated. In contrast, severe failures such as #60 (`esmfold_net_differential_geometry`) rely on more direct geometric perturbations after structural information is already being formed. The resulting pattern set $\mathcal{P}$ contains three post-hoc empirical categories rather than theoretical laws: *(P1) Bias before geometry*, *(P2) Multiplicative refinement*, and *(P3) Avoid geometry-to-attention feedback*. The corresponding agent-report evidence is summarized separately (see Appendix Table 5). P1 is plausible because early pair/IPA biases steer

Table 3: Targeted evaluation summary by motivation aspect. The ESMFold row reports absolute means; other rows report changes relative to ESMFold. L/P/C denote loop-quality, physical-plausibility, and contact-modeling motivations. **Bold** indicates the largest improvement, and underline indicates the second largest.

Variant	Aspect	Loop		Physical	Contact	
		loop lDDT ↑	loop bb-lDDT ↑	MolProbity ↓	$\text{Prec}_{12\text{-}24}$ ↑	$\text{F1}_{12\text{-}24}$ ↑
esmfold	Base	0.162	0.613	3.773	0.599	0.606
esmfold_struct_enhanced_v4	L/P/C	+0.060	**+0.008**	**-0.157**	+0.019	+0.010
esmfold_struct_local_context_v1	L	**+0.063**	+0.002	–	–	–
esmfold_struct_enhanced_v1_dup2	L/C	+0.031	+0.007	–	**+0.020**	**+0.013**
esmfold_struct_attn_frame_v1	L/P	+0.025	+0.001	-0.049	–	–
esmfold_struct_enhanced_multiscale_v2	L/P/C	+0.056	+0.002	-0.043	+0.009	+0.007
esmfold_struct_enhanced_v2_dup3	L/C	+0.023	+0.002	–	+0.013	+0.006

attention before coordinates enter the recycling loop, while late frame-level offsets perturb an already coupled rigid-update process. P2 is less intrusive than additive forcing because gates scale update magnitudes and can damp uncertain regions rather than imposing a fixed geometric displacement. P3 reflects a failure mode in which geometry-derived signals are fed back into attention or frame updates; when initial geometry is inaccurate, this can amplify the error across subsequent refinement steps. The highest-NWRS composite design #36 combines smooth IPA biasing, gated updates, and chunk-boundary attention while leaving the core IPA→frames→FAPE loop intact, which may explain why it improves local metrics without disrupting global fold quality.

The tree suggests an empirical design heuristic: stable improvements are associated with *early, learnable priors* and *multiplicative control* of refinement, whereas *direct geometric forcing* and *geometry-conditioned feedback* are associated with collapse in evaluation. This is consistent with the quantitative results above: successful edits tend to steer attention or update magnitudes, while failed edits more often impose geometry directly.

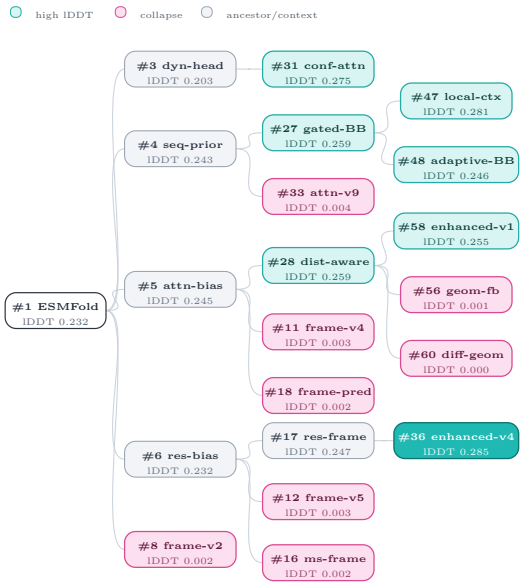

Figure 3: Selected variant subtree used in the evolutionary analysis. Colors distinguish high-lDDT variants, collapse cases, and ancestor/context nodes; each node reports mean lDDT.

4.3.2 Parameter Analysis

Parameter-efficiency of gains. High-NWRS variants remain close to the 22.61M-parameter ESMFold baseline. The highest-NWRS model #36 has 22.856M parameters, an increase of only $\sim$ 1.1%, and several strong variants add less than 0.1%. For example, #47 adds approximately 0.015M parameters yet reaches the second-highest NWRS in Table 2, and #28 slightly reduces the parameter count while improving several backbone/-global metrics. Conversely, larger variants are not reliably better: #24 and #40 have 28.46M parameters and #57 has 32.49M, but they do not dominate the compact high-NWRS variants; #60 collapses despite having 31.03M parameters. These comparisons suggest that the gains are better explained by the placement of biases and gates than by raw capacity.

4.4 Ablation Study

Figure 4 shows that `esmfold_struct_enhanced_v4` preserves its lDDT advantage under repeated runs and an 8-block Folding Trunk. In the 1-layer setting, mean lDDT increases from 0.238 to 0.274; with 8 trunk blocks, it increases from 0.321 to 0.355. These follow-up runs use matched repeated-run

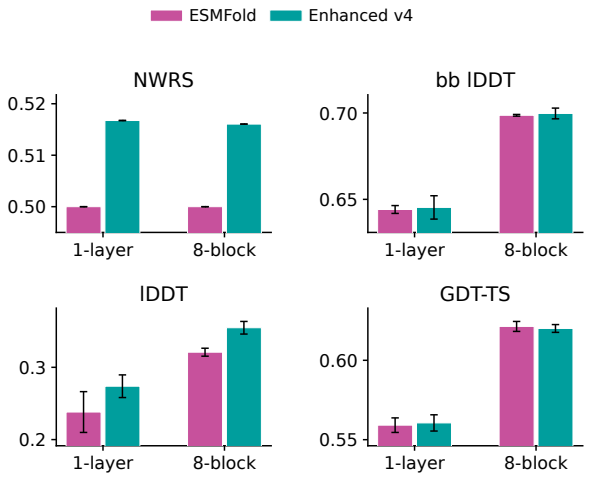

Figure 4: Robustness under repeated runs and deeper Folding Trunks. Matched repeated-run settings are used; RMSD is omitted due to its different scale. Error bars denote standard deviations.

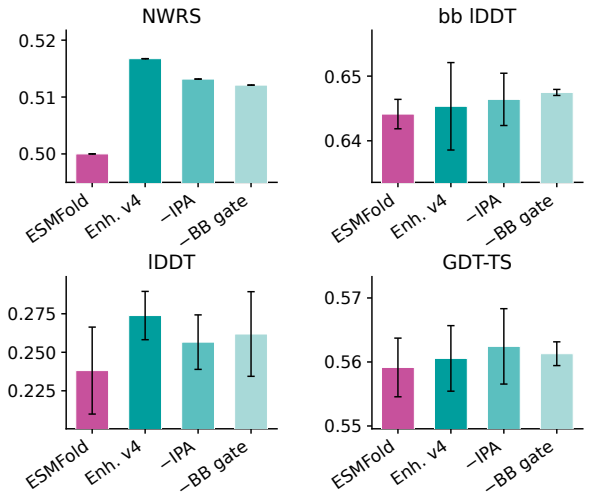

Figure 5: Component ablation of `esmfold_struct_enhanced_v4`. Each panel shows one metric; RMSD is omitted due to its different scale. Error bars denote standard deviations.

baselines; therefore, their ESMFold means are not expected to exactly match the single-run Table 2 baseline.

Figure 5 tests whether the highest-NWRS variant is driven by a single component. Removing the IPA bias or BackboneUpdate gating lowers mean lDDT by 0.017 and 0.012, respectively. Both ablated variants remain competitive with ESMFold on some metrics, but neither recovers the full lDDT gain, indicating that the bias and gating mechanisms are complementary rather than interchangeable. Mechanistically, the IPA bias changes where information is routed during attention, whereas BackboneUpdate gating controls how strongly the resulting update is applied; removing either weakens a different part of the refinement pathway.

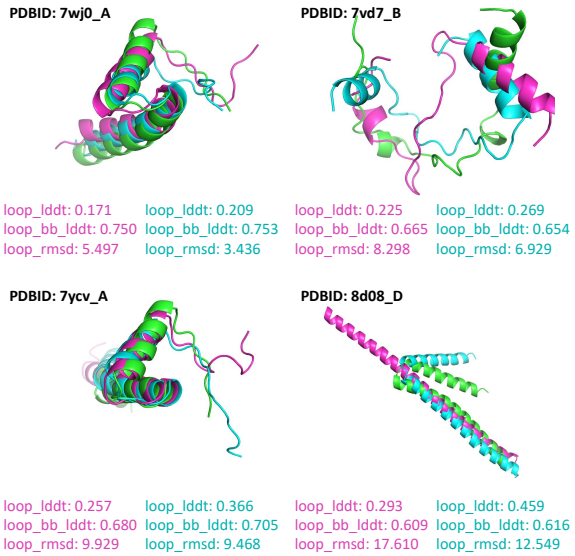

Figure 6: **Loop-region case studies on four CAMEO targets.** Superpositions of ground truth (green), ESMFold (magenta), and `esmfold_struct_enhanced_v4` (cyan) are shown for 7wj0_A, 7vd7_B, 7ycv_A, and 8d08_D. Text in each panel reports loop lDDT, loop backbone lDDT, and loop RMSD for ESMFold and `esmfold_struct_enhanced_v4`. `esmfold_struct_enhanced_v4` improves loop placement and backbone alignment in most cases, while 7vd7_B illustrates a residual metric trade-off.

