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

Anonymous Authors[1]

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. In 33 molecular property prediction tasks, DRUGSAGE ranks first among nine SOTA agents in a single-task setting. With memory accumulated from 16 smaller tasks, DRUGSAGE achieves a averaged normalized score of 0.935 on 17 held-out tasks in a cross-task evaluation setting and outperforms all baseline agents by 10-30% in a zero-test-time search regime. 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 (Wei et al., 2025). In biomedicine and drug discovery, systems such as Biomni (Huang et al., 2025) and STELLA (Jin et al., 2025) have demonstrated that LLM-based agents can automatically build workflows for data processing, feature extraction, and predictive modeling. More recent efforts such as AutoResearch (Karpathy, 2026) 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 (Liaw et al., 2018). 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 (Bai et al., 2024). 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 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

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Submitted to the 2026 Workshop on Generative and Agentic AI for Biology (ICML 2026). Do not distribute.

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 (Huang et al., 2021; Polaris, 2025) 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 achieves an average score of 0.935 on 17 held-out tasks in a cross-task setting and outperforms the best baseline by more than 10-30% in a zero-test-time search regime. 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.
- Unlike previous works such as Autoresearch (Karpathy, 2026) 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 outperforms all baseline agents by more than 10-30% 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 (Guo et al., 2024; Nam et al., 2025). Nevertheless, later work tackles them via searching algorithms (Jiang et al., 2025; Toledo et al., 2025). More recently, a variety of work improve over previous methods from several dimensions, including introducing agent structures (Yang et al., 2025; Li et al., 2025), incorporating better search algorithms (Chen et al., 2026; Feng et al., 2026) and external knowledge (Nadafian et al., 2026). 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 (Shinn et al., 2023), while other approaches distill non-executable insights into semantic memory (Zhao et al., 2024; Chen et al., 2024; Suzgun et al., 2026). For actionable guidance, systems develop procedural memory by maintaining append-only skill libraries (Wang et al., 2024), inducing reusable workflows (Wang et al., 2025b), managing full memory lifecycles (Fang et al., 2025), or evolving shared knowledge via multi-agent reflection (Qu et al., 2026). More recent work explores the accumulation of agent experiences across tasks such as (Zheng et al., 2025; Tang et al., 2025; Xiao et al., 2025). 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 reasoning capacity in codes, tools, and scientific contexts. One line of work embeds LLMs in a verifier-driven hypothesis search loop. FunSearch (Romera-Paredes et al., 2024) uses LLMs as evolutionary operators in a program evolution loop, while a significant amount of later works extend its capability beyond program discovery (Wang et al., 2025a; Shojaee et al., 2025; Novikov et al., 2025). Beyond hypothesis search, an alternative perspective is to build agents that orchestrate tools such as Coscientist (Boiko et al., 2023) and ChemCrow (M. Bran et al., 2024). Later work (Huang et al., 2025; Jin et al., 2025) has furthered agent capabilities to build computational workflows. More recently, SAGA (Du et al., 2025) has taken a step forward in automatically evolving the objectives in the scientific discovery workflow. The most closely related work is Agentomics (Martinek et al., 2026), 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 $\mathcal{D}_*$ stands for training, validation, and test data; μ is the task metric (e.g., AUROC, AUPRC, RMSE, MAE); and B is the budget for training and validation set evaluation runs.
- The **skill library** $\mathcal{K}$ is a collection of validated, task-relevant **model families**. A model family consists of different instances of a model (e.g., Chemprop with different hyperparameters).
- 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. When $B > 0$, it runs the memory-enhanced Monte Carlo tree search algorithm (MCTS) to search for executable solutions (§3.3), where each step is equipped with one of the memory components(§3.2). When $B = 0$, DRUGSAGE performs memory routing to transfer a verified solution directly without additional search. 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.

