DRUGSAGE: Self-evolving Agent Experience for Efficient State-of-the-Art Drug Discovery

Yikun Zhang[1,2*] **Xiwei Cheng**[1,2*] **Tianyu Liu**[3*] **Yuanqi Du**[4] **Wengong Jin**[1,2]

[1]Northeastern University [2]Broad Institute of MIT and Harvard
[3]Yale University [4]Microsoft Research New England

Abstract

Building state-of-the-art (SOTA) predictive models for drug discovery requires expensive search over tools, architectures, and training strategies. Current LLM-based agents can find SOTA solutions through extensive trial and error, but they do not retain the experience accumulated along the way and therefore pay the full search cost on every new task. We propose DRUGSAGE (Self-evolving Agent Experience), a framework that accumulates and reuses experience across tasks to build SOTA drug discovery models efficiently. DRUGSAGE maintains a cross-task memory of verified skills, statistical evidence about effective strategies, and a record of recurring errors and their fixes. In some cases, DRUGSAGE transfers a working solution directly without test-time search. On 22 TDC-ADMET molecular property prediction tasks, DRUGSAGE ranks first among nine SOTA agents in a single-task setting. With memory accumulated from 16 smaller tasks, DRUGSAGE-ZERO achieves an averaged normalized score of 0.935 on 17 held-out tasks, comprising six TDC-ADMET and 11 Polaris tasks, outperforming the best baseline score of 0.588 achieved with 20 search iterations while requiring no test-time search. In summary, our work shows the advantage of cross-task memory for efficient SOTA model development in drug discovery.

1 Introduction

Autonomous agents powered by large language models are increasingly capable of performing scientific research tasks end-to-end [1]. In biomedicine and drug discovery, systems such as Biomni [2] and STELLA [3] have demonstrated that LLM-based agents can automatically build workflows for data processing, feature extraction, and predictive modeling. More recent efforts such as AutoResearch [4] go one step further, moving from assembling a functional pipeline to actively searching over model architectures, training recipes, and preprocessing strategies to achieve state-of-the-art (SOTA) performance on a given benchmark. Achieving SOTA accuracy is especially critical for drug discovery because inaccurate predictions translate directly into experimental failures, which costs thousands of dollars and months of effort.

However, building SOTA models is an expensive search problem [5]. The agent must identify relevant tools and codebases from the literature, adapt them to the dataset at hand, tune training configurations and hyperparameters, and iterate through rounds of trial and error, all of which consume substantial compute and token budgets. This cost grows further when datasets become large, foundation models are expensive to fine-tune, and a single training run takes hours or days [6]. Slow error feedback makes this iterative search loop on which current agents rely increasingly impractical. Critically, most of the existing agents pay for this full cost independently on every new task, discarding all

*Equal contribution.

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

search experience before the next one begins. As a result, their ability to achieve SOTA performance has been difficult to scale across thousands of diverse prediction problems in drug discovery.

In this work, we propose DRUGSAGE (Self-evolving AGent Experience), a framework that accumulates and reuses experience across tasks to build SOTA solution efficiently. The key observation behind DRUGSAGE is that many drug discovery tasks share structural similarities overlooked by current agents. Predicting solubility, binding affinity, and bioactivity all involve the same input/output structure and differ only in their labels, data set sizes, and evaluation metrics. When an agent discovers that a particular molecular featurization performs well or a specific training recipe reliably improves a class of models, there is no reason to discard that knowledge. Therefore, DRUGSAGE treats each task not as an isolated problem, but as an experience that enriches the agent for future tasks. It maintains a cross-task memory that accumulates verified skills, statistical evidence about which strategies generally work better, and a record of recurring failure modes and their fixes. As this memory grows, the agent's search narrows: on a new task, instead of exploring the full space of tools, architectures, and hyperparameters from scratch, DRUGSAGE draws on prior experience to prioritize approaches that are likely to succeed, and in some cases, transfers a working solution directly with no additional search at all.

We evaluate DRUGSAGE on two benchmarks [7, 8] with 33 drug-property prediction tasks spanning absorption, distribution, metabolism, excretion, toxicity, binding, solubility, lipophicity, and bioactivity. In a single-task setting, DRUGSAGE ranks first among eight baseline agents, including autoresearch systems, ML automation agents, and scientific discovery agents. With cross-task memory accumulated from 16 smaller tasks, DRUGSAGE-ZERO achieves an average normalized score of 0.935 on 17 held-out tasks in a cross-task setting without test-time search, compared with 0.588 for the best baseline, Autoresearch. In summary, this work makes three key contributions.

- We introduce DRUGSAGE, an autonomous agent that maintains a cross-task memory integrated into a different stage of an MCTS-based search loop, with a formal guarantee that the memory-augmented selection policy preserves the regret bound of standard UCB for memory-adjusted regret.
- Unlike previous works such as Autoresearch [4] that refine user-provided or top leaderboard models, DRUGSAGE automatically builds a skill library by searching the literature and GitHub repositories. This broadens the search space and removes the need for a human-curated starting point.
- We show that cross-task memory enables a zero-test-time search regime: DRUGSAGE-ZERO transfers verified solutions to new tasks without test-time search. It achieves an average normalized score of 0.935 compared with 0.588 for the strongest baseline, even though the baselines perform 20 search iterations at test time.

2 Related Work

Automated algorithm discovery agents. Recent advances in LLMs have enabled agents to automate the model development in machine learning as a search problem over benchmark datasets. Early attempts solve this problem with retrieval-augmented generation from existing open-source models [9, 10]. Nevertheless, later work tackles them via searching algorithms [11, 12]. More recently, a variety of work improve over previous methods from several dimensions, including introducing agent structures [13, 14], incorporating better search algorithms [15, 16] and external knowledge [17]. DRUGSAGE introduces the accumulation of structured cross-task experience that grows as the agent solves successive tasks, enabling later tasks to directly retrieve verified solutions at zero search cost or to warm-start from empirically validated starting points.

Agent experience. A growing body of work equips LLM agents with persistent, evolving memory, which differs primarily in what is stored and how actionable it is. Episodic memory aids immediate retries with trial-level records [18], while other approaches distill non-executable insights into semantic memory [19–21]. For actionable guidance, systems develop procedural memory by maintaining append-only skill libraries [22], inducing reusable workflows [23], managing full memory lifecycles [24], or evolving shared knowledge via multi-agent reflection [25]. More recent work explores the accumulation of agent experiences across tasks such as [26–28]. DRUGSAGE combines two complementary memory mechanisms, pairing an executable skill library that expands across tasks with a performance-grounded memory that deepens through execution, enabling broader search coverage and more targeted reuse over time.

Scientific discovery agents. Recent work has begun to instantiate LLMs as scientific discovery agents with their growing capacity to reason over codes, tools, and scientific contexts. One line of work embeds LLMs in a verifier-driven hypothesis search loop. FunSearch [29] uses LLMs as evolutionary operators in a program evolution loop, while a significant amount of later works extend its capability beyond program discovery [30–32]. Beyond hypothesis search, an alternative perspective is to build agents that orchestrate tools such as Coscientist [33] and ChemCrow [34]. Later work [2, 3] has furthered agent capabilities to build computational workflows. More recently, SAGA [35] has taken a step forward in automatically evolving the objectives in the scientific discovery workflow. The most closely related work is Agentomics [36], which builds an end-to-end experimentation agent that explores ML modeling strategies for a given biomedical dataset. In contrast, DRUGSAGE formulates this setting as a cross-task search problem, reusing empirically validated strategies from prior tasks to seed experience-conditioned MCTS and avoid redundant exploration.

3 Methodology

Problem formulation. We first define the terminology used in this paper.

- A **target task** is $\tau = (\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}, \mathcal{D}_{\text{test}}, \mu, B)$, where $\mathcal{D}_*$ denotes the training, validation, and test data; μ is the task metric, such as AUROC, AUPRC, RMSE, or MAE; and B is the *search budget*, i.e., the number of new candidate solutions the agent may train and validate during iterative search.
- The **skill library** $\mathcal{K}$ is a collection of validated, task-relevant executable skills, including model families, dataset-specific methods, and training strategies such as featurization, preprocessing, class-imbalance loss, sampling, tuning.
- An **executable solution** is a runnable model built from the skills in $\mathcal{K}$, possibly with typed edits. It specifies the entire pipeline (e.g., model architecture, featurization, training procedure) and is included only if it can be executed in a sandbox. The *best solution* is a solution that receives the best score on the *validation* set according to a given task metric μ.
- The **search tree** contains all executable solutions for a given task. Each *node* in the search tree is a particular instance of a model family. Each *edge* contains **typed edits** of its parent node, including changes in model architecture, training objective, data processing, and ensemble strategies.

Overview. The goal of DRUGSAGE is to find the best solution within the search budget B. As shown in Figure 1, DRUGSAGE first constructs an executable skill library $\mathcal{K}$ (§3.1) by searching the literature. It then maintains a experience memory $\mathcal{Z}$ across tasks (§3.2). When $B > 0$, DRUGSAGE runs the memory-enhanced Monte Carlo tree search algorithm (MCTS) to search for executable solutions, where each step is equipped with one of the memory components (§3.3). When $B = 0$, DRUGSAGE performs memory routing to transfer a verified solution directly without additional search (§3.4). We call this setting DRUGSAGE-ZERO.

3.1 From Literature to Executable Skills

A scientific agent needs an open search space, but not an unconstrained one. DRUGSAGE treats the literature and GitHub repositories as a source of executable search space. Before budgeted search begins, the Explore Agent adds new methods to the shared skill library $\mathcal{K}$. It follows a discovery, grounding, and validation procedure. The discovery stage expands the task to multiple expert perspective queries, searches the literature, selects relevant papers with usable repositories, and writes a task memory of candidate methods. The grounding stage clones selected repositories, extracts an abstract-syntax-tree (AST) API snapshot of public interfaces, model classes, and usage examples, and asks the LLM to write a grounded `SKILL.md` using only symbols observed in that snapshot. The validation stage runs tiered checks, from syntax and import resolution to end-to-end execution in a per-skill sandbox, with automatic repair on failure. Only validated skills are included in $\mathcal{K}$. In this way, the search space can grow with the field while remaining executable. Details of the explore agent are provided in the Appendix H.

3.2 Experience Memory

The agent maintains a persistent memory $\mathcal{Z} = (\mathcal{R}, \mathcal{H}, \mathcal{Q})$ across tasks. These memories provide reusable cross-task evidence for the search tree.

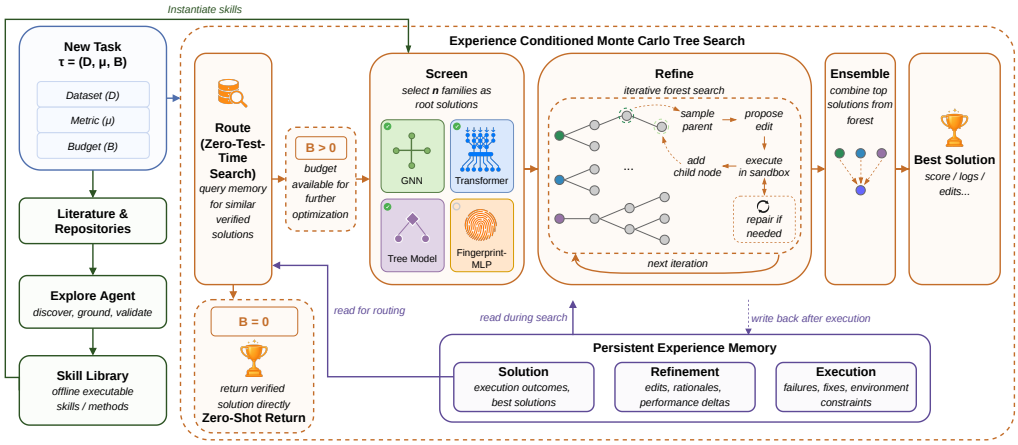

Figure 1: **DRUGSAGE overview.** The Explore Agent builds a shared skill library $\mathcal{K}$ from literature and repositories. Cross-task memory $\mathcal{Z}$ enables two settings: experience-conditioned MCTS ($B > 0$), where $\mathcal{Z}$ guides and is updated by each search step, and zero-test-time routing ($B = 0$), where a verified solution is retrieved from $\mathcal{Z}$ without any new search.

The solution memory $\mathcal{R}$ records the performance of different models in different tasks. Each record in this memory describes a node in the search tree, including its model family, model architecture, hyperparameters, training procedure, data processing, training objectives, task description, and its performance on the validation set. $\mathcal{R}$ is updated whenever a new solution is added to the search tree. This memory helps DRUGSAGE identify the most promising solutions to explore.

The refinement memory $\mathcal{H}$ stores the information associated with each edge in the search tree. Each record contains the description of the parent and child nodes, task description, LLM-proposed edits (e.g., changes of model architecture, training objective, data processing); rationale (LLM-generated reasons why proposed edits are beneficial); and the validation performance difference between the parent and child. $\mathcal{H}$ is updated when a child node spawns from a parent with LLM-generated edits. This memory helps DRUGSAGE identify promising optimization strategies, e.g., what changes in model architecture, training pipeline, and data processing generally work better than others.

The execution memory $\mathcal{Q}$ stores all the logs generated by the sandbox during execution, including failure information, verified fixes, resource profiles, environment traces, and sandbox logs. When the execution of a candidate solution fails, the sandbox matches the error against known failure signatures and applies a verified fix immediately if one exists. Resource profiles keep track of the average runtime and memory of this model family, which is useful for preventing out-of-memory or timeout failures. After each execution, new failure information, fixes, and resource profiles are appended to $\mathcal{Q}$. This memory enables DRUGSAGE to apply quick fixes when it encounters execution failures.

3.3 Experience-Memory-Enhanced MCTS

DRUGSAGE searches over a task-level solution forest. Each root is a baseline executable solution obtained by skill from $\mathcal{K}$ on the target task. Each edge is a typed edit or composition operation, and each node is a concrete executable solution with execution status, metric record, lineage, and artifacts. The search loop alternates between four stages: screening roots, refining completed parents, proposing executable edits, and executing or repairing the resulting candidates. Experience memory $\mathcal{Z} = (\mathcal{R}, \mathcal{H}, \mathcal{Q})$ enters this loop through four distinct interfaces, illustrated in Figure 2.

