TRELLIS: AGENT-ORCHESTRATED DESIGN AND EVOLUTION OF BIOLOGICAL SEQUENCE-BASED PROPERTY PREDICTORS

Anonymous authors
Paper under double-blind review

ABSTRACT

Predicting biological properties such as expression, immunogenicity, stability, and binding affinity of proteins is a critical step in many drug discovery pipelines. New experimental datasets and protein sequence representations create opportunities to improve these predictors, but updating them requires repeated engineering of features, architectures, and training procedures. To address this, we introduce TRELLIS, an agent-orchestrated framework for developing biological sequence based property predictors. Large language model (LLM) agents write and execute code in isolated environments, and a monte Carlo tree search explores diverse combinations of sequence representations and model architectures. Across FLIP, Peptiverse, and major histocompatibility complex (MHC) class I pathway tasks, a multi turn tool calling agent produced predictors with scores above the listed literature references on eight tasks. Tree search improved test performance over the best Qwen3.8-27B baseline on seven of eight selected tasks. We also evaluated sequential updates of an MHC class I elution predictor using Immune Epitope Database (IEDB) release-year holdouts. We release our code which accepts any labeled sequence-property dataset without task-specific modification, and all the generated trajectories at `https://anonymous.4open.science/r/trellis-B12C`.

1 INTRODUCTION

Predicting protein properties from sequence supports candidate selection in protein engineering and drug discovery. Dedicated predictors estimate stability (Blaabjerg et al., 2023), fitness (Didi et al., 2026), solubility (Hebditch et al., 2017), and peptide presentation (Nilsson et al., 2025). Each predictor typically requires manual implementation and tuning, and subsequent updates depend on continued maintenance as experimental data accumulates.

In addition, protein sequence representations used to build these predictors continue to evolve and pretrained protein language models such as ESM-2 support representations that differ substantially from those used in earlier benchmarks (Lin et al., 2023). Updating a predictor therefore requires renewed choices about representations, architectures, and training procedures. These choices also affect generative protein design, where property predictions guide sampling across several objectives (Tang et al., 2025; Hong & Kortemme, 2024; Emami et al., 2023). Maintaining accurate predictors is consequently relevant to both ranking existing candidates and generating new ones.

LLM agents can automate parts of this development process by writing, executing, and revising machine learning models. Scientific agent systems increasingly support hypothesis development and computational experimentation (Gottweis et al., 2026; Lu et al., 2024; Yamada et al., 2025; Swanson et al., 2025). For automated machine learning engineering, AIDE and related systems search over candidate programs using validation feedback (Jiang et al., 2025; Chi et al., 2024; Nam et al., 2025). Biological sequence prediction adds a consequential choice of representation, including protein language model embeddings, chemical fingerprints, and sequence-derived features. A successful training script alone does not establish that this choice generalizes to new sequences.

Here, we introduce TRELLIS to automatically develop and maintain sequence-to-property predictors from labeled datasets and task specifications. We first evaluated a single multi-turn agent on

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

each task to establish baseline performance and code execution reliability. On tasks with baseline scores below their literature references, we then performed tree search over sequence representation-architecture combinations. A diversity judge compares proposed plans with a persistent log, and separate validation splits support search and final selection. To extend this process to changing datasets, we also evaluated a configurable agent loop for updating an elution predictor across IEDB releases.

The evaluation spans FLIP protein fitness tasks (Didi et al., 2026), Peptiverse peptide properties (Zhang et al., 2026), and components of the MHC class I pathway. The first two are fixed benchmarks with published references; the last is versioned, so the same predictor is rebuilt as new measurements become available. Our contributions are as follows:

- We introduce an agentic workflow for protein sequence property prediction with isolated code execution, user-side scoring, and search over representation-architecture plans.
- We evaluate single-agent iteration and diversity-controlled tree search across protein and peptide tasks, including unsuccessful submissions and tasks with limited generalization.
- We examine sequential predictor updates on IEDB release-year holdouts using an outer agent that adjusts the inner development loop while preserving the evaluation protocol.

2 Related Work

Sequence-property predictors. Dedicated predictors support several stages of MHC class I antigen processing and recognition. NetChop estimates proteasomal cleavage (Nielsen et al., 2005), NetMHCpan supports binding and presentation prediction (Nilsson et al., 2025), and MHCflurry combines binding and antigen-processing information (O'Donnell et al., 2020). T-cell receptor (TCR) recognition presents additional generalization challenges when evaluation peptides differ from those used for training (Grazioli et al., 2022). Similar representation and model-selection choices arise across protein fitness and peptide-property benchmarks (Didi et al., 2026; Zhang et al., 2026).

LLM agents for scientific discovery. Co-Scientist supports hypothesis generation and refinement (Gottweis et al., 2026), and The AI Scientist combines ideation, experimentation, and manuscript generation (Lu et al., 2024; Yamada et al., 2025). Using the Virtual Lab, Swanson et al. (2025) applied agent collaboration to nanobody design. Robin and Kosmos combine literature search with data analysis in scientific workflows (Ghareeb et al., 2026; Mitchener et al., 2025). Such applications demonstrate agent-assisted scientific computation and motivate evaluation of repeated biological predictor development.

Agents for machine learning engineering. AIDE searches a tree of candidate programs using validation performance (Jiang et al., 2025). SELA and MLE-STAR also use structured search or refinement for machine learning tasks (Chi et al., 2024; Nam et al., 2025). MLAgentBench and MLE-bench evaluate such capabilities across computational tasks (Huang et al., 2024; Chan et al., 2025). We build on this approach by explicitly searching sequence representations alongside downstream architectures.

Automated biological prediction. BioAutoML and BioAutoML-FAST automate feature engineering and model selection for biological sequences (Bonidia et al., 2022; de Almeida et al., 2026). More recent agent benchmarks examine end-to-end biomedical machine learning, including BioML-bench and BioXArena (Miller et al., 2025; Li et al., 2026). Within this setting, TRELLIS combines code-generating agents with diversity-controlled searches over sequence representations and predictor architectures. We further examined agent-generated changes to the development loop in the sequential-update experiment.

3 METHODS

3.1 PROBLEM SETUP AND DATASETS

A labeled dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ contains N examples, where x_i is a protein or peptide sequence, or a sequence pair, and y_i is its property label. The goal is to construct a predictor f_θ that maps an input x to a predicted label and generalizes to held-out data. The metric used to evaluate the models is passed via task prompt. We used curated tasks from three benchmark families. FLIP is a protein fitness landscape benchmark that contains tasks for predicting the stability or activity of a protein given its sequence and contains different splits for the same task based on the number of mutations (Didi et al., 2026). The reported baseline evaluation covers six datasets, with close-to-far, low-to-high, two-to-many, single-to-double, and by-wild-type splits. Peptiverse includes amino-acid and SMILES inputs for peptide properties, with both classification and regression labels (Zhang et al., 2026). The MHC class I tasks concern proteasomal cleavage, eluted peptide presentation, and TCR recognition (Drost et al., 2025). These are distinct molecular interactions along the MHC Class I pathway.

3.2 CODING MODELS AND ISOLATED EXECUTION

We evaluated Qwen3.8-27B and GLM-5.3 as coding models and self-usered these models on Modal. The agent receives a task description, split identity, training set sizes, input paths to mounted data files, and a tool catalog. It generates the representation, architecture, and training code, executes it with an isolated runtime, and submits a solution by calling the submit tool. CPU environments include NumPy, pandas, scikit-learn, and SciPy. GPU environments also include PyTorch, Transformers, PEFT, fair-esm, Biopython, and multimolecule, with additional packages installed as needed by the agent.

The runtime contains labeled training examples and unlabeled validation inputs. A submitted `solution.py` by the agent should be able to run end to end, fit the predictor, and return validation predictions, which the user evaluator scores against separately stored labels. user-computed metrics determine search updates and selections. Test inputs and labels remain unavailable during the search, and the user evaluates the finalized solution on the test partition.

3.3 SINGLE-AGENT BASELINE

We evaluated a single multi-turn agent trajectory under a six-hour budget. The agent operates in a persistent runtime, with each tool result returned as an observation for the next turn. The four tools support model-usage hints (`lookup_hint`), known runtime and API errors (`lookup_common_errors`), Python execution (`execute_code`), and submission of a complete solution (`submit_answer`). After submission, the user executes the script end to end. A failed execution returns a traceback for agent to debug and resubmit, and an accepted submission ends the trajectory. This interaction continues until submission succeeds or the run budget is exhausted (Figure 1).

3.4 TREE SEARCH OVER REPRESENTATION AND ARCHITECTURE PLANS

Single-agent trajectories often retained an early representation choice. To explore other combinations, we ran Monte Carlo tree search (MCTS) over high-level plans, each pairing a sequence representation with a predictor architecture (Figure 2). Four language-model roles operate with separate contexts: an *orchestrator* proposes plans; a *diversity judge* approves a plan only if it differs from every prior attempt using memory of previous actions in its context; an *implementation subagent* converts an approved plan into a self-contained script, `solution.py`, and chooses its training details; and a *summarizer* compresses each outcome into a short record and add its to the memory. Scoring is performed by the user, not by a language model.

Each node stores its plan, parent, depth, and execution status, and, if execution succeeds, its script and validation score. After every execution, the summarizer receives the plan with the resulting script or error trace and appends a summary to a shared file, `memory.jsonl`. The summarizer never reads this file, so each record describes one attempt independently. Because crashes are also

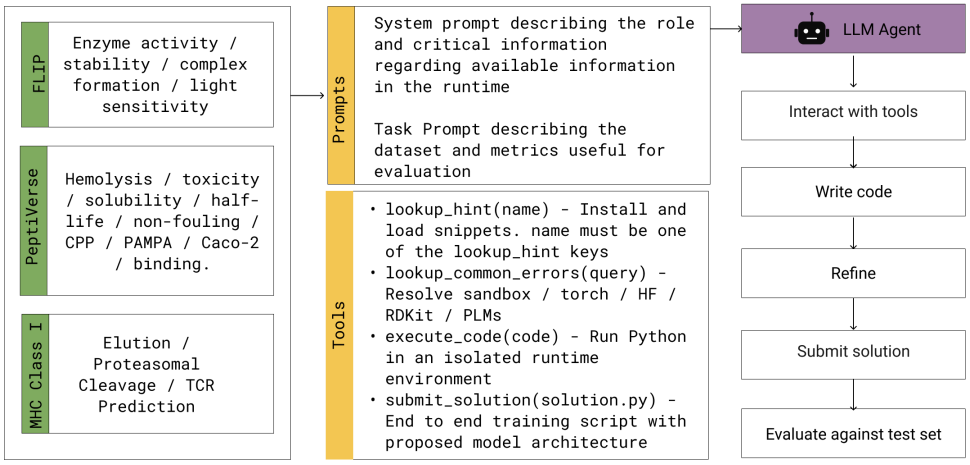

Figure 1: Single-agent predictor pipeline. A system prompt specifies the runtime and tools, and a task prompt describes the dataset and evaluation metric. The agent writes and executes model code across turns, then submits a solution for user-side evaluation. Final test scoring uses held-out labels.