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

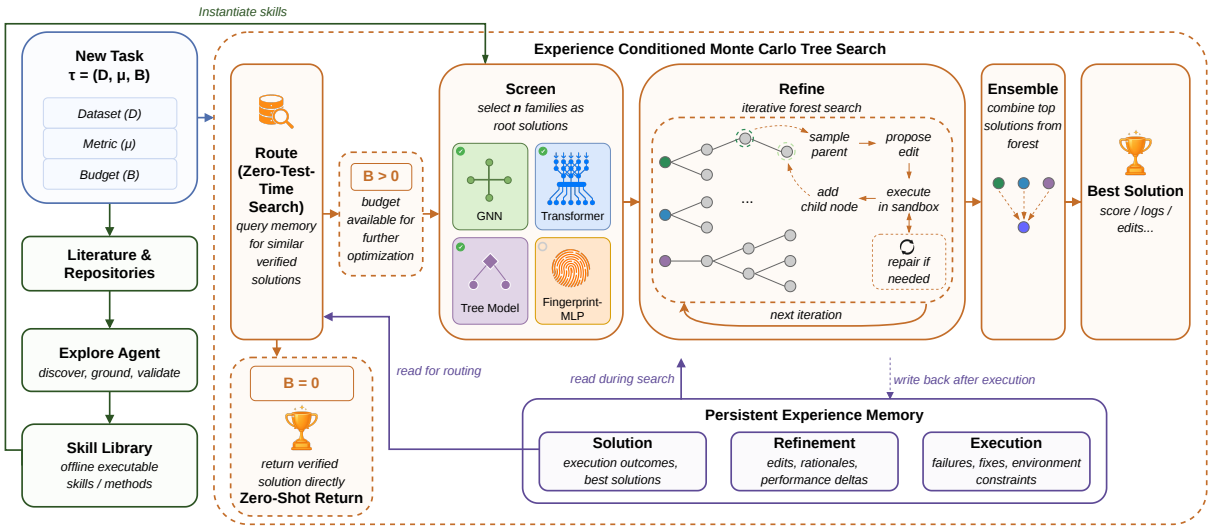

Figure 1. **DRUGSAGE overview.** The Explore Agent builds a shared skill library $\mathcal{K}$ from literature and repositories. Cross-task memory $\mathcal{Z}$ enables two settings: experience-conditioned MCTS ($B > 0$), where $\mathcal{Z}$ guides and is updated by each search step, and zero-test-time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

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 task, penalized for instability and overfit; $\text{transfer}(f)$ computes the weighted average performance of nodes in family f over historical tasks, using the records retrieved from the solution memory. We formally define $\text{transfer}(f)$ in Appendix I. Intuitively, when $t = 0$, the agent tend 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 ($\text{exploit}_t(f)$) rather than the performance on the historical tasks ($\text{transfer}(f)$). 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* $|\text{transfer}(f)| \leq C_0$*, and standard asymptotic convergence of the exploit estimate, the policy in Equation* (1) *attains cumulative regret*

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

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

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

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 $c_i = |\text{children}(v_i)|$ its expansion count. The parent sampler assigns the following weight to each candidate solution:

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

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

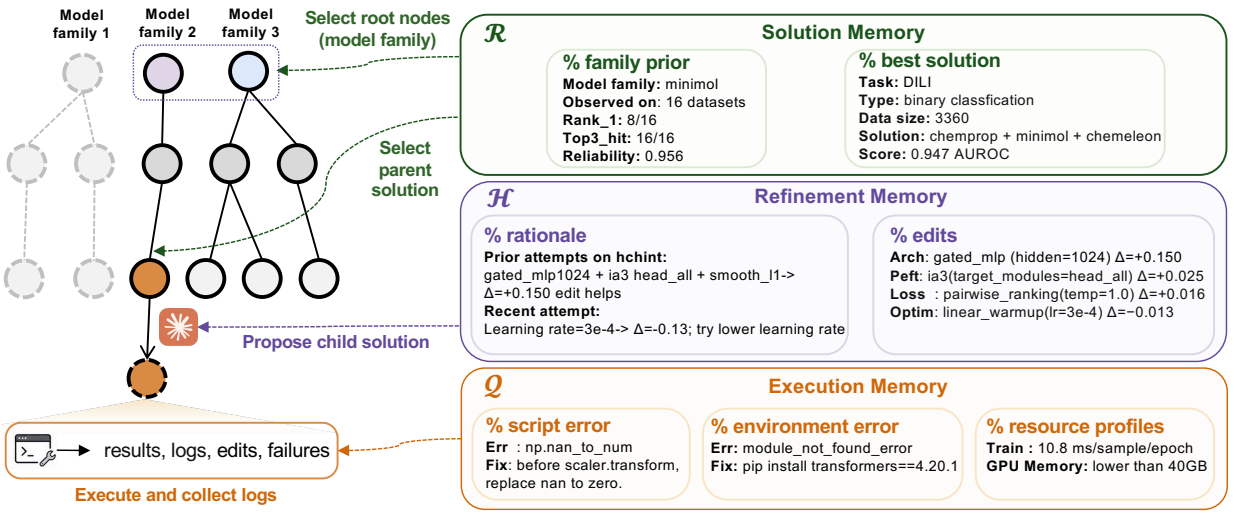

Figure 2. **Experience Memory.** DRUGSAGE stores cross-task experience as $\mathcal{Z} = (\mathcal{R}, \mathcal{H}, \mathcal{Q})$. Solution memory $\mathcal{R}$ stores family priors and verified best solutions for root-family and parent-solution selection. Refinement memory $\mathcal{H}$ stores proposal 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.

