MolMem: Memory-Augmented Agentic Reinforcement Learning for Sample-Efficient Molecular Optimization

Ziqing Wang[1] **Yibo Wen**[1] **Abhishek Pandey**[2] **Han Liu**[1] **Kaize Ding**[1]*
[1]Northwestern University [2]AbbVie
{ziqingwang2029, yibowen2024}@u.northwestern.edu
abhishek.pandey@abbvie.com
{hanliu, kaize.ding}@northwestern.edu

Abstract

In drug discovery, molecular optimization aims to iteratively refine a lead compound to improve molecular properties while preserving structural similarity to the original molecule. However, each oracle evaluation is expensive, making sample efficiency a key challenge for existing methods under a limited oracle budget. Trial-and-error approaches require many oracle calls, while methods that leverage external knowledge tend to reuse familiar templates and struggle on challenging objectives. A key missing piece is long-term memory that can ground decisions and provide reusable insights for future optimizations. To address this, we present MolMem (**Mol**ecular optimization with **Mem**ory), a multi-turn agentic reinforcement learning (RL) framework with a dual-memory system. Specifically, MolMem uses Static Exemplar Memory to retrieve relevant exemplars for cold-start grounding, and Evolving Skill Memory to distill successful trajectories into reusable strategies. Built on this memory-augmented formulation, we train the policy with dense step-wise rewards, turning costly rollouts into long-term knowledge that improves future optimization. Extensive experiments show that MolMem achieves 90% success on single-property tasks (1.5× over the best baseline) and 52% on multi-property tasks using only 500 oracle calls. Our code is available at https://github.com/REAL-Lab-NU/MolMem.

1 Introduction

Molecular optimization is a key step in drug discovery that iteratively refines a lead compound to improve molecular properties while preserving structural similarity to the original molecule (Plowright et al., 2012; Wesolowski and Brown, 2016; Wang et al., 2025d,c). Since each refinement relies on expensive oracle evaluations (e.g., wet-lab assays, high-fidelity simulators, or property predictors), **sample efficiency** becomes the core challenge (Gao et al., 2022; Guo and Schwaller, 2024): how to achieve strong optimization performance under a limited oracle budget?

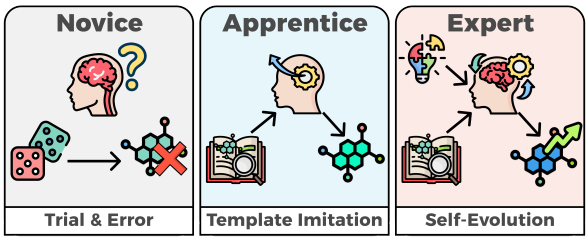


Figure 1: **The Evolution of Molecular Optimization Paradigms.** (1) **Novice**: trial-and-error exploration with low sample efficiency; (2) **Apprentice**: template imitation of external knowledge; (3) **Expert (Ours)**: self-evolution via grounded knowledge and consolidated skills.

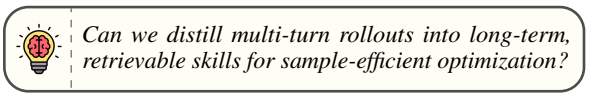
However, most existing methods still struggle with this sample-efficiency challenge. To understand why, we highlight two common paradigms in Fig. 1 and contrast them with how human experts work. ❶ **Novice** methods rely on trial-and-error search driven by oracle feedback, such as genetic algorithms and single-step reinforcement learning (Jensen, 2019; Olivecrona et al., 2017; Loeffler et al., 2024). Without prior knowledge, they typically require many oracle calls to reach good performance. ❷ **Apprentice** methods leverage external knowledge such as large offline datasets, pretrained models, or retrieved exemplars (Guo et al., 2023; Liu et al., 2024; Ye et al., 2025). They often start faster by imitating familiar transformation templates, but struggle when the task demands edits beyond those templates. ❸ Human experts, in contrast, ground decisions in relevant references and turn successful trials into reusable strategies. This ability to learn from experience is key to sample-efficient optimization.

Agentic reinforcement learning (RL) provides a natural framework for capturing this ability: an agent interacts with an environment over multiple turns and learns from feedback (Wang et al., 2025a;

*Corresponding Author

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 43694–43712
July 2-7, 2026

Zhang et al., 2025). Using large language models (LLMs) as the agent backbone makes this framework practical in this domain, since they can follow instructions and incorporate rich textual context to guide molecular edits (Wang et al., 2025d). However, this multi-turn agentic framework remains under-explored in molecular optimization. More importantly, most existing methods provide limited support for long-term memory: once a rollout ends, useful discoveries are lost and cannot be reused in future optimizations. Such memory is crucial for sample efficiency, since discoveries from one run can reduce oracle calls in the next. This leads to a key question:



We answer this question with `MolMem` (**Mol**ecular optimization with **Mem**ory), a multi-turn agentic RL framework with a dual-memory system (Fig. 2). Within each rollout, the agent uses the trajectory history as short-term context. To complement this, `MolMem` maintains two long-term memory components: ❶ **Static Exemplar Memory** retrieves relevant exemplars (e.g., structurally similar high-scoring molecules) to provide cold-start grounding when the agent lacks its own experience; ❷ **Evolving Skill Memory** distills successful trajectories into reusable strategies stored in a skill bank, enabling the agent to improve beyond directly copying exemplars. Long-term memory is queried only when optimization plateaus, encouraging the agent to explore independently before consulting retrieved guidance. To train the multi-turn policy effectively, we use dense step-wise rewards derived from oracle signals, enabling precise credit assignment across the trajectory. To summarize, our main contributions are as follows:

- **Framework.** We propose `MolMem`, a memory-augmented multi-turn agentic RL framework for sample-efficient molecular optimization.
- **Memory System.** We design a dual-memory system with distinct roles: Static Exemplar Memory for cold-start grounding, and Evolving Skill Memory for distilling successful trajectories into reusable strategies.
- **Effectiveness.** Experiments show that `MolMem` achieves **90%** success on single-property tasks (**1.5**× over best baseline) and **52%** on multi-property tasks using only **500** oracle calls.

2 Related Work

Molecular Optimization. To reduce the cost of wet-lab assays, researchers have proposed various computational methods for molecular optimization (Gao et al., 2022). Approaches include genetic algorithms (Jensen, 2019), Bayesian optimization (Korovina et al., 2020), and reinforcement learning for goal-directed generation (Popova et al., 2018; Olivecrona et al., 2017; Loeffler et al., 2024). However, the PMO benchmark shows that many of these methods struggle under realistic oracle budgets and often have difficulty balancing property improvement with structural similarity to the original lead (Gao et al., 2022). More recently, LLMs have been explored as flexible editors for interactive molecular design (Ye et al., 2025; Liu et al., 2024; Wang et al., 2024a; Rivera et al., 2026). Despite their flexibility, these methods are typically used in a single-step manner and do not systematically learn from full optimization trajectories, which limits cross-trajectory improvement under tight budgets. This motivates multi-turn formulations that optimize over trajectories with step-wise feedback.