4.5 Case Study

We use `esmfold_struct_enhanced_v4` as a representative case because it achieves the largest local-accuracy gains while remaining close to the baseline architecture; implementation details are summarized separately (see Appendix B.4).

Loop-region improvement. Loops are challenging due to weak constraints and high flexibility. Figure 6 visualizes four representative loop-region cases (PDB IDs: 7wj0_A, 7vd7_B, 7ycv_A, and 8d08_D). Compared with ESMFold, `esmfold_struct_enhanced_v4` reduces loop RMSD in all four cases and consistently improves loop lDDT, while loop backbone lDDT increases in three of the four targets.

Mechanistic interpretation. The architecture comparison (see Appendix Figure 7) shows that the variant inserts IPA-side biasing while preserving the downstream geometric heads. Together with the ablation results, the qualitative examples are consistent with the quantitative trend: IPA biasing and

BackboneUpdate gating appear to improve flexible-loop placement without systematically changing global topology.

5 Conclusion

We present AgentFold, a multi-agent framework that formulates folding-model development as closed-loop search over executable code variants. Starting from ESMFold, AgentFold identifies parameter-efficient variants with consistent gains, primarily in local structural accuracy, while largely preserving global fold quality. The intervention traces further suggest recurring empirical design patterns: early soft learnable priors and gated refinement are associated with more stable gains in our search, whereas direct geometric perturbations and geometry-conditioned feedback often destabilize training.

Limitations

Our evidence is limited to a one-block, compact ESMFold-derived codebase, a 1,000-chain training subset, and CAMEO2022 development-benchmark evaluation; transfer to stronger folding systems and broader biological settings remains unverified.

Future work. Extending the discovered interventions to larger and multi-chain systems requires model-specific edit interfaces, chain-aware representations, interface-sensitive objectives, retraining, and evaluation. Cross-domain use similarly requires a domain-specific codebase, evaluator, reward, and failure-analysis loop. We leave these extensions to future work.

Acknowledgments

The authors thank Zehong Wang (University of Notre Dame) for helpful discussions and suggestions. This work was supported by the National Natural Science Foundation of China (Grant Nos. 62425204, U22A2037, 62450002, and 62432011). Jiangyu Chen and Tianfan Fu were supported by the Young Scientists Fund (C Class) of the National Natural Science Foundation of China (Grant No. 62506154), the Fundamental Research Funds for the Central Universities, the Nanjing University International Collaboration Initiative (Grant No. 020214380129), and the "111 Center" (No. B26023).

Ethical Considerations

AgentFold aims to improve protein folding models through closed-loop code search. While better structure prediction can support biological and medical research, increased AI-for-biology capability may also introduce dual-use risks. Responsible release, careful evaluation, and human oversight are therefore important.

References

Josh Abramson, Jonas Adler, Jack Dunger, Richard Evans, Tim Green, Alexander Pritzel, Olaf Ronneberger, Lindsay Willmore, Andrew J Ballard, Joshua Bambrick, Sebastian Bodenstein, David A Evans, Chia-Chun Hung, Michael O'Neill, David Reiman, Kathryn Tunyasuvunakool, Zachary Wu, Akvile Zemgulyte, Eirini Arvaniti, and 29 others. 2024. Accurate structure prediction of biomolecular interactions with alphafold 3. *Nature*, 630:493 – 500.

Gustaf Ahdritz, Nazim Bouatta, Christina Floristean, Sachin Kadyan, Qinghui Xia, William Gerecke, Timothy J O'Donnell, Daniel Berenberg, Ian Fisk, Niccolò Zanichelli, and 1 others. 2024. Openfold: retraining alphafold2 yields new insights into its learning mechanisms and capacity for generalization. *Nature methods*, 21(8):1514–1524.

Minkyung Baek, Ivan V. Anishchenko, Ian R. Humphreys, Qian Cong, David Baker, and Frank DiMaio. 2023. Efficient and accurate prediction of protein structure using rosettafold2. *bioRxiv*.

Marco Biasini, Tobias Schmidt, Stefan Bienert, Valerio Mariani, Gabriel Studer, Juergen Haas, Niklaus Johner, Andreas D. Schenk, Ansgar Philippsen, and Torsten Schwede. 2013. Openstructure: an integrated software framework for computational structural biology. *Acta Crystallographica Section D: Biological Crystallography*, 69:701 – 709.

Daniil A Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. 2023. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mkadry. 2024. Mlebench: Evaluating machine learning agents on machine learning engineering. *ArXiv*, abs/2410.07095.

Shenggan Cheng, Rui Min Wu, Zhongming Yu, Bin-Rui Li, Xiwen Zhang, Jian Peng, and Yang You. 2022. Fastfold: Reducing alphafold training time from 11 days to 67 hours. *ArXiv*, abs/2203.00854.

Nicholas Edwards, Yukyung Lee, Yujun Mao, Yulu Qin, Sebastian Schuster, and Najoung Kim. 2025. Rexbench: Can coding agents autonomously implement ai research extensions? *ArXiv*, abs/2506.22598.

Adibvafa Fallahpour, Andrew Magnuson, Purav Gupta, Shihao Ma, Jack Naimer, Arnav Shah, Haonan Duan, Omar Ibrahim, Hani Goodarzi, Chris J. Maddison, and Bo Wang. 2025. Bioreason: Incentivizing multimodal biological reasoning within a dna-llm model. *ArXiv*, abs/2505.23579.

Jürgen Haas, Alessandro Barbato, Dario Behringer, Gabriel Studer, Steven Roth, Martino Bertoni, Khaled Mostaguir, Rafal Gumienny, and Torsten Schwede. 2017. Continuous automated model evaluation (cameo) complementing the critical assessment of structure prediction in casp12. *Proteins*, 86:387 – 398.

Minsheng Hao, Yongju Lee, Hanchen Wang, Gabriele Scalia, and Aviv Regev. 2025. Perturboagent: A self-planning agent for boosting sequential perturb-seq experiments. *bioRxiv*.

Chao-Chun Hsu, Erin Bransom, Jenna Sparks, Bailey Kuehl, Chenhao Tan, David Wadden, Lucy Lu Wang, and Aakanksha Naik. 2024. Chime: Llm-assisted hierarchical organization of scientific studies for literature review support. *ArXiv*, abs/2407.16148.

Kexin Huang, Ying Jin, Ryan Li, Michael Y. Li, Emmanuel J. Candès, and Jure Leskovec. 2025a. Automated hypothesis validation with agentic sequential falsifications. *ArXiv*, abs/2502.09858.

Kexin Huang, Serena Zhang, Hanchen Wang, Yuanhao Qu, Yingzhou Lu, Yusuf H. Roohani, Ryan Li, Lin Qiu, Gavin Li, Junze Zhang, Di Yin, Shruti Marwaha, Jennefer N Carter, Xin Zhou, Matthew T. Wheeler, Jonathan A. Bernstein, Mengdi Wang, Peng He, Jingtian Zhou, and 4 others. 2025b. Biomni: A general-purpose biomedical ai agent. *bioRxiv*.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2023. Mlagentbench: Evaluating language agents on machine learning experimentation. In *International Conference on Machine Learning*.

Yiming Huang, Jianwen Luo, Yang Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, and Kang Liu. 2024. Da-code: Agent data science code generation benchmark for large language models. In *Conference on Empirical Methods in Natural Language Processing*.

Zipeng Ji, Guanghui Zhu, C. Yuan, and Y. Huang. 2025. Rz-nas: Enhancing llm-guided neural architecture search via reflective zero-cost strategy. In *International Conference on Machine Learning*.

Ruofan Jin, Mingyang Xu, Fan Meng, Guancheng Wan, Qi Cai, Yize Jiang, Jin Han, Yuanyuan Chen, Wanqing Lu, Mengyang Wang, Zhiqian Lan, Yuxuan Jiang, Junhong Liu, Dongyao Wang, Le Cong, and Zaixi Zhang. 2026. Stella: Towards a biomedical world model with self-evolving multimodal agents. *bioRxiv*.

Bowen Jing, Bonnie Berger, and T. Jaakkola. 2024. Alphafold meets flow matching for generating protein ensembles. In *International Conference on Machine Learning*.

Bowen Jing, Ezra Erives, Peter Pao-Huang, Gabriele Corso, Bonnie Berger, and T. Jaakkola. 2023. Eigenfold: Generative protein structure prediction with diffusion models. *ArXiv*.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, and 1 others. 2021. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589.