Step 1: selecting model families using the solution memory $\mathcal{R}$**.** The screening step allocates the budget to different model families based on their performance on historical tasks. Specifically, DRUGSAGE selects the next eligible family f using a memory-augmented UCB rule:

$$f_t = \arg\max_{f \in \mathcal{F}_{\text{eligible}}} \underbrace{\text{exploit}_t(f)}_{\text{target task}} + \underbrace{\alpha\sqrt{\frac{\ln(t+1)}{n_t(f)}}}_{\text{exploration}} + \underbrace{\text{transfer}(f)}_{\text{historical tasks}}, \tag{1}$$

where t indexes the screening step; $n_t(f)$ (≥ 1) counts the number of visits of the model family f; $\text{exploit}_t(f) \in [0, 1]$ records the best performance of any node in the model family on the target

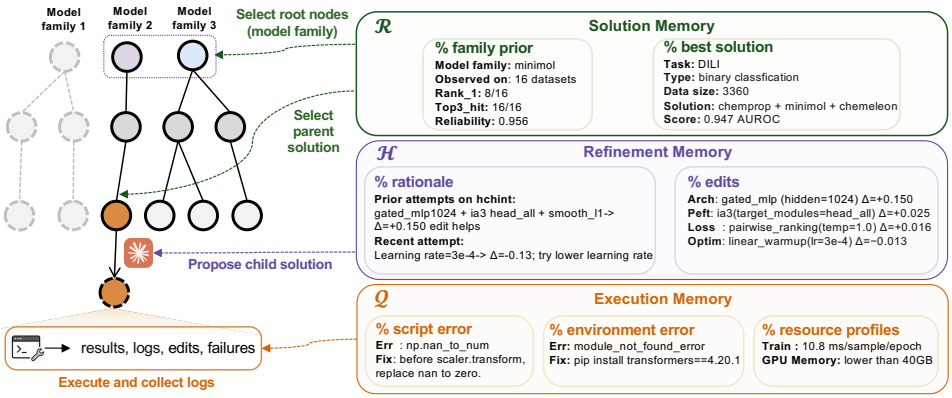

Figure 2: **Experience Memory.** DRUGSAGE stores cross-task experience as $\mathcal{Z} = (\mathcal{R}, \mathcal{H}, \mathcal{Q})$. Solution memory $\mathcal{R}$ stores family priors and verified best solutions for root-family and parent-solution selection. Refinement memory $\mathcal{H}$ stores proposal rationales and typed edit effects for grounding child-solution proposals. Execution memory $\mathcal{Q}$ stores script errors, environment failures, verified fixes, and resource profiles for sandbox repair and resource-aware execution. After each execution, outcomes, edits, rationales, logs, fixes, and resource usage are written back to memory, converting prior train/eval runs into reusable cross-task evidence.

task, penalized for instability and overfit; $\mathrm{transfer}(f)$ computes the weighted average performance of nodes in family f over historical tasks, using the records retrieved from the solution memory. We formally define $\mathrm{transfer}(f)$ in Appendix I. Intuitively, when $t = 0$, the agent tends to choose the model family with the best performance on historical tasks. As the agent explores more solutions on the target task, the agent relies more on performance on the current task ($\mathrm{exploit}_t(f)$) rather than the performance on the historical tasks ($\mathrm{transfer}(f)$). Moreover, we formally show that introducing this bias term does not change the regret bound of standard UCB:

Theorem 3.1. *Assuming bounded historical performance* $|\mathrm{transfer}(f)| \le C_0$*, and standard asymptotic convergence of the exploit estimate, the policy in Equation* (1) *attains cumulative memory-adjusted regret*

$$R_T \le \sum_{f \in \mathcal{F}: \Delta_f > 0} \left(\frac{8(\alpha^2 + C_0^2) \ln T}{\Delta_f} + \left(1 + \tfrac{\pi^2}{3}\right) \Delta_f \right), \tag{2}$$

matching the $O(|\mathcal{F}| \log T)$ *order of standard UCB.*

Proof: We use the boundedness and asymptotic consistency of $\mathrm{transfer}(f)$*, follow the proof steps in UCB1 and AM-GM inequality to prove it. Details are in Appendix J.2.3.*

The regret is defined with respect to the memory-adjusted reward $\tilde{r}(f) + \Delta^*(f)$ in Appendix J.2.1. The node-level regret discussion in that appendix is informal.

Step 2: sampling parent solutions using the solution memory $\mathcal{R}$. Once the roots are selected, the decision shifts from family selection to choosing a specific solution in the model family to expand. Let $\mathcal{V}$ be a pool of completed, non-ensemble nodes with finite primary metrics. For each $v_i \in \mathcal{V}$, let q_i denote its performance on the target task and $c_i = |\mathrm{children}(v_i)|$ its expansion count. The parent sampler assigns the following weight to each candidate solution:

$$w_i = \underbrace{\sigma\left(\beta \, \frac{q_i - \mathrm{median}(q)}{\max(\mathrm{MAD}(q), \epsilon)}\right)}_{\text{target task performance}} \cdot \underbrace{\frac{1}{1 + c_i}}_{\text{breadth}} \cdot \underbrace{\big(1 + \lambda \cdot \mathrm{transfer}(v_i)\big)}_{\text{historical task performance}}, \quad p(v_i) = \frac{w_i}{\sum_j w_j}, \tag{3}$$

where $\sigma(x) = (1 + e^{-x})^{-1}$, $\mathrm{MAD}(q)$ is the median absolute deviation across $\mathcal{V}$, $\epsilon > 0$ guards against vanishing denominator, and $\mathrm{transfer}(v_i)$ is the node-level analog of $\mathrm{transfer}(f)$ (Appendix I). The three factors balance target task performance, expansion breadth (discouraging overly-expanded parents), and historical performance on related tasks.

Step 3: generating child solutions using the refinement memory $\mathcal{H}$. After a parent is sampled, DRUGSAGE asks the LLM to propose a typed edit. The prompt is grounded by $\mathcal{H}$, which retrieves relevant rationales, previous attempts, successful recipes, and recent trajectory summaries for the

current task and parent lineage. The generated edit must parse into the expected schema and pass duplicate and feasibility checks. If the proposal is empty, malformed, or nearly a duplicate of a previous child, DRUGSAGE resamples a parent or retries proposal generation. After execution, the proposed edit, rationale, and the observed outcome are written back into $\mathcal{H}$.

Step 4: construct environments to execute solutions using the execution memory $\mathcal{Q}$. A proposed child is materialized as an executable program and evaluated in a sandbox. Before execution, DRUGSAGE automatically prepares a skill-specific runtime: it resolves dependencies, creates or reuses an isolated environment, installs missing packages when needed, verifies imports with smoke tests, and sets timeout and resource limits using profiles from $\mathcal{Q}$. If setup or execution fails, the repair loop queries $\mathcal{Q}$ for a verified fix matching the observed error; matched fixes are applied immediately and retried without additional LLM debugging. Otherwise, DRUGSAGE escalates to dependency repair, script patching, program regeneration, or environment rebuild. Successful fixes are marked verified; the final status, metrics, logs, resource trace, and repair outcome are written back to $\mathcal{Q}$.

Step 5: update memories. Every executed candidate produces a result bundle that updates both the task-level forest and persistent memory. $\mathcal{R}$ receives the score, status, and the description of generated solutions. $\mathcal{H}$ receives the proposal rationale and trace of each LLM-guided child expansion. $\mathcal{Q}$ receives logs recording failures, repair, and resource profiles of all executed experiments. In this way, the same execution advances the current task and also becomes reusable evidence for later tasks.

3.4 DRUGSAGE-ZERO: Zero-Test-Time Search by Memory Routing

Normally, DRUGSAGE explores candidate skills from the literature, optimizes them through iterative refinement, and returns the best solution found within a given experiment budget. However, this search process often consumes substantial compute and token budgets. With cross-task memory, the agent can draw on the memory accumulated from previous tasks to narrow or bypass the search entirely by reusing strategies that have proven effective in similar tasks. When $B = 0$, DRUGSAGE-ZERO transfers a verified solution directly without launching any additional search.

Cross-task memory routing. This is the key to DRUGSAGE-ZERO's zero-test-time search capability. For a target task $\tau = (\mathcal{D}, \mu, B)$ with dataset description d_τ, the agent forms a task signature

$$\phi(\tau) = \big(\text{type}(\tau),\ \mu,\ \log|\mathcal{D}|,\ \mathbf{e}(d_\tau)\big), \tag{4}$$

where $\mathbf{e}(d_\tau)$ is an embedding of the task description. Using this signature, the agent identifies **analog tasks** with similar task type and evaluation metrics as the target metric. DRUGSAGE-ZERO ranks these analog tasks based on the proximity in training set size $\log|\mathcal{D}_{\text{train}}|$ and the cosine similarity of task description embeddings $\mathbf{e}(d_\tau)$. Lastly, the agent retrieves all the solutions developed for the closest analog task and rank them based on their performance. DRUGSAGE-ZERO returns the best solution in the analog task and deploys it in the target task.

4 Experiments

Our experiments evaluate DRUGSAGE in two settings: (1) we run DRUGSAGE in a single-task setting, where agents need to develop a SOTA solution from scratch, without relying on cross-task experience. This setting aims to evaluate DRUGSAGE's basic problem-solving and coding ability and properly compare our method with previous AI agent frameworks that do not have cross-task memory; (2) we run DRUGSAGE in a cross-task setting, allowing the agent to transfer useful experience from previous tasks to reduce the search budget required for a new task or directly transfers a solution without test-time search.

4.1 Evaluation of DRUGSAGE in the single-task setting

Benchmark tasks. To evaluate whether DRUGSAGE can build SOTA solutions from scratch, we collected 22 molecular property prediction tasks from Therapeutic Data Commons (TDC) [7]. We chose TDC because it has a public leader board of the best solutions curated by human developers, which is important for us to verify if any discovered solution is SOTA. The 22 tasks span a variety of properties critical for drug discovery, including Absorption (6), Distribution (3), Metabolism (6), Excretion (3), and Toxicity (4), with training set sizes ranging from 475 to 13,130.

Table 1: Results on the first scenario, where each method searches for SOTA solutions from scratch. Expl. and Mem. indicate whether the system supports exploration and memory through its standard interface. Avg Rank per Category is averaged over each subset of datasets. Norm. Score is the average min-max-normalized score over 22 datasets. #Wins/Tot. counts datasets ranked first. Per-task absolute metrics (mean ± std over 5 seeds) are reported in Section D.

	Capability		Avg Rank per Category ↓					Overall		
Method	Expl.	Mem.	Abs (6)	Dist (3)	Meta (6)	Excr (3)	Tox (4)	Avg Rank ↓	Norm. Score ↑	#Wins/Tot. ↑
TDC reference										
TDC Leaderboard Official	–	–	2.83	5.33	3.17	**2.33**	3.00	3.23	0.878	2/22
Optimization-only agents (top-3 leaderboard model anchored)										
Autoresearch	✗	✓	5.67	4.67	4.17	4.33	5.00	4.82	0.741	2/22
ShinkaEvolve	✗	✓	5.33	4.00	5.33	5.33	4.25	4.95	0.731	0/22
DRUGSAGE-anchor mode	✗	✓	2.17	3.00	**2.67**	3.00	2.50	2.59	0.875	7/22
ML automation agents										
MLEvolve	✓	✓	7.50	7.67	6.83	8.67	8.25	7.64	0.567	0/22
AIRA-dojo	✓	✓	7.83	8.00	8.00	7.00	6.75	7.59	0.615	0/22
General coding agents										
Claude Code	✓	✓	7.17	5.67	6.67	5.00	7.75	6.64	0.646	0/22
Scientific discovery agents										
Biomni	✓	✗	10.50	10.67	9.17	11.00	10.75	10.27	0.076	0/22
STELLA	✓	✓	7.33	8.00	8.67	7.67	7.00	7.77	0.550	1/22
Agentomics	✓	✓	8.17	8.00	8.67	8.67	8.75	8.45	0.544	0/22
Ours										
DRUGSAGE	✓	✓	**1.33**	**1.00**	**2.67**	3.00	**1.75**	**1.95**	**0.929**	**10/22**

Baselines. Our baselines span two distinct paradigms, and we construct a fair comparison for each. The first paradigm consists of *optimization-only agents* that refine existing algorithms rather than building a pipeline from scratch: Autoresearch [4] and ShinkaEvolve [37], both run by anchoring on the top-three open-sourced TDC leaderboard models. For fair comparison, we include DRUGSAGE-anchor, which searches from the same starting points as optimization-only agents. The second paradigm consists of agents that develop a full solution pipeline from scratch: two *ML-automation agents*, MLEvolve [16] and AIRA-Dojo [12]; one *general-purpose coding agent*, Claude Code [38]; three *scientific-discovery agents*, Biomni [2][2], STELLA [3], and Agentomics [36]. All from-scratch agents, including DRUGSAGE, receive the same shared task prompt (Section C), while each baseline retains its original system prompt and native harness. We also include TDC-Leaderboard as a human-curated reference. All agents are powered by Claude-sonnet-4.6, except that Biomni and STELLA use their own default API configurations because their implementation does not support Claude.

Protocol. We evaluated DRUGSAGE on all 22 TDC datasets independently, so every task searches for the SOTA solution from scratch. We use the official 5-seed train-validation-test split and collect each task's native metric, so our scores are directly comparable to public TDC leaderboard entries. All agents selects best solutions by validation performance and we report the average test metrics. We use min-max normalization to compute a normalized score: $\text{score}_{m,d} = (x_{m,d} - \min_d)/(\max_d - \min_d)$ for metrics that are higher the better and $\text{score}_{m,d} = (\max_d - x_{m,d})/(\max_d - \min_d)$ for metrics that are lower the better. We report the average normalized score for all 22 tasks in Table 1.

Results. Table 1 evaluates from-scratch search without cross-task memory. Full DRUGSAGE achieves the best average rank (1.95) and the most wins ($10/22$). To control for the search space, DRUGSAGE-anchor disables the Explore Agent and uses the same top-3 TDC leaderboard models as Autoresearch and ShinkaEvolve. Under this matched setting, DRUGSAGE-anchor still reaches an average rank of 2.59 and $7/22$ wins, outperforming Autoresearch (4.82, $2/22$) and ShinkaEvolve (4.95, $0/22$). The gap between full DRUGSAGE and DRUGSAGE-anchor measures the benefit of automatically building the executable skill library. We also compare DRUGSAGE-anchor and MLEvolve-anchor with the same top-three starting models, no cross-task memory, the same task prompt, and matched compute and evaluation budgets. Their normalized scores are 0.875 and 0.630, respectively; the remaining differences are in search, refinement, repair, and execution (Appendix K.1).

[2]Biomni results are provided by the Biomni team from their beta platform and are not independently reproduced by us.

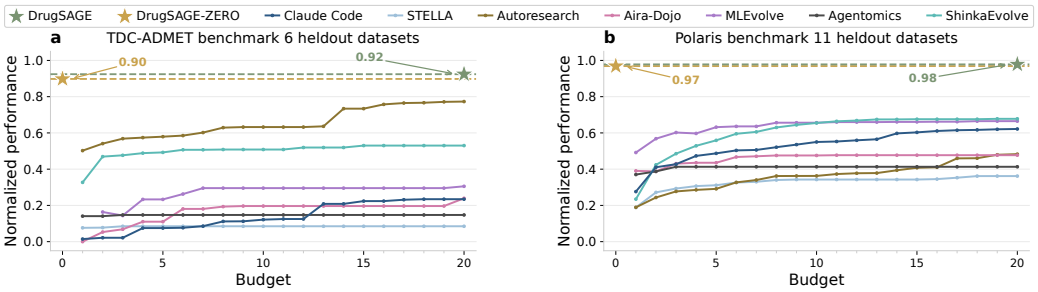

Figure 3: **Cross-task efficiency on held-out tasks. (a)** Six TDC-ADMET held-out tasks. **(b)** Eleven Polaris held-out tasks. Scores are min-max normalized per task before averaging; higher is better. The yellow dashed line is DRUGSAGE-ZERO with $B = 0$, and the green curve is DRUGSAGE with the same memory and up to 20 target-task search steps. TDC-ADMET and Polaris absolute metrics and per-task curves are in Section E and Section F.