Multi-Turn Agentic Reinforcement Learning. Multi-turn reinforcement learning provides a natural way to model sequential decision making with LLM agents (Du et al., 2023; Wang et al., 2024b; Ma et al., 2026; Hao et al., 2025; Zhang et al., 2026). It also aligns well with molecular optimization, where an optimizer proposes a sequence of edits under step-wise oracle feedback. Under the broader umbrella of agentic RL, LLMs are treated as policies that learn through interaction with an environment rather than as one-shot generators (Zhang et al., 2025). Recent systems such as RAGEN (Wang et al., 2025a) show that optimizing over full trajectories can improve long-horizon behavior. While multi-turn RL has advanced rapidly in language-based tasks, its use for sample-efficient molecular optimization remains relatively under-explored. Moreover, most multi-turn setups primarily use the trajectory history as short-term context: once an episode ends, useful intermediate insights are difficult to carry over to future runs (Wang et al., 2025c; Wang and Ding, 2018). This gap motivates mechanisms that can store and retrieve experience across trajectories.

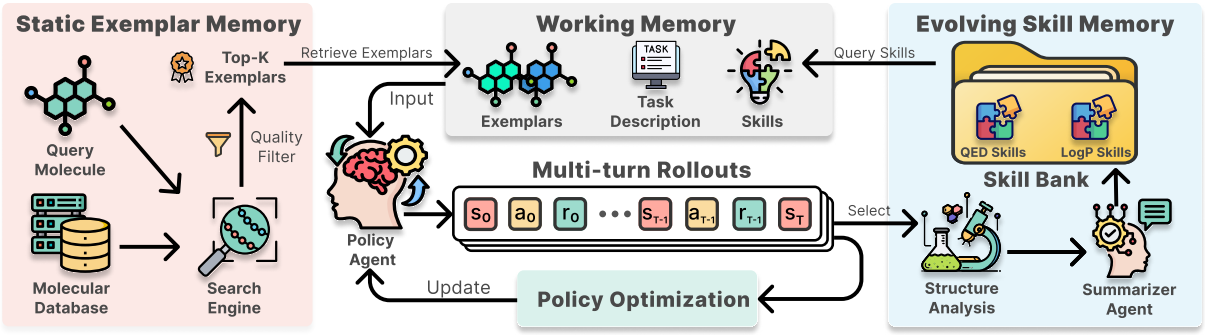


Figure 2: **Overview of the `MolMem`.** The framework consists of two complementary memory modules: **(Left) Static Exemplar Memory** retrieves structurally similar, high-scoring molecules from an external database as references for the current objective. **(Right) Evolving Skill Memory** distills successful trajectories into reusable textual strategies. Retrieved exemplars or skills are injected into the **Working Memory** (center) to augment the LLM agent's context during **Multi-turn Rollouts**.

Memory-Augmented Agents. A growing line of work equips LLM agents with memory to reuse information and experience across steps and episodes (Hu et al., 2025; Huang et al., 2026). In molecular design, memory is often implemented as static retrieval: systems such as ChatDrug (Liu et al., 2024) and DrugAssist (Ye et al., 2025) retrieve similar molecules or references from databases to guide editing, but this context is typically used as read-only guidance rather than being updated from optimization outcomes. Beyond molecular design, experience-centric memory has been explored across domains, including retrieval-augmented reasoning in clinical settings (Ou et al., 2025; Shen et al., 2026; Wang et al., 2025b). Moreover, Voyager builds a library of executable code skills from task feedback (Wang et al., 2023), and ExpeL distills successful trajectories into exemplars for future reuse (Zhao et al., 2024). Chemistry-focused agents have also begun to build self-updating libraries for general chemistry tasks (e.g., ChemAgent) (Tang et al., 2025). However, these ideas have not been tailored to molecular optimization, where an agent must improve properties while preserving structural similarity under tight oracle budgets. `MolMem` builds on this direction by combining static exemplar retrieval for grounding with trajectory-based skill distillation for reuse across runs.

3 Preliminary

3.1 Problem Formulation

Molecular optimization aims to refine a lead compound by proposing structural edits that improve one or more molecular properties under a limited oracle-call budget. Let $\mathcal{M}_{\text{mol}}$ denote the space of valid molecules. Given a lead molecule $m \in \mathcal{M}_{\text{mol}}$, the goal is to find an optimized molecule $m' \in \mathcal{M}_{\text{mol}}$ that solves:

$$\max_{m' \in \mathcal{M}_{\text{mol}}} \sum_{i=1}^{n} w_i F_i(m') \text{ s.t. } \begin{cases} \text{sim}(m, m') \geq \gamma, \\ \sum_{i=1}^{n} c_i \leq B, \end{cases} \tag{1}$$

where $F_i : \mathcal{M}_{\text{mol}} \to \mathbb{R}$ are black-box property oracles (e.g., binding affinity, solubility), w_i are their weights, $\text{sim}(\cdot, \cdot)$ denotes Tanimoto similarity with threshold γ to encourage structural similarity, c_i denotes the oracle call of evaluating F_i, and B is the total oracle-call budget.

3.2 Multi-Turn MDP Formulation

We model molecular optimization as a finite-horizon Markov Decision Process (MDP) in which an agent iteratively proposes candidate molecules and receives oracle evaluations after each turn. Formally, the MDP is defined as $\langle \mathcal{S}, \mathcal{A}, P, R \rangle$ with horizon T:

- **State $\mathcal{S}$.** At turn t, the state s_t contains the task objective and the optimization context, including the lead molecule m_0, the previously proposed molecules $(m_1, \ldots, m_t)$, and their reward $(r_0, \ldots, r_{t-1})$.
- **Action $\mathcal{A}$.** The action $a_t \sim \pi_\theta(\cdot \mid s_t)$ corresponds to proposing the next candidate molecule m_{t+1}, represented as a SMILES string.
- **Transition and reward P, R.** After taking action a_t, the environment evaluates the proposed molecule m_{t+1} with oracle(s) and returns a step reward $r_t = R(s_t, a_t)$ (Appendix E.2). The next state s_{t+1} is obtained by updating the history with (m_{t+1}, r_t). The transition is written as $s_{t+1} \sim P(\cdot \mid s_t, a_t)$.

This multi-turn formulation allows the agent to learn from intermediate feedback over a trajectory, rather than optimizing each edit in isolation.

4 Method

We present `MolMem`, a multi-turn agentic reinforcement learning (RL) approach for sample-efficient molecular optimization. Building on the MDP formulation in Section 3.2, we first describe the dual-memory system (Section 4.1) and then introduce the policy optimization procedure. (Section 4.2).

4.1 Agentic Memory System

In the multi-turn MDP, the trajectory history provides short-term context within a rollout. However, once a rollout ends, the agent has no mechanism to retain and reuse effective edits discovered along the way, so similar oracle calls may be repeated across rollouts. To support reuse for better sample efficiency, `MolMem` maintains a dual-memory system $\mathcal{M} = (\mathcal{M}^{\text{static}}, \mathcal{M}^{\text{evolve}})$.

4.1.1 Static Exemplar Memory

The static component $\mathcal{M}^{\text{static}}$ provides **cold-start grounding** by maintaining a large molecule bank constructed from ChEMBL (Zdrazil et al., 2024) (2.8M molecules) with precomputed physicochemical properties (e.g., QED, LogP). Each molecule is indexed by its Morgan fingerprint (ECFP4; radius $= 2$, 2048-bit) using FAISS (Johnson et al., 2019) for efficient similarity search.