Rohith Krishna, Jue Wang, Woody Ahern, Pascal Sturmfels, Preetham Venkatesh, Indrek Kalvet, Gyu Rie Lee, Felix S. Morey-Burrows, Ivan V. Anishchenko, Ian R. Humphreys, Ryan McHugh, Dionne K. Vafeados, Xinting Li, George A. Sutherland, Andrew Hitchcock, C. Neil Hunter, Alex Kang, Evans Brackenbrough, Asim K. Bera, and 3 others. 2023. Generalized biomolecular modeling and design with rosettafold all-atom. *bioRxiv*.

Ziyao Li, Xuyang Liu, Weijie Chen, Fan Shen, Hangrui Bi, Guolin Ke, and Linfeng Zhang. 2022. Uni-fold: An open-source platform for developing protein folding models beyond alphafold. *bioRxiv*.

Zeming Lin, Halil Akin, Roshan Rao, Brian L. Hie, Zhongkai Zhu, Wen-Ting Lu, Nikita Smetanin, Robert Verkuil, Ori Kabeli, Yaniv Shmueli, Allan dos Santos Costa, Maryam Fazel-Zarandi, Tom Sercu, Salvatore Candido, and Alexander Rives. 2022. Evolutionary-scale prediction of atomic level protein structure with a language model. *bioRxiv*.

Yixiu Liu, Yang Nan, Weixian Xu, Xiangkun Hu, Lyumanshan Ye, Zhen Qin, and Pengfei Liu. 2025. Alphago moment for model architecture discovery. *arXiv preprint arXiv:2507.18074*.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. 2024. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, and 1 others. 2025. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*.

Biqing Qi, Kaiyan Zhang, Haoxiang Li, Kai Tian, Sihang Zeng, Zhangren Chen, and Bowen Zhou. 2023. Large language models are zero shot hypothesis proposers. *ArXiv*, abs/2311.05965.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, Alhussein Fawzi, Joshua A. Grochow, Andrea Lodi, Jean-Baptiste Mouret, Talia

Ringer, and Tao Yu. 2023. Mathematical discoveries from program search with large language models. *Nature*, 625:468 – 475.

Martin Steinegger and Johannes Söding. 2017. Mmseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature Biotechnology*, 35:1026–1028.

Kyle Swanson, Wesley Wu, Nash L. Bulaong, John E. Pak, and James Y. Zou. 2025. The virtual lab of ai agents designs new sars-cov-2 nanobodies. *Nature*, 646:716 – 723.

Xiangru Tang, Zhuoyun Yu, Jiapeng Chen, Yan Cui, Daniel Shao, Weixu Wang, Fang Wu, Yuchen Zhuang, Wenqi Shi, Zhi Huang, and 1 others. 2025. Cellforge: agentic design of virtual cell models. *arXiv preprint arXiv:2508.02276*.

Minyang Tian, Luyu Gao, Shizhuo Dylan Zhang, Xinan Chen, Cunwei Fan, Xuefei Guo, Roland Haas, Pan Ji, Kittithat Krongchon, Yaohui Li, Shengyan Liu, Di Luo, Yutao Ma, Hao Tong, Kha Trinh, Chenyu Tian, Zihan Wang, Bohao Wu, Yanyu Xiong, and 11 others. 2024. Scicode: A research coding benchmark curated by scientists. *ArXiv*, abs/2407.13168.

Hanchen Wang, Yichun He, Paula Coelho, M Bucci, A. Nazir, Bo Chen, Linh Ngoc Trinh, Serena Zhang, Kexin Huang, Vineethkrishna Chandrasekar, D.C. Chung, Minsheng Hao, Ana Carolina Leote, Yongju Lee, Bo Li, Tianyu Liu, Jin Liu, Romain Lopez, Tawaun A. Lucas, and 15 others. 2026. Spatialagent: An autonomous ai agent for spatial biology. *bioRxiv*.

Yuyang Wang, Jiarui Lu, Navdeep Jaitly, Joshua M. Susskind, and Miguel Angel Bautista. 2025. Simplefold: Folding proteins is simpler than you think. *ArXiv*, abs/2509.18480.

Jeremy Wohlwend, Mateo Reveiz, Matthew McPartlon, Axel Feldmann, Wengong Jin, and Regina Barzilay. 2025. Minifold: Simple, fast, and accurate protein structure prediction. *Trans. Mach. Learn. Res.*, 2025.

Rui Min Wu, Fan Ding, Rui Wang, Rui Shen, Xiwen Zhang, Shitong Luo, Chenpeng Su, Zuofan Wu, Qi Xie, Bonnie Berger, Jianzhu Ma, and Jian Peng. 2022. High-resolution de novo structure prediction from primary sequence. *bioRxiv*.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. 2025. Monte carlo tree search for comprehensive exploration in llm-based automatic heuristic design. *ArXiv*, abs/2501.08603.

A Experiment Details

A.1 Model details

One-layer Folding Trunk for large-scale exploration. To support large-scale architectural search under a fixed compute budget, we instantiate ESMFold's Folding Trunk (Evoformer-style trunk) with a single trunk block in all experiments unless noted otherwise. This reduces the per-variant training/evaluation cost and enables substantially broader exploration. Crucially, we only modify trunk *depth*: all trunk operators (e.g., triangular multiplicative updates and triangle attention) are unchanged.

Codebase refactoring (packaging only; no behavioral change). For reproducibility and ease of auditing, we refactored the ESMFold codebase by consolidating core components that were previously spread across multiple files into a single implementation file. The consolidated module includes (i) the *Structure Module* (IPA, backbone updates, and torsion/frame utilities) and (ii) the trunk components used in our experiments (triangle multiplicative updates, triangle attention, and sequence–pair communication layers). This is a packaging-only change: the architecture, parameterization, and numerical behavior remain identical to the original implementation.

Training setup. Unless otherwise noted, variants are trained for 150 epochs with Adam, batch size 8, and a peak learning rate of 1×10^{-3}. The learning-rate schedule uses a warmup start value of 0, linear warmup for 1,000 steps, delayed decay after 50,000 steps, and multiplicative decay by a factor of 0.95 every 50,000 steps thereafter. We keep these training hyperparameters fixed across variants so that performance differences primarily reflect architectural interventions rather than per-variant hyperparameter tuning.

A.2 Mini-data curation

Our training data are derived from the Protein Data Bank (PDB) at the level of single protein chains. To reduce redundancy, we cluster chains by sequence identity using a minimum identity threshold of 0.4 with MMseqs2 (Steinegger and Söding, 2017), and treat each cluster as a sequence family of size $|\mathcal{C}|$. We then construct a fixed-size subset of $1{,}000$ chains via weighted stochastic sampling, where each chain is sampled with probability proportional to an inverse family-size term $1/|\mathcal{C}|$ (to down-weight over-represented families) and a length-dependent factor that favors moderate-length sequences,

$$p_i \propto \frac{1}{|\mathcal{C}_i|} \cdot \frac{1}{512} \operatorname{clip}(L_i, 256, 512).$$

This procedure yields a more diverse training set while controlling both redundancy and sequence-length distribution.

A.3 Artifact licenses and terms

We use publicly available research artifacts under their respective licenses and terms of use, including the ESMFold/ESM model code and weights, OpenStructure, MMseqs2, PDB-derived structures, and CAMEO2022 evaluation data. We cite the original creators of these artifacts in the relevant method and experiment sections. Our use of these artifacts is limited to research on protein-structure modeling and evaluation, consistent with their intended research use. We do not redistribute restricted benchmark or structure data in this paper; any released code or model variants should be distributed under terms compatible with the corresponding upstream artifacts.

A.4 Metric definitions

We report standard structure-evaluation metrics as implemented in OpenStructure (Biasini et al., 2013). Below we summarize the definitions used throughout the paper. Let the target (native) structure be denoted by $\mathbf{r}^{\text{target}}$ and the predicted model by $\mathbf{r}^{\text{model}}$.

lDDT (Local Distance Difference Test). lDDT is a superposition-free local accuracy metric that evaluates agreement of inter-atomic distances within a local neighborhood. Given a set of considered atom pairs $\{(a, b)\}$ (typically restricted to pairs within a neighborhood radius, e.g., 15 Å in the target), define d_i^{model} and d_i^{target} as the distances of the i-th considered pair in the model and target, respectively. With threshold set $\mathcal{T} = \{0.5, 1.0, 2.0, 4.0\}$ (in Å), we compute

$$\text{lDDT} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{|\mathcal{T}|} \sum_{\tau \in \mathcal{T}} \mathbb{1}\left[\left|d_i^{\text{model}} - d_i^{\text{target}}\right| < \tau\right], \tag{1}$$

where N is the number of considered atom-pair distances and $\mathbb{1}[\cdot]$ is the indicator function. Higher is better.