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

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 Appendix 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**

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 deploy 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) (Huang et al., 2021). 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.

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 (Karpathy, 2026) and ShinkaEvolve (Lange et al., 2025), 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 (Feng et al., 2026) and AIRA-Dojo (Toledo et al., 2025); one *general-purpose coding agent*, Claude Code (Anthropic, 2025); three *scientific-discovery agents*, Biomni (Huang et al., 2025)[1], STELLA (Jin et al., 2025), and Agentomics (Martinek et al., 2026). All from-scratch

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

Table 2. The cross-task performance (without normalization) of DRUGSAGE-ZERO on six TDC held-out tasks. Full cross-task results for all individual tasks are in the Appendix 8 and 9.

Task (size)	Metric	DRUGSAGE-ZERO	DRUGSAGE	TDC Leaderboard Best
Ames (7,255)	AUROC ↑	0.871 ± 0.003	0.878 ± 0.0126	0.871 ± 0.002
LD50 (7,385)	MAE ↓	0.547 ± 0.010	0.502 ± 0.0131	0.552 ± 0.009
Solubility (9,982)	MAE ↓	$\mathbf{0.686 \pm 0.006}$	0.722 ± 0.0109	0.741 ± 0.013
CYP2C9 Inhibition (12,092)	AUPRC ↑	0.852 ± 0.003	0.861 ± 0.0019	0.859 ± 0.001
CYP3A4 Inhibition (12,328)	AUPRC ↑	0.914 ± 0.002	0.914 ± 0.0013	0.916 ± 0.000
CYP2D6 Inhibition (13,130)	AUPRC ↑	$\mathbf{0.779 \pm 0.007}$	0.724 ± 0.0022	0.790 ± 0.001

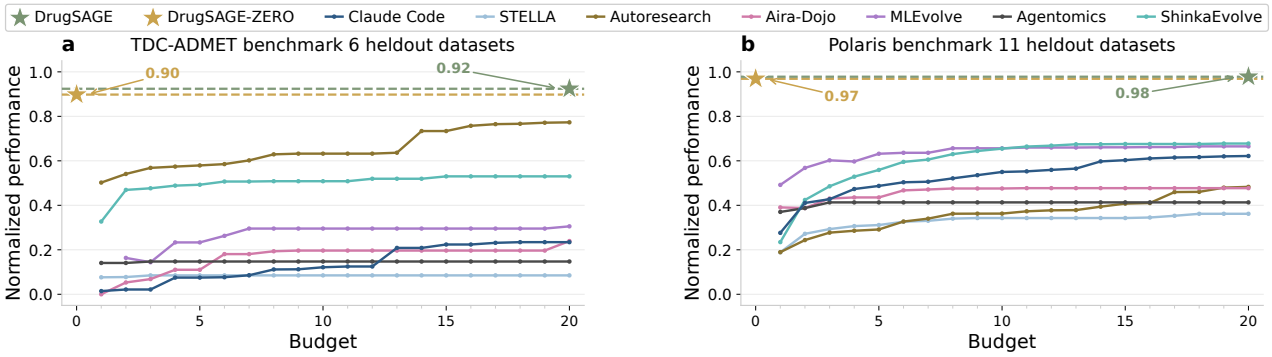

Figure 3. Results on the second scenario, where DRUGSAGE leverages cross-task memory to search for SOTA solutions with minimal search budget. **(a)** The 6 TDC-ADMET tasks held out from the pool. **(b)** 11 held-out tasks from the Polaris Hub. Per-task scores are min-max normalized within each task before averaging; higher is better. The dashed line is DRUGSAGE's zero-shot score. Polaris per-task absolute metrics (mean ± std over 5 seeds) are in Appendix E. Per-task curves are in Appendix F.

agents, including DRUGSAGE, receive the same shared task prompt (Appendix C). 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 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.