To avoid over-reliance on external guidance, we trigger exemplar retrieval only when progress plateaus (e.g., no reward improvement for consecutive turns), rather than at every turn or from the start. This encourages independent exploration in early turns before incorporating exemplar-based grounding. Given the current molecule m_t at the triggered turn and the lead molecule m_0, retrieval follows a two-stage procedure:

Candidate Retrieval. We first perform approximate nearest-neighbor retrieval in fingerprint space to obtain a candidate set:

$$\mathcal{C}_t = \text{ANN}\big(\phi(m_t);\, \mathcal{M}^{\text{static}}\big)\,, \tag{2}$$

where $\phi(\cdot)$ denotes the ECFP4 fingerprint and ANN is implemented with FAISS.

Constrained Reranking. We then enforce lead similarity by filtering candidates with Tanimoto similarity to m_0, and return the top-K exemplars ranked by target property score:

$$c_t^{\text{static}} = \text{Top-}K_{\,m' \in \mathcal{C}_t:\ \text{sim}(m', m_0) \geq \gamma_{\text{ex}}}\ F_{\mathcal{Q}}(m'). \tag{3}$$

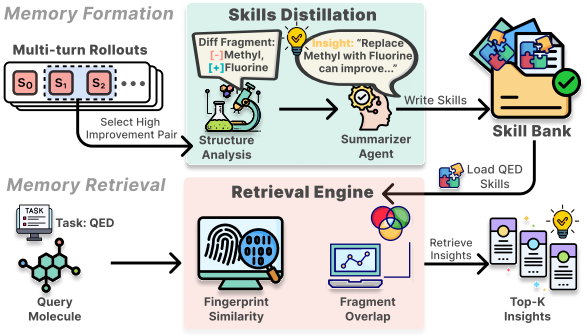


Figure 3: **Mechanism of the Evolving Skill Memory. (Top) Memory Formation**: High-reward pairs in multi-turn rollouts are distilled into textual skills by extracting structural changes and synthesizing insights via a summarizer agent. **(Bottom) Memory Retrieval**: Relevant skills are retrieved from the skill bank using a hybrid matching of fingerprint similarity and functional group overlap to guide the policy agent.

where $\text{sim}(\cdot,\cdot)$ is Tanimoto similarity, γ_{ex} is a similarity threshold, and $F_{\mathcal{Q}}(\cdot)$ is the task-specific score. This design uses m_t for broad retrieval while enforcing similarity to m_0, ensuring retrieved exemplars both explore relevant chemical space and remain similar to the lead.

Crucially, exemplars serve as *references* rather than targets to copy. We penalize exact copying by assigning a negative reward if the generated molecule is identical to any retrieved exemplar, discouraging rote replication and encouraging the agent to learn from structural patterns. Implementation details are provided in Appendix G.

4.1.2 Evolving Skill Memory

Unlike the static, external exemplar bank, the evolving component $\mathcal{M}^{\text{evolve}}$ grows a library of reusable strategies consolidated from the agent's own high-reward rollouts (Fig. 3). We implement this component via *memory formation* and *memory retrieval*.

Memory Formation. After each training iteration, we extract step-wise experiences from multi-turn rollouts. For each transition (m_t, m_{t+1}) with high reward improvement $\Delta r > \delta$ (where δ is a threshold), we build a structured *edit card* κ_t that summarizes the transformation, including: (i) MCS-based edit decomposition (modification type and removed/added fragments), (ii) scaffold analysis (before/after scaffold and scaffold type, e.g., scaffold hop), (iii) functional-group additions/removals, and (iv) cheap descriptor deltas (e.g., MW, PSA, HBD/HBA, ring count). A summarizer LLM then converts this card into a single actionable strategy sentence following a fixed *Action–What–Where–Effect* template, e.g., *"Replace methoxy (-*

OCH_3) with fluorine (-F) on the aromatic ring to improve the target score."

$$e = \mathrm{LLM}_{\mathrm{summarizer}}(\kappa_t,\ \Delta r,\ \mathcal{Q})\,. \tag{4}$$

Each skill e is stored in a task-specific bank $\mathcal{M}_{\mathcal{Q}}^{\mathrm{evolve}}$ and indexed by the molecule m_t's Morgan fingerprint and a set of functional-group tags.

Memory Retrieval. Given the current molecule m_t and objective $\mathcal{Q}$, we query $\mathcal{M}_{\mathcal{Q}}^{\mathrm{evolve}}$ via two parallel matching signals. Fingerprint matching uses Tanimoto similarity over Morgan fingerprints, and functional-group matching uses Jaccard similarity over functional-group sets. Each skill e is stored with its pre-edit source molecule $m(e)$ and functional-group tags $G(e)$; we compute $\mathrm{sim}_{\mathrm{FP}}(m_t, e) = \mathrm{sim}(\phi(m_t), \phi(m(e)))$ and $\mathrm{sim}_{\mathrm{FG}}(m_t, e) = J(G(m_t), G(e))$. After threshold filtering, we return top-K_{fp} and top-K_{fg} skills respectively:

$$\begin{aligned} c_t^{\mathrm{fp}} &= \text{Top-}K_{\mathrm{fp}}\{e \in \mathcal{M}_{\mathcal{Q}}^{\mathrm{evolve}} : \mathrm{sim}_{\mathrm{FP}}(m_t, e) \geq \gamma_{\mathrm{fp}}\}\,, \\ c_t^{\mathrm{fg}} &= \text{Top-}K_{\mathrm{fg}}\{e \in \mathcal{M}_{\mathcal{Q}}^{\mathrm{evolve}} : \mathrm{sim}_{\mathrm{FG}}(m_t, e) \geq \gamma_{\mathrm{fg}}\}\,, \end{aligned} \tag{5}$$

and set $c_t^{\mathrm{sk}} = c_t^{\mathrm{fp}} \cup c_t^{\mathrm{fg}}$. Among candidates that pass the threshold, we rank skills by their improvement Δr, ensuring retrieved skills are not only structurally relevant but also empirically effective.

Similar to exemplar retrieval, skill retrieval is triggered only when optimization plateaus, encouraging the agent to first explore independently before consulting accumulated experience. Retrieved skills are injected into the agent's working memory as high-level guidance, enabling cross-rollout reuse of successful edit principles without additional oracle evaluations. Implementation details are in Appendix H.

4.1.3 Memory-Augmented Rollouts

As described above, exemplar and skill retrieval are triggered only when optimization plateaus. If both trigger conditions are satisfied simultaneously, we stochastically select one memory source to keep the LLM context budget and prevent over-reliance on a single modality. The retrieved items are assembled into the policy input (working memory), forming the memory-augmented state:

$$s_t = \big(\mathcal{Q},\ \mathcal{H}_t \oplus c_t^{\mathrm{mem}}\big), \tag{6}$$

where $\mathcal{Q}$ is the task objective, $\mathcal{H}_t = (m_0, m_1, \ldots, m_t; r_0, \ldots, r_{t-1})$ is the trajectory history, $\oplus$ denotes context concatenation, and $c_t^{\mathrm{mem}} \in \{c_t^{\mathrm{static}}, c_t^{\mathrm{evolve}}, \varnothing\}$ denotes the injected memory depending on trigger status. This design combines within-rollout context with cross-rollout knowledge via on-demand retrieval.