4.2 Evaluation of DRUGSAGE in the cross-task setting

Experience pool and held-out benchmark tasks. In this scenario, we want to test whether the memory $\mathcal{Z}$ can transfer experience to a new target task and eliminate the need for test-time search. For this purpose, we partition the 22 TDC datasets by training-set size: the 16 smallest tasks ($< 5{,}000$ samples) form the *experience pool*, from which DRUGSAGE builds the cross-task memory $\mathcal{Z}$ and the skill library $\mathcal{K}$; the rest of the six largest tasks serve as held-out tasks. This size-based split allows DRUGSAGE-ZERO to gain experience on small tasks before the agent encounters the larger targets, where each trial is more expensive. In addition, we collect 11 tasks from the Polaris Hub [39], including six ADMET tasks from Fang et al. [40] and five kinase-inhibition tasks based on the PKIS2 data of Drewry et al. [41]. These 11 Polaris tasks form a separate held-out benchmark set used only at evaluation time and never entering $\mathcal{Z}$. The two held-out sets together give 17 tasks on which we measure cross-task memory transfer.

Setup. According to the zero-test-time routing regime defined in §3.4, DRUGSAGE-ZERO transfers a verified solution from memory with $B = 0$, without test-time search on the held-out tasks. Baseline agents also have their own memory mechanisms, but they do not maintain the cross-task memory studied here. We therefore report their best-so-far performance over $B \in \{1, \ldots, 20\}$ search steps under each agent's native budget definition. All methods are evaluated on the same held-out targets using the benchmark metrics. The skill library and repositories are fixed before evaluation, and each held-out task uses the same frozen memory independently without updating the experience pool. In deployment, memory is incrementally updated as tasks are completed.

Results. Figure 3 compares DRUGSAGE-ZERO against baseline agents with increasing search budgets. On the six held-out TDC-ADMET tasks, DRUGSAGE-ZERO reaches an average normalized performance of 0.90, exceeding the strongest baseline at $B = 20$. On the 11 held-out Polaris tasks, it reaches 0.97, again outperforming all baselines using their full search budget. Across the 17 held-out tasks, DRUGSAGE-ZERO obtains an averaged normalized score of 0.935, compared with 0.588 for the strongest baseline, Autoresearch. On the five PKIS2 kinase tasks, DRUGSAGE and DRUGSAGE-ZERO obtain scores of 0.96 and 0.91, respectively, using ADMET-only memory (Appendix K.6).

The Polaris results test whether zero-test-time routing works beyond the TDC benchmark, which is partly used to build memory. Table 2 shows the performance (without normalization) for each TDC held-out tasks. We find that DRUGSAGE-ZERO achieves competitive performance than top TDC leaderboard models and DRUGSAGE without cross-task memory with $B = 20$ test-time search. In some cases like Solubility and CYP2D6 Inhibition, DRUGSAGE-ZERO even outperforms DRUGSAGE without cross-task memory (0.686 vs. 0.722 MAE; 0.779 vs. 0.724 AUPRC). LD50 is the only target where DRUGSAGE-ZERO underperforms by a non-trivial margin (0.547 vs. 0.502 MAE), but it stays close to the best leaderboard model. In summary, these results show that experience accumulated on the historical tasks transfers to held-out tasks and can provide a high-quality solution for related problems without test-time search. Code of solution and analysis are shown in Section G.3.

Table 2: The cross-task performance (without normalization) of DRUGSAGE-ZERO on six TDC held-out tasks. The DRUGSAGE column reports search without cross-task memory at $B = 20$. Full cross-task results for all individual tasks are in Table 8 and Table 9.

Task (size)	Metric	DRUGSAGE-ZERO	DRUGSAGE (no memory)	TDC Leaderboard Best
Ames (7,255)	AUROC ↑	0.871 ± 0.003	0.878 ± 0.013	0.871 ± 0.002
LD50 (7,385)	MAE ↓	0.547 ± 0.010	0.502 ± 0.013	0.552 ± 0.009
Solubility (9,982)	MAE ↓	$\mathbf{0.686 \pm 0.006}$	0.722 ± 0.011	0.741 ± 0.013
CYP2C9 Inhibition (12,092)	AUPRC ↑	0.852 ± 0.003	0.861 ± 0.002	0.859 ± 0.001
CYP3A4 Inhibition (12,328)	AUPRC ↑	0.914 ± 0.002	0.914 ± 0.001	0.916 ± 0.000
CYP2D6 Inhibition (13,130)	AUPRC ↑	$\mathbf{0.779 \pm 0.007}$	0.724 ± 0.002	0.790 ± 0.001

4.3 Ablation Study

Benefit of the explore agent. In Table 1, we compare the performance of DRUGSAGE against DRUGSAGE-anchor where explore agent is disabled and replaced with top TDC leaderboard models. We find that DRUGSAGE outperforms DRUGSAGE-anchor (normalized score 0.929 vs 0.875), proving the benefit of skill library automatically constructed by the explore agent.

LLM cost. Figure 4(a) compares the total LLM API cost on the six held-out TDC-ADMET tasks $B = 20$. Compared with general coding and optimization baselines, DRUGSAGE variants use substantially lower LLM cost. In particular, DRUGSAGE-ZERO performs no LLM generation at test time and only calls the lightweight `text-embedding-3-small` model for memory routing, resulting in near-zero task-level LLM API cost. These test-time costs exclude the one-time skill-library build (about USD 10.5) and the 16 prior-task searches used to construct memory. Appendix K.2 reports the cost breakdown and token usage.

Importance of cross-task memory. Figure 4 (a, b) evaluates the contribution of cross-task experience memory $\mathcal{Z}$ under the same experimental setting. From without $\mathcal{Z}$, to with $\mathcal{Q}$, to full memory, the normalized score increases monotonically, while the cost decreases monotonically, showing that the gain is not only from the base search procedure but also from reusing experience across tasks. Removing $\mathcal{H}$, $\mathcal{Q}$, or $\mathcal{R}$ separately gives scores of 0.91, 0.89, and 0.83, compared with 0.99 for full memory. Across 10 independently sampled, stratified memory-growth chains, the mean transfer score increases from 0.39 at two memory tasks to 0.90 at 16 tasks (Figure 4d; Appendix K.4).

Impact of analog tasks. To examine the impact of analog tasks on the performance of DRUGSAGE-ZERO, we conduct ablation studies on four target tasks (Solubility, CYP2C9, CYP3A4, CYP2D6) that have tasks in the experience pool belonging to the same category. We run DRUGSAGE-ZERO with the matched category removed from the experience pool. This forces the agent to select a less related source task from a different task category, often with a different metric. As shown in Figure 4(c), Solubility MAE changes from 0.686 to 0.763, while AUPRC changes from 0.852 to 0.822 on CYP2C9, 0.914 to 0.892 on CYP3A4, and 0.779 to 0.744 on CYP2D6. The 17-task averaged normalized score decreases from 0.935 to 0.787, but remains above the strongest baseline at 0.588. The other 13 held-out tasks already route across categories. These results show that the performance of DRUGSAGE-ZERO is not merely copying exact same-category analogs on these targets. Transfer can also hurt: on LD50, MAE is 0.502 without memory, 0.547 for DRUGSAGE-ZERO, and 0.539 for full DRUGSAGE (Appendix K.5).

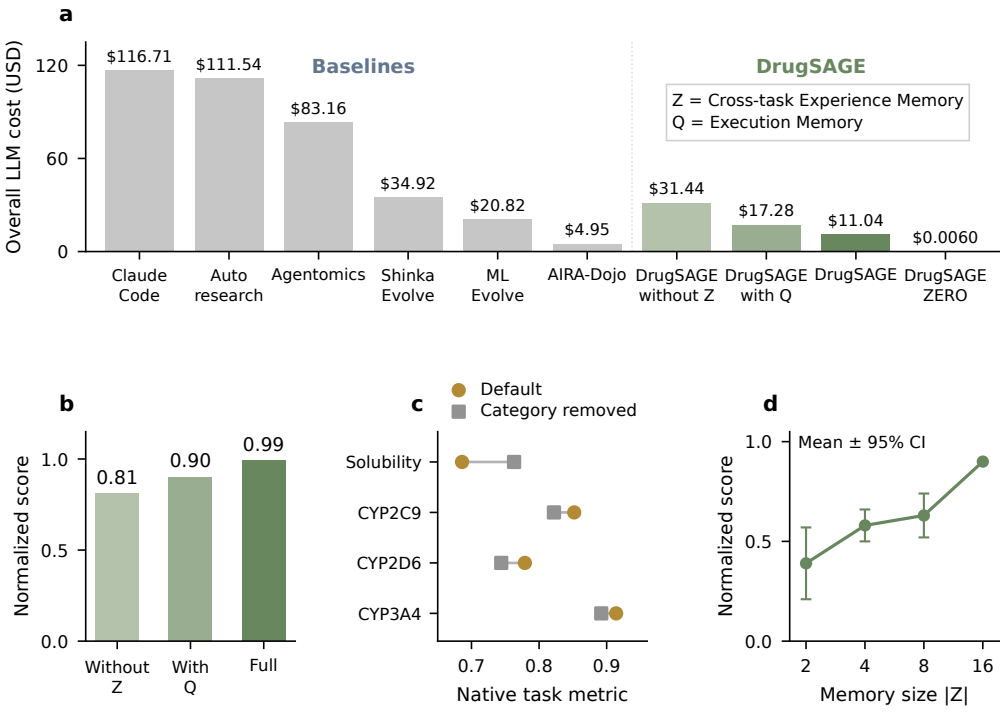

Figure 4: **(a)** Total LLM API cost on the six held-out TDC-ADMET tasks. Compared with baselines, DRUGSAGE variants require substantially lower LLM cost. DRUGSAGE-ZERO uses nearly zero LLM cost by reusing cross-task experience without test-time search. **(b)** Performance improves from DRUGSAGE without $\mathcal{Z}$, to DRUGSAGE with $\mathcal{Q}$, and further to full DRUGSAGE, showing each module's contribution. **(c)** Ablation study of analog-task impact on four target tasks before and after removing the matched category. Lower MAE and higher AUPRC are better. **(d)** Memory growth on the six held-out TDC-ADMET tasks: mean $\pm$ 95% CI across 10 independently sampled, stratified task-arrival chains. All chains reach the same complete memory pool at 16 tasks. Details are in Appendix K.4.

5 Discussion

In this paper, we introduce an agentic framework DRUGSAGE for efficient drug discovery powered by agent-driven exploration, cross-task memory, and automatic experiment refinement. We demonstrate that DRUGSAGE can leverage the knowledge learned from different tasks to improve algorithm performances compared with other baselines for both in-distribution and out-of-distribution problems. There are many interesting directions to further improve our framework. For example, our agentic framework prioritizes skill sets based on prior knowledge such as citation number, GitHub number, etc. Investigating the possibility of selecting important skills based on the combination of prior knowledge and experiments on datasets could be helpful. Finally, we test our framework mainly based on drug discovery tasks, but it could be also generalized to other scientific research areas. We plan to work on these directions in the future.

Acknowledgments

Funding. We gratefully acknowledge support from the Google Research Scholar Award, the NVIDIA Academic Grant Program, the Google TPU Research Cloud Award, and NSF ACCESS.

Competing interests. The authors declare no competing interests at this time.

References

[1] Jiaqi Wei, Yuejin Yang, Xiang Zhang, Yuhan Chen, Xiang Zhuang, Zhangyang Gao, Dongzhan Zhou, Guangshuai Wang, Zhiqiang Gao, Juntai Cao, et al. From ai for science to agentic science: A survey on autonomous scientific discovery. *arXiv preprint arXiv:2508.14111*, 2025.

[2] Kexin Huang, Serena Zhang, Hanchen Wang, Yuanhao Qu, Yingzhou Lu, Yusuf Roohani, Ryan Li, Lin Qiu, Gavin Li, Junze Zhang, et al. Biomni: A general-purpose biomedical ai agent. *biorxiv*, 2025.

[3] Ruofan Jin, Zaixi Zhang, Mengdi Wang, and Le Cong. Stella: Self-evolving llm agent for biomedical research. *arXiv preprint arXiv:2507.02004*, 2025.

[4] Andrej Karpathy. autoresearch: Ai agents running research on single-gpu nanochat training automatically. `https://github.com/karpathy/autoresearch`, 2026.

[5] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

[6] Guangji Bai, Zheng Chai, Chen Ling, Shiyu Wang, Jiaying Lu, Nan Zhang, Tingwei Shi, Ziyang Yu, Mengdan Zhu, Yifei Zhang, et al. Beyond efficiency: A systematic survey of resource-efficient large language models. *arXiv preprint arXiv:2401.00625*, 2024.

[7] Kexin Huang, Tianfan Fu, Wenhao Gao, Yue Zhao, Yusuf Roohani, Jure Leskovec, Connor W Coley, Cao Xiao, Jimeng Sun, and Marinka Zitnik. Therapeutics data commons: Machine learning datasets and tasks for drug discovery and development. *Advances in Neural Information Processing Systems*, 2021.

[8] Polaris. Polaris hub. `https://github.com/polaris-hub/polaris`, 2025.

[9] Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. Ds-agent: Automated data science by empowering large language models with case-based reasoning. In *Forty-first International Conference on Machine Learning*, 2024.

[10] Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, Jinwoo Shin, Sercan O Arik, and Tomas Pfister. Mle-star: Machine learning engineering agent via search and targeted refinement. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[11] 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.

[12] Edan Toledo, Karen Hambardzumyan, Martin Josifoski, RISHI HAZRA, Nicolas Baldwin, Alexis Audran-Reiss, Michael Kuchnik, Despoina Magka, Minqi Jiang, Alisia Maria Lupidi, et al. Ai research agents for machine learning: Search, exploration, and generalization in mle-bench. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[13] Xu Yang, Xiao Yang, Shikai Fang, Bowen Xian, Yuante Li, Jian Wang, Minrui Xu, Haoran Pan, Xinpeng Hong, Weiqing Liu, et al. R&d-agent: Automating data-driven ai solution building through llm-powered automated research, development, and evolution. *arXiv e-prints*, pages arXiv–2505, 2025.

[14] Annan Li, Chufan Wu, Zengle Ge, Yee Hin Chong, Zhinan Hou, Lizhe Cao, Cheng Ju, Jianmin Wu, Huaiming Li, Haobo Zhang, et al. The fm agent. *arXiv preprint arXiv:2510.26144*, 2025.

[15] Jiefeng Chen, Bhavana Dalvi Mishra, Jaehyun Nam, Rui Meng, Tomas Pfister, and Jinsung Yoon. Mars: Modular agent with reflective search for automated ai research. *arXiv preprint arXiv:2602.02660*, 2026.

[16] Shiyang Feng, Runmin Ma, Xiangchao Yan, Yue Fan, Yusong Hu, Songtao Huang, Shuaiyu Zhang, Zongsheng Cao, Tianshuo Peng, Jiakang Yuan, Zijie Guo, Zhijie Zhong, Shangheng Du, Weida Wang, Jinxin Shi, Yuhao Zhou, Xiaohan He, Zhiyin Yu, Fangchen Yu, Bihao Zhan, Qihao Zheng, Jiamin Wu, Mianxin Liu, Chi Zhang, Shaowei Hou, Shuya Li, Yankai Jiang, Wenjie Lou, Lilong Wang, Zifu Wang, Jiong Wang, Wanghan Xu, Yue Deng, Dongrui Liu, Yiheng Wang, Wenlong Zhang, Fenghua Ling, Shufei Zhang, Xiaosong Wang, Shuangjia Zheng, Xun Huang, Siqi Sun, Shuyue Hu, Peng Ye, Chunfeng Song, Bin Wang, Conghui He, Yihao Liu, Xin Li, Qibin Hou, Tao Chen, Xiangyu Yue, Bin Wang, Liang He, Dahua Lin, Bowen Zhou, Bo Zhang, and Lei Bai. Internagent-1.5: A unified agentic framework for long-horizon autonomous scientific discovery. *arXiv preprint arXiv:2602.08990*, 2026.