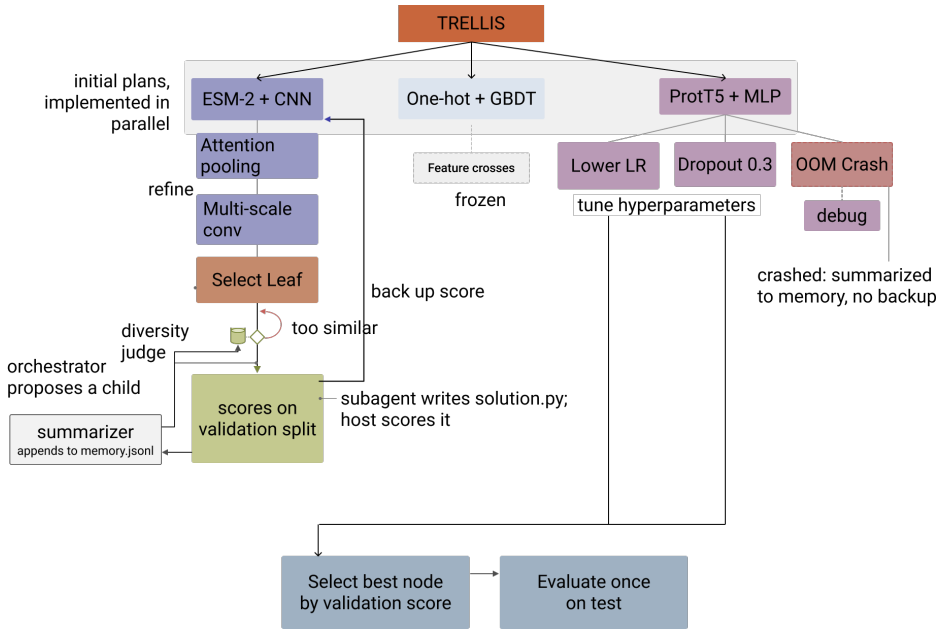

Figure 2: Tree search over predictor plans. Seed plans are proposed sequentially, approved by the diversity judge, and implemented in parallel. Each later iteration either proposes a new seed or selects a leaf by UCT, from which the orchestrator proposes a child plan using the leaf's lineage. The judge checks each proposal against the full memory of prior attempts, and the user scores the implementation on validation data. Every outcome, including crashes, is summarized into shared memory, and successful rewards are backed up to the root.

summarized, the memory records both what was tried and why it failed. Only the judge sees the full memory, and it rejects any proposal that duplicates an earlier attempt anywhere in the tree. The orchestrator sees only the summaries of the selected leaf and its parent, and the subagent sees only the approved plan and the training data in its runtime. The orchestrator thus builds on a branch's history while the judge enforces novelty across the whole search.

The search runs in two phases under a budget of $B = 16$ nodes. First, the orchestrator proposes $n_{\text{seed}} = 4$ seed plans one at a time. The judge checks each against the seeds already approved, and the approved seeds are implemented in parallel. Seeds are children of a virtual root. In the second phase, which runs sequentially, each iteration proposes a new seed with probability $p_{\text{seed}} = 0.25$, or always if no seed has succeeded. Otherwise it expands an existing branch. Selection descends from the root through successful, unpruned nodes, choosing at each level the child that maximizes

$$\begin{aligned} \mathrm{UCT}(n) = Q(n) + c\sqrt{\frac{\ln N_{\text{parent}}}{N_n}} - \lambda_{\text{fam}}\sqrt{\frac{n_{\text{fam}}}{N}} \\ - \lambda_{\text{sib}}\left(\sqrt{\frac{n_{\text{sib}}}{N_{\text{sib}}}} + \sqrt{\frac{n_{\text{crowd}}}{n_{\text{crowd}}+1}}\right). \end{aligned} \tag{1}$$

Here $Q(n)$ is the node's mean reward, N_n and N_{parent} count visits to the node and its parent, and unvisited nodes score $+\infty$. The three penalties discourage concentration. The first scales with n_{fam}, the number of expansions in the node's architecture family (including crashes) among all N expansions. The second scales with n_{sib}, the number of same-family siblings among its N_{sib} siblings. The third scales with n_{crowd}, the node's number of successful children. We use $c = 0.3$, $\lambda_{\text{fam}} = 0.45$, and $\lambda_{\text{sib}} = 0.25$.

Under the selected leaf, the orchestrator must shift to a new architecture with probability $p_{\text{shift}} = 0.3$. Otherwise it may refine the plan, debug it, or shift. The orchestrator proposes until the judge approves, and the subagent implements the approved plan. The user fits `solution.py` on the full training set, and its validation score is the reward r. After a successful evaluation, each node on the path to the root increments its visit count and updates $Q(n) \leftarrow Q(n) + (r - Q(n))/N_n$. Crashed evaluations are summarized but not backed up. The search ends when the tree reaches B nodes, or after 32 consecutive iterations that add no node. The successful node with the highest validation score is then evaluated once on the test set. The test set is never shown to any agent or used in any search decision. Algorithm 1 gives the full procedure.

3.5 Sequential MHC class I elution updates

The fixed benchmark experiments use prepared datasets. To examine maintenance as new assay results become available, we also evaluated sequential elution-predictor updates using IEDB, which supports programmatic queries (Vita et al., 2025). We used January 29, 2025 as the reference cutoff since that was the publication date of the source of our training data. Eligible new records had a reference year of at least 2025 and a peptide absent from the earlier curated splits. Peptides had 8 to 14 residues and an allele mapping to a 180 residue IMGT $\alpha 1/\alpha 2$ sequence; records whose allele could not be mapped were dropped. For release year 2026, training included the initial data and eligible records from earlier years. The records for year 2026 were divided evenly into validation and test partitions, stratified by allele and label. Each example is a peptide and allele pair with a binary presentation label, and the inputs are the peptide, the allele sequence, and the two flanking regions.

The predictor consists of a nested agentic loop. An outer agent adjusts allowed settings of the inner agentic loop, including training subset sizes, hints, epoch size, and per run time budgets. It evaluates a small number of rollouts and reads the resulting scores before revising the settings. Dataset splits and metrics remain fixed and are withheld from the outer agent too. The inner agent then chooses a new sequence representation and model architecture or updates the previous trained predictor using the newly available training data. The outer agent must submit a predictor that the main runtime can score. This experiment evaluates repeated predictor updates under a fixed protocol.

Algorithm 1 Tree search over predictor plans

Require: training and validation data; node budget B; seed count n_{seed}; probabilities $p_{\text{seed}}, p_{\text{shift}}$; UCT coefficients

Initialize a tree with a virtual root and an empty memory $\mathcal{M}$
for $i = 1, \ldots, n_{\text{seed}}$ **do**
    Orchestrator proposes seed plans until the judge approves one against earlier seeds
end for
for all approved seeds **in parallel do**
    Implement, score on validation, and append the summary to $\mathcal{M}$
end for
Back up successful rewards to the root
while the tree has fewer than B nodes and the search has not stalled **do**
    if no seed has succeeded or a new-seed action is sampled with probability p_{seed} **then**
        Orchestrator proposes a new seed plan
    else
        Select a leaf ℓ by descending with Eq. equation 1
        With probability p_{shift}, require an architecture shift; otherwise allow refine, debug, or shift
        Orchestrator proposes a child plan from the summaries of ℓ and its parent
    end if
    Repeat the proposal until the judge, given all of $\mathcal{M}$, approves it
    Subagent implements the plan; user fits on training data and scores on validation
    Summarizer appends the outcome to $\mathcal{M}$; back up the reward if execution succeeded
end while
Evaluate the successful node with the highest validation score once on test data

4 RESULTS

We evaluated held-out performance across FLIP, Peptiverse, and MHC class I pathway tasks, then examined tree search on selected baseline shortfalls. The final experiment concerns sequential data updates and reports validation performance for the selected predictor at each release. Published reference values include different evaluation protocols, so comparisons with these values are descriptive.

4.1 SINGLE-AGENT PERFORMANCE ACROSS BENCHMARKS

The baseline experiments were designed for three replicates per task and coding model. Reported ranges include runs with scorable test predictions, including additional retries, and exclude runs that exhausted their budget without a valid submission (Table 1). Qwen returned at least one scored result on 20 tasks. Its best replicate exceeded the listed reference on five of these tasks, comprising Hydro, RhoMax, Half-life AA, Nonfouling AA, and Solubility SMILES. GLM-5.3 additionally exceeded the references for Elution, Solubility AA, and Toxicity, bringing the total to eight distinct task-split pairs across the two coding models.

Performance varied substantially across runs. Qwen scores ranged from -0.084 to 0.696 on RhoMax, 0.474 to 0.786 on Half-life AA, and 0.169 to 0.384 on Hydro. Several tasks had only one scorable submission or none. The best-score summaries therefore describe achieved predictive performance conditional on a valid submission, and do not measure the probability of completing a run successfully. Both coding models exceeded the listed references for Hydro, Half-life AA, and Nonfouling AA, and both remained below the Caco-2 and PAMPA references.

Shortfalls were concentrated in FLIP extrapolation tasks and several SMILES peptide tasks. On Amylase, the best GLM-5.3 score was 0.614, compared with the reference 0.617 and Qwen's 0.462. For MHC class I elution, GLM-5.3 reached AP 0.957 against the listed MUNIS reference of 0.952, and Qwen reached 0.924. These differences motivated examination of the generated representations and architectures, with examples and prompts included in Appendix A.