4.2 Memory-Augmented Policy Optimization

Given the memory-augmented state formulation (Eq. 6), we train `MolMem` in two stages:

Stage I: Supervised Fine-Tuning. We initialize the policy π_θ via supervised fine-tuning (SFT) on an offline dataset of molecular edit pairs that satisfy the similarity constraint. This warm start teaches the model to propose valid edits and provides a stable prior before RL (details in Appendix D).

Stage II: Memory-Aware Policy Optimization. Building on the SFT-initialized policy, we apply Proximal Policy Optimization (PPO) (Schulman et al., 2017) to further refine the agent's multi-turn decision making. Unlike single-turn methods that treat each edit independently, our formulation optimizes over entire trajectories while providing *dense step-wise feedback*: the agent receives an immediate reward r_t after each modification, enabling precise credit assignment rather than relying solely on terminal outcomes. Given a rollout τ collected by the policy $\pi_{\theta_{\mathrm{old}}}$, we optimize:

$$\begin{aligned} J_{\mathrm{PPO}}(\theta) &= \mathbb{E}_{\tau \sim \pi_{\theta_{\mathrm{old}}}}\left[\sum_{t=0}^{T-1} L_t(\theta)\right], \\ L_t(\theta) &= \min\Big(\rho_t \hat{A}_t,\ \mathrm{clip}(\rho_t, 1-\epsilon, 1+\epsilon)\hat{A}_t\Big), \end{aligned} \tag{7}$$

where $\rho_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t|s_t)}$ is the importance ratio and $\hat{A}_t$ is computed via GAE (Schulman et al., 2015) from the dense reward sequence $(r_t, r_{t+1}, \ldots)$, which balances property improvement with similarity preservation (details in Appendix E.2). Since the state s_t (Eq. 6) incorporates retrieved memory when triggered (Section 4.1), training naturally teaches the policy to interpret and leverage memory guidance for long-term improvement.

5 Experiments

5.1 Experimental Setup

Baselines. We compare `MolMem` with baselines from two *baseline* paradigms (Fig. 1): **(1) Novice:** Graph-GA (Jensen, 2019), QMO (Hoffman et al., 2022), and Reinvent 4 (Loeffler et al., 2024); **(2) Apprentice:** (i) **Direct Retrieval**, a retrieval-only variant that retrieves molecules using the same

Table 1: **Single-Property Optimization Results.** Methods are categorized by optimization paradigm: Novice, Apprentice, and Expert. SR (%) denotes the success rate, Sim denotes the Tanimoto similarity to the original molecule (we require Sim ≥ 0.4), and RI denotes relative improvement. For SR, **gold** = best, silver = second, and bronze = third.

Method	Backbone	QED			plogP			SA			DRD2			JNK3		
		SR (%)	Sim	RI	SR (%)	Sim	RI	SR (%)	Sim	RI	SR (%)	Sim	RI	SR (%)	Sim	RI
❶ Novice																
Graph-GA	N/A	59.5	0.49	0.13	61.5	0.49	9.64	46.0	0.48	0.20	34.0	0.56	5.13	6.5	0.59	1.73
QMO	GRU	16.5	0.53	0.16	8.0	0.55	5.87	6.5	0.52	0.15	9.5	0.51	3.91	0.5	0.56	1.20
Reinvent 4	Transformer	33.5	0.50	0.14	51.5	0.48	10.37	18.5	0.47	0.21	34.5	0.50	8.71	28.0	0.51	1.91
❷ Apprentice																
Retrieval-based																
Direct Retrieval	N/A	68.5	0.43	0.21	53.5	0.43	13.13	38.0	0.44	0.30	28.5	0.43	8.10	21.0	0.44	2.45
Prompting & SFT																
Direct Prompt	Qwen2.5-1.5B	53.0	0.66	0.18	38.0	0.65	12.48	13.0	0.69	0.16	13.5	0.64	4.92	19.5	0.66	2.23
	Qwen2.5-3B	67.5	0.63	0.20	52.5	0.63	14.65	34.0	0.66	0.27	25.5	0.68	8.14	31.5	0.64	1.79
SFT-only	Qwen2.5-1.5B	37.0	0.62	0.14	24.5	0.64	9.09	8.0	0.62	0.13	20.0	0.62	7.04	8.5	0.63	0.72
	Qwen2.5-3B	48.0	0.61	0.17	23.5	0.63	9.67	10.5	0.62	0.16	23.0	0.64	6.70	10.5	0.64	0.81
Task-Specific LLM																
MOLLEO	BioT5	15.0	0.68	0.08	12.5	0.70	3.37	8.0	0.81	0.02	4.0	0.78	0.63	10.0	0.88	0.04
LlaSMol	Mistral-7B	38.0	0.45	0.14	32.5	0.49	9.98	20.5	0.50	0.19	15.5	0.47	5.94	6.0	0.44	0.71
ChemLLM	ChemLLM-7B	69.0	0.65	0.21	61.5	0.64	19.86	35.5	0.69	0.27	46.0	0.64	11.71	44.0	0.65	2.48
GeLLM^{3}O	Llama3.1-8B	61.5	0.56	0.19	57.0	0.52	12.80	14.5	0.57	0.16	49.0	0.56	11.16	8.5	0.55	1.19
PEIT-LLM	Llama3.1-8B	82.5	0.47	0.23	81.5	0.49	15.41	45.0	0.50	0.31	50.5	0.50	11.86	41.0	0.50	2.51
❸ Expert																
`MolMem` **(Ours)**	Qwen2.5-1.5B	**91.0**	0.47	0.24	**100.0**	0.47	19.44	**63.5**	0.45	0.34	**96.0**	0.49	16.96	**98.5**	0.47	8.87

static retriever as `MolMem`; (ii) **Direct Prompt** with general-purpose LLMs using the same instruction prompt as `MolMem` for fair comparison; (iii) **SFT-only**, using the same supervised data and recipe as Stage I but without RL; (iv) **Task-Specific LLMs:** MOLLEO (Wang et al., 2024a), LlaSMol (Yu et al., 2024), ChemLLM (Zhang et al., 2024), PEIT-LLM (Lin et al., 2024), and GeLLM^{3}O (Dey et al., 2025). Direct Retrieval and SFT-only also serve as ablations of `MolMem`. Notably, `MolMem` uses a compact **Qwen2.5-1.5B** backbone, while most task-specific LLM baselines use 7–8B parameters. Details of these baselines are in Appendix B.

Tasks and Constraints. We evaluate on five single-property tasks (QED, plogP, SA, DRD2, JNK3) and five multi-property tasks (QED+plogP, plogP+DRD2, QED+SA, DRD2+SA, DRD2+QED+plogP) using 200 lead molecules randomly sampled from ZINC-250k (Irwin and Shoichet, 2005). We enforce two practical constraints: (1) a Tanimoto similarity threshold of $\gamma = 0.4$ to preserve similarity to the lead, and (2) an oracle-call budget of $B = 500$ per lead molecule.

Evaluation Metrics. Following prior work (Dey et al., 2025), we report three complementary metrics: **(1) Success Rate (SR):** the percentage of leads that meet the task-specific success criterion while satisfying the similarity constraint; **(2) Similarity (Sim):** the average Tanimoto similarity between the lead and the final optimized molecule; **(3) Relative Improvement (RI):** the average relative improvement from the lead to the final molecule over the target property. Detailed definitions and task-specific success criteria are provided in Appendix C.