[17] Alireza Nadafian, Alireza Mohammadshahi, and Majid Yazdani. Kapso: A knowledge-grounded framework for autonomous program synthesis and optimization. *arXiv preprint arXiv:2601.21526*, 2026.

[18] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

[19] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642, 2024.

[20] Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. Automanual: Constructing instruction manuals by llm agents via interactive environmental learning. *Advances in Neural Information Processing Systems*, 37:589–631, 2024.

[21] Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7080–7106, 2026.

[22] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL `https://openreview.net/forum?id=ehfRiF0R3a`.

[23] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Forty-second International Conference on Machine Learning*, 2025. URL `https://openreview.net/forum?id=NTAhi2JEEE`.

[24] Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. *arXiv preprint arXiv:2508.06433*, 2025.

[25] Ao Qu, Han Zheng, Zijian Zhou, Yihao Yan, Yihong Tang, Shao Yong Ong, Fenglu Hong, Kaichen Zhou, Chonghe Jiang, Minwei Kong, et al. Coral: Towards autonomous multi-agent evolution for open-ended discovery. *arXiv preprint arXiv:2604.01658*, 2026.

[26] Boyuan Zheng, Michael Y Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, et al. Skillweaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025.

[27] Xiangru Tang, Tianrui Qin, Tianhao Peng, Ziyang Zhou, Daniel Shao, Tingting Du, Xinming Wei, Peng Xia, Fang Wu, He Zhu, et al. Agent kb: Leveraging cross-domain experience for agentic problem solving. *arXiv preprint arXiv:2507.06229*, 2025.

[28] Yunzhong Xiao, Yangmin Li, Hewei Wang, Yunlong Tang, and Zora Zhiruo Wang. Toolmem: Enhancing multimodal agents with learnable tool capability memory. *arXiv preprint arXiv:2510.06664*, 2025.

[29] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

[30] Haorui Wang, Marta Skreta, Cher Tian Ser, Wenhao Gao, Lingkai Kong, Felix Strieth-Kalthoff, Chenru Duan, Yuchen Zhuang, Yue Yu, Yanqiao Zhu, et al. Efficient evolutionary search over chemical space with large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

[31] Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K Reddy. Llm-sr: Scientific equation discovery via programming with large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

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

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

[34] Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. Augmenting large language models with chemistry tools. *Nature machine intelligence*, 6(5):525–535, 2024.

[35] Yuanqi Du, Botao Yu, Tianyu Liu, Tony Shen, Junwu Chen, Jan G Rittig, Kunyang Sun, Yikun Zhang, Zhangde Song, Bo Zhou, et al. Accelerating scientific discovery with autonomous goal-evolving agents. *arXiv preprint arXiv:2512.21782*, 2025.

[36] Vlastimil Martinek, Andrea Gariboldi, Dimosthenis Tzimotoudis, Mark Galea, Elissavet Zacharopoulou, Aitor Alberdi Escudero, Edward Blake, David Čechák, Luke Cassar, Alessandro Balestrucci, et al. Agentomics: An agentic system that autonomously develops novel state-of-the-art solutions for biomedical machine learning tasks. *bioRxiv*, pages 2026–01, 2026.

[37] Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*, 2025.

[38] Anthropic. Claude code documentation. `https://code.claude.com/docs/en/overview`, 2025. Accessed: 2026-05-06.

[39] Jeremy R Ash, Cas Wognum, Raquel Rodríguez-Pérez, Matteo Aldeghi, Alan C Cheng, Djork-Arné Clevert, Ola Engkvist, Cheng Fang, Daniel J Price, Jacqueline M Hughes-Oliver, et al. Practically significant method comparison protocols for machine learning in small molecule drug discovery. *Journal of chemical information and modeling*, 65(18):9398–9411, 2025.

[40] Cheng Fang, Ye Wang, Richard Grater, Sudarshan Kapadnis, Cheryl Black, Patrick Trapa, and Simone Sciabola. Prospective validation of machine learning algorithms for absorption, distribution, metabolism, and excretion prediction: An industrial perspective. *Journal of Chemical Information and Modeling*, 63(11):3263–3274, 2023.

[41] David H Drewry, Carrow I Wells, David M Andrews, Richard Angell, Hassan Al-Ali, Alison D Axtman, Stephen J Capuzzi, Jonathan M Elkins, Peter Ettmayer, Mathias Frederiksen, et al. Progress towards a public chemogenomic set for protein kinases and a call for contributions. *PloS one*, 12(8):e0181585, 2017.

[42] William Green, Jackson Burns, Akshat Shirish Zalte, Charlles Abreu, Jochen Sieg, Christian Feldmann, and Miriam Mathea. Deep learning foundation models from classical molecular descriptors. 2026.

[43] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

Appendix for DrugSAGE

A Benchmark Dataset Summary **16**

B Dataset Partition **17**

C Per-Baseline Protocols and Prompts **18**

C.1 Autoresearch 18

C.2 ShinkaEvolve 19

C.3 MLEvolve 19

C.4 AIRA-Dojo 20

C.5 Claude Code 20

C.6 STELLA 20

C.7 Agentomics 21

C.8 Per-Baseline Budget Mapping 22

D Per-Task Scores for the TDC-ADMET Benchmark **22**

E Per-Task Scores on the Polaris Benchmark **24**

F Per-Task Budget Trajectories on the 17 Held-Out Tasks **24**

G Per-Task Zero-Test-Time Search Results **24**

G.1 TDC-ADMET Benchmark Held-Out Tasks 24

G.2 Polaris Benchmark Held-Out Tasks 26

G.3 Zero-Test-Time Routing Case Study 26

H Explore Agent Workflow Details **26**

I Formal Definition of Cross-Task Transfer Scores **27**

I.1 Per-Task Standardized Score 28

I.2 Task-Similarity Weights 28

I.3 Formal Definition of Transfer Scores 28

I.4 Boundedness Guarantee 29

J Proof of adding the cross-task causal-factor term will not change the upperbound of speed **29**

J.1 Problem Setup 29

J.1.1 Assumptions 29

J.2 Family-Level Selection 29

J.2.1 Setup and Regret Definition 29
J.2.2 Lemma: Causal Bonus Is Order-Compatible with UCB Exploration 30
J.2.3 Theorem: Family-Level Regret Bound 30

K Additional Analyses of Cross-Task Experience **32**
K.1 Matched Starting Models and Search-System Differences 32
K.2 Skill Library and Full Cost Accounting 32
K.3 Memory Components and Search Controls 33
K.4 Memory Growth and Composition 33
K.5 Leave-Category-Out Routing and Negative Transfer 33
K.6 Transfer from ADMET to Kinase Activity 34
K.7 Extension to Generative Tasks 34

A Benchmark Dataset Summary

Table 3 provides a complete description of all benchmark datasets used in our evaluation, covering 22 ADMET datasets from the Therapeutics Data Commons (TDC) [7] and 11 held-out test datasets from the Polaris Hub [39].

Table 3: All benchmark datasets. Size is the total number of items in the training and testing sets.

Dataset	Description	Size	Metric
tdcommons-caco2	Predict intestinal permeability in Caco-2 cell assay.	906	MAE
tdcommons-hia	Predict human intestinal absorption.	578	AUROC
tdcommons-pgp	Classify P-glycoprotein (Pgp) inhibition for absorption risk assessment.	1,212	AUROC
tdcommons-bioavailability	Predict oral bioavailability.	640	AUROC
tdcommons-lipophilicity	Predict lipophilicity (logP).	4,200	MAE
tdcommons-solubility	Predict aqueous solubility.	9,982	MAE
tdcommons-bbb-martins	Classify blood–brain barrier penetration.	1,975	AUROC
tdcommons-ppbr	Predict human plasma protein binding rate.	1,797	MAE
tdcommons-vdss	Predict volume of distribution at steady state (VDss).	1,130	Spearman
tdcommons-cyp2c9-substrate	Classify CYP2C9 enzyme substrates for drug metabolism.	666	AUPRC
tdcommons-cyp2d6-substrate	Classify CYP2D6 enzyme substrates for drug–drug interaction risk.	664	AUPRC
tdcommons-cyp3a4-substrate	Classify CYP3A4 enzyme substrates for drug metabolism.	667	AUROC
tdcommons-cyp2c9-inhibition	Classify CYP2C9 enzyme inhibitors for drug–drug interaction risk.	12,092	AUPRC
tdcommons-cyp2d6-inhibition	Classify CYP2D6 enzyme inhibitors for drug–drug interaction risk.	13,130	AUPRC
tdcommons-cyp3a4-inhibition	Classify CYP3A4 enzyme inhibitors for drug–drug interaction risk.	12,328	AUPRC
tdcommons-half-life	Predict drug half-life duration.	667	Spearman
tdcommons-clearance-hepatocyte	Predict hepatocyte intrinsic drug clearance.	1,020	Spearman
tdcommons-clearance-microsome	Predict microsomal intrinsic drug clearance.	1,102	Spearman
tdcommons-herg	Classify hERG potassium channel blockers to assess cardiotoxicity risk.	648	AUROC
tdcommons-dili	Classify drug-induced liver injury (DILI) risk.	475	AUROC
tdcommons-ames	Classify mutagenicity via the Ames bacterial reverse mutation assay.	7,255	AUROC
tdcommons-ld50	Predict acute oral toxicity (LD50).	7,385	MAE
polaris-adme-fang-hclint	Predict human liver microsomal intrinsic clearance.	2,806	Pearson
polaris-adme-fang-rclint	Predict rat liver microsomal intrinsic clearance.	2,779	Pearson
polaris-adme-fang-perm	Predict MDR1-MDCK efflux ratio (permeability).	2,403	Pearson
polaris-adme-fang-solu	Predict compound solubility using standardised ADME protocols.	1,978	Pearson
polaris-adme-fang-hppb	Predict human plasma protein binding.	160	Pearson

Continued on next page...

Dataset	Description	Size	Metric
polaris-adme-fang-rppb	Predict rat plasma protein binding.	135	Pearson
polaris-pkis2-egfr-wt-reg	Predict EGFR wild-type kinase inhibition (% inhibition).	640	MSE
polaris-pkis2-ret-wt-cls	Classify RET wild-type kinase inhibitors for cancer target engagement.	640	AUPRC
polaris-pkis2-ret-wt-reg	Predict RET wild-type kinase inhibition (% inhibition).	640	MSE
polaris-pkis2-kit-wt-cls	Classify KIT wild-type kinase inhibitors for cancer target engagement.	640	AUPRC
polaris-pkis2-kit-wt-reg	Predict KIT wild-type kinase inhibition (% inhibition).	640	MSE

B Dataset Partition

Table 4 lists the experience pool and target set used in the cross-task amortization experiments (Section 4.2). The split is determined by training-set size: the 6 largest datasets ($\geq 5{,}000$ training samples; the next-smallest dataset is Lipophilicity at $4{,}200$) constitute the target set; the remaining 16 form the experience pool.

Table 4: Partition of the 33 benchmark datasets into experience pool (16) and target set (17). The target set contains the 6 largest TDC ADMET datasets and 11 Polaris datasets.

Role	Dataset	Size	Task type	Metric
Experience pool	DILI	475	Binary cls.	AUROC
	HIA	578	Binary cls.	AUROC
	Bioavailability	640	Binary cls.	AUROC
	hERG	648	Binary cls.	AUROC
	CYP2D6 Substrate	664	Binary cls.	AUPRC
	CYP2C9 Substrate	666	Binary cls.	AUPRC
	CYP3A4 Substrate	667	Binary cls.	AUROC
	Half Life	667	Regression	Spearman
	Caco-2	906	Regression	MAE
	Clearance (hepatocyte)	1,020	Regression	Spearman
	Clearance (microsome)	1,102	Regression	Spearman
	VDss	1,130	Regression	Spearman
	Pgp	1,212	Binary cls.	AUROC
	PPBR	1,797	Regression	MAE
	BBB	1,975	Binary cls.	AUROC
	Lipophilicity	4,200	Regression	MAE
Target set	Ames	7,255	Binary cls.	AUROC
	LD50	7,385	Regression	MAE
	Solubility (AqSolDB)	9,982	Regression	MAE
	CYP2C9 Inhibition	12,092	Binary cls.	AUPRC
	CYP3A4 Inhibition	12,328	Binary cls.	AUPRC
	CYP2D6 Inhibition	13,130	Binary cls.	AUPRC
	Polaris HClint	2,806	Regression	Pearson
	Polaris RClint	2,779	Regression	Pearson
	Polaris Perm	2,403	Regression	Pearson
	Polaris Solu	1,978	Regression	Pearson
	Polaris HPPB	160	Regression	Pearson
	Polaris RPPB	135	Regression	Pearson
	Polaris EGFR WT Reg	640	Regression	MSE
	Polaris RET WT Cls	640	Binary cls.	AUPRC
	Polaris RET WT Reg	640	Regression	MSE

Continued on next page...

Role	Dataset	Size	Task type	Metric
	Polaris KIT WT Cls	640	Binary cls.	AUPRC
	Polaris KIT WT Reg	640	Regression	MSE

C Per-Baseline Protocols and Prompts

This section documents the run protocol, prompt structure, and non-default hyperparameters for every baseline evaluated in Section 4. Except for Biomni, all models use the 5-seed split of train and validation sets for training and the fixed `prepare.py` evaluation pipeline.

Computational setup. All agents, including DRUGSAGE and all baselines, use Claude Sonnet 4.6 as the backbone LLM via the Anthropic API unless the baseline's original paper specifies a different model (see per-baseline notes below). Each experiment is run on a single NVIDIA L40S GPU, wall-clock budget per agent per task is capped at 24 hours.

Shared task prompt (from-scratch baselines). All baselines except Autoresearch and ShinkaEvolve receive the same per-dataset task description. This description specifies (i) the task name and task type (classification or regression); (ii) the primary metric and its optimization direction; (iii) the train/validation pool size and held-out test size; (iv) the `prepare.py` API contract (`load_data`, `load_seed_split`, `evaluate`, `save_predictions`, `SEEDS`); (v) the required output format `[result] METRIC = MEAN +/- STD`; and (vi) constraints forbidding modification of `prepare.py`, test-label leakage, or package installation outside the pre-built conda environment. For Polaris tasks the description is identical in structure, substituting `prepare_polaris.py` and the Polaris-specific metric. The full template is available in the released codebase.

C.1 Autoresearch

Autoresearch [4] is an automatic ML pipeline optimization agent. For each TDC dataset it starts from the top-3 open-sourced leaderboard models, and for each Polaris dataset it starts from Chemelon [42].

Prompt structure. Each anchor model directory contains an auto-generated `CLAUDE.md` system prompt (one per dataset/model pair); placeholders below are filled per task. The pipeline automatically records outcomes in `results.tsv`.