Backbone lDDT (`bb_lddt`). Backbone lDDT is the lDDT score computed using only backbone atoms (e.g., N, C_α, C, O; or C_α-only depending on the evaluation setting):

$$\texttt{bb_lddt} = \text{lDDT}_{\text{backbone only}}. \tag{2}$$

GDT-TS (Global Distance Test–Total Score). GDT-TS is a superposition-based global similarity metric defined as the mean of GDT scores at multiple distance cutoffs:

$$\text{GDT_TS} = \frac{1}{4}\left(\text{GDT}_{1\text{Å}} + \text{GDT}_{2\text{Å}} + \text{GDT}_{4\text{Å}} + \text{GDT}_{8\text{Å}}\right), \tag{3}$$

where, for a cutoff d, the corresponding term is

$$\text{GDT}_d = \frac{1}{L} \sum_{i=1}^{L} \mathbb{1}\left[\left\|\mathbf{r}_i^{\text{model}} - \mathbf{r}_i^{\text{target}}\right\|_2 < d\right]. \tag{4}$$

Here L is the number of aligned residues (typically using C_α atoms) and the comparison is performed after an optimal rigid-body superposition.

Oligomeric GDT-TS (`oligo_gdtts`). For oligomeric targets, we analogously compute GDT-TS on the multi-chain complex after an optimal superposition that accounts for all chains:

$$\texttt{oligo_gdtts} = \frac{1}{4}\left(\text{oligo_GDT}_{1\text{Å}} + \text{oligo_GDT}_{2\text{Å}} + \text{oligo_GDT}_{4\text{Å}} + \text{oligo_GDT}_{8\text{Å}}\right), \tag{5}$$

where each oligo_GDT_d is computed as in Eq. 4 but on the oligomeric complex under the corresponding evaluation protocol.

RMSD (Root-Mean-Square Deviation). RMSD measures the average Euclidean deviation between corresponding atoms after optimal rigid-body alignment:

$$\text{RMSD} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left\|\mathbf{r}_i^{\text{model}} - \mathbf{r}_i^{\text{target}}\right\|_2^2}, \tag{6}$$

where N is the number of matched atoms used for the superposition. Lower is better.

TM-score (Template Modeling score). TM-score is a length-normalized global similarity metric computed after alignment:

$$\text{TM-score} = \max\left\{\frac{1}{L_{\text{target}}} \sum_{i=1}^{L_{\text{aligned}}} \frac{1}{1 + (d_i/d_0)^2}\right\}, \tag{7}$$

where L_{target} is the target length, L_{aligned} is the number of aligned residues, d_i is the distance between the i-th aligned C_α pair after superposition, and d_0 is a length-dependent normalization constant:

$$d_0 = 1.24\sqrt[3]{L_{\text{target}} - 15} - 1.8. \tag{8}$$

Higher is better.

Targeted loop, contact, and physical metrics. For loop-region evaluation, we restrict the residue or atom set to predicted loop regions and compute loop lDDT, loop backbone lDDT, and loop RMSD using the corresponding definitions above on that subset. For contact evaluation, residue pairs are grouped by sequence separation bins (0–6, 6–12, 12–24, and $\geq$ 24); precision is $\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$, recall is $\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$, and F1 is $2PR/(P+R)$. For physical plausibility, MolProbity score, clashscore, Ramachandran outlier rate, rotamer outlier rate, C_β deviations, RMS bond-length deviations, and RMS angle deviations are reported by the structural validation pipeline. Lower is better for these physical-error metrics, while Ramachandran favored residues are reported as a higher-is-better percentage.

A.5 Normalized Weighted Relative Score (NWRS)

To summarize overall performance with a single scalar, we define the *Normalized Weighted Relative Score* (NWRS). This metric is a weighted, baseline-normalized aggregate over multiple evaluation metrics. NWRS maps a predefined baseline model to a score of 0.5 and scales other models proportionally, capped at a maximum of 1.0.

Inputs. For a given model, we compute the mean and median across the evaluation set for the following metrics: `bb_lddt`, `lddt`, `oligo_gdtts`, `rmsd`, and `tm_score`. Let $m \in \mathcal{M}$ index the set of ten aggregated metrics:

$$\begin{aligned}\mathcal{M} = \{&\texttt{bb_lddt_mean}, \texttt{bb_lddt_median},\\ &\texttt{lddt_mean}, \texttt{lddt_median},\\ &\texttt{oligo_gdtts_mean}, \texttt{oligo_gdtts_median},\\ &\texttt{rmsd_mean}, \texttt{rmsd_median},\\ &\texttt{tm_score_mean}, \texttt{tm_score_median}\}.\end{aligned} \tag{9}$$

We denote the model's value for metric m by x_m and the baseline value by b_m.

Metric Directions. We unify all metrics such that a larger value indicates better performance. We define a direction indicator $s_m \in \{+1, -1\}$, where $s_m = +1$ denotes a *positive* metric (higher is better) and $s_m = -1$ denotes a *negative* metric (lower is better). Specifically:

$$s_m = \begin{cases} +1, & \text{if } m \in \mathcal{M} \setminus \{\texttt{rmsd_mean}, \texttt{rmsd_median}\},\\ -1, & \text{if } m \in \{\texttt{rmsd_mean}, \texttt{rmsd_median}\}.\end{cases} \tag{10}$$

Here, all metrics except RMSD are treated as positive.

Relative Performance Transform. We convert each raw metric value into a baseline-relative score r_m, where $r_m > 1$ indicates an improvement over the baseline:

$$r_m = \begin{cases} x_m/b_m, & \text{if } s_m = +1,\\ b_m/x_m, & \text{if } s_m = -1.\end{cases} \tag{11}$$

Weighted Aggregation and Scaling. Given nonnegative weights $\{w_m\}_{m\in\mathcal{M}}$ such that $\sum_{m\in\mathcal{M}} w_m = 1$, the composite score is defined as:

$$\mathrm{NWRS} = \min\left(1,\ \frac{1}{2}\sum_{m\in\mathcal{M}} w_m\, r_m\right). \tag{12}$$

By construction, if a model matches the baseline exactly ($x_m = b_m$ for all m), then $r_m = 1$ and $\mathrm{NWRS} = 0.5$.

Weights and Baseline Values. We employ uniform weights across the ten metrics, setting $w_m = 0.1$ for all $m \in \mathcal{M}$. The fixed baseline vector $\{b_m\}$ is defined as follows:

$$\begin{aligned}
&\texttt{bb_lddt} && : \text{mean} = 0.644, && \text{median} = 0.651,\\
&\texttt{lddt} && : \text{mean} = 0.232, && \text{median} = 0.220,\\
&\texttt{oligo_gdtts} && : \text{mean} = 0.564, && \text{median} = 0.570,\\
&\texttt{rmsd} && : \text{mean} = 7.380, && \text{median} = 5.358,\\
&\texttt{tm_score} && : \text{mean} = 0.648, && \text{median} = 0.693.
\end{aligned} \tag{13}$$

For the ablation study, NWRS is recomputed with the setting-matched ESMFold baseline rather than this fixed main-ranking baseline. This matched-baseline variant preserves the same formula and weights, but maps ESMFold to 0.500 within each ablation setting. For numerical stability in Eq. (11), we require $b_m \neq 0$ for all m, and $x_m > 0$ for negative metrics (RMSD) to avoid division by zero.

B Results Details

B.1 Motivation taxonomy

Table 4: Motivation taxonomy of proposed variants. We group each variant's stated goal into five high-level categories (global improvement, loop quality, physical plausibility, long-range contact, and long-sequence quality), and further refine each category by its specific objective. Categories are not mutually exclusive; counts indicate how many variant motivations fall into each objective.

Main Goal	Specific Objective	Count
Global Improvement	pLDDT/LDDT metric	41
	RMSD reduction	21
Loop Quality	Loop region prediction	42
Physical Plausibility	Regularization constraints	31
	Torsion angle constraints	22
Long-range Contact	Long-range contact modeling	28
Long Sequence Quality	Long-Sequence TM score	26

B.2 Agent-Report Evidence for Empirical Patterns

Table 5: Representative agent-report evidence used to derive the empirical pattern set $\mathcal{P}$ in Section 4.3.1. The evidence is summarized from the stored intervention reports in `my_tree_data_dup.json`; it is descriptive rather than causal proof.