Table 1: Baseline test performance and published reference scores. Each cell reports the best score, with the minimum scorable score in brackets. The maximum equals the best score. Three replicates were intended per task and model, with scorable retries included. Equal endpoints can represent one successful submission. Ref. denotes the source-paper score, including validation or cross-validation scores for Peptiverse. Bold marks values above the listed reference, and a dash denotes no scorable result. AA denotes amino-acid input and CPP cell-penetrating peptides.

					3.8-27B		GLM-5.3	
Bench	Task	Split	Metric	Paper	Best $[\min, \max]$	n	Best $[\min, \max]$	n
FLIP	Amylase	close→far	ρ	0.617	0.462 [0.462, 0.462]	1/3	0.614 [0.614, 0.614]	1/3
	Hydro	low→high	ρ	0.308	**0.384** [0.169, 0.384]	2/3	**0.392** [−0.050, 0.392]	2/3
	IRED	two→many	ρ	0.211	0.208 [0.173, 0.208]	2/3	—	0/3
	NucB	two→many	ρ	0.723	0.424 [0.394, 0.424]	2/3	—	0/3
	PDZ3	single→double	ρ	0.512	0.494 [0.279, 0.494]	2/3	0.537 [0.526, 0.537]	2/3
	RhoMax	by wild type	ρ	0.379	**0.696** [−0.084, 0.696]	2/3	0.159 [0.159, 0.159]	1/3
Peptiverse	Half-life	AA	ρ	0.582	**0.786** [0.474, 0.786]	2/3	**0.692** [0.239, 0.692]	2/3
		SMILES	ρ	0.542	0.329 [0.258, 0.329]	2/3	0.501 [0.416, 0.501]	2/3
	Hemolysis	AA	F1	0.596	0.525 [0.486, 0.525]	2/3	0.571 [0.558, 0.571]	2/3
		SMILES	F1	0.530	0.440 [0.363, 0.440]	2/3	—	0/3
	Nonfouling	AA	F1	0.736	**0.739** [0.728, 0.739]	2/3	**0.742** [0.742, 0.742]	1/3
		SMILES	F1	0.735	0.624 [0.624, 0.624]	1/3	—	0/3
	CPP	AA	F1	0.929	0.903 [0.886, 0.903]	2/3	0.897 [0.897, 0.897]	1/3
		SMILES	F1	0.909	0.903 [0.890, 0.903]	2/3	—	0/3
	Solubility	AA	F1	0.749	0.688 [0.688, 0.688]	1/3	**0.763** [0.763, 0.763]	1/3
		SMILES	F1	0.668	**0.732** [0.732, 0.732]	1/3	—	0/3
	Caco-2	SMILES	ρ	0.799	0.666 [0.491, 0.666]	2/3	0.674 [0.658, 0.674]	2/3
	PAMPA	SMILES	ρ	0.688	0.665 [0.632, 0.665]	2/3	0.667 [0.667, 0.667]	1/3
	Toxicity	SMILES	F1	0.853	0.754 [0.754, 0.754]	1/3	**0.865** [0.865, 0.865]	1/3
MHC-I	Elution	default	AP	0.952	0.924 [0.924, 0.924]	1/3	**0.957** [0.957, 0.957]	1/3
	Prot. cleavage	N-term	AUROC	0.806	0.791 [0.777, 0.791]	2/3	0.797 [0.797, 0.797]	1/3
		C-term	AUROC	0.898	0.878 [0.865, 0.878]	3/3	0.880 [0.875, 0.880]	3/3
	TCR binding	cleaned	AUROC	—	0.846 [0.841, 0.846]	2/3	0.893 [0.888, 0.893]	2/3

4.2 Tree search on selected baseline shortfalls

To test whether broader exploration improved prediction, we ran tree search on eight Qwen task-split pairs whose best baseline score was below the listed reference. These comprised Amylase, IRED, NucB, and PDZ3, together with the SMILES protocols for Half-life, Hemolysis, Nonfouling, and Caco-2. This subset was selected after examining baseline performance. Within each search, model ranking used validation scores, and final reporting used the selected model's test score.

Tree search improved seven of the eight selected tasks (Table 2). Spearman correlation increased from 0.462 to 0.525 on Amylase, 0.424 to 0.490 on NucB, and 0.494 to 0.533 on PDZ3. IRED decreased from 0.208 to 0.190. All four peptide tasks improved over their Qwen baselines, with gains from 0.054 to 0.137 in the reported primary metrics. Most resulting scores remained below the literature references, indicating that further gains were needed on these tasks.

The selected plans used several representation-architecture combinations. Amylase used ESM-2 sequence-likelihood features, and NucB paired frozen ESMC residue embeddings with a one-dimensional convolutional head. Peptide intermediate solutions also included ChemBERTa embeddings and Morgan/ECFP fingerprints with learned prediction heads. Each search took approximately two to four hours. Across these runs, broader representation search coincided with improved scores on most selected tasks. A separate ablation is needed to determine the contribution of the diversity judge.

4.3 Sequential elution-predictor updates

We next evaluated whether the development loop could incorporate additional IEDB records across release years. Starting from the MUNIS data, the sequence proceeded through IEDB 2025 and IEDB 2026, with eligible earlier-release records added to training before the next phase. All selected predictors in this experiment were generated with Qwen.

Task	Baseline	TRELLIS	Selected plan
FLIP, Spearman correlation			
Amylase	0.462	**0.525**	Frozen ESM-2 650M delta vs wT at mutated positions, multi pooling and 3 layer MLP
IRED	0.208	0.190	Frozen ESM2-t30 150M mean pooled per residue and MLP regressor with Huber loss
NucB	0.424	**0.490**	Frozen ESMC-300M with a 1D-CNN
PDZ3	0.494	**0.533**	Frozen ESM-2 pair embeddings per position mean pooled embeddings + lightGBM and engineered epistasis features
Peptiverse, SMILES input			
Half-life	0.329	**0.413**	RDKit ECFP4, physiochemical features and stacked generalization: 5-fold ridge regression
Hemolysis	0.440	**0.498**	Morgan/ECFP and physicochemical features with LightGBM with class weight balancing
Nonfouling	0.624	**0.707**	ECFP4, physiochemical features and LightGBM class weighted ensemble with oof threshold tuning
Caco-2	0.666	**0.720**	Morgan/ECFP and physicochemical features with a GBDT

Table 2: Tree search on eight selected tasks with baseline scores below the source-paper reference. Baseline is the best Qwen replicate from Table 1. TRELLIS is the test score of the model selected by validation. Bold marks improvements over the baseline. ECFP denotes extended-connectivity fingerprints, CNN a convolutional neural network, MLP a multilayer perceptron, and GBDT gradient-boosted decision trees.

Release	Holdout	Alleles	AP	Selected plan
MUNIS	6500	117	0.894	LightGBM with peptide/flank encodings and allele identity
IEDB 2025	849	49	0.835	LightGBM + ESM-2 35M embeddings

Table 3: Sequential elution-predictor updates using a January 29, 2025 reference cutoff. AP denotes the reported validation average precision of the selected model, pooled over alleles.

The reported validation AP for the 2026 model, based on training data up until the 2026 release, was 0.835. MUNIS contains over 3 million training data points, and we sampled only 6500 out of a larger test set to enable faster test score computation during the loop. The selected predictors used LightGBM with peptide and flanking-sequence one-hot encodings, followed by ESM-2 35M features in the later phases.

Between turns, the outer agent adjusted allowed settings, including training-subset sizes, per-node time limits, and hints for allele-conditioned LightGBM predictors with ESM features. Independent test results and a fixed-model comparison on the same release cohorts are needed to quantify the predictive benefit of these updates. Even though the AP score looks promising, the evaluation needs to be expanded to include true negative and true positive rates along with median precision over alleles, as well as performance among rarer alleles. This benchmark was intended as a test of the agents ability to ingest new data and retrain the predictor and should not be considered state of the art.

5 DISCUSSION

We developed TRELLIS to automate the construction and revision of biological sequence-property predictors. Agents produced predictors that exceeded published reference scores on six tasks with no human intervention in the development loop, and tree search improved seven of the eight tasks where the baseline fell short, by exploring additional representations and architectures. Extending the workflow to IEDB release-year holdouts let the predictor be updated as new records accumulate. Submission success and score variance across replicates mean that this does not happen on every run, so the relevant cost is not a single run but the number of attempts needed to obtain a usable

predictor. Tree search compounds this, since each node requires model fitting and inference, so additional benchmarks under fixed compute budgets are needed to establish whether the gains in performance justify the cost of tree-based search.

Several aspects of the evaluation were limited by computational cost. Search tasks were selected retrospectively on the basis of baseline performance across three replicates, so the reported gains are conditioned on that selection. We did not ablate the diversity judge, since a controlled comparison requires re-running the full search on each task, so its contribution relative to undirected expansion is unclear. Published reference values also follow different validation and test splits; we matched data partitions across most of the tasks but not uniformly across the MHC class I tasks.

Along with evaluating under various conditions mentioned above, next step is widening what is covered during the tree search. We searched over representations and architectures but directly optimizing the agentic loop itself is a natural next step. Diversity memory is also logged with a single task type, and sharing it across datasets may allow modeling decisions to transfer between related properties. Apart from IEDB, this work used pre-curated datasets, and a genuinely end-to-end workflow would need web search and paper parsing so the agent could assemble its own training data rather than receive it. Despite these shortcomings, as agents take on more of the scientific process, self-updating predictors are a useful component, both for scientists who need up to date predictors and for use in design tasks.

Use of Large Language Models

We used AI assistants in three ways during this work. First, for writing, we often dictated rough, informal drafts using Wispr Flow and then used Claude to turn them into prose. We revised the resulting text over several rounds to ensure it was accurate and said what we intended it to say. We also used assistants to assist with LaTex formatting. Second, we used AI coding agents (Cursor and Claude Code) to help write and debug the code for our experiments. Third, we occasionally used Claude Code to help us work through specific points where we were stuck. In these cases, we evaluated its suggestions ourselves and did not rely on LLM judgments for any research decision. For method figures, initial canvas drafts in one of the cases here was made and connected to figma and the authors then edited and expanded it. All research ideas were conceived by the authors, and we found relevant literature primarily through our own web searches. The authors take full responsibility for the content of this paper.