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 (Ash et al., 2025), including six ADMET tasks from Fang et al. (2023) and five kinase-inhibition tasks based on the PKIS2 data of Drewry et al. (2017). 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.

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 4. 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.

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.

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. In both benchmarks, DRUGSAGE-ZERO outperforms the baselines with more than 10-30% gain.

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 with $B = 20$ test-time search. In some cases like Solubility and CYP2D6 Inhibition, DRUGSAGE-ZERO even outperforms DRUGSAGE (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 the Appendix **??**.

Case study: why zero-shot routing outperforms search. Taking Solubility as a concrete example (Figure 4), the router matches it 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, achieving MAE = 0.686. In contrast, 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. 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.

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 5(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.

Importance of cross-task memory. Figure 5 (a, b) evaluates the contribution of cross-task experience memory $\mathcal{Z}$ un-

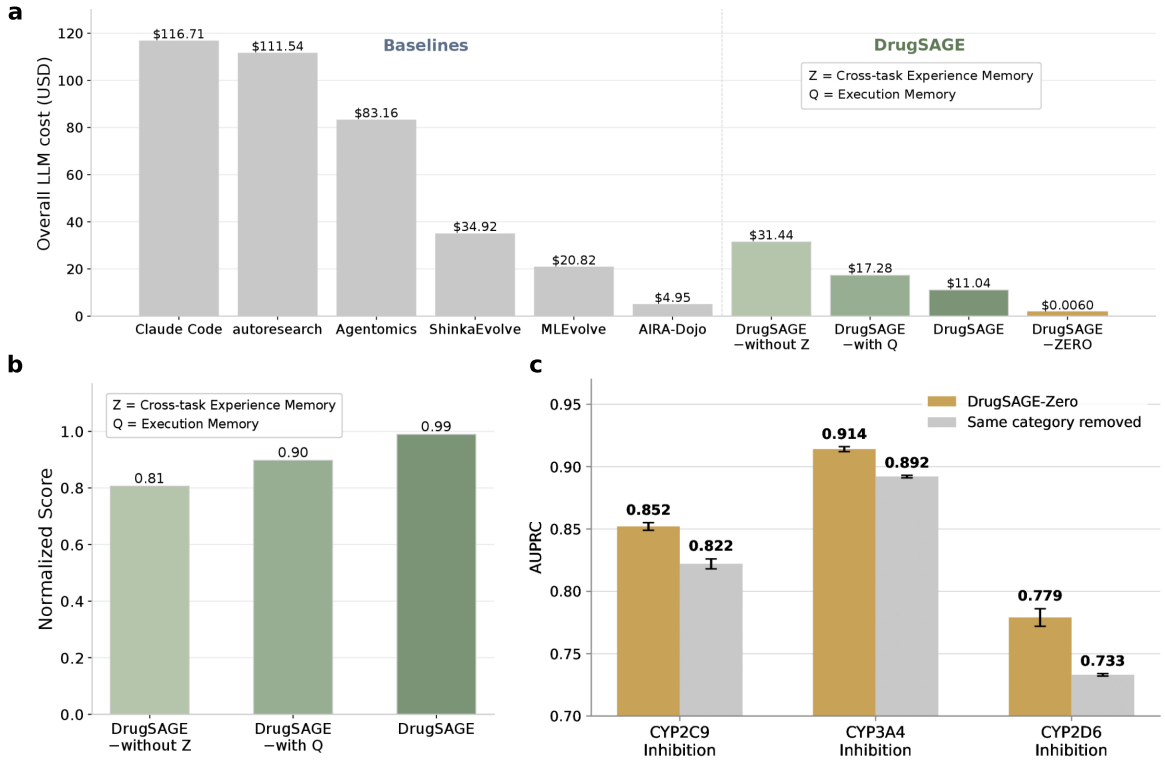

Figure 5. **(a)** We report total LLM API cost on the six held-out TDC-ADMET tasks. Compared with existing agent baselines, DRUGSAGE variants require substantially lower LLM cost. DRUGSAGE-ZERO uses nearly zero LLM cost because it reuses cross-task experience without test-time search. **(b)** Performance drops when $\mathcal{Z}$ is removed, and drops further without $\mathcal{Q}$, showing each module's contribution. **(c)** Ablation study of the impact of analog tasks on the three target tasks.

der 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.