Autoresearch / Claude Code system prompt (`CLAUDE.md`, abbreviated)

```
You are an autonomous ML researcher optimizing a drug
property prediction model.

Task: Improve {METRIC} ({DIRECTION}) on the {DATASET}
benchmark using the {MODEL} model.
Train/val: {N_TRAIN_VAL} | Test: {N_TEST}
Conda env: {ENV} | Experiment budget: 20

Constraints:
- train.py is the ONLY file you modify.
- prepare.py is FIXED -- DO NOT modify.
- 5-seed evaluation protocol (seeds [1,2,3,4,5]).
- No test-label leakage; no new package installation.

Experiment loop (repeat until budget exhausted):
1. Check results.tsv for experiment history.
2. Read train.py; plan one focused change.
3. Edit train.py.
4. Run: python pipeline.py run-exp {DATASET}/{MODEL}
     --desc "..." --gpu 0
5. Keep if val metric improved; auto-revert otherwise.

Strategy guide:
  Quick wins: tune LR, batch size, regularization.
  Medium effort: change model family, add CV.
```

```
  High effort: ensemble methods, custom featurization.

NEVER STOP -- continue autonomously until budget exhausted.
```

C.2 ShinkaEvolve

ShinkaEvolve [37] is an evolutionary algorithm optimization agent. It maintains a population of candidate programs, samples parents from the archive, and mutates them via LLM-generated code blocks. The starting point of ShinkaEvolve is the same as Autoresearch.

Prompt structure. The task-level system message is set via `task_sys_msg` in `shinka_config.yaml`:

ShinkaEvolve task system message (`shinka_config.yaml`)

```
You are an expert in ADMET property prediction and
machine learning for drug discovery.

TASK: {TASK_NAME}
This is a {TASK_TYPE} task. Metric: {METRIC} ({DIR}).
Your goal is to {maximize/minimize} {METRIC}.
Dataset: {N} train+val, {M} test molecules.

RULES:
1. Use prepare.py: load_data, load_seed_split, evaluate,
   save_predictions, save_summary, SEEDS
2. Iterate over ALL 5 seeds and train/evaluate each
3. Call save_predictions for each seed
4. Call save_summary() at the end
5. DATASET, METRIC, OUT variables are FIXED
6. Ensure all predictions are finite (no NaN/inf)

STRATEGY:
- Tune hyperparameters, feature engineering, architectures
- Consider ensemble methods, stacking, or blending
- Change ML library if beneficial
- Classification: probabilities; Regression: continuous
```

Mutation prompts are constructed by a `PromptSampler`: for *diff* patches (70% probability), the prompt appends SEARCH/REPLACE format instructions requesting a unified diff; for *full* patches (30%), it requests a complete rewrite. Both include the parent program's code, its performance metrics, and the code and metrics of archive inspiration programs sorted in ascending-score order.

C.3 MLEvolve

MLEvolve [16] is an MLE agent with tree-search that explores a solution tree via draft–debug–improve cycles. It generates multiple initial drafts, executes them, and iteratively expands the tree by selecting promising nodes, proposing code improvements, and executing them.

Prompt structure. Each dataset has a `description.md` providing the task prompt. Internally, MLEvolve has specialized sub-agents for drafting, debugging, improving, code review, data-leakage checking, result parsing, and multi-branch fusion, each with its own prompt template.

MLEvolve task description (`description.md`, **abbreviated**)

```
# {TASK_NAME}

Predict the {TASK_NAME} from feature provided in {DATA}.
- Task type: {TASK_TYPE}
- Metric: {METRIC} ({DIRECTION})
- Train/val pool: {N} molecules | Test: {M} molecules

prepare.py API:
  load_data(name) -> (train_val_df, test_df, meta)
```

```
  load_seed_split(name, seed) -> (train_df, val_df)
  evaluate(y_true, y_pred, metric) -> float
  save_predictions(Path(out), seed, test, y_pred, metric)

Required output (per seed):
  [seed N] val_{metric}=VALUE
Final output:
  [result] METRIC = MEAN +/- STD

Constraints: do not modify prepare.py; no test-label
leakage; all 5 seeds mandatory; classification outputs
must be positive-class probabilities.
```

C.4 AIRA-Dojo

AIRA-Dojo [12] is an ML research agent that iterates through draft, improve, debug, analyze operators via MCTS, each backed by a prompted LLM call that generates a self-contained Python script.

Prompt structure. Each operator has a Jinja2-templated system prompt defined in YAML; the *draft* operator is shown below. The *improve* prompt is similar but also injects the previous solution's code and execution output. The *debug* prompt focuses on fixing a buggy script. A shared `instructions.txt` prepends the benchmark contract (use `prepare.py`, 5 seeds, output format) to all operator prompts. Draft complexity is varied across rounds (`simple`, `normal`, `complex`).

AIRA-Dojo draft operator (Jinja2 template, abbreviated)

```
You are an expert machine learning researcher for
molecular property prediction. Carefully study the ADMET
task description, the fixed five-seed evaluation protocol,
the available data overview, and the available packages.
Propose exactly one promising initial approach and
implement it as a single self-contained Python script.

# TASK DESCRIPTION
{{task_desc}}

# DATA OVERVIEW
{{data_overview}}

# CONSTRAINTS
- Code must complete within {{execution_timeout}}.
- Use ./data/prepare.py; fixed seed splits 1..5.
- Print: [result] METRIC = MEAN +/- STD
- Do not perform exploratory data analysis.

# RESPONSE FORMAT
Provide "Idea to implement", then one Python code block
that imports prepare.py, trains on all 5 seeds, and
prints the final [result] line.
```

C.5 Claude Code

Claude Code [38] is Anthropic's agentic coding assistant, used as a general-purpose baseline. The `CLAUDE.md` system prompt is identical to Autoresearch (see the prompt box in §C.1), except the setting of anchor methods.

C.6 STELLA

STELLA [3] is a self-evolving LLM agent for biomedical research. We run STELLA in its default self-evolve mode with the shared task prompt.

STELLA prompt (abbreviated)

```
You can only read files under the current path. The goal: **improve the benchmark metric
    ** beyond the leaderboard baseline by having a better algorithm.

## Setup

1. **Create the task files for each task**:
  - ‘train.py‘ : the file you modify. Featurization, model, hyperparameters, training
    loop.
  - ‘prepare.py‘ (two levels up) : fixed data loading, evaluation, output.
2. **Review experiment history**: Check ‘results.tsv‘ for what has already been tried.

## Constraints

**What you CAN modify:**
- ‘train.py‘ : everything is fair game: model hyperparameters, featurization, feature
    engineering, model architecture, ensemble strategies, preprocessing, training
    procedure.

**What you CANNOT modify:**
- ‘prepare.py‘ : read-only. Contains fixed evaluation (‘evaluate()‘), data loading (‘
    load_data()‘), prediction saving, and summary generation.
- The 5-seed evaluation protocol : all models run seeds [1,2,3,4,5] and report mean and
    std.
- The test set : no data leakage. Train only on ‘train_val‘ data.

**What you CAN add:**
- Use any algorithms you prefer.
- Create a new conda environment.

**What you CANNOT do:**
- Modify the evaluation metric or data splits.
- Access test labels during training.

## Metrics

Each benchmark has one primary metric (specified in ‘train.py‘ as ‘METRIC‘).

## The Experiment Loop

Default: **20 experiments** per task.

### LOOP (up to budget):

1. **Plan**: Look at experiment history in ‘results.tsv‘, and make changes to ‘train.py‘.
2. **Modify ‘train.py‘**: Make your change. Keep it focused one idea per experiment.
3. **Snapshot**: The pipeline automatically saves a copy of ‘train.py‘.
4. **Run**: Run experiments.
5. **Check results**: Read the output.
6. **Record**: Log results.
7. **Decide**: Take results or not.
8. **Repeat** until budget exhausted.
## Output Format
The ‘save_summary()‘ call writes ‘results/summary.json‘ with full details.

## NEVER STOP

You need to save the best results and standard deviation in folder ‘./{task}_result/‘.
```

C.7 Agentomics

Agentomics [36] is a multi-step ML agent that follows a fixed step sequence per iteration: iteration planning → data exploration → data split → data representation → model architecture → model training → model inference → prediction exploration → validation evaluation.

Prompt structure. The system prompt is assembled at runtime by `prompt_builder.py`:

Agentomics system prompt (abbreviated)

```
Your goal is to create a robust machine learning model
that will generalize to new unseen data.

Multi-step architecture (per iteration):
  - Iteration Planning
  - Data Exploration
  - Data Split
  - Data Representation
  - Model Architecture
  - Model Training
  - Model Inference
  - Prediction Exploration
  - Validation Evaluation

Resources: {AVAILABLE_RESOURCES}
Conda env: {CONDA_PATH}
Dataset: {DATASET_PATH}

{DATASET_DESCRIPTION}

ADMET benchmark protocol:
- No anchor or starting solution is provided.
- Use numeric_label as the target column.
- Model selection uses validation {METRIC}.
- Runtime handles repeated training/evaluation across
  5 seed splits.
```

The per-iteration user prompt provides the instruction *"Develop a machine learning model that generalizes well to new unseen data."*, workspace rules (writable directory, read-only previous iterations), and references to archived iteration outputs.

C.8 Per-Baseline Budget Mapping

Figure 3 plots performance against experiment budget B, but different baselines count their iterations differently. Table 5 shows how we convert each baseline's native unit to B: one unit of B corresponds to one complete train-and-evaluate cycle on the target dataset, so the comparison is approximately compute-equalized across systems.

Table 5: Mapping from each baseline's native iteration unit to budget B.

System	One unit of B corresponds to
Autoresearch	one revise-and-evaluate cycle
ShinkaEvolve	one mutation round
MLEvolve	one tree-search expansion + evaluation
AIRA-Dojo	one search step
Claude Code	one user-agent iteration
STELLA	one self-evolve round
Agentomics	one full ML iteration
Ours	one full training run on the target task

D Per-Task Scores for the TDC-ADMET Benchmark

Table 6 reports per-task results for all 22 TDC-ADMET benchmark datasets. Biomni scores are provided by the Biomni team from their beta platform with no standard deviation reported. All other systems were run by us under the unified evaluation protocol described in Appendix C.

Table 6: Per-task scores on the TDC-ADMET benchmark, reported as mean ± std over 5 seeds. Top 2 results are highlighted with **bold text** and underlined text, respectively. Biomni scores are from the official beta release (single point, no std). (↑) / (↓) denotes a larger / smaller number is better.

Dataset	Metric	Leaderboard	Autoresearch	ShinkaEvolve	Claude Code	MLEvolve	Aira-Dojo	Biomni	STELLA	Agentomics	DRUGSAGE (Ours)	DRUGSAGE (anchor)
Absorption												
Bioavailability	AUROC ↑	0.748 ± 0.033	0.7298 ± 0.0195	0.7287 ± 0.0187	0.7205 ± 0.0084	0.7178 ± 0.0033	0.7152 ± 0.0206	0.5920	0.7316 ± 0.0317	0.7239 ± 0.0056	0.7731 ± 0.0216	$\mathbf{0.7739 \pm 0.0151}$
Caco-2	MAE ↓	0.256 ± 0.006	0.2857 ± 0.0086	0.2700 ± 0.0057	0.2613 ± 0.0025	0.2965 ± 0.0072	0.2783 ± 0.0050	0.5070	0.2743 ± 0.0064	0.2905 ± 0.0114	$\mathbf{0.2466 \pm 0.0030}$	0.2619 ± 0.0071
HIA	AUROC ↑	0.993 ± 0.005	0.9781 ± 0.0027	0.9873 ± 0.0026	0.9630 ± 0.0156	0.9636 ± 0.0247	0.9861 ± 0.0061	0.9740	0.9785 ± 0.0035	0.9725 ± 0.0071	0.9930 ± 0.0008	$\mathbf{0.9936 \pm 0.0014}$
Lipophilicity	MAE ↓	0.456 ± 0.008	0.4088 ± 0.0053	0.4177 ± 0.0056	0.5181 ± 0.0031	0.5006 ± 0.0101	0.5280 ± 0.0102	0.7910	0.5466 ± 0.0039	0.5663 ± 0.0066	$\mathbf{0.3753 \pm 0.0034}$	0.4009 ± 0.0050
Pgp	AUROC ↑	0.938 ± 0.002	0.9241 ± 0.0075	0.9345 ± 0.0048	0.9287 ± 0.0013	0.9304 ± 0.0018	0.8978 ± 0.0082	0.8940	0.9031 ± 0.0138	0.9118 ± 0.0058	$\mathbf{0.9519 \pm 0.0026}$	0.9404 ± 0.0027
Solubility	MAE ↓	0.741 ± 0.013	0.7446 ± 0.0107	0.7886 ± 0.0219	0.7854 ± 0.0033	0.7465 ± 0.0174	0.7814 ± 0.0125	1.1450	0.8421 ± 0.0180	0.7668 ± 0.0131	$\mathbf{0.7215 \pm 0.0109}$	0.7435 ± 0.0090
Distribution												
BBB Martins	AUROC ↑	0.924 ± 0.003	0.9256 ± 0.0031	0.9247 ± 0.0035	0.8988 ± 0.0016	0.9013 ± 0.0098	0.9021 ± 0.0045	0.8460	0.9029 ± 0.0138	0.9108 ± 0.0077	$\mathbf{0.9377 \pm 0.0015}$	0.9228 ± 0.0038
PPBR	MAE ↓	7.44 ± 0.02	7.34 ± 0.27	7.23 ± 0.15	7.27 ± 0.05	7.71 ± 0.25	7.57 ± 0.14	9.87	7.42 ± 0.11	7.58 ± 0.12	$\mathbf{7.11 \pm 0.19}$	7.19 ± 0.12
VDss	Spearman ↑	0.713 ± 0.007	0.7015 ± 0.0089	0.7084 ± 0.0062	0.7211 ± 0.0047	0.7196 ± 0.0104	0.6636 ± 0.0230	0.3880	0.3247 ± 0.0331	0.5177 ± 0.0367	$\mathbf{0.7392 \pm 0.0029}$	0.7260 ± 0.0052
Metabolism												
CYP2C9 Sub	AUPRC ↑	0.474 ± 0.025	0.4635 ± 0.0247	0.4361 ± 0.0059	0.4292 ± 0.0196	0.3967 ± 0.0413	0.3899 ± 0.0152	0.3870	0.3739 ± 0.0181	0.3836 ± 0.0198	$\mathbf{0.5237 \pm 0.0202}$	0.4535 ± 0.0468
CYP2C9 Inh	AUPRC ↑	0.859 ± 0.001	0.8800 ± 0.0009	0.7999 ± 0.0027	0.7895 ± 0.0018	0.7918 ± 0.0031	0.7860 ± 0.0067	0.6320	0.7043 ± 0.0057	0.7871 ± 0.0066	0.8605 ± 0.0019	$\mathbf{0.8828 \pm 0.0011}$
CYP2D6 Sub	AUPRC ↑	0.736 ± 0.024	0.7041 ± 0.0130	0.6823 ± 0.0145	0.7407 ± 0.0093	0.6934 ± 0.0064	0.6708 ± 0.0254	0.5740	0.6895 ± 0.0068	0.6895 ± 0.0267	0.7542 ± 0.0152	$\mathbf{0.8237 \pm 0.0154}$
CYP2D6 Inh	AUPRC ↑	0.790 ± 0.001	0.7286 ± 0.0037	0.7324 ± 0.0016	0.6968 ± 0.0013	0.7177 ± 0.0049	0.7133 ± 0.0033	0.5620	0.7005	0.7116 ± 0.0051	0.7237 ± 0.0022	$\mathbf{0.8232 \pm 0.0023}$
CYP3A4 Sub	AUROC ↑	0.667 ± 0.019	0.6394 ± 0.0262	0.6472 ± 0.0261	0.6553 ± 0.0086	0.6386 ± 0.0150	0.6571 ± 0.0155	0.6730	0.6574 ± 0.0040	0.6411 ± 0.0195	$\mathbf{0.6757 \pm 0.0150}$	0.6671 ± 0.0086
CYP3A4 Inh	AUPRC ↑	0.916 ± 0.000	$\mathbf{0.9247 \pm 0.0014}$	0.9176 ± 0.0003	0.8838 ± 0.0008	0.8901 ± 0.0020	0.8829 ± 0.0015	0.7470	0.8642 ± 0.0024	0.8810 ± 0.0029	0.9143 ± 0.0013	0.8878 ± 0.0012
Half-Life	Spearman ↑	0.576 ± 0.025	0.5473 ± 0.0268	0.5835 ± 0.0188	0.5463 ± 0.0169	0.4985 ± 0.0236	0.5545 ± 0.0161	0.1500	0.5370 ± 0.0222	0.2771 ± 0.0553	0.5734 ± 0.0314	$\mathbf{0.6054 \pm 0.0197}$
Excretion												
CL Hepatocyte	Spearman ↑	$\mathbf{0.536 \pm 0.020}$	0.4517 ± 0.0087	0.4376 ± 0.0273	0.4977 ± 0.0050	0.3689 ± 0.0329	0.4382 ± 0.0115	0.3020	0.4608 ± 0.0048	0.4489 ± 0.0170	0.4821 ± 0.0119	0.4787 ± 0.0144
CL Microsome	Spearman ↑	0.630 ± 0.010	$\mathbf{0.6399 \pm 0.0113}$	0.5953 ± 0.0143	0.5753 ± 0.0080	0.5714 ± 0.0126	0.5656 ± 0.0229	0.5040	0.5421 ± 0.0272	0.5554 ± 0.0167	0.6327 ± 0.0078	0.6100 ± 0.0212
Toxicity												
Ames	AUROC ↑	0.871 ± 0.002	0.8718 ± 0.0019	0.8723 ± 0.0021	0.8688 ± 0.0047	0.8714 ± 0.0037	0.8691 ± 0.0014	0.7260	0.8643 ± 0.0008	0.8640 ± 0.0037	0.8776 ± 0.0126	$\mathbf{0.8794 \pm 0.0026}$
DILI	AUROC ↑	$\mathbf{0.956 \pm 0.006}$	0.9292 ± 0.0025	0.9306 ± 0.0064	0.9059 ± 0.0081	0.8965 ± 0.0192	0.9180 ± 0.0027	0.9050	0.9151 ± 0.0038	0.9263 ± 0.0180	0.9369 ± 0.0100	0.9315 ± 0.0014
hERG	AUROC ↑	0.880 ± 0.002	0.8447 ± 0.0093	0.8526 ± 0.0053	0.8436 ± 0.0043	0.8411 ± 0.0117	0.8480 ± 0.0120	0.7340	$\mathbf{0.8941 \pm 0.0212}$	0.8350 ± 0.0126	0.8862 ± 0.0123	0.8800 ± 0.0082
LD50	MAE ↓	0.552 ± 0.009	0.5813 ± 0.0072	0.5898 ± 0.0110	0.5994 ± 0.0015	0.6149 ± 0.0162	0.6124 ± 0.0088	0.7340	0.6190 ± 0.0081	0.6181 ± 0.0083	$\mathbf{0.5019 \pm 0.0131}$	0.5682 ± 0.0102