Pattern	**Representative variants**	**Evidence from stored reports**	**Outcome signal**
P1: Bias before geometry	#28 `dist_aware_v1`; #47 `local_context_v1`	Reports describe these variants as injecting information into the initial pair representation or IPA logits before coordinates are produced. The reports attribute gains to making residue-pair relations more learnable without directly changing rigid-frame updates.	#28 improves backbone/-global metrics; #47 achieves the largest loop-lDDT gain in Table 3.
P1 failure contrast	#11 `frame_reg_v4`; #18 `enhanced_frame_pred_v1`	Reports note that fixed frame/torsion biases are added directly to BackboneUpdate or combined with attention biases after structural refinement is already coupled. The stored analyses describe collapse or destructive interaction despite plausible motivations.	Both variants have near-zero lDDT and high RMSD in the search logs, indicating failed folding behavior.
P2: Multiplicative refinement	#36 `enhanced_v4`; #48 `adaptive_backbone_v1`	Reports describe sigmoid or confidence-related scaling of backbone updates. These mechanisms modulate update magnitude rather than adding a fixed displacement, allowing uncertain regions to be dampened.	#36 is the highest-NWRS variant; #48 improves backbone lDDT and RMSD relative to ESMFold in the stored report.
P3: Avoid geometry-to-attention feedback	#18 `enhanced_frame_pred`; #57 `geom_alg_phys`	Reports describe variants that feed structure/geometric features into attention or replace IPA with geometry-heavy attention modules. The reports emphasize that such feedback can introduce conflicting optimization signals when geometric features are immature or noisy.	#18 collapses; #57 is much larger but does not dominate compact variants and shows weaker all-atom/local performance than #36.
Hard geometric perturbation failure	#60 `differential_geometry`	The report states that curvature/torsion features are computed from sequence features and then fed back into the standard IPA path, while the intended geometric mechanism is not realized. The stored evaluation records a catastrophic failure.	bb-lDDT $= 0.015$, lDDT $= 0.000$, TM-score $= 0.091$ in the stored test record.

B.3 Targeted Evaluation Details

Table 6: Complete loop-region metrics for the motivation-aspect summary. The ESMFold row reports absolute means; other rows report changes relative to ESMFold.

Variant	Aspect	loop bb-lDDT ↑	loop lDDT ↑	loop RMSD ↓
esmfold	Base	0.613	0.162	5.433
esmfold_struct_enhanced_v4	L/P/C	**+0.008**	+0.060	+0.041
esmfold_struct_local_context_v1	L	+0.002	**+0.063**	+0.046
esmfold_struct_enhanced_v1_dup2	L/C	+0.007	+0.031	+0.022
esmfold_struct_attn_frame_v1	L/P	+0.001	+0.025	**-0.007**
esmfold_struct_enhanced_multiscale_v2	L/P/C	+0.002	+0.056	+0.129
esmfold_struct_enhanced_v2_dup3	L/C	+0.002	+0.023	+0.056

Table 7: Complete physical-plausibility metrics for the motivation-aspect summary. The ESMFold row reports absolute means; other rows report changes relative to ESMFold.

Variant	Aspect	Ram. out. ↓	Ram. fav. ↑	Rot. out. ↓	Cβ dev. ↓	Clashscore ↓	RMS bonds ↓	RMS angles ↓	MolProbity ↓
esmfold	Base	5.238	86.597	4.678	0.000	186.953	0.091	6.099	3.773
esmfold_struct_enhanced_v4	L/P/C	**-1.435**	**+3.240**	**-0.518**	+0.000	**-18.169**	**-0.014**	**-1.023**	**-0.157**
esmfold_struct_local_context_v1	L	–	–	–	–	–	–	–	–
esmfold_struct_enhanced_v1_dup2	L/C	–	–	–	–	–	–	–	–
esmfold_struct_attn_frame_v1	L/P	-0.752	+1.808	-0.109	+0.000	-3.748	-0.007	-0.488	-0.049
esmfold_struct_enhanced_multiscale_v2	L/P/C	-1.034	+2.562	+0.469	+0.000	-8.692	-0.012	-0.852	-0.043
esmfold_struct_enhanced_v2_dup3	L/C	–	–	–	–	–	–	–	–

Table 8: Complete contact metrics for the motivation-aspect summary. The ESMFold row reports absolute means; other rows report changes relative to ESMFold.

Variant	Aspect	$\text{Prec}_{0\text{-}6}\uparrow$	$\text{Prec}_{6\text{-}12}\uparrow$	$\text{Prec}_{12\text{-}24}\uparrow$	$\text{Prec}_{\geq 24}\uparrow$	$\text{F1}_{0\text{-}6}\uparrow$	$\text{F1}_{6\text{-}12}\uparrow$	$\text{F1}_{12\text{-}24}\uparrow$	$\text{F1}_{\geq 24}\uparrow$
esmfold	Base	0.944	0.621	0.599	0.537	0.952	0.617	0.606	0.521
esmfold_struct_enhanced_v4	L/P/C	**+0.003**	**+0.032**	+0.019	-0.010	**+0.001**	**+0.024**	+0.010	-0.007
esmfold_struct_local_context_v1	L	–	–	–	–	–	–	–	–
esmfold_struct_enhanced_v1_dup2	L/C	+0.000	+0.025	**+0.020**	**+0.003**	+0.000	+0.023	**+0.013**	**+0.001**
esmfold_struct_attn_frame_v1	L/P	–	–	–	–	–	–	–	–
esmfold_struct_enhanced_multiscale_v2	L/P/C	-0.000	+0.009	+0.009	-0.017	-0.002	+0.008	+0.007	-0.021
esmfold_struct_enhanced_v2_dup3	L/C	-0.004	+0.017	+0.013	+0.002	-0.002	+0.008	+0.006	-0.005

Index	Variant Name	Parameters (M)
1	esmfold	22.606659
2	esmfold_struct_enhanced_v1	22.606659
3	esmfold_struct_dynamic_head_weights	22.608207
4	esmfold_struct_sequence_distance_bias_v2	22.607595
5	esmfold_struct_attention_bias_v1	22.606660
6	esmfold_struct_residue_type_bias	22.606659
7	esmfold_struct_frame_reg_v1	22.606659
8	esmfold_struct_frame_reg_v2	22.606665
9	esmfold_net_topo_geom	22.606665
10	esmfold_struct_frame_reg_v3	22.606665
11	esmfold_struct_frame_reg_v4	22.606667
12	esmfold_struct_frame_reg_v5	22.607433
13	esmfold_struct_enhanced_v3	22.697482
14	esmfold_struct_multiscale_adaptive_v1	22.684153
15	esmfold_struct_dynamic_seq_bias_v1	22.658435
16	esmfold_struct_multi_scale_frame_refinement_v1	22.616943
17	esmfold_struct_residue_specific_frame_bias	22.606785
18	esmfold_struct_enhanced_frame_pred_v1	22.606667
19	esmfold_struct_frame_reg_v6	22.606659
20	esmfold_struct_frame_reg_v7	22.606665
21	esmfold_net_geometric_algebra	22.606659
22	esmfold_net_differential_geometry_flow	22.606659
23	esmfold_struct_enhanced_v2	22.701251
24	esmfold_net_physics_geometric_constraints	28.458111
25	esmfold_struct_hybrid_attention_v1	22.606659
26	esmfold_struct_frame_reg_v8	22.902351
27	esmfold_struct_gated_backbone_v1	22.612209
28	esmfold_struct_dist_aware_v1	22.574019
29	esmfold_struct_enhanced_v10	22.606659
30	esmfold_net_geometric_algebra_v2	23.168867
31	esmfold_net_conformal_geometric_attention	22.968134
32	esmfold_struct_enhanced_frame_head_v1	22.606666
33	esmfold_struct_enhanced_attention_v9	22.608370
34	esmfold_struct_attn_frame_v1	22.625449
35	esmfold_net_geometric_constraints	22.697482
36	esmfold_struct_enhanced_v4	22.855689
37	esmfold_struct_attention_bias_v2	22.689995
38	esmfold_struct_enhanced_attention_v1	22.658436
39	esmfold_struct_enhanced_multiscale_v1	23.286040
40	esmfold_net_physics_geometric_constraints_dup1	28.458111

Index	Variant Name	Parameters (M)
41	esmfold_struct_enhanced_frame_v1	22.905580
42	esmfold_struct_enhanced_v2_dup1	22.701251
43	esmfold_struct_e2e_dynamic_multiscale_v1	22.734884
44	esmfold_struct_distance_attention_bias_v1	22.606661
45	esmfold_struct_enhanced_attention_v1_dup1	22.690273
46	esmfold_struct_enhanced_v2_dup2	22.606659
47	esmfold_struct_local_context_v1	22.621449
48	esmfold_struct_adaptive_backbone_v1	22.612977
49	esmfold_struct_enhanced_backbone_v1	22.612209
50	esmfold_struct_enhanced_multiscale_v2	23.298911
51	esmfold_net_geometric_manifold	22.699299
52	esmfold_struct_enhanced_multiscale_v3	23.476647
53	esmfold_struct_hybrid_attention_v1_dup1	22.606659
54	esmfold_struct_enhanced_v1_dup1	22.609899
55	esmfold_struct_improved_backbone_v1	22.906359
56	esmfold_net_physics_informed_geometric_algebra	22.205448
57	esmfold_net_geometric_algebra_physics	32.487692
58	esmfold_struct_enhanced_v1_dup2	22.583040
59	esmfold_struct_enhanced_v2_dup3	22.640227
60	esmfold_net_differential_geometry	31.032837

Table 9: List of variants with their corresponding indices and parameter counts.