Reproducibility Statement

Section 3 describes the evaluation procedure, isolated execution environment, search score, model-selection procedure, and sequential-update protocol. Algorithm 1 specifies the search workflow, and Appendix A contains representative generated solutions, diversity-judge outputs, and prompt templates. The accompanying source includes the displayed scripts and prompts, and the anonymous code link is `https://anonymous.4open.science/r/trellis-B12C`.

Ethics Statement

This study concerns computational prediction using existing protein, peptide, and immune-assay datasets. Assay and allele representation can affect prediction quality, so aggregate benchmark scores should be interpreted within the reported data coverage. These predictors require experimental validation before biological or clinical use. Automated code generation also requires controlled execution and independent checking of outputs. Reuse of the underlying datasets and pretrained models remains subject to their licenses and access terms, and downstream biological applications require appropriate oversight.

References

Lasse M Blaabjerg, Maher M Kassem, Lydia L Good, Nicolas Jonsson, Matteo Cagiada, Kristoffer E Johansson, Wouter Boomsma, Amelie Stein, and Kresten Lindorff-Larsen. Rapid protein stability

prediction using deep learning representations. *eLife*, 12:e82593, may 2023. ISSN 2050-084X. doi: 10.7554/eLife.82593. URL https://doi.org/10.7554/eLife.82593.

Robson P Bonidia, Anderson P Avila Santos, Breno L S de Almeida, Peter F Stadler, Ulisses N da Rocha, Danilo S Sanches, and André C P L F de Carvalho. Bioautoml: automated feature engineering and metalearning to predict noncoding rnas in bacteria. *Briefings in Bioinformatics*, 23(4):bbac218, 07 2022. ISSN 1477-4054. doi: 10.1093/bib/bbac218. URL https://doi.org/10.1093/bib/bbac218.

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 Mądry. Mle-bench: Evaluating machine learning agents on machine learning engineering, 2025. URL https://arxiv.org/abs/2410.07095.

Yizhou Chi, Yizhang Lin, Sirui Hong, Duyi Pan, Yaying Fei, Guanghao Mei, Bangbang Liu, Tianqi Pang, Jacky Kwok, Ceyao Zhang, Bang Liu, and Chenglin Wu. Sela: Tree-search enhanced llm agents for automated machine learning, 2024. URL https://arxiv.org/abs/2410.17238.

Breno L. S. de Almeida, Robson P. Bonidia, Martin Bole, Anderson Avila-Santos, Peter F. Stadler, Ulisses N. da Rocha, and André C. P. L. F. de Carvalho. Bioautoml-fast: an automated machine-learning platform for reusable and benchmarked biological sequence models. *bioRxiv*, 2026. doi: 10.64898/2026.04.18.719383. URL https://www.biorxiv.org/content/early/2026/04/22/2026.04.18.719383.

Kieran Didi, Sarah Alamdari, Alex X. Lu, Bruce Wittmann, Kadina E. Johnston, Ava P. Amini, Ali Madani, Maya Czeneszew, Christian Dallago, and Kevin K. Yang. Flip2: Expanding protein fitness landscape benchmarks for real-world machine learning applications. *bioRxiv*, 2026. doi: 10.64898/2026.02.23.707496. URL https://www.biorxiv.org/content/early/2026/08/07/2026.02.23.707496.

Felix Drost, Anna Chernysheva, Mahmoud Albahah, Katharina Kocher, Kilian Schober, and Benjamin Schubert. Benchmarking of t cell receptor-epitope predictors with epytope-tcr. *Cell Genomics*, 5(8), 2025.

Patrick Emami, Aidan Perreault, Jeffrey Law, David Biagioni, and Peter St. John. Plug amp; play directed evolution of proteins with gradient-based discrete mcmc. *Machine Learning: Science and Technology*, 4(2):025014, April 2023. ISSN 2632-2153. doi: 10.1088/2632-2153/accacd. URL http://dx.doi.org/10.1088/2632-2153/accacd.

Ali Essam Ghareeb, Benjamin Chang, Ludovico Mitchener, Angela Yiu, Caralyn J Szostkiewicz, Dmytro Shved, Gavin J Gyimesi, Jon M Laurent, Samantha M Wright, Muhammed T Razzak, et al. A multi-agent system for automating scientific discovery. *Nature*, pp. 1–3, 2026.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Petar Sirkovic, Artiom Myaskovsky, Grzegorz Glowaty, Felix Weissenberger, Alessio Orlandi, Dan Popovici, et al. Accelerating scientific discovery with co-scientist. *Nature*, pp. 1–3, 2026.

Filippo Grazioli, Anja Mösch, Pierre Machart, Kai Li, Israa Alqassem, Timothy J. O'Donnell, and Martin Renqiang Min. On tcr binding predictors failing to generalize to unseen peptides. *Frontiers in Immunology*, Volume 13 - 2022, 2022. ISSN 1664-3224. doi: 10.3389/fimmu.2022.1014256. URL https://www.frontiersin.org/journals/immunology/articles/10.3389/fimmu.2022.1014256.

Max Hebditch, M Alejandro Carballo-Amador, Spyros Charonis, Robin Curtis, and Jim Warwicker. Protein–sol: a web tool for predicting protein solubility from sequence. *Bioinformatics*, 33(19):3098–3100, 10 2017. ISSN 1367-4803. doi: 10.1093/bioinformatics/btx345. URL https://doi.org/10.1093/bioinformatics/btx345.

Lu Hong and Tanja Kortemme. An integrative approach to protein sequence design through multiobjective optimization. *PLOS Computational Biology*, 20(7):1–37, 07 2024. doi: 10.1371/journal.pcbi.1011953. URL https://doi.org/10.1371/journal.pcbi.1011953.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: Evaluating language agents on machine learning experimentation, 2024. URL https://arxiv.org/abs/2310.03302.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. Aide: Ai-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*, 2025.

Loka Li, Duzhen Zhang, Xingbo Du, Leonard Song, Zixiao Wang, Assanali Aukenov, Noel Thomas, Shakhnazar Sailaukan, Yonghan Yang, Feilong Chen, Jiahua Dong, Kun Zhang, Bin Zhang, and Le Song. Bioxarena: Benchmarking llm agents on multi-modal biomedical machine learning tasks, 2026. URL https://arxiv.org/abs/2605.15766.

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. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637):1123–1130, 2023. doi: 10.1126/science.ade2574. URL https://www.science.org/doi/abs/10.1126/science.ade2574.

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

Henry E. Miller, Matthew Greenig, Benjamin Tenmann, and Bo Wang. Bioml-bench: Evaluation of ai agents for end-to-end biomedical ml. *bioRxiv*, 2025. doi: 10.1101/2025.09.01.673319. URL https://www.biorxiv.org/content/early/2025/09/28/2025.09.01.673319.

Ludovico Mitchener, Angela Yiu, Benjamin Chang, Mathieu Bourdenx, Tyler Nadolski, Arvis Sulovari, Eric C. Landsness, Daniel L. Barabasi, Siddharth Narayanan, Nicky Evans, Shriya Reddy, Martha Foiani, Aizad Kamal, Leah P. Shriver, Fang Cao, Asmamaw T. Wassie, Jon M. Laurent, Edwin Melville-Green, Mayk Caldas, Albert Bou, Kaleigh F. Roberts, Sladjana Zagorac, Timothy C. Orr, Miranda E. Orr, Kevin J. Zwezdaryk, Ali E. Ghareeb, Laurie McCoy, Bruna Gomes, Euan A. Ashley, Karen E. Duff, Tonio Buonassisi, Tom Rainforth, Randall J. Bateman, Michael Skarlinski, Samuel G. Rodriques, Michaela M. Hinks, and Andrew D. White. Kosmos: An ai scientist for autonomous discovery, 2025. URL https://arxiv.org/abs/2511.02824.

Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, Jinwoo Shin, Sercan Ö. Arık, and Tomas Pfister. Mle-star: Machine learning engineering agent via search and targeted refinement, 2025. URL https://arxiv.org/abs/2506.15692.

Morten Nielsen, Claus Lundegaard, Ole Lund, and Can Keşmir. The role of the proteasome in generating cytotoxic t-cell epitopes: insights obtained from improved predictions of proteasomal cleavage. *Immunogenetics*, 57(1):33–41, 2005.

Jonas Birkelund Nilsson, Jason Greenbaum, Bjoern Peters, and Morten Nielsen. Netmhcpan-4.2: improved prediction of cd8+ epitopes by use of transfer learning and structural features. *Frontiers in Immunology*, 16:1616113, 2025.

Timothy J. O'Donnell, Alex Rubinsteyn, and Uri Laserson. Mhcflurry 2.0: Improved pan-allele prediction of mhc class i-presented peptides by incorporating antigen processing. *Cell Systems*, 11 (1):42–48.e7, 2020. doi: 10.1016/j.cels.2020.06.010. URL https://doi.org/10.1016/j.cels.2020.06.010. doi: 10.1016/j.cels.2020.06.010.

Kyle Swanson, Wesley Wu, Nash L Bulaong, John E Pak, and James Zou. The virtual lab of ai agents designs new sars-cov-2 nanobodies. *Nature*, 646(8085):716–723, 2025.

Sophia Tang, Yinuo Zhang, and Pranam Chatterjee. Peptune: De novo generation of therapeutic peptides with multi-objective-guided discrete diffusion, 2025. URL https://arxiv.org/abs/2412.17780.

Randi Vita, Nina Blazeska, Daniel Marrama, IEDB Curation Team Members, Sebastian Duesing, Jason Bennett, Jason Greenbaum, Marcus De Almeida Mendes, Jarjapu Mahita, Daniel K Wheeler, Jason R Cantrell, James A Overton, Darren A Natale, Alessandro Sette, and Bjoern Peters. The immune epitope database (iedb): 2024 update. *Nucleic Acids Research*, 53(D1):D436–D443, 01 2025. ISSN 1362-4962. doi: 10.1093/nar/gkae1092. URL https://doi.org/10.1093/nar/gkae1092.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025.

Yinuo Zhang, Sophia Tang, Tong Chen, Elizabeth Mahood, Sophia Vincoff, and Pranam Chatterjee. Peptiverse: A unified platform for therapeutic peptide property prediction. *Nature Communications*, 2026.