Impact of analog tasks. To examine the impact of analog tasks on the performance of DRUGSAGE-ZERO, we conduct ablation studies on three target tasks (CYP2C9, CYP3A4, CYP2D6) that have tasks in the experience pool belonging to the same category. We run DRUGSAGE-ZERO with these tasks 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 5(c), DRUGSAGE-ZERO remains competitive with the TDC leaderboard best on all three targets: 0.822 versus 0.820 on CYP2C9 inhibition, 0.892 versus 0.898 on CYP3A4 inhibition, and 0.733 versus 0.728 on CYP2D6 inhibition. Removing same-category tasks reduces performance only by 0.03–0.05, but the routed solutions remain close to the leaderboard level when $B = 0$. These results show that the performance of DRUGSAGE-ZERO is not merely copying exact same-category analogs on these CYP targets.

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-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 of 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.

References

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

Ash, J. R., Wognum, C., Rodríguez-Pérez, R., Aldeghi, M., Cheng, A. C., Clevert, D.-A., Engkvist, O., Fang, C., Price, D. J., Hughes-Oliver, J. M., 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.

Bai, G., Chai, Z., Ling, C., Wang, S., Lu, J., Zhang, N., Shi, T., Yu, Z., Zhu, M., Zhang, Y., et al. Beyond efficiency: A systematic survey of resource-efficient large language models. *arXiv preprint arXiv:2401.00625*, 2024.

Boiko, D. A., MacKnight, R., Kline, B., and Gomes, G. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023.

Chen, J., Mishra, B. D., Nam, J., Meng, R., Pfister, T., and Yoon, J. Mars: Modular agent with reflective search for automated ai research. *arXiv preprint arXiv:2602.02660*, 2026.

Chen, M., Li, Y., Yang, Y., Yu, S., Lin, B., and He, X. Automanual: Constructing instruction manuals by llm agents via interactive environmental learning. *Advances in Neural Information Processing Systems*, 37:589–631, 2024.

Drewry, D. H., Wells, C. I., Andrews, D. M., Angell, R., Al-Ali, H., Axtman, A. D., Capuzzi, S. J., Elkins, J. M., Ettmayer, P., Frederiksen, M., et al. Progress towards a public chemogenomic set for protein kinases and a call for contributions. *PloS one*, 12(8):e0181585, 2017.

Du, Y., Yu, B., Liu, T., Shen, T., Chen, J., Rittig, J. G., Sun, K., Zhang, Y., Song, Z., Zhou, B., et al. Accelerating scientific discovery with autonomous goal-evolving agents. *arXiv preprint arXiv:2512.21782*, 2025.

Fang, C., Wang, Y., Grater, R., Kapadnis, S., Black, C., Trapa, P., and Sciabola, S. 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.

Fang, R., Liang, Y., Wang, X., Wu, J., Qiao, S., Xie, P., Huang, F., Chen, H., and Zhang, N. Memp: Exploring agent procedural memory. *arXiv preprint arXiv:2508.06433*, 2025.

Feng, S., Ma, R., Yan, X., Fan, Y., Hu, Y., Huang, S., Zhang, S., Cao, Z., Peng, T., Yuan, J., Guo, Z., Zhong, Z., Du, S., Wang, W., Shi, J., Zhou, Y., He, X., Yu, Z., Yu, F., Zhan, B., Zheng, Q., Wu, J., Liu, M., Zhang, C., Hou, S., Li, S., Jiang, Y., Lou, W., Wang, L., Wang, Z., Wang, J., Xu, W., Deng, Y., Liu, D., Wang, Y., Zhang, W., Ling, F., Zhang, S., Wang, X., Zheng, S., Huang, X., Sun, S., Hu, S., Ye, P., Song, C., Wang, B., He, C., Liu, Y., Li, X., Hou, Q., Chen, T., Yue, X., Wang, B., He, L., Lin, D., Zhou, B., Zhang, B., and Bai, L. Internagent-1.5: A unified agentic framework for long-horizon autonomous scientific discovery. *arXiv preprint arXiv:2602.08990*, 2026.

Green, W., Burns, J., Zalte, A. S., Abreu, C., Sieg, J., Feldmann, C., and Mathea, M. Deep learning foundation models from classical molecular descriptors. 2026.

Guo, S., Deng, C., Wen, Y., Chen, H., Chang, Y., and Wang, J. Ds-agent: Automated data science by empowering large language models with case-based reasoning. In *Forty-first International Conference on Machine Learning*, 2024.