E Per-Task Scores on the Polaris Benchmark

Table 7 reports per-task results on the 11 Polaris hold-out tasks introduced in Section 4.2. Among agents with budgeted search ($B = 20$), DRUGSAGE ranks first on eight of eleven tasks. DRUGSAGE-Zero achieves the top score overall on `adme-fang-hclint` and second best results on six of eleven tasks, illustrating that DRUGSAGE-Zero is able to maintain competitive performance at zero experiment budget.

F Per-Task Budget Trajectories on the 17 Held-Out Tasks

Figure 3 in the main paper averages best-so-far performance over tasks within each evaluation set. Figure 5 breaks this down and shows the best-so-far score on each individual held-out task as a function of experiment budget B.

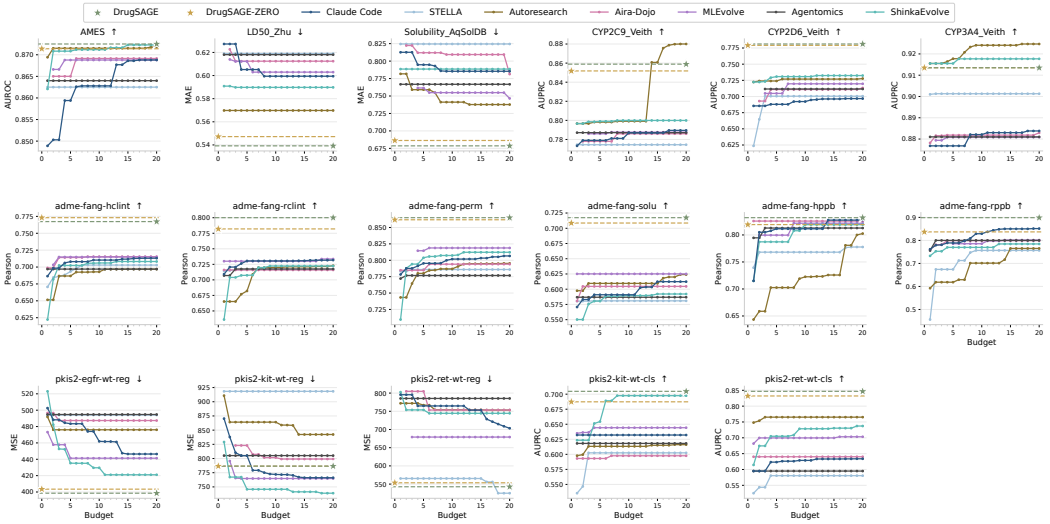

Figure 5: Per-task budget trajectories on all 17 held-out tasks: 6 TDC-ADMET tasks (top one row) and 11 Polaris tasks (bottom two rows).

G Per-Task Zero-Test-Time Search Results

This section explains the routing decisions made by DRUGSAGE-Zero on the 17 held-out tasks. For each task we report which task in $\mathcal{Z}$ the router matched it to, the solution it transferred, and the resulting zero-shot score (mean $\pm$ std over 5 seeds).

G.1 TDC-ADMET Benchmark Held-Out Tasks

Table 8 reports the analog task memory from $\mathcal{Z}$ that the router selected, the transferred solution, and its corresponding performance for each of the 6 ADMET target tasks. Ames matches to BBB_Martins, both of which are AUROC binary classification tasks, while the biological domains differ (mutagenicity vs. blood–brain barrier penetration). LD50 and Solubility both match to Lipophilicity_AstraZeneca, the largest MAE regression task in $\mathcal{Z}$, and inherit the same three-model ensemble. The three CYP inhibition tasks (CYP2C9, CYP2D6, CYP3A4 Veith) all match to CYP2C9_Substrate, despite the task-type difference (substrate vs. inhibition classification); the router generalizes across this distinction because the metric (AUPRC) and molecular domain (CYP enzyme) align. Across all three patterns, the matched task's best solution transfers without modification, indicating that metric and task-type alignment in $\mathcal{Z}$ may be sufficient for effective zero-shot transfer even when the biological context does not fully overlap.

Table 7: Per-task scores on the 11 Polaris hold-out datasets, reported as mean ± std over 5 seeds. Top 2 results are highlighted with **bold text** and underlined text, respectively. (↑) / (↓) denotes a larger / smaller number is better.

			Budgeted ($B = 20$)								
Dataset	Metric	Leaderboard	Autoresearch	ShinkaEvolve	Claude Code	MLEvolve	Aira-Dojo	STELLA	Agentomics	DRUGSAGE (Ours)	DRUGSAGE-Zero
ADME-Fang regression											
adme-fang-hclint	Pearson ↑	—	0.6949 ± 0.0070	0.7082 ± 0.0067	0.7131 ± 0.0032	0.7148 ± 0.0057	0.7015 ± 0.0057	0.6706 ± 0.0038	0.6970 ± 0.0063	*0.7673 ± 0.0027*	**0.7731 ± 0.0014**
adme-fang-rclint	Pearson ↑	—	0.7215 ± 0.0038	0.6764 ± 0.0014	0.7318 ± 0.0026	0.7336 ± 0.0089	0.7058 ± 0.0047	0.7168 ± 0.0021	0.7175 ± 0.0052	**0.8001 ± 0.0031**	*0.7820 ± 0.0012*
adme-fang-perm	Pearson ↑	0.725	0.7952 ± 0.0073	0.8087 ± 0.0042	0.8065 ± 0.0021	0.8187 ± 0.0077	0.7855 ± 0.0062	0.7914 ± 0.0065	0.7720 ± 0.0032	**0.8650 ± 0.0029**	*0.8618 ± 0.0047*
adme-fang-solu	Pearson ↑	**0.781**	0.6218 ± 0.0163	0.3994 ± 0.1485	0.6124 ± 0.0030	0.6090 ± 0.0139	0.6046 ± 0.0103	0.5841 ± 0.0115	0.5868 ± 0.0128	*0.7174 ± 0.0090*	0.7086 ± 0.0108
adme-fang-hppb	Pearson ↑	**0.886**	0.8080 ± 0.0302	0.7284 ± 0.0424	0.8276 ± 0.0004	0.8217 ± 0.0189	0.8013 ± 0.0185	0.7612 ± 0.0481	0.8128 ± 0.0228	*0.8320 ± 0.0180*	0.8192 ± 0.0072
adme-fang-rppb	Pearson ↑	*0.892*	0.7167 ± 0.0739	0.5261 ± 0.1282	0.8514 ± 0.0037	0.7641 ± 0.0259	0.7991 ± 0.0263	0.7486 ± 0.0833	0.8000 ± 0.0267	**0.8988 ± 0.0058**	0.8370 ± 0.0123
PKIS2 kinase regression											
pkis2-egfr-wt-reg	MSE ↓	459.93	485.30 ± 18.72	432.78 ± 25.35	446.41 ± 2.92	471.17 ± 22.66	475.80 ± 17.60	469.91 ± 32.25	494.72 ± 31.39	**398.52 ± 12.10**	*403.46 ± 10.13*
pkis2-kit-wt-reg	MSE ↓	849.61	856.88 ± 19.36	805.95 ± 7.55	**766.19 ± 6.91**	853.62 ± 25.76	799.07 ± 53.90	918.33 ± 20.45	805.15 ± 51.65	*786.60 ± 23.56*	786.63 ± 22.95
pkis2-ret-wt-reg	MSE ↓	589.94	766.01 ± 40.66	882.78 ± 74.50	703.35 ± 6.37	785.25 ± 81.87	794.20 ± 57.86	**524.97 ± 62.54**	785.26 ± 48.29	*542.69 ± 37.80*	553.62 ± 27.85
PKIS2 kinase classification											
pkis2-kit-wt-cls	AUPRC ↑	0.646	0.6110 ± 0.0238	0.6749 ± 0.0137	0.6273 ± 0.0026	0.5843 ± 0.0263	0.5976 ± 0.0172	—	0.6182 ± 0.0197	**0.7047 ± 0.0227**	*0.6873 ± 0.0174*
pkis2-ret-wt-cls	AUPRC ↑	**0.885**	0.7375 ± 0.0462	0.7372 ± 0.0067	0.6338 ± 0.0218	0.5353 ± 0.0760	0.6043 ± 0.0688	0.5805 ± 0.0150	0.5950 ± 0.0707	*0.8467 ± 0.0345*	0.8321 ± 0.0199

Table 8: Zero-shot routing results on the 6 TDC-ADMET held-out tasks, performance reported as mean $\pm$ std over 5 seeds. *Matched task*: the task in $\mathcal{Z}$ the router selected.

ADMET task	Metric	Matched Task	Transferred Solution	DRUGSAGE-ZERO
Ames	AUROC ↑	BBB_Martins	minimol + chemeleon + lantern-radr-ensemble	0.8714 ± 0.0026
LD50	MAE ↓	Lipophilicity_AstraZeneca	minimol + chemprop-rdkit + chemprop	0.5472 ± 0.0104
Solubility	MAE ↓	Lipophilicity_AstraZeneca	minimol + chemprop-rdkit + chemprop	0.6863 ± 0.0057
CYP2C9 Inh	AUPRC ↑	CYP2C9_Substrate_CarbonMangels	maplight-gnn + minimol + lantern-radr-ensemble	0.8518 ± 0.0031
CYP2D6 Inh	AUPRC ↑	CYP2C9_Substrate_CarbonMangels	maplight-gnn + minimol + lantern-radr-ensemble	0.7791 ± 0.0067
CYP3A4 Inh	AUPRC ↑	CYP2C9_Substrate_CarbonMangels	maplight-gnn + minimol + lantern-radr-ensemble	0.9135 ± 0.0015

G.2 Polaris Benchmark Held-Out Tasks

Table 9 reports zero-shot routing results across the 11 Polaris held-out tasks. For the nine regression tasks, neither Pearson nor MSE appears in $\mathcal{Z}$, so the router cannot find an exact metric match and instead selects the most similar pool task by task type and molecular domain. The six ADME-Fang regression tasks route to Excretion tasks with Spearman metrics, and the three PKIS2 potency tasks route to Absorption tasks with MAE metrics. The two PKIS2 kinase classification tasks both use AUPRC, which is present in $\mathcal{Z}$, and the router matches them directly to CYP substrate tasks on the basis of shared metric and task type, despite the domain gap between kinase inhibition and enzyme substrate activity.

Table 9: Zero-shot routing results on the 11 Polaris held-out tasks, performance reported as mean $\pm$ std over 5 seeds. *Matched task / category*: the selected source task; source categories are shown for the nine regression tasks.

Polaris task	Metric	Matched Task / Category	Transferred Solution	DRUGSAGE-ZERO
ADME-Fang regression				
adme-fang-hclint	Pearson ↑	Excretion (Spearman)	minimol + chemprop-rdkit + chemprop	0.7731 ± 0.0014
adme-fang-rclint	Pearson ↑	Excretion (Spearman)	minimol + chemprop-rdkit + chemprop	0.7820 ± 0.0012
adme-fang-perm	Pearson ↑	Excretion (Spearman)	minimol + chemprop-rdkit + chemprop	0.8618 ± 0.0047
adme-fang-solu	Pearson ↑	Excretion (Spearman)	maplight-gnn + minimol + lantern-radr-ensemble	0.7086 ± 0.0108
adme-fang-hppb	Pearson ↑	Excretion (Spearman)	minimol + chemprop-rdkit + chemprop	0.8192 ± 0.0072
adme-fang-rppb	Pearson ↑	Excretion (Spearman)	minimol + chemprop-rdkit + chemprop	0.8370 ± 0.0123
PKIS2 kinase regression				
pkis2-egfr-wt-reg	MSE ↓	Absorption (MAE)	minimol + chemprop-rdkit + chemprop	403.46 ± 10.13
pkis2-kit-wt-reg	MSE ↓	Absorption (MAE)	maplight-gnn + minimol + lantern-radr-ensemble	786.63 ± 22.95
pkis2-ret-wt-reg	MSE ↓	Absorption (MAE)	maplight-gnn + minimol + lantern-radr-ensemble	553.62 ± 27.85
PKIS2 kinase classification				
pkis2-kit-wt-cls	AUPRC ↑	CYP2D6_Substrate_CarbonMangels	admetrix	0.6873 ± 0.0174
pkis2-ret-wt-cls	AUPRC ↑	CYP2C9_Substrate_CarbonMangels	maplight-gnn + minimol + lantern-radr-ensemble	0.8321 ± 0.0199

G.3 Zero-Test-Time Routing Case Study

Figure 6 illustrates a concrete instance in which zero-test-time routing produces a better solution than budgeted search from scratch. Router matches Solubility task to Lipophilicity_AstraZeneca, the largest MAE regression task in $\mathcal{Z}$, and transfers its best solution without modification. The transferred solution featurizes molecules with a wide 1900-dimensional concatenation of minimol, Morgan, RDKit, and MACCS descriptors and aggregates predictions from a 10-member homogeneous dense ensemble. The search loop converges on a 4-architecture heterogeneous ensemble that combines minimol, AttentiveFP, Admetrix, and NovoExpert with LoRA-ResNet heads, achieving MAE $= 0.722$. The gap suggests that the wide, homogeneous featurization strategy learned on Lipophilicity transfers more effectively to Solubility than the ensemble the agent independently discovers, a pattern consistent with the shared physicochemical nature of the two regression targets.