A ADDITIONAL ANALYSES AND PROMPTS

All code excerpts and prompt templates are collected in Section A.5 and referenced from the analyses below.

A.1 FLIP

FLIP tasks evaluate protein fitness under distribution shifts in mutation count, sequence distance, fitness, and wild-type identity. Qwen's best baseline score exceeded the listed references for Hydro and RhoMax. Tree search improved Amylase, NucB, and PDZ3 by approximately 0.03-0.10 Spearman correlation, but reduced IRED performance.

The selected Hydro baseline concatenated ESM-2 mean-pooled embeddings with hand-built classical features, namely mutation count and hydrophobicity deltas against the wild type, and used ridge regression (Listing 1). The RhoMax solution used concatenated mean and max pooling with a stronger ridge penalty (Listing 2). Among the tree-search solutions, the selected Amylase solution used ESM-2 sequence-likelihood features (Listing 7), and the selected NucB solution paired frozen ESMC embeddings with a residual one-dimensional convolutional head (Listing 8).

On FLIP, the diversity LLM judge rejected repeated representation-head combinations, including mean-pooled ESM-2, ESMC, or ProtBERT embeddings with MLP or ridge heads. Similar LoRA plans with changed Hub identifiers also prompted rejections. Rejection traffic varied widely across tasks: PDZ3 produced 118 rejections and 28 proposals that exhausted their retries without acceptance, whereas IRED produced none. Listing 10 gives representative rejection text. These counts describe search behavior, without isolating the judge's effect on predictive performance.

A.2 PEPTIVERSE

Peptiverse includes amino-acid and SMILES inputs. The best Qwen baseline scores exceeded the listed references for Half-life AA, Nonfouling AA, and Solubility SMILES. The selected solutions combined pretrained embeddings with linear prediction heads: frozen ESMC-300M with ridge regression on a $\log_{10}$ target for Half-life (Listing 3), concatenated ESM-2 650M and 150M pooling with L2 logistic regression for Nonfouling (Listing 4), and concatenated ESM-2 and PeptideCLM-2 features with logistic regression for Solubility (Listing 5).

On the four SMILES tasks selected for tree search, scores increased by approximately 0.05-0.14 over the baseline, with different feature choices across tasks. The selected Hemolysis solution combined Morgan/ECFP fingerprints and physicochemical features with LightGBM (Listing 9). Judge rejections on the SMILES runs followed the same pattern as on FLIP, and Listing 11 gives representative text.

A.3 MHC CLASS I

Elution prediction uses peptide and HLA information, with optional flanking sequences, and is evaluated by AP. The selected GLM-5.3 baseline combined dual-stream ESM-2 features, cross-attention between peptide and parent-protein context, allele embeddings, and LoRA fine-tuning. Its test AP exceeded the listed MUNIS reference. Qwen's elution score and GLM-5.3's cleavage AUROC were below their corresponding references (Table **??**). We did not include MHC class I tasks in the eight-task tree-search comparison. Sequential-update results appear in Section 4.3.

The elution architecture used cross-attention blocks over peptide and context features, followed by staged LoRA fine-tuning; Listing 6 gives the core model setup.

A.4 PROMPT STRUCTURE

The supplied baseline and tree-search prompt templates share a common structure across benchmarks, with task-specific dataset descriptions, metrics, and search hints. The sources are `baseline/prompt.py`, `mcts/prompts.py`, and `mcts/diversity.py`. The accompanying `solutions/` directory contains the supplied full scripts for selected baseline outcomes. The templates themselves are reproduced in Section A.5.4: the baseline system prompt and tool

contract (Listings 12 and 13), the orchestrator and subagent prompts (Listings 14 and 15), and the diversity judge's two system prompts and tool schema (Listings 16–18).

A.5 Code and prompt listings

A.5.1 Selected baseline solutions

Listing 1. Hydro low→high baseline solution: classical features, ESM-2, and ridge regression (`solutions/good/flip_hydro_low_to_high_qwen.py`).

```
def cls_feats(seq):
    w = WT[len(seq)]
    d = [(a, b) for a, b in zip(seq, w) if a != b]
    return np.array([
        len(d),
        sum(a in 'FILMV' for a, b in d),
        sum(KD[a] - KD[b] for a, b in d),
        sum(KD[a] for a in seq),
        float(len(seq) == 63),
        float(len(seq) == 65),
    ])


def embed(model, tok, seqs, bs=64):
    out = []
    with torch.no_grad():
        for i in range(0, len(seqs), bs):
            b = seqs[i:i + bs]
            enc = tok(b, return_tensors='pt', padding=True).to('cuda')
            h = model(**enc).last_hidden_state
            mask = enc['attention_mask'].unsqueeze(-1)
            mean = (h * mask).sum(1) / mask.sum(1)
            out.append(mean.cpu().numpy())
    return np.concatenate(out)


def main():
    tr = pd.read_csv('/data/train.csv')
    va = pd.read_csv('/data/validation.csv')
    y = tr.target.values

    tok = EsmTokenizer.from_pretrained('facebook/esm2_t33_650M_UR50D')
    model = EsmModel.from_pretrained('facebook/esm2_t33_650M_UR50D').cuda().eval()

    Etr = embed(model, tok, list(tr.sequence))
    Eva = embed(model, tok, list(va.sequence))

    Ctr = np.array([cls_feats(s) for s in tr.sequence])
    Cva = np.array([cls_feats(s) for s in va.sequence])

    Xtr = np.hstack([Etr, Ctr])
    Xva = np.hstack([Eva, Cva])

    m = Ridge(alpha=0.1).fit(Xtr, y)
```

Listing 2. RhoMax by-wild-type baseline solution: ESM-2 mean and max pooling with ridge regression (`solutions/good/flip_rhomax_by_wild_type_qwen.py`).

```
# ESM2 650M embeddings (mean + max pool)
tok = EsmTokenizer.from_pretrained('facebook/esm2_t33_650M_UR50D')
model = EsmModel.from_pretrained('facebook/esm2_t33_650M_UR50D').cuda().eval()

def embed(seqs, bs=8):
    means, maxs = [], []
    with torch.no_grad():
        for i in range(0, len(seqs), bs):
            batch = seqs[i:i+bs]
            enc = tok(batch, padding=True, truncation=True, max_length=340,
     return_tensors='pt').to('cuda')
            out = model(**enc)
            h = out.last_hidden_state
            mask = enc['attention_mask'].unsqueeze(-1).float()
            pooled_mean = (h * mask).sum(1) / mask.sum(1)
```

```
            h_masked = h.masked_fill(~enc['attention_mask'].unsqueeze(-1).bool(), -1e9)
            pooled_max = h_masked.max(1).values
            means.append(pooled_mean.cpu().numpy())
            maxs.append(pooled_max.cpu().numpy())
    return np.vstack(means), np.vstack(maxs)

print("Embedding train...")
tr_mean, tr_max = embed(seqs_tr)
print("Embedding val...")
va_mean, va_max = embed(seqs_va)

Xtr = np.hstack([tr_mean, tr_max])
Xva = np.hstack([va_mean, va_max])

sc = StandardScaler().fit(Xtr)
Xtr_s = sc.transform(Xtr)
Xva_s = sc.transform(Xva)

# Ridge with alpha=50 (best group-CV spearman ~0.57)
m = Ridge(alpha=50.0).fit(Xtr_s, y)
preds = m.predict(Xva_s)

print("PREDS_JSON=" + json.dumps([float(p) for p in preds]))
```

Listing 3. Half-life AA baseline solution: frozen ESMC-300M with ridge regression on a $\log_{10}$ target (`solutions/good/pep_halflife_aa_qwen.py`).

```
tr = pd.read_csv('/data/train.csv')
va = pd.read_csv('/data/validation.csv')

name = 'biohub/ESMC-300M'
tok = AutoTokenizer.from_pretrained(name, trust_remote_code=True)
model = AutoModel.from_pretrained(name).cuda().eval()

def embed(seqs, bs=16):
    embs = []
    with torch.no_grad():
        for i in range(0, len(seqs), bs):
            batch = seqs[i:i+bs]
            enc = tok(batch, return_tensors='pt', padding=True, truncation=True,
     max_length=512).to('cuda')
            out = model(**enc).last_hidden_state
            mask = enc['attention_mask'].unsqueeze(-1).float()
            mean = (out * mask).sum(1) / mask.sum(1)
            embs.append(mean.cpu().numpy())
    return np.vstack(embs)

Xtr = embed(list(tr.sequence))
Xva = embed(list(va.sequence))

yt = np.log10(tr.target.values)
m = Ridge(alpha=1.0)
m.fit(Xtr, yt)
preds = m.predict(Xva)

print("PREDS_JSON=" + json.dumps([float(p) for p in preds]))
```

Listing 4. Nonfouling AA baseline solution: concatenated ESM-2 650M and 150M mean and max pooling with L2 logistic regression (`solutions/good/pep_nonfouling_aa_qwen.py`).

```
def embed_all(model_name, cache_path, batch_size=32):
    if os.path.exists(cache_path):
        return np.load(cache_path)
    tok = EsmTokenizer.from_pretrained(model_name)
    model = EsmModel.from_pretrained(model_name).cuda().eval()
    D = model.config.hidden_size
    emb = np.zeros((len(seqs), 2 * D), dtype=np.float32)
    with torch.no_grad():
        for i in range(0, len(seqs), batch_size):
            b = seqs[i:i + batch_size]
            batched = tok(b, padding=True, truncation=True, max_length=512,
                          return_tensors='pt').to('cuda')
            h = model(**batched).last_hidden_state
            m = batched.attention_mask.unsqueeze(-1).float()
```

```
            mean = (h * m).sum(1) / m.sum(1)
            mx = torch.where(m.bool(), h, h.new_full((), float('-inf'))).max(1).values
            emb[i:i + len(b)] = torch.cat([mean, mx], 1).cpu().numpy()
    np.save(cache_path, emb)
    del model
    torch.cuda.empty_cache()
    return emb

e650 = embed_all('facebook/esm2_t33_650M_UR50D', '/tmp/esm2_650_emb.npy', batch_size=32)
e150 = embed_all('facebook/esm2_t30_150M_UR50D', '/tmp/esm2_150_emb.npy', batch_size=64)

X = np.hstack([e650[:n_tr], e150[:n_tr]])
y = tr.target.values
Xv = np.hstack([e650[n_tr:], e150[n_tr:]])

sc = StandardScaler().fit(X)
clf = LogisticRegression(max_iter=10000, C=0.001).fit(sc.transform(X), y)
probs = clf.predict_proba(sc.transform(Xv))[:, 1]
probs = np.clip(probs, 1e-6, 1 - 1e-6)

os.makedirs('/out', exist_ok=True)
np.save('/out/preds.npy', probs)
print('PREDS_JSON=' + json.dumps([float(p) for p in probs]))
```