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

Huang, K., Fu, T., Gao, W., Zhao, Y., Roohani, Y., Leskovec, J., Coley, C. W., Xiao, C., Sun, J., and Zitnik, M. Therapeutics data commons: Machine learning datasets and tasks for drug discovery and development. *Advances in Neural Information Processing Systems*, 2021.

Huang, K., Zhang, S., Wang, H., Qu, Y., Lu, Y., Roohani, Y., Li, R., Qiu, L., Li, G., Zhang, J., et al. Biomni: A general-purpose biomedical ai agent. *biorxiv*, 2025.

Jiang, Z., Schmidt, D., Srikanth, D., Xu, D., Kaplan, I., Jacenko, D., and Wu, Y. Aide: Ai-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*, 2025.

Jin, R., Zhang, Z., Wang, M., and Cong, L. Stella: Self-evolving llm agent for biomedical research. *arXiv preprint arXiv:2507.02004*, 2025.

Karpathy, A. autoresearch: Ai agents running research on single-gpu nanochat training automatically. https://github.com/karpathy/autoresearch, 2026.

Lange, R. T., Imajuku, Y., and Cetin, E. Shinkaevolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*, 2025.

Li, A., Wu, C., Ge, Z., Chong, Y. H., Hou, Z., Cao, L., Ju, C., Wu, J., Li, H., Zhang, H., et al. The fm agent. *arXiv preprint arXiv:2510.26144*, 2025.

Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., and Stoica, I. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

M. Bran, A., Cox, S., Schilter, O., Baldassari, C., White, A. D., and Schwaller, P. Augmenting large language models with chemistry tools. *Nature machine intelligence*, 6(5):525–535, 2024.

Martinek, V., Gariboldi, A., Tzimotoudis, D., Galea, M., Zacharopoulou, E., Escudero, A. A., Blake, E., Čechák, D., Cassar, L., Balestrucci, A., et al. Agentomics: An agentic system that autonomously develops novel state-of-the-art solutions for biomedical machine learning tasks. *bioRxiv*, pp. 2026–01, 2026.

Nadafian, A., Mohammadshahi, A., and Yazdani, M. Kapso: A knowledge-grounded framework for autonomous program synthesis and optimization. *arXiv preprint arXiv:2601.21526*, 2026.

Nam, J., Yoon, J., Chen, J., Shin, J., Arik, S. O., and Pfister, T. Mle-star: Machine learning engineering agent via search and targeted refinement. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Novikov, A., Vũ, N., Eisenberger, M., Dupont, E., Huang, P.-S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B., Ruiz, F. J., Mehrabian, A., et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Polaris. Polaris hub. https://github.com/polaris-hub/polaris, 2025.

Qu, A., Zheng, H., Zhou, Z., Yan, Y., Tang, Y., Ong, S. Y., Hong, F., Zhou, K., Jiang, C., Kong, M., et al. Coral: Towards autonomous multi-agent evolution for open-ended discovery. *arXiv preprint arXiv:2604.01658*, 2026.

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J., Ellenberg, J. S., Wang, P., Fawzi, O., et al. Mathematical discoveries from program search with large language models. *Nature*, 625 (7995):468–475, 2024.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Shojaee, P., Meidani, K., Gupta, S., Farimani, A. B., and Reddy, C. K. Llm-sr: Scientific equation discovery via programming with large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Suzgun, M., Yuksekgonul, M., Bianchi, F., Jurafsky, D., and Zou, J. 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)*, pp. 7080–7106, 2026.

Tang, X., Qin, T., Peng, T., Zhou, Z., Shao, D., Du, T., Wei, X., Xia, P., Wu, F., Zhu, H., et al. Agent kb: Leveraging cross-domain experience for agentic problem solving. *arXiv preprint arXiv:2507.06229*, 2025.

Toledo, E., Hambardzumyan, K., Josifoski, M., HAZRA, R., Baldwin, N., Audran-Reiss, A., Kuchnik, M., Magka, D., Jiang, M., Lupidi, A. M., 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.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. 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.

Wang, H., Skreta, M., Ser, C. T., Gao, W., Kong, L., Strieth-Kalthoff, F., Duan, C., Zhuang, Y., Yu, Y., Zhu, Y., et al. Efficient evolutionary search over chemical space with large language models. In *The Thirteenth International Conference on Learning Representations*, 2025a.

Wang, Z. Z., Mao, J., Fried, D., and Neubig, G. Agent workflow memory. In *Forty-second International Conference on Machine Learning*, 2025b. URL https://openreview.net/forum?id=NTAhi2JEEE.