H Explore Agent Workflow Details

This appendix expands the skill-construction pipeline summarized in Section 3.1. DRUGSAGE separates skill construction from the online optimization loop. The Explore Agent runs offline, before budgeted search on a target task, and converts task-relevant literature into executable skills that can

Zero-shot routed MAE↓ = 0.686

Input (1900-d)

```
x = concat(minimol(512), Morgan(1024),
      RDKit(~200), MACCS(167))
```

Head & ensemble

```
head = Dense(2048) x 4 + skip
pred = mean(head_k(x) for k in 1..10)
```

Best self-searched MAE↓ = 0.722

Input (per architecture)

```
minimol_comp.x = minimol(512)
attentive_fp.x = graph
admetrix.x   = own pipeline
novoexpert.x  = own pipeline
```

Head & ensemble

```
head = LoRA-ResNet (rank 8, 3 layers)
pred = mean(comp_i(x_i) for 4 archs)
```

Figure 6: Solubility task case: a zero-test-time routing solution routed from Lipophilicity outperforms the best solution DRUGSAGE finds by searching Solubility from scratch. The routed solution combines wide multi-source features with a 10-member homogeneous ensemble; the from-scratch search converges on a 4-architecture heterogeneous ensemble.

be read by the downstream search system. Its output is a set of directories `skills/{name}/`, each containing a structured `SKILL.md`, an API snapshot, and repository metadata.

Phase 1: LiteratureScout. The Explore Agent begins from the task name, task description, and dataset metadata. A single query often retrieves only a narrow view of the relevant literature, so LiteratureScout asks an LLM to rewrite the task from multiple expert perspectives, such as molecular-property prediction, data modality, model architecture, training objective, domain-specific constraints, and implementation availability. These perspective-specific queries are submitted to heterogeneous literature and code backends, including web search, paper indexes, preprint servers, and repository search. The returned pool is deduplicated and hard filtered to remove methods that do not match the task, papers without usable implementations, and repositories that cannot be resolved. The remaining candidates are ranked by combining LLM relevance judgements with explicit implementation signals such as recency, repository activity, documentation quality, and the presence of training or inference examples. LiteratureScout writes the selected papers, repository links, and short selection rationales to `task_memory.md` and a machine-readable companion file.

Phase 2: SkillBuilder. SkillBuilder turns each selected paper–repository pair into an executable skill. It clones the referenced repository and constructs an abstract-syntax-tree (AST) snapshot: a structural parse of the source code that records importable modules, public functions and classes, model constructors, and nearby usage examples. The snapshot is stored with the skill as an implementation contract. Given the paper summary, repository metadata, and AST snapshot, the LLM writes a structured `SKILL.md` describing the task type, dependencies, callable entry points, and a minimal quick-start example. The prompt requires executable snippets to use only identifiers observed in the AST snapshot, which targets the main failure mode of LLM-written integration code: plausible but non-existent class names, methods, and import paths.

Tiered validation and repair. A generated skill is not admitted to the shared library immediately. DrugSAGE validates it through escalating checks. Tier 0 verifies the skill file itself, including syntax, package identity, dependency metadata, and import resolvability. Tier 1 performs a dependency dry run of the quick-start section against the installed package, checking that referenced calls and objects are consistent with the repository snapshot. Tier 2 executes the quick-start end-to-end in a per-skill conda sandbox, catching dependency drift, missing data assumptions, and runtime errors. When a tier fails, SkillBuilder repairs the offending section using the AST snapshot, package metadata, and observed error message, then reruns validation. Only skills that pass all tiers are admitted to the library $\mathcal{K}$ and exposed to the online search loop through the tool catalog.

I Formal Definition of Cross-Task Transfer Scores

This section provides the formal definition of the per-task standardized score underlying the cross-task transfer terms $\mathrm{transfer}(f)$ (Equation (1)) and $\mathrm{transfer}(v_i)$ (Equation (3)) introduced in Section 3.3.

I.1 Per-Task Standardized Score

Since historical tasks use heterogeneous metrics whose raw values are not directly comparable, all historical scores are first converted to a bounded, normalized utility score before aggregation. Each node v evaluated on a historical task τ with evaluation metric μ_τ receives a raw score $x_\tau(v)$. We convert it to a standardized score $\bar{s}_\tau(v) \in [-1, 1]$ in three steps.

Step 1: Direction alignment. Convert all metrics to a higher-is-better orientation:

$$s_\tau(v) = \begin{cases} x_\tau(v), & \text{if } \mu_\tau \text{ is higher-is-better,} \\ -x_\tau(v), & \text{if } \mu_\tau \text{ is lower-is-better.} \end{cases} \tag{5}$$

Step 2: Within-task robust normalization. Normalize $s_\tau(v)$ relative to all completed nodes V_τ evaluated on task τ with the median as location and the median absolute deviation (MAD) as scale:

$$\tilde{s}_\tau(v) = \frac{s_\tau(v) - \mathrm{median}_{u \in V_\tau}\ s_\tau(u)}{\max\big(\mathrm{MAD}_{u \in V_\tau}\ s_\tau(u),\ \epsilon\big)}, \tag{6}$$

where $\mathrm{MAD}_{u \in V_\tau}\ s_\tau(u) \triangleq \mathrm{median}_{u \in V_\tau} \big| s_\tau(u) - \mathrm{median}_{u \in V_\tau}\ s_\tau(u) \big|$ and $\epsilon > 0$ is a small constant that prevents division by zero when all solutions achieve identical scores. The choice of median and MAD over mean and standard deviation makes the normalization robust to outlier solutions.

Step 3: Map to $[-1, 1]$. Apply the logistic sigmoid σ to map the unbounded $\tilde{s}_\tau(v)$ to $[-1, 1]$:

$$\bar{s}_\tau(v) = 2\,\sigma\big(\tilde{s}_\tau(v)\big) - 1 \ \in\ [-1,\, 1]. \tag{7}$$

The monotone sigmoid maps any finite normalized score to the bounded interval, ensuring no single historical task can contribute an unbounded signal to the transfer prior.

I.2 Task-Similarity Weights

Given a target task τ_0, the weight $w(\tau, \tau_0) \geq 0$ measures the relevance of each historical task $\tau \in \mathcal{Z}$:

$$w(\tau, \tau_0) = w_{\text{metric}} \cdot w_{\text{type}} \cdot w_{\text{size}} \cdot w_{\text{emb}}, \tag{8}$$

where each factor captures one dimension of task similarity:

- **Metric match.** $w_{\text{metric}} = \mathbf{1}[\mu_\tau = \mu_{\tau_0}] + \delta$ rewards exact metric match with a small fallback $\delta > 0$ for metrics in the same family (e.g., both are correlation-based);
- **Task match.** $w_{\text{type}} = \mathbf{1}\big[\mathrm{type}(\tau) = \mathrm{type}(\tau_0)\big]$ matches the task type (classification vs. regression);
- **Dataset size match.** $w_{\text{size}} = \exp\big(-\gamma \big|\log|\mathcal{D}_\tau| - \log|\mathcal{D}_{\tau_0}|\big|\big)$ penalizes dataset-size mismatch on a log scale, with decay rate $\gamma > 0$;
- **Description similarity.** $w_{\text{emb}} = \max\big(\cos(\mathbf{e}_\tau,\, \mathbf{e}_{\tau_0}),\ 0\big)$ is the clamped cosine similarity between task-description embeddings $\mathbf{e}_\tau$ obtained from an LLM embedding API.

I.3 Formal Definition of Transfer Scores

Using the standardized scores from Appendix I.1 and the weights from Appendix I.2, we define the transfer terms in Equations (1) and (3) as weighted averages over all historical tasks in $\mathcal{Z}$.

Family-level transfer. The transfer score for model family f aggregates the standardized scores of all solutions in f across historical tasks:

$$\mathrm{transfer}(f) = \frac{\displaystyle\sum_{\tau \in \mathcal{Z}} w(\tau, \tau_0)\ \cdot\ \mathrm{mean}_{v \in f,\, \tau}\ \bar{s}_\tau(v)}{\displaystyle\sum_{\tau \in \mathcal{Z}} w(\tau, \tau_0)}, \tag{9}$$

where the inner mean is over all solutions in family f evaluated on task τ.

Node-level transfer. For an individual solution v_i, we search for the solution in each historical task τ that shares the same model family and modification type as v_i, denoted $\mathrm{matched}(v_i, \tau)$, and aggregate its standardized score:

$$\mathrm{transfer}(v_i) = \frac{\sum_{\tau \in \mathcal{Z}} w(\tau, \tau_0) \cdot \bar{s}_\tau\big(\mathrm{matched}(v_i, \tau)\big)}{\sum_{\tau \in \mathcal{Z}} w(\tau, \tau_0)}. \tag{10}$$

I.4 Boundedness Guarantee

Since $\bar{s}_\tau(v) \in [-1, 1]$ for all v and τ by construction (Equation (7)), and all weights $w(\tau, \tau_0) \geq 0$, both transfer scores satisfy $|\mathrm{transfer}(f)| \leq 1$ and $|\mathrm{transfer}(v_i)| \leq 1$. This directly satisfies the bounded-transfer assumption $|\mathrm{transfer}(f)| \leq C_0$ in Assumption J.1 of Appendix J with $C_0 = 1$, ensuring that the cross-task prior cannot introduce unbounded bias into the UCB selection policy and the regret order of Theorem J.5 is preserved.

J Proof of adding the cross-task causal-factor term will not change the upperbound of speed

J.1 Problem Setup

We prove that the transfer bonus term $\mathrm{transfer}(f)$ for family-level selection and w_v for node-level selection in the family-level selector and node-level selector, respectively, *do not break the sublinear regret guarantee* of the underlying UCB1 policy. In particular, the asymptotic order of the regret bound remains identical to that of standard UCB1 algorithm, and the causal terms affect only the leading constant. Therefore, our modification does not change the upperbound of complexity after introducing cross-task memory.

J.1.1 Assumptions

Assumption J.1 (Boundedness)**.** *Rewards satisfy $r \in [0, 1]$. The clipped causal bonus terms are uniformly bounded:*

$$|\mathrm{transfer}(f)| \leq C_0 < \infty \quad \forall\, f \in \mathcal{F}$$

Assumption J.2 (Asymptotic Consistency)**.** *The CEG estimator is consistent: as $n(f) \to \infty$,*

$$\mathrm{transfer}(f) \;\to\; \Delta^*(f),$$

where $\Delta^*(f)$ is the ground-truth cross-task transfer value for family f. More precisely, we construct the estimation error satisfies

$$\big|\mathrm{transfer}(f) - \Delta^*(f)\big| \;\leq\; \frac{C_0}{\sqrt{n(f)}}$$

for some constant $C_0 > 0$.

Assumption J.3 (Optimism)**.** *The CEG bonus is (asymptotically) optimistic, consistent with the normalized score designed in UCB:*

$$\mathrm{transfer}(f) \;\geq\; \Delta^*(f) - \epsilon_t, \quad \epsilon_t \to 0 \text{ as } t \to \infty.$$

J.2 Family-Level Selection

J.2.1 Setup and Regret Definition

The family-level UCB index is

$$f_t = \arg\max_{f \in \mathcal{F}} \; \tilde{r}(f) + \alpha \sqrt{\frac{\ln(t+1)}{n(f)}} + \mathrm{transfer}(f). \tag{11}$$

Let $f^* = \arg\max_f \mathbb{E}[\tilde{r}(f) + \Delta^*(f)]$ be the optimal family under the adjusted reward. Define the sub-optimality gap

$$\Delta_f \;=\; \big(\tilde{r}(f^*) + \Delta^*(f^*)\big) - \big(\tilde{r}(f) + \Delta^*(f)\big) \;>\; 0 \quad \text{for } f \neq f^*.$$

The T-round cumulative regret is

$$R_T = \sum_{t=1}^{T} \Big[\big(\tilde{r}(f^*) + \Delta^*(f^*)\big) - \big(\tilde{r}(f_t) + \Delta^*(f_t)\big) \Big].$$

J.2.2 Lemma: Causal Bonus Is Order-Compatible with UCB Exploration

Lemma J.4 (Order Compatibility)**.** *Under Assumption J.2, decompose the CEG estimator as*

$$\text{transfer}(f) = \Delta^*(f) + \xi_t(f), \quad |\xi_t(f)| \le \frac{C_0}{\sqrt{n(f)}}.$$

Then the augmented UCB index (11) *can be written as*

$$\text{UCB}_t(f) = \underbrace{\tilde{r}(f) + \Delta^*(f)}_{\textit{adjusted true value}} + \underbrace{\alpha\sqrt{\frac{\ln(t+1)}{n(f)}} + \xi_t(f)}_{\textit{confidence term}}, \tag{12}$$

where the confidence term satisfies

$$\left| \alpha\sqrt{\frac{\ln(t+1)}{n(f)}} + \xi_t(f) \right| \;\le\; \frac{\alpha\sqrt{\ln(t+1)} + C_0}{\sqrt{n(f)}} \;=\; O\left(\sqrt{\frac{\ln t}{n(f)}}\right).$$

Hence the causal bonus introduces no new asymptotic order.

Proof. The decomposition follows directly from Assumption J.2. The bound on the confidence term follows from the triangle inequality:

$$\left| \alpha\sqrt{\frac{\ln(t+1)}{n(f)}} + \xi_t(f) \right| \;\le\; \alpha\sqrt{\frac{\ln(t+1)}{n(f)}} + |\xi_t(f)| \;\le\; \frac{\alpha\sqrt{\ln(t+1)} + C_0}{\sqrt{n(f)}}.$$

Since $\alpha\sqrt{\ln(t+1)} + C_0 = O(\sqrt{\ln t})$, the entire confidence term is $O\big(\sqrt{\ln t / n(f)}\big)$, matching the standard UCB rate. □

J.2.3 Theorem: Family-Level Regret Bound

Theorem J.5 (Family-Level Regret Upper Bound)**.** *Under Assumptions J.1–J.3, the augmented UCB policy* (11) *achieves cumulative regret*

$$R_T \;\le\; \sum_{\substack{f \in \mathcal{F} \\ \Delta_f > 0}} \left(\frac{8(\alpha^2 + C_0^2)\ln T}{\Delta_f} + \left(1 + \frac{\pi^2}{3}\right)\Delta_f \right). \tag{13}$$

This is $O(|\mathcal{F}| \ln T)$, identical in asymptotic order to standard UCB; the causal factor only modifies the leading constant via C_0.