Listing 5. Solubility SMILES baseline solution: concatenated ESM-2 150M, ESM-2 35M, and PeptideCLM-2 features with logistic regression (`solutions/good/pep_solubility_smiles_qwen.py`).

```
def embed(model_name, seqs, batch=32, max_length=512):
    tok = AutoTokenizer.from_pretrained(model_name)
    model = AutoModel.from_pretrained(model_name).cuda().eval()
    embs = []
    for i in range(0, len(seqs), batch):
        b = seqs[i:i+batch]
        enc = tok(list(b), return_tensors='pt', padding=True, truncation=True,
     max_length=max_length)
        ids = enc['input_ids'].cuda()
        mask = enc['attention_mask'].cuda()
        with torch.no_grad():
            out = model(input_ids=ids, attention_mask=mask)
        h = (out.last_hidden_state * mask.unsqueeze(-1)).sum(1) / mask.sum(1).unsqueeze(-1)
        embs.append(h.cpu().numpy())
    return np.vstack(embs)

def embed_pclm2(seqs, batch=64):
    name = 'aaronfeller/peptideclm-2-hybrid-small'
    tok = AutoTokenizer.from_pretrained(name, trust_remote_code=True)
    model = AutoModel.from_pretrained(name, trust_remote_code=True,
     use_safetensors=True).cuda().eval()
    embs = []
    for i in range(0, len(seqs), batch):
        b = seqs[i:i+batch]
        enc = tok(list(b), return_tensors='pt', padding=True)
        ids = enc['input_ids'].cuda()
        mask = enc['attention_mask'].cuda()
        with torch.no_grad():
            out = model(input_ids=ids, attention_mask=mask)
        h = (out.last_hidden_state * mask.unsqueeze(-1)).sum(1) / mask.sum(1).unsqueeze(-1)
        embs.append(h.cpu().numpy())
    return np.vstack(embs)

def main():
    tr = pd.read_csv('/data/train.csv')
    va = pd.read_csv('/data/validation.csv')
    y = tr['target'].values
    tr_seqs = tr['sequence'].tolist()
    va_seqs = va['sequence'].tolist()

    t0 = time.time()
    X1 = embed('facebook/esm2_t30_150M_UR50D', tr_seqs)
    print('esm2_150m train', X1.shape, time.time()-t0)
    X2 = embed('facebook/esm2_t12_35M_UR50D', tr_seqs)
    print('esm2_35m train', X2.shape, time.time()-t0)
    X3 = embed_pclm2(tr_seqs)
    print('pclm2 train', X3.shape, time.time()-t0)
    Xtr = np.hstack([X1, X2, X3])
```

```
    t0 = time.time()
    V1 = embed('facebook/esm2_t30_150M_UR50D', va_seqs)
    V2 = embed('facebook/esm2_t12_35M_UR50D', va_seqs)
    V3 = embed_pclm2(va_seqs)
    Xva = np.hstack([V1, V2, V3])
    print('val embeds', Xva.shape, time.time()-t0)

    sc = StandardScaler().fit(Xtr)
    m = LogisticRegression(max_iter=3000, C=0.1).fit(sc.transform(Xtr), y)
    p = m.predict_proba(sc.transform(Xva))[:, 1]
    print('preds', p.shape, 'mean', p.mean())

    os.makedirs('/out', exist_ok=True)
    np.save('/out/preds.npy', p)
    print('PREDS_JSON=' + json.dumps([float(x) for x in p]))
```

Listing 6. MHC-I elution baseline win under GLM-5.3: cross-attention model core (`solutions/good/mhc_elution_glm.py`, excerpt).

```
class SelfBlock(nn.Module):
    def __init__(self, d, heads, ff, drop):
        super().__init__()
        self.ln1 = nn.LayerNorm(d); self.attn = nn.MultiheadAttention(d, heads,
     dropout=drop, batch_first=True)
        self.ln2 = nn.LayerNorm(d); self.ff = nn.Sequential(nn.Linear(d, ff), nn.GELU(),
     nn.Dropout(drop), nn.Linear(ff, d), nn.Dropout(drop))
    def forward(self, x, pad):
        h = self.ln1(x)
        a, _ = self.attn(h, h, h, key_padding_mask=pad, need_weights=False)
        x = x + a
        return x + self.ff(self.ln2(x))

class CrossBlock(nn.Module):
    def __init__(self, d, heads, ff, drop):
        super().__init__()
        self.ln1 = nn.LayerNorm(d); self.attn = nn.MultiheadAttention(d, heads,
     dropout=drop, batch_first=True)
        self.ln2 = nn.LayerNorm(d); self.ff = nn.Sequential(nn.Linear(d, ff), nn.GELU(),
     nn.Dropout(drop), nn.Linear(ff, d), nn.Dropout(drop))
    def forward(self, x, mem):
        h = self.ln1(x)
        a, _ = self.attn(h, mem, mem, need_weights=False)
        return x + self.ff(self.ln2(x + a))

class Model(nn.Module):
    def __init__(self, d=512, heads=8, ff=2048, drop=0.1, n_hla=0):
        super().__init__()
        self.pep_proj = nn.Sequential(nn.Linear(1280, d), nn.LayerNorm(d))
        self.pure_proj = nn.Sequential(nn.Linear(1280, d), nn.LayerNorm(d))
        self.hla_proj = nn.Sequential(nn.Linear(1280, d), nn.LayerNorm(d))
        self.hla_param = nn.Parameter(torch.from_numpy(hla_emb_np).float().clone())
        self.pos = nn.Parameter(torch.randn(1, 15, d) * 0.02)
        self.aa_emb = nn.Embedding(21, 48); self.aa_proj = nn.Linear(48, d)
        self.self_blocks = nn.ModuleList([SelfBlock(d, heads, ff, drop) for _ in range(2)])
        self.cross_blocks = nn.ModuleList([CrossBlock(d, heads, ff, drop) for _ in range(2)])
        self.pool_score = nn.Linear(d, 1)
        self.flank_net = nn.Sequential(nn.LayerNorm(2560), nn.Linear(2560, 128), nn.GELU(),
     nn.Linear(128, 64))
        self.len_emb = nn.Embedding(8, 16)
        din = 2 * d + d + 64 + 16
        self.norm = nn.LayerNorm(din)
        self.out = nn.Sequential(nn.Linear(din, 512), nn.GELU(), nn.Dropout(drop),
     nn.Linear(512, 1))
    def forward(self, ctx, pure, aa, pep_len, start, T, hla_idx):
        s_ = start[:, None]; L_ = pep_len[:, None]
        AR15 = torch.arange(15, device=ctx.device)[None]
        AR5 = torch.arange(5, device=ctx.device)[None]
        idx = (s_ + AR15).clamp(max=ctx.shape[1] - 1)
        valid = AR15 < L_
        span = ctx.gather(1, idx[..., None].expand(-1, -1, ctx.shape[-1])) * valid[...,
     None].to(ctx.dtype)
        p = self.pep_proj(span) + self.pure_proj(pure) + self.pos +
     self.aa_proj(self.aa_emb(aa))
        pad = ~valid
        for b in self.self_blocks: p = b(p, pad)
        h = self.hla_proj(self.hla_param[hla_idx])
        for b in self.cross_blocks: p = b(p, h)
```

```
        sc = self.pool_score(p).squeeze(-1).masked_fill(pad, -1e4)
        w = sc.softmax(-1)
        v = (p * w[..., None]).sum(1)
        vmax = p.masked_fill(pad[..., None], -1e4).max(1).values
        nidx = AR5.clamp(max=ctx.shape[1]-1)
        nv = (AR5 < s_).to(ctx.dtype)
        nemb = (ctx.gather(1, nidx[..., None].expand(-1, -1, ctx.shape[-1])) * nv[...,
    None]).sum(1) / nv.sum(1, keepdim=True).clamp(min=1)
        cidx = (s_ + L_ + AR5).clamp(max=ctx.shape[1]-1)
        cv = (AR5 < (T[:, None] - s_ - L_)).to(ctx.dtype)
        cemb = (ctx.gather(1, cidx[..., None].expand(-1, -1, ctx.shape[-1])) * cv[...,
    None]).sum(1) / cv.sum(1, keepdim=True).clamp(min=1)
        f = self.flank_net(torch.cat([nemb, cemb], 1))
        le = self.len_emb((pep_len - 8).clamp(min=0, max=7))
        return self.out(self.norm(torch.cat([v, vmax, h.mean(1), f, le], 1))).squeeze(-1)

# ---------------------------- caches as tensors ----------------------------
pep_t = load_big(f'{CACHE}/tr_pep_emb_650.npy')
ctx_t = load_big(f'{CACHE}/tr_ctx_emb_650.npy')
va_ctx_t = load_big(f'{CACHE}/va_ctx_emb_650.npy')
AA_BY_CODE_t = torch.from_numpy(AA_BY_CODE).to(DEV)
log('big caches in memory')

# ---------------------------- training ----------------------------
def dedupe_rows():
    key = pep_codes.astype(np.int64) * 100000 + hla_codes
    uk, first_idx, cnt = np.unique(key, return_index=True, return_counts=True)
    inv = np.searchsorted(uk, key)
    dup = cnt[inv] > 1
    sub = pd.DataFrame({'k': key[dup], 'y': y[dup]})
    g = sub.groupby('k').y.nunique(); conf = set(g[g > 1].index)
    conf_u = np.array([k in conf for k in uk])
    keep = (cnt[inv] == 1)
    keep[first_idx[~conf_u]] = True
    return np.where(keep)[0]

rows_all = dedupe_rows()
rng0 = np.random.default_rng(SEED0)
if SMOKE:
    rows_all = rng0.choice(rows_all, size=min(SMOKE_ROWS, len(rows_all)), replace=False)
    va_rows = np.arange(min(SMOKE_VAL, len(va)))
else:
```