Agent Engine. We use GPT-4o as the skill summarizer in `MolMem` (Appendix H).

5.2 Single-Property Optimization

Table 1 reports five single-property tasks. `MolMem` achieves the highest success rate across all tasks while maintaining comparable similarity.

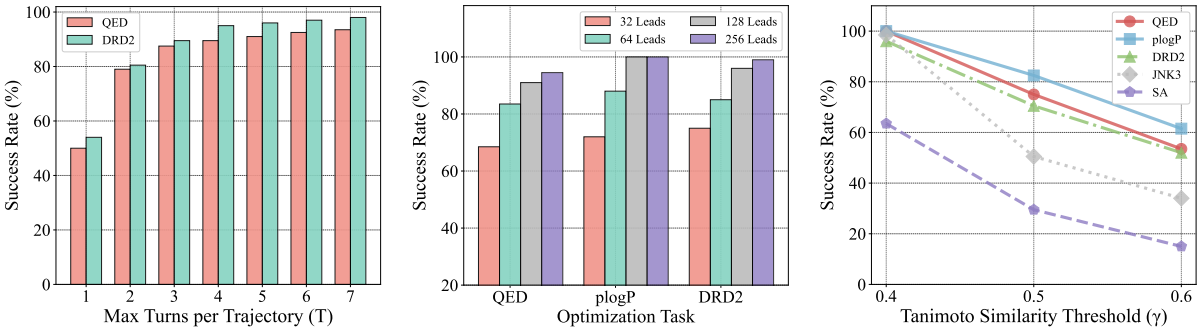
Largest gains emerge on bioactivity targets. Compared with strong task-specific LLM baselines, `MolMem` improves SR from 82.5% to 91.0% on QED and from 81.5% to 100.0% on plogP. The gap widens on bioactivity targets: 50.5% to 96.0% on DRD2 and 44.0% to 98.5% on JNK3. These tasks require precise navigation of structure-activity relationships, suggesting that `MolMem` is more effective on such complex landscapes.

Smaller backbone, larger gains. Despite using a compact Qwen2.5-1.5B backbone, `MolMem` outperforms much larger baselines (ChemLLM 7B, GeLLM^{3}O 8B, PEIT-LLM 8B). The multi-turn framework compensates for limited model capacity by allowing iterative refinement, while the dual-memory system supplies chemical knowledge to retrievable exemplars and accumulated skills, reducing the need to encode such knowledge purely in model parameters.

Retrieval provides cold-start, learning enables scaling. Direct Retrieval performs well on QED

Table 2: **Multi-Property Optimization Results.** Methods are categorized by optimization paradigm: Novice, Apprentice, and Expert. SR (%) denotes the success rate, Sim denotes the Tanimoto similarity to the original molecule (we require Sim ≥ 0.4), and RI denotes relative improvement. For SR (Success Rate), **gold** = best, silver = second, bronze = third.

Method	Backbone	QED+plogP			plogP+DRD2			QED+SA			DRD2+SA			DRD2+QED+plogP		
		SR (%)	Sim	RI	SR (%)	Sim	RI	SR (%)	Sim	RI	SR (%)	Sim	RI	SR (%)	Sim	RI
❶ Novice																
Graph-GA	N/A	8.0	0.50	4.11	2.5	0.48	6.61	8.0	0.52	0.12	0.0	0.53	4.15	0.0	0.51	3.79
QMO	GRU	3.0	0.52	6.40	2.0	0.51	4.20	1.5	0.53	0.11	1.0	0.50	2.49	0.0	0.52	1.14
Reinvent 4	Transformer	6.0	0.49	8.60	50.0	0.48	7.94	17.0	0.66	0.05	37.5	0.50	5.54	3.0	0.51	4.28
❷ Apprentice																
Retrieval-based																
Direct Retrieval	N/A	24.0	0.43	4.51	21.5	0.44	7.52	31.0	0.43	0.25	23.5	0.43	2.64	8.0	0.43	3.72
Prompting & SFT																
Direct Prompt	Qwen2.5-1.5B	10.0	0.58	3.49	8.0	0.55	8.98	24.0	0.49	0.22	6.0	0.59	3.59	2.0	0.54	4.60
	Qwen2.5-3B	12.5	0.54	4.85	7.0	0.57	8.58	21.5	0.55	0.16	5.0	0.58	3.36	1.0	0.58	3.68
SFT-only	Qwen2.5-1.5B	14.0	0.60	3.46	11.5	0.62	7.16	20.0	0.59	0.13	6.5	0.62	2.74	1.0	0.61	2.61
	Qwen2.5-3B	18.0	0.59	2.89	7.5	0.62	7.86	21.0	0.60	0.14	7.0	0.61	2.41	1.5	0.61	2.83
Task-Specific LLM																
LlaSMol	Mistral-7B	15.0	0.49	3.58	8.0	0.46	9.16	21.0	0.48	0.19	16.0	0.47	3.79	1.5	0.47	3.59
ChemLLM	ChemLLM-7B	11.5	0.51	6.36	8.0	0.53	8.06	20.5	0.51	0.21	22.0	0.55	5.40	0.0	0.52	2.85
GeLLM^{3}O	Llama3.1-8B	19.5	0.60	2.89	16.0	0.58	7.41	22.5	0.57	0.14	9.0	0.59	2.76	0.0	0.61	2.92
PEIT-LLM	Llama3.1-8B	12.5	0.44	5.95	25.0	0.47	12.82	41.5	0.44	0.26	34.5	0.46	5.61	4.0	0.45	5.51
❸ Expert																
`MolMem` **(Ours)**	Qwen2.5-1.5B	**58.0**	0.49	12.12	**54.0**	0.48	13.78	**64.5**	0.47	0.24	**69.5**	0.48	7.11	**15.0**	0.46	6.26



(a) Impact of Trajectory Length (T) (b) Impact of Training Data Size (c) Impact of Similarity Constraint (γ)

Figure 4: **Analysis of key hyperparameters.** (a) Success rate improves with more turns and saturates around T=5. (b) Performance improves with more training leads, though even 64 leads yield competitive results. (c) Stricter similarity constraints reduce the success rate.

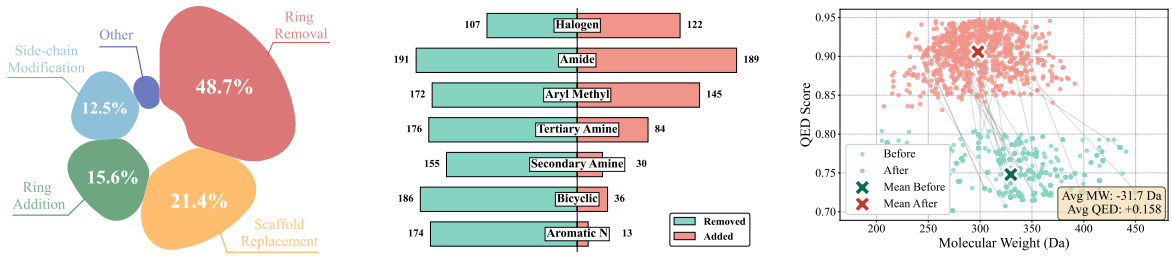
(68.5% SR) but struggles on bioactivity targets (28.5% on DRD2 and 21.0% on JNK3), where relevant exemplars are scarce. `MolMem` sustains improvement by learning from multi-turn RL and distilling successful edits into reusable skills, moving beyond exemplar copying.