Proof. We follow the standard UCB analysis and carefully track the causal error term $\xi_t(f)$.

Step 1: High-probability bound on the optimal arm.

By Hoeffding's inequality [43], with probability at least $1 - t^{-4}$,

$$\tilde{r}(f^*) \;\ge\; \hat{r}_{n(f^*)}(f^*) - \sqrt{\frac{2\ln t}{n(f^*)}}.$$

Therefore, using Assumption J.2 for the CEG term,

$$\mathrm{UCB}_t(f^*) \;\geq\; \tilde{r}(f^*) + \Delta^*(f^*) - \frac{C_0}{\sqrt{n(f^*)}}. \tag{14}$$

Step 2: Necessary condition for selecting a suboptimal arm.

Arm $f \neq f^*$ is selected at round t only if $\mathrm{UCB}_t(f) \geq \mathrm{UCB}_t(f^*)$. Combining with (14) and using the decomposition (12):

$$\tilde{r}(f) + \Delta^*(f) + \alpha\sqrt{\frac{\ln(t+1)}{n(f)}} + \frac{C_0}{\sqrt{n(f)}} \;\geq\; \tilde{r}(f^*) + \Delta^*(f^*).$$

Rearranging:

$$\left(\alpha + \frac{C_0}{\sqrt{\ln(t+1)}}\right)\sqrt{\frac{\ln(t+1)}{n(f)}} \;\geq\; \Delta_f. \tag{15}$$

Step 3: Upper bound on arm pull count.

Squaring both sides of (15) and solving for $n(f)$:

$$n(f) \;\leq\; \frac{\left(\alpha + C_0/\sqrt{\ln(t+1)}\right)^2 \ln(t+1)}{\Delta_f^2} \;\leq\; \frac{2(\alpha^2 + C_0^2)\ln T}{\Delta_f^2},$$

where the last step uses $(\alpha + C_0/\sqrt{\ln(t+1)})^2 \leq 2(\alpha^2 + C_0^2)$ by the AM–GM inequality, valid for all $t \leq T$.

Step 4: Expected number of pulls.

Using the standard UCB analysis (tail-sum decomposition),

$$\mathbb{E}[N_T(f)] \;\leq\; \frac{8(\alpha^2 + C_0^2)\ln T}{\Delta_f^2} + 1 + \frac{\pi^2}{3}.$$

Step 5: Cumulative regret.

Summing over all suboptimal families:

$$R_T = \sum_{f:\,\Delta_f > 0} \Delta_f \cdot \mathbb{E}[N_T(f)] \leq \sum_{f:\,\Delta_f > 0} \left(\frac{8(\alpha^2 + C_0^2)\ln T}{\Delta_f} + \left(1 + \frac{\pi^2}{3}\right)\Delta_f\right).$$

Key observation. The bound is $O(\ln T)$, matching standard UCB. The causal factor contributes only through C_0, which modifies the constant but not the asymptotic order.

If we also consider the node-level regret bound, under the weighted sampler computation in 3, the bonus $\max(\Delta^{\mathrm{CEG}}(v_i), 0)$ now serves as a non-negative multiplicative determined by w_i. It therefore only increases sampling probability for nodes with positive CEG signal, and leaves the original node-level UCB asymptotic regret order $O(\sqrt{T \ln T})$ unchanged while potentially improving the leading constant. □

K Additional Analyses of Cross-Task Experience

K.1 Matched Starting Models and Search-System Differences

We compare DRUGSAGE-anchor and MLEvolve-anchor on the 22 TDC-ADMET tasks using the same top-three TDC leaderboard models and no cross-task memory. Both use Claude Sonnet 4.6, one NVIDIA L40S, a 24-hour wall-clock budget per task, the same task prompt, and the same data and evaluation constraints. One unit of budget is one train-and-evaluate cycle. Each agent retains its native system prompts and execution harness. With the starting models, task prompt, and budget

Table 10: Matched-anchor comparison without cross-task memory on 22 TDC-ADMET tasks.

Agent	Starting models	Normalized score ↑
MLEvolve-anchor	Same top-three models	0.630
DRUGSAGE-anchor	Same top-three models	0.875

held fixed, the remaining 0.245 score difference reflects the search, refinement, repair, and execution systems. Within DRUGSAGE, enabling cross-task memory under the same skill library increases the six-task held-out normalized score from 0.81 to 0.99. In the inspected MLEvolve trajectories, a

Table 11: Search-loop comparison. Both systems support tree search, refinement, and debugging.

Dimension	DRUGSAGE	MLEvolve
Roots	Multi-root forest from executable skills	LLM-generated branches under a virtual root
Selection	Family UCB and parent sampling with cross-task priors	Task-local UCT and time-dependent top-k
Proposals	Typed edits using parent and refinement memory	Edits using task and within-run memory
Repair	Reuses verified fixes and resource profiles	Prompts patches or rewrites from within-run errors
Persistence	Solution, refinement, and execution memory across tasks	Within-run experimental history
Routing	Verified-solution transfer at $B = 0$	New search for each task

CUDA initialization error involving multi-worker data loading over GPU-resident tensors recurs on 16 of the 22 tasks. These nodes account for about 4.2% of execution failures. DRUGSAGE stores the verified fix in execution memory and reuses it when the same error occurs.

K.2 Skill Library and Full Cost Accounting

The skill library contains 55 models from nine architecture families, with original publication years roughly spanning 2017–2025. It is constructed once and held fixed across all 33 evaluation tasks; source repositories and generated skills do not change between runs. In deployment, the library can be extended incrementally. The build uses the first two rule-based validation tiers. The third tier, end-to-end execution and repair, is charged to the task that selects the skill. Library construction is distinct from experience accumulation: the latter requires searching the 16 pool tasks from scratch under a 20-experiment budget for each task. The offline cost includes the library build and the 16 pool-task searches. Averaging the library-build cost over 33 tasks gives about USD 0.32 per task; it does not amortize the pool-task training or search costs. Figure 4(a) reports held-out test-time API costs and excludes these offline expenses and GPU training costs. Total tokens per pool task range from about 0.6M to 1.4M, with a maximum about 1.6 times the median. In contrast, calls within a from-scratch task are long-tailed: the top 20% of calls use about 0.54M of 0.85M total tokens (about 64%), mainly for code generation and repair. The other 80% use about 0.31M. Zero-test-time routing avoids LLM generation but still uses a task-description embedding and target-task training and evaluation.

Table 12: Architecture-family distribution in the fixed skill library.

Model family	Models
2D GNN / MPNN	16
SMILES / sequence model	9
3D / geometric GNN	7
Fingerprint / classical ML (gradient boosting)	7
Graph transformer	6
Ensemble / stacking	5
CNN	3
Tabular in-context (TabPFN)	1
Multi-task network	1
Total	55

Table 13: One-time Explore-Agent library build. Total is rounded.

Stage	Work	Cost (USD)
Discovery	Literature and repository search, ranking, extraction, synthesis	3.0
Grounding	Skill documentation, examples, and scripts	7.4
Validation	Rule-based repair and package-identity checks	0.11
Total		≈ 10.5

K.3 Memory Components and Search Controls

Removing solution memory causes the largest leave-one-out drop. Separately, replacing the Explore-Agent library with the top-three leaderboard models reduces the 22-task from-scratch normalized score from 0.929 to 0.875 (Table 1). Routing is evaluated by DRUGSAGE-Zero, which uses no target-task search and therefore does not use the family- or parent-selection transfer terms.

Users can set the experiment budget, early-stopping threshold and patience, allocation across search phases, and numbers of model families and refinements. By default, search stops when improvement in the best score is less than 0.001 over five consecutive experiments. Full DRUGSAGE reaches its best-so-far performance after about five experiments within the 20-experiment evaluation. Figure 3 and Appendix F show the supplied budget trajectories.

K.4 Memory Growth and Composition

We evaluate DRUGSAGE-Zero on the same six held-out TDC-ADMET tasks using nested subsets of the 16-task experience pool. A single random arrival chain (seed 0) gives scores of 0.10, 0.43, 0.40, and 0.90 at $|\mathcal{Z}| = 2, 4, 8, 16$, respectively. The decrease from four to eight tasks is caused by CYP2C9 inhibition routing to a less suitable analog; the other five tasks hold or improve. Thus adding an individual task is not guaranteed to improve transfer.

We additionally evaluate 10 independently sampled, stratified task-arrival chains with fixed baseline-based normalization. Each chain accumulates the same 16 pool tasks in a different random order, producing different memory compositions at $|\mathcal{Z}| = 2, 4, 8$ and the same complete pool at $|\mathcal{Z}| = 16$. Table 16 reports the mean and 95% confidence interval across chains on the same six held-out TDC-ADMET tasks. Average performance increases from 0.39 to 0.90 and is monotonic across the evaluated memory sizes. Across the pooled observations, memory size is positively associated with performance ($\rho = 0.44$, $p = 1.1 \times 10^{-11}$). The intervals at smaller pool sizes quantify sensitivity to memory composition; the complete-pool endpoint is shared across chains. The self-evolution claim concerns improved expected transfer with broader experience, rather than improvement after every individual task addition.

K.5 Leave-Category-Out Routing and Negative Transfer

Only four of the 17 held-out tasks normally route to a same-category analog: Solubility and the three CYP inhibition tasks. AMES, LD50, and all 11 Polaris tasks already route across categories. We

Table 14: Per-task input and output tokens at budget 20. Pool-task ranges and medians are measured across the 16 from-scratch tasks; held-out values are approximate.

Task group	Input tokens	Output tokens
16 pool tasks, no memory	0.5M–1.2M	0.07M–0.17M
Pool-task median	0.74M	0.11M
17 held-out tasks, full DRUGSAGE	≈ 0.29M	≈ 0.05M
17 held-out tasks, DRUGSAGE-Zero	≈ 0	≈ 0

Table 15: Leave-one-out memory ablation on the six held-out TDC-ADMET tasks. Each component is removed separately.

Configuration	Normalized score ↑
Full DRUGSAGE	0.99
Without refinement memory $\mathcal{H}$	0.91
Without execution memory $\mathcal{Q}$	0.89
Without solution memory $\mathcal{R}$	0.83
Without cross-task memory $\mathcal{Z}$	0.81

remove the matched category for each of the four same-category cases and rerun DRUGSAGE-Zero. The 17-task average normalized score decreases from 0.935 to 0.787, but remains above the strongest baseline, Autoresearch, at 0.588. Solubility and CYP2D6 benefit from memory. On LD50, both transferred routing and memory-augmented search perform worse than no-memory search, providing a case of negative transfer.

Eight of the 17 tasks retrieve an exact metric match. The remaining nine use related metrics: Pearson and Spearman for the six ADME-Fang regression tasks, and MSE and MAE for the three kinase regression tasks. Related endpoints occur for the three CYP inhibition tasks, which route to CYP substrate prediction, and Solubility, which routes to Lipophilicity. Matched TDC-ADMET source tasks, transferred pipelines, and Polaris source categories are listed in Appendix G.

K.6 Transfer from ADMET to Kinase Activity

The five PKIS2 tasks measure molecule-by-target kinase activity, rather than ADMET properties. They include two classification and three potency-regression tasks. Memory contains only the 16 TDC-ADMET pool tasks, with no target-activity task. DRUGSAGE and DRUGSAGE-Zero obtain normalized scores of 0.96 and 0.91, respectively, and rank first and second in this tested domain shift. Across the same five tasks, normalized scores are 0.19 for MLEvolve, 0.30 for Agentomics, 0.31 for AIRA-Dojo, 0.34 for Autoresearch, 0.35 for STELLA, 0.54 for ShinkaEvolve, and 0.56 for Claude Code. These results show that experience learned from ADMET tasks can transfer to kinase activity prediction, even when the memory contains no target-activity task.

K.7 Extension to Generative Tasks

DRUGSAGE can be applied to generative design by changing the task interface. The search requires candidate solutions and a score for each candidate. In predictive tasks, a candidate is a model pipeline and its score is the validation metric. In generative tasks, a candidate is a molecular generator and its score is given by the task oracle. Solution-tree search, UCB selection, LLM refinement, and cross-task memory follow the same search formulation. We plan to continue exploring this extension in future work.

Table 16: Memory growth across 10 independently sampled, stratified task-arrival chains. Values are mean $\pm$ 95% CI across chains using fixed baseline-based normalization. At $|\mathcal{Z}| = 16$, all chains use the same complete memory pool; the zero interval reflects this shared endpoint.

Memory size $\lvert\mathcal{Z}\rvert$	Normalized score (mean $\pm$ 95% CI)
2	0.39 ± 0.18
4	0.58 ± 0.08
8	0.63 ± 0.11
16	0.90 ± 0.00

Table 17: Default and leave-category-out routing. Lower MAE and higher AUPRC are better.

Held-out task	Metric	Default route	Category removed
Solubility_AqSolDB	MAE $\downarrow$	0.686	0.763
CYP2C9_Veith	AUPRC $\uparrow$	0.852	0.822
CYP2D6_Veith	AUPRC $\uparrow$	0.779	0.744
CYP3A4_Veith	AUPRC $\uparrow$	0.914	0.892

Table 18: Beneficial and harmful transfer relative to search without cross-task memory. All numbers are native task metrics.

Task	Metric	No-memory search	DRUGSAGE-Zero	Full DRUGSAGE
Solubility	MAE $\downarrow$	0.722	0.686	0.679
CYP2D6 inhibition	AUPRC $\uparrow$	0.724	0.779	0.781
LD50	MAE $\downarrow$	0.502	0.547	0.539

Table 19: Transfer from ADMET-only memory to five PKIS2 tasks. Ranks are among agents with available results.

Kinase task	Metric	DRUGSAGE-Zero (rank)	DRUGSAGE (rank)
KIT active/inactive	AUPRC $\uparrow$	0.687 (2/8)	0.705 (1/8)
RET active/inactive	AUPRC $\uparrow$	0.832 (2/9)	0.847 (1/9)
EGFR potency	MSE $\downarrow$	403.5 (2/9)	398.5 (1/9)
KIT potency	MSE $\downarrow$	786.6 (3/9)	786.6 (2/9)
RET potency	MSE $\downarrow$	553.6 (3/9)	542.7 (2/9)

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The claims match the stated method in Section 3 and experiments in Section 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

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

   Answer: [Yes]

   Justification: We discussed the limitations in Section 5.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

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

   Answer: [Yes]

Justification: Assumptions and proofs are provided in Section J.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

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

Answer: [Yes]

Justification: Datasets, splits, metrics, and protocols are specified in Section 4 and Section C.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

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

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

   Answer: [Yes]

   Justification: Public data and codes are provided in supplementary material.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: Experimental settings are described in the Section 4 and Section C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: Results are reported over multiple seeds and results are reported in Table 6 and Table 7.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: Computational resources are reported in Appendix C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The work follows the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: Broader impacts are discussed in Section 5.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [N/A]

    Justification: Released artifacts are limited to predictive modeling workflows.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: Existing datasets, tools, models, and codes are cited and properly respected.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

    Answer: [Yes]

    Justification: New released artifacts are documented.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

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

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

    Answer: [N/A]

    Justification: This work does not involve human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

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

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

    Answer: [N/A]

    Justification: No human-subject research is involved.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

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

Answer: [Yes]

Justification: LLMs are a core component of the proposed system. Claude Sonnet 4.6 (accessed via the Anthropic API) serves as the backbone of the proposed agent.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.