A.5.2 SELECTED TREE-SEARCH SOLUTIONS

Listing 7. Amylase tree-search solution: ESM-2 likelihood features (node `b17b2f16`).

```
# [host] submit_epochs=10
EPOCHS = 10
import json
import numpy as np
import pandas as pd
import torch
from transformers import EsmForMaskedLM, EsmTokenizer
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler

NLL_CACHE = '/tmp/esm2_650m_nll.pt'

def nll_features(seqs, cache_path=NLL_CACHE):
    try:
        cached = torch.load(cache_path)
        if cached.shape[0] == len(seqs):
            return cached.numpy()
    except Exception:
        pass
    dev = 'cuda' if torch.cuda.is_available() else 'cpu'
    tok = EsmTokenizer.from_pretrained('facebook/esm2_t33_650M_UR50D')
    model = EsmForMaskedLM.from_pretrained('facebook/esm2_t33_650M_UR50D').to(dev).eval()
    feats = []
    bs = 64
    with torch.no_grad():
        for i in range(0, len(seqs), bs):
            b = seqs[i:i+bs]
            enc = tok(b, return_tensors='pt', padding=True).to(dev)
            out = model(**enc)
            logp = torch.log_softmax(out.logits, dim=-1)
```

```
            lab = enc['input_ids']
            m = enc['attention_mask']
            toklp = logp.gather(-1, lab.unsqueeze(-1)).squeeze(-1)
            nll = (toklp * m).sum(1) / m.sum(1)
            feats.append(nll.cpu())
    feats = torch.cat(feats)
    try:
        torch.save(feats, cache_path)
    except Exception:
        pass
    return feats.numpy()

def main():
    tr = pd.read_csv('/data/train.csv')
    va = pd.read_csv('/data/validation.csv')
    all_seqs = tr['sequence'].tolist() + va['sequence'].tolist()
    nll = nll_features(all_seqs)
    n_tr = len(tr)
    y = tr['target'].values
```

Listing 8. NucB tree-search solution: frozen ESMC-300M with a residual 1D-CNN head (node `196f50ff`).

```
import torch, numpy as np, pandas as pd, json
from transformers import AutoTokenizer, AutoModel
NAME = 'biohub/ESMC-300M'

def encode(seqs, tok, model, bs=64):
    (outs, masks) = ([], [])
    for i in range(0, len(seqs), bs):
        chunk = seqs[i:i + bs]
        enc = tok(chunk, return_tensors='pt', padding=True, truncation=True, max_length=256)
        with torch.no_grad():
            o = model(**{k: v.cuda() for (k, v) in enc.items()})
        outs.append(o.last_hidden_state.cpu())
        masks.append(enc['attention_mask'].cpu())
    X = torch.cat(outs, 0)
    M = torch.cat(masks, 0)
    return (X, M)

class ResBlock(torch.nn.Module):

    def __init__(self, c, k=3):
        super().__init__()
        self.conv = torch.nn.Conv1d(c, c, k, padding=k // 2)
        self.bn = torch.nn.BatchNorm1d(c)
        self.act = torch.nn.GELU()

    def forward(self, x):
        r = self.act(self.bn(self.conv(x)))
        return self.act(x + r)

class Net(torch.nn.Module):

    def __init__(self, din=960, c=256):
        super().__init__()
        self.stem = torch.nn.Sequential(torch.nn.Conv1d(din, c, 3, padding=1),
     torch.nn.BatchNorm1d(c), torch.nn.GELU())
        self.blocks = torch.nn.Sequential(*[ResBlock(c) for _ in range(4)])
        self.head = torch.nn.Sequential(torch.nn.Linear(c, 128), torch.nn.GELU(),
     torch.nn.Linear(128, 1))

    def forward(self, x, m):
        x = x.permute(0, 2, 1)
        x = self.stem(x)
        x = self.blocks(x)
        m = m.unsqueeze(1).float()
        x = (x * m).sum(2) / m.sum(2).clamp(min=1)
        return self.head(x).squeeze(-1)

def main():
    torch.manual_seed(0)
    np.random.seed(0)
    tok = AutoTokenizer.from_pretrained(NAME)
    model = AutoModel.from_pretrained(NAME).eval().cuda()
    for p in model.parameters():
        p.requires_grad_(False)
```

```
    tr = pd.read_csv('/data/train.csv')
    va = pd.read_csv('/data/validation.csv')
    (Xtr, Mtr) = encode(tr['sequence'].tolist(), tok, model)
```

Listing 9. Hemolysis SMILES tree-search solution: Morgan/ECFP fingerprints and physicochemical features with LightGBM (node `b8809feb`).

```
# [host] submit_epochs=10
EPOCHS = 10
import warnings, logging
warnings.filterwarnings('ignore')
logging.disable(logging.CRITICAL)
import os
os.environ['PYTHONWARNINGS'] = 'ignore'

import json
import numpy as np
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem, Descriptors, rdMolDescriptors, QED
import lightgbm as lgb

def get_fps(mols, radius, nbits):
    return np.array([AllChem.GetMorganFingerprintAsBitVect(m, radius, nBits=nbits) for m in
     mols], dtype=np.int8)

def get_descs(mols):
    out = []
    for m in mols:
        out.append([
            Descriptors.MolWt(m),
            Descriptors.MolLogP(m),
            Descriptors.TPSA(m),
            rdMolDescriptors.CalcNumRotatableBonds(m),
            Descriptors.FractionCSP3(m),
            rdMolDescriptors.CalcNumHBD(m),
            rdMolDescriptors.CalcNumHBA(m),
            rdMolDescriptors.CalcNumAromaticRings(m),
            rdMolDescriptors.CalcNumAliphaticRings(m),
            QED.qed(m),
            rdMolDescriptors.CalcNumAtoms(m),
        ])
    return np.array(out)

def build_features(smiles_list):
    mols = [Chem.MolFromSmiles(s) for s in smiles_list]
    X2 = get_fps(mols, 2, 2048)
    X3 = get_fps(mols, 3, 2048)
    D = get_descs(mols)
    return np.hstack([X2, X3, D])

def main():
    tr = pd.read_csv('/data/train.csv')
    va = pd.read_csv('/data/validation.csv')

    X = build_features(tr['sequence'].tolist())
    y = tr['target'].values
    Xv = build_features(va['sequence'].tolist())

    model = lgb.LGBMClassifier(
        n_estimators=600,
        learning_rate=0.05,
        num_leaves=63,
```

A.5.3 DIVERSITY-JUDGE REJECTIONS

Listing 10. Excerpts from FLIP diversity-judge rejections.

```
# Diversity REJECT examples , FLIP2 (stamp 20260925T061032Z)

## NucB
REJECT seed family=esm2 similar_to='esmc (frozen ESMC-300M + MLP regression head)'
reason: same plan as the ESMC seed , frozen ESM-family PLM, mean-pooled residues, small MLP
     head.
```

```
REJECT seed family=protbert similar_to='frozen ESMC-300M + MLP / frozen SaProt + MLP'
reason: minor checkpoint swap within frozen protein-transformer + regression-head pattern.

REJECT seed family=lora_ft similar_to='LoRA fine-tuning of ESM-2 650M + regression head'
reason: same LoRA-on-ESM core; only the backbone id changed.

## PDZ3
REJECT seed family=lora_ft similar_to='lora_ft (facebook/esm2_t33_650M_UR50D + supervised
    head)'
reason: same ESM-2 LoRA backbone already tried successfully.

## Amylase
REJECT seed family=esmc similar_to='ESM-C 300M + MLP (memory entry 2)'
reason: minor refinement of existing ESM-C + MLP (same backbone + head class).
```

Listing 11. Excerpts from Peptiverse SMILES diversity-judge rejections.

```
# Diversity REJECT examples , PeptiVerse SMILES (stamp 20260925T061032Z)

## Nonfouling SMILES
REJECT seed family=esmc similar_to='esm2 (Frozen ESM-2 650M + MLP head, f1=0.7254)'
reason: swapping ESM-2 for ESM-C is a minor checkpoint swap inside frozen-PLM + MLP.

## Hemolysis SMILES
REJECT seed family=esm2 similar_to='esm2 (frozen ESM-2 650M + MLP head on AA sequence)'
reason: essentially the same plan as the existing ESM-2 + MLP seed.

## Half-life SMILES
REJECT seed family=esm2 similar_to='esm2 seed (frozen ESM-2 650M mean-pooled + small MLP)'
reason: same frozen ESM-2 mean-pool + MLP core already in memory.
```