5.3 Multi-Property Optimization

Table 2 reports five multi-property tasks. `MolMem` achieves the highest success rate on all tasks, with particularly large margins on combinations involving bioactivity targets. For example, on DRD2+SA, `MolMem` reaches 69.5% SR compared to 37.5% for a strong baseline (Reinvent 4). The three-property task (DRD2+QED+plogP) proves challenging for all methods, yet `MolMem` still leads with 15.0% versus 8.0% (Direct Retrieval). These results suggest that the benefits of memory-augmented multi-turn optimization extend from single-property to multi-property settings, especially when the objective involves complex bioactivity constraints.

5.4 Analysis of Key Hyperparameters

Figure 4 analyzes the impact of critical hyperparameters on optimization performance. **(1) Importance of multi-turn optimization.** As shown in Figure 4a, allowing the agent to optimize over multiple turns yields substantial improvements over single-turn baselines. Performance gains saturate around T=5, indicating that this range provides an effective balance between optimization quality and computational cost. **(2) Data efficiency.** Figure 4b shows that performance scales with training data size, but even 64 leads yield competitive results across tasks. This efficiency stems from the dual-memory system, which allows the agent to accumulate and reuse successful strategies across rollouts, amplifying the utility of limited training



(a) Scaffold Modification Types (b) Functional Group Flux (c) Chemical Space Trajectories

Figure 5: **What the skill bank learns (QED task).** (a) Distribution of scaffold modifications, where ring removal is the most frequent operation. (b) Functional-group flux suggests a tendency to remove amine groups while adding halogen substitutions. (c) Optimization trajectories indicate that QED improvements often coincide with reductions in molecular weight.

data. **(3) Robustness to similarity constraints.** Figure 4c shows that stricter similarity thresholds reduce success rates across all tasks, with harder targets (SA, JNK3) showing larger drops. Nevertheless, `MolMem` maintains reasonable performance, demonstrating the robustness of `MolMem`.

5.5 What the Skill Bank Learns

To interpret the strategies captured by the evolving skill memory, we analyze learned skills on the QED task (Figure 5). The skill bank is dominated by scaffold-level transformations: ring removal accounts for 48.7% of edits, followed by scaffold replacement (21.4%) and ring addition (15.6%) (Figure 5a). This tendency toward simplification is consistent with common drug-likeness guidelines that favor avoiding overly large or complex structures (Lipinski, 2000; Bickerton et al., 2012). At the functional-group level, the learned skills more often remove amine-related motifs while slightly favoring halogen additions, with amide changes remaining roughly balanced (Figure 5b). Such edits reflect standard medicinal-chemistry levers used to tune drug-like properties during optimization (Gleeson, 2008). Consistent with these patterns, optimization trajectories shift toward lower molecular weight and higher QED (Avg MW -31.7 Da; Avg QED $+0.158$) (Figure 5c). Overall, the skill bank captures reusable structure-level heuristics that move candidates toward more QED-favorable regions of chemical space.

5.6 Ablation Study

We conduct an ablation study to quantify the contribution of each component, as summarized in Table 3. **(1) Multi-turn optimization is foundational.** Restricting the agent to single-turn optimization (T=1) substantially degrades success rate, from 91.0% to 50.0% on QED and from 96.0% to 54.0% on DRD2, highlighting the importance of iterative refinement with intermediate feedback. **(2) RL training is essential beyond a supervised prior.** Removing RL and using SFT-only reduces SR to 37.0% (QED) and 20.0% (DRD2), indicating that multi-turn optimization is necessary to realize consistent gains under a fixed oracle budget. **(3) SFT initialization is particularly critical for bioactivity.** Without SFT warm-start, DRD2 SR drops from 96.0% to 34.5%, suggesting that learning effective edits for challenging targets benefits from a strong chemical prior. **(4) Memory components are complementary.** Removing either Static Exemplar Memory or Evolving Skill Memory reduces SR on both tasks, and removing both causes the larger degradation, supporting that cold-start grounding and consolidated strategies provide additive benefits.

Table 3: **Ablation Study of Component Contributions.**

Model Configuration	QED			DRD2		
	SR (%)	Sim	RI	SR (%)	Sim	RI
`MolMem` (Full)	**91.0**	**0.47**	**0.24**	**96.0**	**0.49**	**16.97**
— Ablation of Training Components —						
w/o SFT Initialization	68.0	0.54	0.19	34.5	0.58	8.64
w/o Multi-turn (T=1)	50.0	0.55	0.19	54.0	0.58	11.39
w/o RL Training (SFT-only)	37.0	0.62	0.14	20.0	0.62	7.04
— Ablation of Memory Components —						
w/o Static Exemplar Memory	81.0	0.52	0.20	88.5	0.49	15.56
w/o Evolving Skill Memory	83.0	0.52	0.21	85.5	0.49	14.78
w/o Both Memories	80.0	0.51	0.20	81.0	0.50	14.12

6 Conclusion

We presented `MolMem`, a memory-augmented multi-turn agentic RL framework for sample-efficient molecular optimization. In our dual-memory system, Static Exemplar Memory provides cold-start grounding, while Evolving Skill Memory distills successful trajectories into reusable optimization strategies. Experiments demonstrate strong performance on both single-property (90% SR) and multi-property (52% SR) tasks using only 500 oracle calls, substantially outperforming existing methods. An analysis of the learned skill bank further

suggests that `MolMem` captures chemically meaningful patterns (e.g., frequent ring removal and systematic amine reduction) that are consistent with established medicinal chemistry heuristics. We believe this work takes a step toward more practical molecular optimization agents.

7 Limitations

First, our evaluation relies on computational oracles. These surrogates, while widely used in molecular optimization benchmarks, may not fully reflect outcomes measured by wet-lab assays or other higher-fidelity evaluations. Second, skill summarization currently depends on an external LLM (GPT-4o) to convert structured edit cards into natural language strategies. While effective, this introduces additional inference cost. Integrating summarization directly into the RL training loop in an end-to-end fashion is an important next step.

Acknowledgments

This work was supported in part by the NVIDIA Academic Grant Program. We thank NVIDIA for providing the computational resources that enabled this research.

References

G Richard Bickerton, Gaia V Paolini, Jérémy Besnard, Sorel Muresan, and Andrew L Hopkins. 2012. Quantifying the chemical beauty of drugs. *Nature chemistry*, 4(2):90–98.

Ziqi Chen, Martin Renqiang Min, Srinivasan Parthasarathy, and Xia Ning. 2021. A deep generative model for molecule optimization via one fragment modification. *Nature machine intelligence*, 3(12):1040–1049.

Vishal Dey, Xiao Hu, and Xia Ning. 2025. Generalizing large language models for multi-property molecule optimization. *arXiv preprint arXiv:2502.13398*.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first International Conference on Machine Learning*.

Wenhao Gao, Tianfan Fu, Jimeng Sun, and Connor Coley. 2022. Sample efficiency matters: a benchmark for practical molecular optimization. *Advances in neural information processing systems*, 35:21342–21357.