Wei, J., Yang, Y., Zhang, X., Chen, Y., Zhuang, X., Gao, Z., Zhou, D., Wang, G., Gao, Z., Cao, J., et al. From ai for science to agentic science: A survey on autonomous scientific discovery. *arXiv preprint arXiv:2508.14111*, 2025.

Xiao, Y., Li, Y., Wang, H., Tang, Y., and Wang, Z. Z. Toolmem: Enhancing multimodal agents with learnable tool capability memory. *arXiv preprint arXiv:2510.06664*, 2025.

Yang, X., Yang, X., Fang, S., Xian, B., Li, Y., Wang, J., Xu, M., Pan, H., Hong, X., Liu, W., et al. R&d-agent: Automating data-driven ai solution building through llm-powered automated research, development, and evolution. *arXiv e-prints*, pp. arXiv–2505, 2025.

Zhao, A., Huang, D., Xu, Q., Lin, M., Liu, Y.-J., and Huang, G. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Zheng, B., Fatemi, M. Y., Jin, X., Wang, Z. Z., Gandhi, A., Song, Y., Gu, Y., Srinivasa, J., Liu, G., Neubig, G., et al. Skillweaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025.

Appendix for DrugSAGE

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) (Huang et al., 2021) and 11 held-out test datasets from the Polaris Hub (Ash et al., 2025).

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
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

Continued on next page...

Dataset	Description	Size	Metric
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
	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 (Karpathy, 2026) 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 (Green et al., 2026).

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 (Lange et al., 2025) 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 (Feng et al., 2026) 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 (Toledo et al., 2025) 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 (Anthropic, 2025) 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 (Jin et al., 2025) 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 (Martinek et al., 2026) 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*	**0.7739 ± 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	**0.2466 ± 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	**0.9936 ± 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	**0.3753 ± 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	**0.9519 ± 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	**0.7215 ± 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	**0.9377 ± 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	**7.11 ± 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	**0.7392 ± 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	**0.5237 ± 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	**0.8828 ± 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*	**0.8237 ± 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	**0.8232 ± 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	**0.6757 ± 0.0150**	0.6671 ± 0.0086
CYP3A4 Inh	AUPRC ↑	0.916 ± 0.000	**0.9247 ± 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	**0.6054 ± 0.0197**
Excretion												
CL Hepatocyte	Spearman ↑	**0.536 ± 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	**0.6399 ± 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*	**0.8794 ± 0.0026**
DILI	AUROC ↑	**0.956 ± 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	**0.8941 ± 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	**0.5019 ± 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 6 breaks this down and shows the best-so-far score on each individual held-out task as a function of experiment budget B.