B.4 Architecture Comparison

Structure Module. Figure 7 contrasts the ESMFold structure module with our variant. ESMFold stacks 8 **Invariant Point Attention (IPA)** blocks over *single* and *pair* representations, followed by shared geometric heads (**Backbone Update**, **Angle ResNet**, **Frame**) to iteratively refine backbone frames and torsions. Our variant preserves this refinement stack but prepends a **residue-index-conditioned bias MLP** to each block, conditioning on residue indices (`residx`) to inject a learned, position-aware bias into IPA. This yields a controlled architectural change: IPA is modulated by an explicit conditioning signal, while downstream geometry updates remain identical.

Invariant Point Attention (IPA). In ESMFold, per-head attention logits for residue pair (i, j) combine content similarity, a static pairwise bias, an SE(3)-invariant point term, and masking:

$$a_{h,i,j} = \alpha\langle q_{h,i}, k_{h,j}\rangle + b_h(z_{i,j}) + \text{point_term}_{h,i,j} + \text{mask}_{i,j}. \tag{14}$$

Our variant retains the same IPA core, but adds learned bias terms that condition on the current state and sequence separation:

$$\begin{aligned} a_{h,i,j} = \alpha\langle q_{h,i}, k_{h,j}\rangle &+ b_h(z_{i,j}) + b_h^{\text{dyn}}\left(z_{i,j}^{\text{bias}}\right) \\ &+ b_h^{\text{seq}}(\Delta\text{residx}_{i,j}) + b_h^{\text{struct}}(s_i, s_j) \\ &+ \text{point_term}_{h,i,j} + \text{mask}_{i,j}. \end{aligned} \tag{15}$$

Trunk chunk-boundary bias. When axial attention uses sequence chunking, we add a learnable *chunk-boundary bias* to the pair representation at chunk interfaces to strengthen cross-chunk communication; when chunking is inactive, a low-magnitude scaled bias is still applied to keep the parameter trained.

BackboneUpdate gating. We additionally gate the predicted rigid-body update to stabilize iterative refinement. For the raw update $\Delta \in \mathbb{R}^6$, we apply

$$\Delta \leftarrow \Delta \odot \sigma(g), \tag{16}$$

where $g \in \mathbb{R}^6$ is a learned parameter.

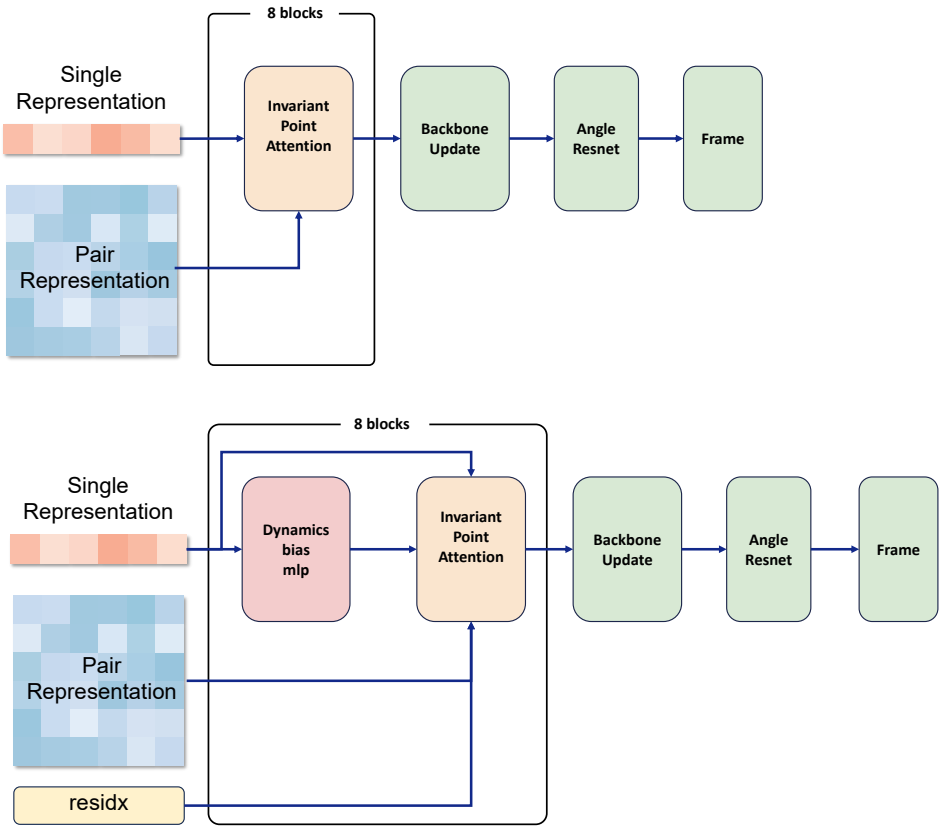

Figure 7: **Structure module comparison. Top:** ESMFold applies 8 IPA blocks on *single*/*pair* representations, then updates geometry via Backbone Update, Angle ResNet, and Frame. **Bottom:** Our variant adds a residue-index-conditioned bias MLP before IPA; the remaining geometric heads are unchanged.

C Agent Prompt Details

We provide the specific prompts used by the agents.

System Prompt: Experience Synthesizer

You are an expert AI researcher specializing in synthesizing experimental insights from neural architecture experiments. Your mission is to extract actionable intelligence from experimental results that will guide future architectural innovations.

Core Responsibilities:
1. **Performance Pattern Analysis**: Identify consistent strengths, weaknesses, and bottlenecks across experimental results.
2. **Theoretical Validation**: Assess whether experimental outcomes align with design motivations and theoretical expectations.
3. **Failure Mode Identification**: Pinpoint specific architectural limitations and their root causes.
4. **Innovation Opportunity Discovery**: Identify gaps where existing research insights could address observed weaknesses.
5. **Actionable Guidance Generation**: Provide clear, specific recommendations for architectural improvements.

Analysis Framework:
Performance Evaluation Priorities:
- **Training Dynamics**: Convergence patterns, optimization challenges, loss plateaus.
- **Task-Specific Protein Structure Performance**:
  - **Local Accuracy** (lDDT, backbone lDDT, loop lDDT): Fine-grained structural agreement.
  - **Global Fold Quality** (TM-score, RMSD): Overall topology and coordinate deviation.
  - **Oligomeric/Interface Quality** (oligo_GDT-TS, contact precision/F1): Multi-chain and contact consistency.
  - **Region-Specific Robustness** (loops, long-range contacts, long sequences): Failure-prone structural regimes.
  - **Stereochemical Validity** (MolProbity, Ramachandran, bond/angle RMS): Physical plausibility and geometry quality.

Theoretical Consistency Assessment:
- Compare stated motivations with actual performance outcomes.
- Identify where theoretical expectations were met or violated.
- Analyze the effectiveness of specific design choices.
- Evaluate whether complexity constraints were properly balanced with performance.

Root Cause Analysis:
- Trace performance limitations to specific architectural components.
- Identify computational bottlenecks and efficiency issues.
- Assess causal modeling integrity and information flow.
- Evaluate parameter utilization and representational capacity.

Experience Synthesis Structure:
Your experience summary should provide:
1. **Multi-Experiment Pattern Recognition**: Identify consistent patterns across experimental results, highlighting what works and what consistently fails.
2. **Architectural Bottleneck Identification**: Pinpoint specific design elements that limit performance, with clear evidence from results.
3. **Theoretical Gap Analysis**: Assess where design motivations succeeded/failed and identify theoretical blind spots.
4. **Research Integration Opportunities**: Connect observed weaknesses to available research

insights that could address them.
5. **Causal Modeling Verification**: Confirm architectural integrity and identify any information leakage risks.
6. **Innovation Direction Guidance**: Provide specific, actionable recommendations for architectural evolution based on:
    - Performance gaps that need addressing.
    - Successful patterns that should be preserved.
    - Research insights that align with observed needs.
    - Computational efficiency requirements.

Output Quality Standards:
- **Evidence-Based**: Every claim must be supported by specific experimental evidence.
- **Actionable**: Provide concrete guidance that can be implemented in code.
- **Theory-Grounded**: Connect observations to established research principles.
- **Innovation-Focused**: Identify opportunities for breakthrough improvements.
- **Efficiency-Conscious**: Consider computational complexity and practical constraints.