M Paul Gleeson. 2008. Generation of a set of simple, interpretable admet rules of thumb. *Journal of medicinal chemistry*, 51(4):817–834.

Jeff Guo and Philippe Schwaller. 2024. Augmented memory: sample-efficient generative molecular design with reinforcement learning. *Jacs Au*, 4(6):2160–2172.

Taicheng Guo, Bozhao Nan, Zhenwen Liang, Zhichun Guo, Nitesh Chawla, Olaf Wiest, Xiangliang Zhang, et al. 2023. What can large language models do in chemistry? a comprehensive benchmark on eight tasks. *Advances in Neural Information Processing Systems*, 36:59662–59688.

Zhezheng Hao, Hong Wang, Haoyang Liu, Jian Luo, Jiarui Yu, Hande Dong, Qiang Lin, Can Wang, and Jiawei Chen. 2025. Rethinking entropy interventions in rlvr: An entropy change perspective. *arXiv preprint arXiv:2510.10150*.

Samuel C Hoffman, Vijil Chenthamarakshan, Kahini Wadhawan, Pin-Yu Chen, and Payel Das. 2022. Optimizing molecules using efficient queries from property evaluations. *Nature Machine Intelligence*, 4(1):21–31.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, et al. 2025. Memory in the age of ai agents. *arXiv preprint arXiv:2512.13564*.

Wei-Chieh Huang, Weizhi Zhang, Yueqing Liang, Yuanchen Bei, Yankai Chen, Tao Feng, Xinyu Pan, Zhen Tan, Yu Wang, Tianxin Wei, et al. 2026. Rethinking memory mechanisms of foundation agents in the second half. *arXiv preprint arXiv:2602.06052*.

John J Irwin and Brian K Shoichet. 2005. Zinc- a free database of commercially available compounds for virtual screening. *Journal of chemical information and modeling*, 45(1):177–182.

Jan H Jensen. 2019. A graph-based genetic algorithm and generative model/monte carlo tree search for the exploration of chemical space. *Chemical science*, 10(12):3567–3572.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547.

Ksenia Korovina, Sailun Xu, Kirthevasan Kandasamy, Willie Neiswanger, Barnabas Poczos, Jeff Schneider, and Eric Xing. 2020. Chembo: Bayesian optimization of small organic molecules with synthesizable recommendations. In *International Conference on Artificial Intelligence and Statistics*, pages 3393–3403. PMLR.

Xuan Lin, Long Chen, Yile Wang, Xiangxiang Zeng, and Philip S Yu. 2024. Property enhanced instruction tuning for multi-task molecule generation with large language models. *arXiv preprint arXiv:2412.18084*.

Christopher A Lipinski. 2000. Drug-like properties and the causes of poor solubility and poor permeability. *Journal of pharmacological and toxicological methods*, 44(1):235–249.

Shengchao Liu, Jiongxiao Wang, Yijin Yang, Chengpeng Wang, Ling Liu, Hongyu Guo, and Chaowei Xiao. 2024. Conversational drug editing using retrieval and domain feedback. In *The twelfth international conference on learning representations*.

Hannes H Loeffler, Jiazhen He, Alessandro Tibo, Jon Paul Janet, Alexey Voronov, Lewis H Mervin, and Ola Engkvist. 2024. Reinvent 4: modern ai–driven generative molecule design. *Journal of Cheminformatics*, 16(1):20.

Shichao Ma, Zhiyuan Ma, Ming Yang, Xiaofan Li, Xing Wu, Jintao Du, Yu Cheng, Weiqiang Wang, Qiliang Liu, Zhengyang Zhou, et al. 2026. Tspo: Breaking the double homogenization dilemma in multi-turn search policy optimization. *arXiv preprint arXiv:2601.22776*.

Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. 2017. Molecular de-novo design through deep reinforcement learning. *Journal of cheminformatics*, 9:1–14.

Justice Ou, Tinglin Huang, Yilun Zhao, Ziyang Yu, Peiqing Lu, and Rex Ying. 2025. Experience retrieval-augmentation with electronic health records enables accurate discharge qa. *Preprint*, arXiv:2503.17933.

Alleyn T Plowright, Craig Johnstone, Jan Kihlberg, Jonas Pettersson, Graeme Robb, and Richard A Thompson. 2012. Hypothesis driven drug design: improving quality and effectiveness of the design-make-test-analyse cycle. *Drug discovery today*, 17(1-2):56–62.

Mariya Popova, Olexandr Isayev, and Alexander Tropsha. 2018. Deep reinforcement learning for de novo drug design. *Science advances*, 4(7):eaap7885.

Oscar Rivera, Ziqing Wang, Matthieu Dagommer, Abhishek Pandey, and Kaize Ding. 2026. Glassmol: Interpretable molecular property prediction with concept bottleneck models. *arXiv preprint arXiv:2603.01274*.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Yifei Shen, Yilun Zhao, Justice Ou, Tinglin Huang, and Arman Cohan. 2026. Patient-similarity cohort reasoning in clinical text-to-SQL. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1367–1412, Rabat, Morocco. Association for Computational Linguistics.

Xiangru Tang, Tianyu Hu, Muyang Ye, Yanjun Shao, Xunjian Yin, Siru Ouyang, Wangchunshu Zhou, Pan Lu, Zhuosheng Zhang, Yilun Zhao, et al. 2025. Chemagent: Self-updating library in large language models improves chemical reasoning. *arXiv preprint arXiv:2501.06590*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Haorui Wang, Marta Skreta, Yuanqi Du, Wenhao Gao, Lingkai Kong, Cher Tian Ser, Felix Strieth-Kalthoff, Chenru Duan, Yuchen Zhuang, Yue Yu, et al. 2024a. Efficient evolutionary search over chemical space with large language models. In *ICML 2024 AI for Science Workshop*.

Qineng Wang, Zihao Wang, Ying Su, Hanghang Tong, and Yangqiu Song. 2024b. Rethinking the bounds of llm reasoning: Are multi-agent discussions the key? *arXiv preprint arXiv:2402.18272*.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, et al. 2025a. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*.

Ziqing Wang and Kaize Ding. 2018. Remol: Llm-guided molecular optimization with reinforcement learning.

Ziqing Wang, Chengsheng Mao, Xiaole Wen, Yuan Luo, and Kaize Ding. 2025b. Amanda: Agentic medical knowledge augmentation for data-efficient medical visual question answering. *arXiv preprint arXiv:2510.02328*.

Ziqing Wang, Yibo Wen, William Pattie, Xiao Luo, Weimin Wu, Jerry Yao-Chieh Hu, Abhishek Pandey, Han Liu, and Kaize Ding. 2025c. Polo: Preference-guided multi-turn reinforcement learning for lead optimization. *arXiv preprint arXiv:2509.21737*.

Ziqing Wang, Kexin Zhang, Zihan Zhao, Yibo Wen, Abhishek Pandey, Han Liu, and Kaize Ding. 2025d. A survey of large language models for text-guided molecular discovery: from molecule generation to optimization. *arXiv preprint arXiv:2505.16094*.

Steven S Wesolowski and Dean G Brown. 2016. The strategies and politics of successful design, make, test, and analyze (dmta) cycles in lead generation. *Lead Generation*, pages 487–512.