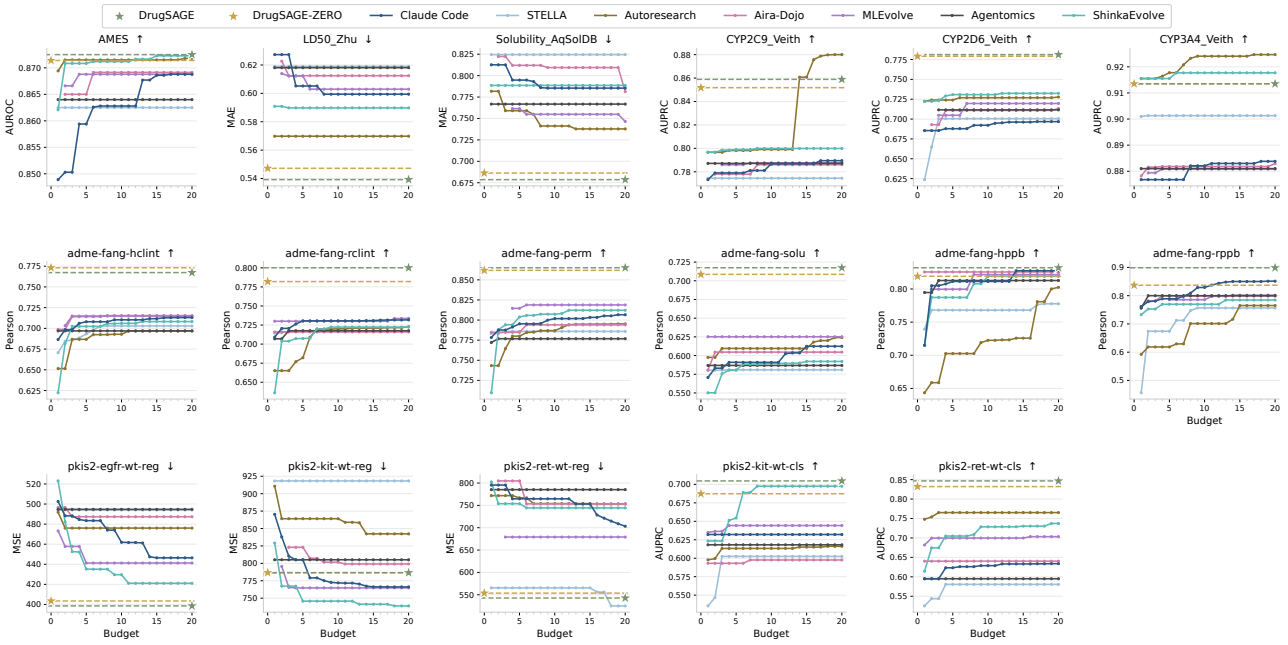

Figure 6. Per-task budget trajectories on all 17 held-out tasks: 6 TDC-ADMET tasks (top one row) and 11 Polaris tasks (bottom two rows).

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 ± 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 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 ± std over 5 seeds. *Matched task*: the task in $\mathcal{Z}$ the router selected.

Polaris task	Metric	Matched Task	Transferred Solution	DRUGSAGE-ZERO
ADME-Fang regression				
adme-fang-hclint	Pearson ↑	Lipophilicity_AstraZeneca	minimol + chemprop-rdkit + chemprop	0.7731 ± 0.0014
adme-fang-rclint	Pearson ↑	Lipophilicity_AstraZeneca	minimol + chemprop-rdkit + chemprop	0.7820 ± 0.0012
adme-fang-perm	Pearson ↑	Lipophilicity_AstraZeneca	minimol + chemprop-rdkit + chemprop	0.8618 ± 0.0047
adme-fang-solu	Pearson ↑	CYP2C9_Substrate_CarbonMangels	maplight-gnn + minimol + lantern-radr-ensemble	0.7086 ± 0.0108
adme-fang-hppb	Pearson ↑	Lipophilicity_AstraZeneca	minimol + chemprop-rdkit + chemprop	0.8192 ± 0.0072
adme-fang-rppb	Pearson ↑	Lipophilicity_AstraZeneca	minimol + chemprop-rdkit + chemprop	0.8370 ± 0.0123
PKIS2 kinase regression				
pkis2-egfr-wt-reg	MSE ↓	Lipophilicity_AstraZeneca	minimol + chemprop-rdkit + chemprop	403.46 ± 10.13
pkis2-kit-wt-reg	MSE ↓	CYP2C9_Substrate_CarbonMangels	maplight-gnn + minimol + lantern-radr-ensemble	786.63 ± 22.95
pkis2-ret-wt-reg	MSE ↓	CYP2C9_Substrate_CarbonMangels	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

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 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.** $w_{\text{type}} = \mathbf{1}\big[\text{type}(\tau) = \text{type}(\tau_0)\big]$ matches the task type (classification vs. regression);
- **Dataset size match.** $w_{\text{size}} = \exp\left(-\gamma \left|\log|\mathcal{D}_\tau| - \log|\mathcal{D}_{\tau_0}|\right|\right)$ penalizes dataset-size mismatch on a log scale, with decay rate $\gamma > 0$;
- **Description similarity.** $w_{\text{emb}} = \max\left(\cos(\mathbf{e}_\tau, \mathbf{e}_{\tau_0}),\ 0\right)$ 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:

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

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 $\text{matched}(v_i, \tau)$, and aggregate its standardized score:

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

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 $|\text{transfer}(f)| \leq 1$ and $|\text{transfer}(v_i)| \leq 1$. This directly satisfies the bounded-transfer assumption $|\text{transfer}(f)| \leq C_0$ 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 $\text{transfer}(f)$ 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:*

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

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

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

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

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

for some constant $C_0 > 0$.

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

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

J.2. Family-Level Selection

J.2.1. Setup and Regret Definition

The family-level UCB index is

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

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 (Hoeffding, 1963), 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,

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

Step 2: Necessary condition for selecting a suboptimal arm.

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

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

Rearranging:

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

Step 3: Upper bound on arm pull count.

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

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

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

Step 4: Expected number of pulls.

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

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

Step 5: Cumulative regret.

Summing over all suboptimal families:

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

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. □