A.5.4 PROMPT TEMPLATES

Listing 12. Baseline agent system prompt (`baseline.prompt.SYSTEM_TEMPLATE`).

```
You are a machine learning engineer with a background in protein engineering and
computational biology. Train the strongest protein / peptide property (or fitness)
predictor you can for the given task.

Data access:
- Train: labeled , use it to fit models and for cross-validation / selection.
- Validation: sequences only in the sandbox , predict on it at submit time; you cannot
  score the competition metric yourself there.
- Test: not mounted. The host re-runs your solution.py on held-out sequences after submit.

Budget: at most 2.0 hours wall-clock. Prefer a solid model over tiny CV /
hparam increments.

Critical workflow hints:
1. For now, prefer fitting on the **entire** labeled train set , do not burn turns on
   elaborate CV / hparam grids. Quick sanity checks are fine; selection via deep CV is
   optional. solution.py should retrain on all of `/data/train.csv`.
2. You have 20 turns. When you have a strong candidate, call submit_answer with a
   full self-contained solution.py that prints PREDS_JSON=<json list of floats> for
   /data/validation.csv (or writes /out/preds.npy).
3. If submit fails (crash / bad preds), fix and resubmit. One accepted submit ends the run ,
   do not submit then keep exploring.

Avoid these common failure modes (they tank held-out Spearman):
- Reject collapsed predictors: if preds have very few unique values (or are almost
  constant), the ranking is dead , use sequence-discriminative features, not only
  n_mut or mean-over-positions aggregates.
- Prefer models/features that transfer under the stated `split_type` (epistasis /
  multi-mutant / new WT / far mutants) over ones that only fit train idiosyncrasies.
- On install / API / traceback issues: call `lookup_hint(<id>)` or
  `lookup_common_errors(<id>)` with an id from the catalogs below , do not invent APIs.

Models (full rein within budget / GPU):
- Sandbox: GPU (default T4); torch / transformers / peft available.
- Use any listed PLM, transformer / attention model, classical features + ridge (or
  similar), frozen embeds + light head, or other reasonable approaches.
- Optimize the model / representation, not exhaustive hyperparameter search. Pick
  reasonable defaults; tweak slightly across rounds if needed.
```

```
- Fine-tuning large LMs: LoRA / peft adapters only , not full-parameter FT. Keep rank,
  seq length, and batch size modest for T4; checkpoint early; fall back on OOM.
- Embedding cache (important): each execute_code is a new process , in-memory tensors
  do not carry over. If you embed the same sequences with the same model / pooling,
  save once (e.g. /tmp/train_emb_<model>.npy, /tmp/val_emb_<model>.npy) and np.load
  on later turns. Do not re-run from_pretrained + full-dataset embed every probe;
  only re-embed when the model, tokenizer, or pooling changes. Shared Hub *weights*
  live under /cache/models/huggingface/hub and /cache/models/torch/hub , READ for
  from_pretrained / fair-esm only. NEVER write embeds, preds, CSVs, LoRA adapters,
  or fine-tuned checkpoints there (cross-job leakage). Save adapters / embeds under
  /tmp or /work only. Do NOT set HF_HOME=/tmp (forces a re-download).
  Pattern for frozen PLM embeds (cache is automatic for HF EsmModel + fair-esm
  pretrained loaders , same tokenized sequence hits /cache/models/embeds/esm):
    import hashlib, numpy as np
    from pathlib import Path
    # Prefer: let the patched model.forward use the disk cache.
    # Explicit helper also available:
    #   from trellis_plm_cache import embed_sequences
    #   emb = embed_sequences(seqs, model_id='facebook/esm2_t33_650M_UR50D', pool='mean')
    key = hashlib.sha1((model_id + "|" + str(len(seqs))).encode()).hexdigest()[:12]
    path = f"/tmp/emb_{key}.npy"
    if not Path(path).exists():
        emb = encode(seqs)   # your forward pass (cache fills underneath)
        np.save(path, emb)
    else:
        emb = np.load(path)

[PLM catalog via format_available_lookups , omitted here; see baseline/prompt.py REP_INSTALL]
Finish with one submit_answer of your best solution.py.
```

Listing 13. Baseline tool list and output contract.

```
lookup_hint(name)
  Install / load snippets. name must be one of the lookup_hint keys listed in the
  system catalogs (or 'list').

lookup_common_errors(query)
  Resolve sandbox / torch / HF / RDKit / PLM footguns. query must be one of the
  lookup_common_errors keys listed in the system catalogs (or 'list' / a matching
  traceback substring).

execute_code(code, timeout_s?)
  Run Python in the sandbox (/data mounted; pip + network OK). Disk under /tmp and
  package installs persist across turns; RAM does not , cache embeddings to /tmp.

submit_answer(code, hypothesis?)
  Write /work/solution.py and run it. Need exit 0 and PREDS_JSON (or /out/preds.npy).
  Host returns validation scores; test scores stay private. One accepted submit ends the run.

solution.py must:
- train on /data/train.csv
- predict on unlabeled /data/validation.csv
- print PREDS_JSON=[...] (len=n_val) or write /out/preds.npy
- be a single seeded, self-contained file that retrains from scratch each run
  (host may re-execute it on other sequences with the same path)
- do NOT score metrics yourself (host scores). If you need Spearman, use
  `from scipy.stats import spearmanr` , never `sklearn.metrics.spearmanr` (ImportError).
```

Listing 14. Orchestrator system prompt (`mcts.prompts.orchestrator_system`).

```
Orchestrator: find the best predictor; beat the paper bar when given. Never write code.

Each expand locks CORE only: featurization × model family
  (e.g. frozen ESM2+MLP, ChemBERTa+ridge, one-hot+LightGBM, LoRA on ESM).
Subagent picks all other knobs (pool/lr/epochs/head/LoRA rank/...).

Hub ids in architecture: allowlisted only
  (facebook/esm2_t33_650M_UR50D, DeepChem/ChemBERTa-77M-MLM, biohub/ESMC-300M,
   Rostlab/prot_bert, westlake-repl/SaProt_650M_AF2). No invented esmif*/prot_t5 ids.
Or name the family ("frozen ESM-2 650M + MLP") and let the subagent lookup_hint.

NO incremental refine. Next plan must be dissimilar: new PLM / feats / model class /
large HP regime. Diversity judge must ACCEPT. Tiny same-core polish is forbidden.
```

```
Actions (tool enum this turn only):
  seed_architecture , new core (root)
  major_shift , dissimilar core under parent
  debug , same core, fix crash
  full_train_heldout , rare; only when offered
  stop , only when offered

architecture = short core. instructions = short goal (not a recipe).
Metric=spearman.
```

Listing 15. Subagent system prompt for implementing a new plan (`mcts.prompts.subagent_system`).

```
Subagent: ship working solution.py for the locked core. Host scores VAL (not you).

ORDER:
  1) lookup_hint('<family>') before from_pretrained / pip , use returned Hub id + snippet.
  2) Preinstalled: torch, transformers, fair-esm, peft, sklearn, lightgbm, pandas, numpy.
     pip only if import fails.
  3) solution.py: train on full /data/train.csv; predict all /data/validation.csv rows;
     prefer print PREDS_JSON=<json floats> (no /out needed). If np.save: mkdir /out first.
     Never paste long SMILES/seqs into source (SyntaxError).
  4) On traceback: lookup_common_errors, then fix. No invented APIs/Hub ids.
  LoRA only if locked plan says so → lookup_hint('lora').

SPEED: ≤6 short execute_code probes then submit. No LOO / multi-config CV / multi-seed
tournaments in-sandbox. One solid fit; host re-scores VAL. No train subsample / DIY val.

LOCKED CORE (keep):
  frozen ESM-2 650M + MLP head
ACTION: seed
INSTRUCTIONS: Implement this core plan.
Tweaks OK: pool, head, lr, epochs≤cap, batch, reg, LoRA rank, optim, scaling.
Do NOT change PLM / featurization / model family (orch's next expand).
SEED: from scratch. First: lookup_hint('<family>'), then code.

Tools: lookup_hint, lookup_common_errors, execute_code, submit_answer
  (submit: code + hypothesis with confidence in [0,1])
Budget: ≤10 epochs/train; ≤10 turns; submit ~600s.
Metric: spearman.

LOOKUP (keys only , call tool for snippets; never invent Hub ids):
  lookup_hint: aaindex, blosum, carp, chemberta, ecfp, esm1b, esm2, esmc, lora, one-hot,
    peptideclm, protbert, saprot
  lookup_common_errors: chemberta_load, cuda_numpy, esm_logits, fair_esm_unpack, hub_404,
    no_smiles_in_source, peft_targets, preds_contract, rdkit_api, smiles_esm,
    spearman_sklearn, tensor_dataset
HF transformers Hub ids (from_pretrained / AutoModel|EsmModel|... ONLY , not other libs):
  facebook/esm2_t33_650M_UR50D|t30_150M|t12_35M; biohub/ESMC-300M;
  DeepChem/ChemBERTa-77M-MLM; Rostlab/prot_bert; westlake-repl/SaProt_650M_AF2;
  aaronfeller/peptideclm-2-hybrid-small; multimolecule/carp-640m
  alt ESM-2: fair-esm esm.pretrained.esm2_t33_650M_UR50D() (not a transformers id)
BAN: facebook/esmif*, invented saprot/prot_t5 ids, ESM-IF1 (needs PDB), full-param FT
API: EsmModel→last_hidden_state; EsmForMaskedLM→.logits; LoRA ESM→lookup_hint('lora')
  (target_modules=['query','key','value'] , not q_proj/v_proj)

(search hints omitted)
```

Listing 16. Judge system prompt for new plans and major shifts.

```
You are the diversity judge for protein/peptide ML experiment plans.
You see MEMORY (prior *successful* plans + scores) and one NEW PROPOSAL.
Crashed / failed attempts are NOT in memory , those architectures may be tried again; do not
     reject a proposal merely because a similar plan might have failed earlier (you will not
     see those failures).
Decide accept/reject by calling judge_diversity once.

ACCEPT only if the proposal is meaningfully dissimilar to memory:
  - a new PLM / checkpoint family, OR
  - a new featurization / representation, OR
  - a new model class / head architecture, OR
  - a clearly LARGE hyperparameter regime change (qualitatively different setup, not a small
     nudge).
```

```
REJECT incremental refine / polish of something already tried:
  - same backbone + same features with a tiny delta (pooling, small lr/epoch tweak, light
     regularization, 'improve parent'),
  - near-duplicate architecture or instructions vs any memory entry,
  - same plan restated with cosmetic wording changes.

When unsure whether it is a real shift vs a refine, REJECT (diverse=false). Cite the similar
     memory entry in similar_to / reason.
```

Listing 17. Judge system prompt for debug expansions.

```
You are the diversity judge for a DEBUG fix under a crashed parent.
Decide accept/reject by calling judge_diversity once.
REJECT if the fix instructions are a near-duplicate of any prior expand.
ACCEPT a clear new failure-mode fix.
```

Listing 18. The `judge_diversity` tool schema (`mcts.diversity.JUDGE_TOOL`).

```
[
  {
    "type": "function",
    "function": {
      "name": "judge_diversity",
      "description": "Judge whether a proposed plan is diverse enough vs the diversity log
     and tree. For seeds/major_shifts: REJECT incremental refine / same-family polish;
     ACCEPT only new PLM, featurization, model class, or a clearly large HP regime change.",
      "parameters": {
        "type": "object",
        "properties": {
          "diverse": {
            "type": "boolean",
            "description": "True if the proposal is a meaningfully new plan."
          },
          "reason": {
            "type": "string",
            "description": "Short reason (cite which log/tree entry it clashes with if not)."
          },
          "similar_to": {
            "type": "string",
            "description": "Family or short plan snippet it is too similar to, else empty."
          }
        },
        "required": [
          "diverse",
          "reason"
        ],
        "additionalProperties": false
      }
    }
  }
]
```