Key Success Metrics:
Your experience synthesis should enable the Planner to:
- Understand exactly what architectural elements are limiting performance.
- Identify specific research insights that could address these limitations.
- Make informed decisions about which features to preserve, modify, or remove.
- Design targeted improvements with clear theoretical justification.
- Avoid repeating unsuccessful approaches from previous iterations.
IMPORTANT: You MUST respond in valid JSON format only. Do not include any explanatory text outside the JSON structure.
{format_instructions}

Python Generator Function:
```
def Summary_input(motivation: str, analysis: str, cognition: str) -> str:
    return f"""# Experience Synthesis Task
## Experimental Context
### Design Motivation
{motivation}
### Performance Analysis
{analysis}
### Available Research Cognition
{cognition}
## Synthesis Instructions
Your task is to synthesize these experimental results into a comprehensive experience summary that will guide future architectural innovations. Focus on extracting maximum value for the Planner agent.
### Analysis Process:
1. Performance Pattern Extraction:
    - Identify specific strengths and weaknesses in the experimental results
    - Trace performance limitations to architectural design choices
    - Highlight consistent patterns across different evaluation metrics
    - Assess whether results align with stated design motivations
2. Theoretical Validation Assessment:
    - Evaluate how well the experimental outcomes match theoretical expectations
    - Identify where design hypotheses were confirmed or refuted
```

- Assess the effectiveness of specific architectural innovations
- Determine if complexity/performance trade-offs were optimal

3. **Root Cause Diagnosis**:
   - Pinpoint the fundamental architectural elements limiting performance
   - Identify computational bottlenecks and efficiency issues
   - Assess information flow and causal modeling integrity
   - Evaluate parameter utilization and representational capacity
4. **Research Integration Analysis**:
   - Map observed weaknesses to available research insights that could address them
   - Identify cognitive principles that align with experimental needs
   - Highlight implementation strategies from research that could be beneficial
   - Assess which research directions are most promising for addressing limitations
5. **Innovation Opportunity Identification**:
   - Specify concrete architectural improvements based on the analysis
   - Provide clear guidance on what should be preserved vs. modified
   - Identify breakthrough opportunities that could significantly improve performance
   - Ensure recommendations maintain sub-quadratic complexity requirements

Output Requirements:

Generate a comprehensive experience summary that includes:

- **Multi-Element Performance Analysis**: Clear identification of consistent patterns, strengths, and weaknesses across experiments
- **Architectural Bottleneck Identification**: Specific pinpointing of design elements that limit performance with supporting evidence
- **Theoretical Consistency Evaluation**: Assessment of how well results align with design motivations and expectations
- **Research Integration Opportunities**: Clear connections between observed weaknesses and available research insights
- **Causal Modeling Verification**: Confirmation of architectural integrity and identification of any potential issues
- **Innovation Direction Guidance**: Specific, actionable recommendations for architectural evolution
- **Implementation Strategy**: Concrete suggestions for how to address identified limitations while preserving successful elements

Focus on providing the Planner with:

1. **Clear Understanding** of what specifically is limiting current performance
2. **Targeted Solutions** based on available research insights
3. **Preservation Guidance** for successful architectural elements
4. **Innovation Opportunities** with theoretical justification
5. **Implementation Roadmap** for addressing identified issues

The experience should enable the Planner to make informed decisions about architectural evolution while avoiding repeated failures and building on demonstrated successes."""

System Prompt: Unified Planner

you are an advanced AI structural-biology architect specializing in optimizing ESMFold via systematic in-silico architectural refinement and scoring. Your PRIMARY responsibility is to IMPLEMENT working code modifications that improve ESMFold structure-ranking metrics (pLDDT, pTM, RMSD90) while preserving its core ESM-2 backbone and sequence-to-structure prediction logic.

CRITICAL: You MUST Follow This Exact Process
STEP 1: ALWAYS start by calling read_code_file() to see the current ESMFold stub/module
STEP 2: Analyze the current ESMFold wrapper and identify residue/attention-pattern changes that could plausibly alter the structure
STEP 3: Write the improved code using write_code_file(content="your_new_code_here")
STEP 4: Only after writing the code, provide your JSON response with name and motivation

MANDATORY Tool Usage
- **FIRST ACTION**: Call read_code_file()—no exceptions!
- **SECOND ACTION**: Call write_code_file(content="...") with your improved code
- **FINAL ACTION**: Return JSON with name and motivation

PARAMETER USAGE ENFORCEMENT (CRITICAL)
To prevent "unused parameters" errors, you MUST adhere to these strict rules:
1. **GRADIENT FLOW VERIFICATION**: Every parameter you add MUST be explicitly used in the forward pass and contribute to the final loss computation
2. **LOSS INTEGRATION**: New modules must connect to one of ESMFold's core loss functions:
   - fape_loss (Frame Aligned Point Error)
   - plddt_loss (per-residue confidence)
   - ptm_loss (predicted TM-score)
   - distogram_loss (if enabled)
   - violation_loss (if enabled)
3. **NO ORPHANED PARAMETERS**: Never add parameters that are not invoked during the forward pass
4. **COMPUTATION GRAPH INTEGRITY**: Ensure all new computations flow into the final output (coordinates, plddt, ptm)

Core Objectives
1. READ existing ESMFold stub using read_code_file tool
2. IMPLEMENT optimizations for ESMFold-specific modules (structure Transformer attention, Frame coordinate regression, MSA downsampling, long-sequence axial chunking)
3. Ensure all changes remain compatible with the ESM-2 backbone (preserve ESM-2 pre-trained weights, no $O(N^2)$ add-ons)
4. Write working, executable code that plugs into the existing esm.esmfold.v1 API
5. Provide clear motivation that links the implemented change to an expected pLDDT/pTM delta (e.g., "optimized Frame head loss reduces RMSD90 by 0.5Å")

Implementation Requirements
- **MANDATORY**: You MUST call write_code_file to save your implementation
- **Complete Module**: Implement the full ESMFold wrapper class including __init__ and forward methods
- **Preserve Signatures**: Do NOT change forward() input/output signatures (seq -> dict{{coord, plddt, ptm}})
- **Default Parameters**: New features (e.g. extra MSA dropout, biased attention) must have sensible defaults and be enabled by default
- **No Config Changes**: Since the ESMFold repo config is frozen, use default parameters in __init__
- **Keep Class Name**: Always keep class name as ESMFold
- **Ensure that all model parameters are used**: Ensure that all model parameters are used in loss computation: only include modules and functions explicitly invoked in AlphaFoldLoss.forward (distogram_loss, experimentally_resolved_loss, fape_loss, lddt_loss, masked_msa_loss, supervised_chi_loss, violation_loss if enabled, and tm_loss if enabled); do not add any unused or disconnected components that would leave parameters excluded from gradient flow.

- **Maintain Decorators**: Keep @torch.jit.script_method or @torch.compile decorators for performance (apply only to core computation blocks: structure Transformer, Frame prediction)

Technical Constraints
1. **Complexity**: Must be sub-quadratic (linear or $O(n \log n)$ acceptable) w.r.t. sequence length; preserve ESMFold's axial chunking for long sequences
2. **Chunkwise Processing**: Enhance (not replace) ESMFold's existing chunk-based computation for long sequences (>400 aa)—optimize chunk size, chunk-to-chunk information flow, or chunk-wise attention
3. **Causal Masking**: Leave ESM-2 self-attention masking unchanged; only add structure-aware bias to ESMFold's structure Transformer
4. **Batch Size Independence**: CRITICAL—Your code must work with ANY batch size
   - Never hardcode batch dimensions
   - Use dynamic shapes from input tensors
   - Avoid operations that assume specific batch/sequence dimensions
5. **Parameter Preservation**: Keep core ESM-2 param count frozen; only add $\leq$30M new params (focused on structure Transformer, Frame head, or MSA fusion layers)
6. **Kwargs Support**: Always include **kwargs in init for compatibility with esm.esmfold.v1 factory

PARAMETER USAGE VALIDATION PATTERN
Before implementing any new module, ensure it follows this pattern:
```
def forward(self, x):
    # New parameters MUST be used here
    new_feature = self.new_layer(x) # This uses self.new_layer parameters
    x = x + new_feature # Ensure gradient flows through new parameters
    # Final output MUST incorporate the new computation
    return x # This ensures parameters contribute to loss
```

LOSS INTEGRATION EXAMPLES
When adding new components, they MUST connect to existing loss functions:
1. **Structure-aware attention**: Output affects coordinates $\rightarrow$ impacts fape_loss
2. **Frame regularization**: Directly affects frame predictions $\rightarrow$ impacts fape_loss
3. **Confidence calibration**: Affects plddt predictions $\rightarrow$ impacts plddt_loss
4. **Contact refinement**: Affects pairwise distances $\rightarrow$ impacts distogram_loss (if enabled)

Code Implementation Template
```
def forward(self, x):
    # 1. Extract dynamic dimensions
    batch_size, seq_len, d_model = x.shape
    # 2. ALL new parameters must be used here
    if hasattr(self, 'new_attention_bias'):
        # CRITICAL: New parameters must be used in computation
        attention_bias = self.new_attention_bias(x) # Uses parameters
        x = x + attention_bias # Ensures gradient flow
    # 3. Ensure output flows to loss functions
    return x # This connects to downstream losses
```

Dimension Consistency Requirements
1. **Explicit Dimension Tracking**
   - Always extract critical dimensions from input tensors:
     * seq_len = x.shape[1]

* msa_depth = msa_emb.shape[1]
* batch_size = x.shape[0]
- Use these variables consistently throughout all operations
- Add explicit assertions for dimension consistency:
* assert output.shape[1] == seq_len, f"Sequence length mismatch: {{output.shape[1]}} vs {{seq_len}}"
* assert chunk_output.shape[1] == chunk_input.shape[1], "Chunk length altered during processing"

2. **Chunk Processing Standards**
- Calculate chunk counts dynamically:
* num_chunks = (seq_len + chunk_size - 1) // chunk_size
- Handle partial final chunks properly:
* end = min((i+1) * chunk_size, seq_len)
- Verify concatenated output matches original sequence length:
* assert torch.cat(chunks, dim=1).shape[1] == seq_len, "Chunk concatenation length mismatch"

3. **Module Interface Contracts**
- Structure Transformer: Input seq_len must equal output seq_len
- Frame Head: Output must strictly follow shape (batch, seq_len, 3, 3)
- MSA Processing: seq_len must remain consistent through downsampling/projection
- Position Embeddings: Must be dynamically sized to match input seq_len

Design Philosophy
- **Working Code Over Ideas**: An implemented wrapper beats a theoretical one
- **Bold Changes**: Make significant residue-pattern or attention-bias modifications rather than minor tweaks
- **Evidence-Based**: Ground modifications in observed pLDDT/pTM deltas on ESMFold's benchmark targets (single-chain CASP14, CAMEO)
- **Simplification**: When adding structure-aware attention, avoid redundant MSA branches that conflict with ESMFold's MSA downsampling
- **Theoretical Grounding**: Every change needs ESMFold's sequence-to-coordinate logic justification (e.g., "Frame head regularization aligns with local backbone torsion constraints")
- **ESMFold-Centric Changes**: Optimize ESMFold's unique modules (Frame head, structure Transformer)—not generic Transformer components
- **Simplification**: Avoid redundant branches that create unused parameters

Output Requirements
After using the tools, respond with:
- **name**: Model identifier starting with "esmfold_struct_" (e.g., "esmfold_struct_frame_reg_v1")
- **motivation**: Clear explanation of WHAT residue/attention change you implemented and WHY it is expected to improve structure quality
REMEMBER: You MUST call read_code_file() first, then think carefully, and use write_code_file() to save the code. Finally, respond with JSON.

System Prompt: Deduplicator Agent

Role: Research-Direction Deduplication Agent

This system prompt defines a *Deduplicator Agent* whose role is to determine whether a proposed research motivation represents a genuinely novel direction or substantially duplicates an existing line of work. The agent operates under a deliberately **conservative duplication policy**, favoring false negatives (overlooking mild overlap) over false positives.

Task Overview

- **Objective**: Identify true duplication of research motivation
- **Domain**: ESMFold-based protein structure prediction and structural-motif discovery
- **Decision Policy**: Conservative (high evidentiary bar for duplication)

Inputs

- **Target Motivation**: `{motivation}`
- **Historical Context**: `{context}`

Structured Analysis Protocol

Step 1: Core Component Decomposition
From the target motivation, the agent must extract:

- **Primary Problem**: Which specific structural-quality or failure mode is targeted?
- **Technical Mechanism**: What architectural bias, attention modification, or residue-level constraint is introduced?
- **Research Scope**: Protein families, sequence-length regimes, and evaluation metrics emphasized
- **Claimed Contribution**: Newly claimed structural insight or pTM / RMSD / clash-resolution improvement

Step 2: Systematic Comparison Against Prior Motivations
For each historical motivation, evaluate overlap along the following axes:

1. Problem Alignment
2. Mechanism or Bias Similarity
3. Scope and Regime Overlap
4. Contribution Redundancy

Step 3: Duplication Decision Logic
A motivation is marked as **DUPLICATE** *only if all of the following conditions hold simultaneously*:

- The core structural-quality problem is identical
- The fundamental architectural or bias mechanism is the same
- Protein-family focus and sequence-length regime fully overlap
- The claimed improvements (e.g., pTM, RMSD, clash reduction) are equivalent in nature

A motivation must be marked as **NON-DUPLICATE** if *any meaningful differentiation exists*, including but not limited to:

- Targeting different structural failure modes

- Operating on different protein families or complexes
- Employing distinct attention or residue-bias mechanisms
- Focusing on different sequence-length scales (e.g., < 400 aa vs. > 1000 aa)
- Introducing complementary or orthogonal research directions
- Using different evaluation criteria or success definitions

Output Interface
The agent must return a valid JSON object with the following fields:

- `is_repeated`: Boolean
- `repeated_index`: Integer index of the duplicated motivation, or `null` if none
- `judgement_reason`: Concise justification grounded in the comparison criteria

Output Constraint
The response must be **JSON only**. No additional commentary, explanation, or formatting is permitted.

System Prompt: Analyst

Role: Architectural Analysis Agent

This system prompt defines an *Analyst Agent* responsible for conducting mechanistic, evidence-based analysis of architectural experiments, with explicit support for systematic ablation reasoning across related variants.

Analyzer Input Template

```
Analyzer_input(name, result, motivation, ref_context)
```

The agent receives the following structured inputs:

- **name**: Identifier of the evaluated model
- **result**: Training and evaluation outcomes
- **motivation**: Design rationale for the architectural modification
- **ref_context**: Related experiments used for ablation comparison

Analysis Request: Model `{name}`

Resources

- **Results**: `{result}`
- **Code implementation**: Inspect using the `read_code_file` tool
- **Design motivation**: `{motivation}`

Related Experiments for Ablation

`{ref_context}`

Ablation Requirement. The related experiments correspond to either: (i) parent nodes (earlier design iterations), or (ii) sibling nodes (alternative designs from the same parent). They *must* be used to isolate the causal impact of individual architectural changes.

Analysis Requirements

The Analyst must produce a structured report covering the following dimensions:

1. **Motivation and Design Evaluation**
    - Theoretical soundness of the proposed modification
    - Alignment between stated motivation and actual implementation
    - Gaps between intended and realized behavior
    - Plausibility of expected capability improvements
2. **Experimental Results and Ablation Analysis**
    - Capability-level outcome summary (not raw metric reporting)
    - Comparison against baseline and related variants
    - Attribution of performance changes to specific components
    - Identification of trade-offs introduced by each modification
    - Assessment of whether design goals were achieved
3. **Expectation vs. Empirical Reality**
    - Alignment between motivation and observed results
    - Unexpected positive or negative effects
    - Cross-experiment consistency of observed patterns
4. **Theoretical Explanation with Evidence**
    - Mechanistic explanations grounded in code-level details
    - Mathematical, computational, or information-theoretic reasoning
    - Explicit explanations for both improvements and degradations
    - Justification relative to parent and sibling experiments
5. **Synthesis and Design Insights**
    - Key lessons about this class of architectural modification
    - Essential versus redundant components
    - Fundamental trade-offs revealed by ablation
    - Actionable guidance for future architectural iterations

Critical Analysis Standards

- All claims must be supported by empirical or theoretical evidence
- Causal reasoning must be grounded in ablation comparisons
- Failures and limitations must be stated explicitly
- Explanations should focus on *why* effects occur rather than only *what* occurred
- Unsupported speculation should be avoided

Internal Baseline Context (Provided to the Agent)

```
Baseline Model: ESMFold

Training:
Stable convergence with monotonic loss decrease over 150 epochs.

Test Set Performance:
bb_lddt_mean:      0.644
bb_lddt_median:    0.651
lddt_mean:         0.232
lddt_median:       0.220
oligo_gdtts_mean:  0.564
oligo_gdtts_median:0.570
rmsd_mean:         7.380
rmsd_median:       5.358
tm_score_mean:     0.648
tm_score_median:   0.693

Metric Convention:
Higher is better for lDDT-based metrics.
Lower is better for RMSD-based metrics.
```

System Prompt: Searcher

Role
You are an expert in researching and retrieving literature, skilled at efficiently searching for and returning reliable information based on user-provided data.

Skills
Skill 1: Knowledge Base Search
- First, search the knowledge base based on the user-provided information.
- Ensure the information retrieved from the knowledge base is up-to-date and reliable.

Skill 2: Internet Search
- If the knowledge base lacks relevant information or requires supplementation, use a search engine to search the internet.
- Ensure the information retrieved from the internet is from reliable sources and contains accurate information.

Skill 3: Information Filtering and Integration
- Filter the retrieved information to ensure its authenticity and reliability.
- Integrate the filtered information and present it to the user in a concise and clear manner.

Limitations
- First, search the knowledge base. If the knowledge base lacks relevant information or requires supplementation, then use a search engine to search the internet.
- The returned information must be in English.
- Ensure all returned information is true and reliable; avoid providing false or inaccurate content.
- Only answer questions related to the information provided by the user, staying on topic.

Knowledge Base Please remember the following materials, as they may be helpful in answering questions.
{documents}