Geyan Ye, Xibao Cai, Houtim Lai, Xing Wang, Junhong Huang, Longyue Wang, Wei Liu, and Xiangxiang Zeng. 2025. Drugassist: A large language model for

molecule optimization. *Briefings in Bioinformatics*, 26(1):bbae693.

Botao Yu, Frazier N Baker, Ziqi Chen, Xia Ning, and Huan Sun. 2024. Llasmol: Advancing large language models for chemistry with a large-scale, comprehensive, high-quality instruction tuning dataset. *arXiv preprint arXiv:2402.09391*.

Barbara Zdrazil, Eloy Felix, Fiona Hunter, Emma J Manners, James Blackshaw, Sybilla Corbett, Marleen De Veij, Harris Ioannidis, David Mendez Lopez, Juan F Mosquera, et al. 2024. The chembl database in 2023: a drug discovery platform spanning multiple bioactivity data types and time periods. *Nucleic acids research*, 52(D1):D1180–D1192.

Di Zhang, Wei Liu, Qian Tan, Jingdan Chen, Hang Yan, Yuliang Yan, Jiatong Li, Weiran Huang, Xiangyu Yue, Wanli Ouyang, et al. 2024. Chemllm: A chemical large language model. *arXiv preprint arXiv:2402.06852*.

Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhongzhi Li, Xiangyuan Xue, Yijiang Li, et al. 2025. The landscape of agentic reinforcement learning for llms: A survey. *arXiv preprint arXiv:2509.02547*.

Peixuan Zhang, Chang Zhou, Ziyuan Zhang, Hualuo Liu, Chunjie Zhang, Jingqi Liu, Xiaohui Zhou, Xi Chen, Shuchen Weng, Si Li, and Boxin Shi. 2026. A benchmark and multi-agent system for instruction-driven cinematic video compilation. *arXiv preprint arXiv:2604.10456*.

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

Appendix Table of Contents

A LLM Usage

We used Large Language Models (ChatGPT/-Claude/Gemini) exclusively for grammatical correction in this manuscript. The LLMs played no role in research ideation, methodology, or scientific content generation. All technical contributions and scientific insights are original work by the authors.

B Baselines and Implementation Details

B.1 Baseline Methods

We compare `MolMem` with baselines that cover two common optimization paradigms in Fig. 1: **(i) Novice** trial-and-error search and **(ii) Apprentice** methods that rely on external knowledge (retrieval, pretrained LLMs, or supervised fine-tuning).

Novice baselines.

- **Graph-GA** (Jensen, 2019) is a graph-based genetic algorithm that iteratively mutates and recombines molecules with chemistry-aware operators, using oracle scores as selection signals.
- **QMO** (Hoffman et al., 2022) performs black-box optimization in the latent space of a pretrained molecular autoencoder, updating candidates via zeroth-order gradient estimates from oracle evaluations.
- **Reinvent 4** (Loeffler et al., 2024) is an on-policy RL method that trains a SMILES generator (e.g., Transformer) to maximize a scoring function under reward shaping.

Apprentice baselines.

- **Direct Retrieval** uses the same static retriever as `MolMem` to retrieve structurally similar molecules from the external database and selects the best candidate under the task objective, subject to the similarity constraint. This baseline isolates the effect of retrieval without learning.
- **Direct Prompt** queries general-purpose instruction-tuned LLMs using the same task instruction as `MolMem` and treats each proposal independently.
- **SFT-only** uses the same supervised data and training recipe as Stage I of `MolMem`, but does not apply RL. At test time, it generates candidates by prompting the fine-tuned model without memory or policy optimization.
- **Task-specific LLMs** include MOLLEO (Wang et al., 2024a), LlaSMol (Yu et al., 2024), ChemLLM (Zhang et al., 2024), PEIT-LLM (Lin et al., 2024), and GeLLM^{3}O (Dey et al., 2025). These models are designed or fine-tuned for chemistry and/or molecular optimization settings. We follow the evaluation protocols described in their respective papers when applicable, while enforcing the same oracle-call budget and similarity constraint as in our setup.

B.2 Implementation Details

Budget and constraints. For each lead molecule, all methods are evaluated under the same oracle-call budget $B{=}500$ and the same similarity constraint (Tanimoto similarity ≥ 0.4). An oracle call is counted whenever a candidate molecule is evaluated by the property oracle(s). We report the best feasible molecule found within the budget. For fair comparison, all methods are evaluated with the same constraints and budget, and we apply identical success criteria and metrics across methods (Appendix C).

Table 4: Task-specific success criteria. For multi-property tasks, $\Delta F = F(m') - F(m)$ is measured relative to the lead molecule m.

Property	Single-property success	Multi-property success
QED	$F_{\text{QED}}(m') \geq 0.9$	$\Delta F_{\text{QED}} \geq 0.1$
plogP	$F_{\text{plogP}}(m') \geq 2.0$	$\Delta F_{\text{plogP}} \geq 1.0$
JNK3	$F_{\text{JNK3}}(m') \geq 0.1$	$\Delta F_{\text{JNK3}} \geq 0.1$
DRD2	$F_{\text{DRD2}}(m') \geq 0.8$	$\Delta F_{\text{DRD2}} \geq 0.5$
SA	$F_{\text{SA}}(m') \leq -2.5$	$\Delta F_{\text{SA}} \leq -0.5$

Traditional methods. For Graph-GA, QMO, and Reinvent 4, we use official implementations when available and otherwise use widely adopted public re-implementations. We keep their default hyperparameters, and we modify the optimization loop to explicitly track oracle calls so that each method respects the same budget.

LLM-based methods. For Direct Prompt and task-specific LLM baselines, we use a shared prompting template (Appendix D) and a common parsing and validation pipeline. Each model produces candidate SMILES strings, which are canonicalized and checked for validity. Valid candidates are then evaluated by the oracle and counted toward the budget. We repeat the generation until the budget is exhausted. We use the temperature τ=0.9 for sampling.

C Evaluation Metrics Details

C.1 Success Rate (SR)

Success Rate (SR) is the percentage of lead molecules for which the method finds at least one *feasible* optimized molecule m' within the oracle budget. A molecule is feasible if it satisfies the similarity constraint $\text{sim}(m, m') \geq 0.4$ and the task-specific success criterion in Table 4. For multi-property tasks, a run is counted as successful only if *all* target properties meet their respective thresholds.

C.2 Similarity (Sim)

We report the average Tanimoto similarity between each lead molecule m_i and the best molecule returned by the method m_i':

$$\text{Sim} = \frac{1}{N} \sum_{i=1}^{N} \text{sim}(m_i, m_i'), \tag{8}$$

where N=200. If a method fails to return any valid candidate that satisfies the constraints for lead m_i, we set $m_i' = m_i$ for that instance. This convention avoids dropping failures when aggregating results. As a consequence, Sim should be interpreted jointly with SR, since a method that frequently fails may obtain a high Sim by defaulting to the lead molecule.

C.3 Relative Improvement (RI)

Relative Improvement (RI) measures the average normalized improvement over the target properties. For a task with n properties, we compute:

$$\text{RI} = \frac{1}{n} \sum_{j=1}^{n} \text{sgn}(w_j) \cdot \frac{F_j(m') - F_j(m)}{|F_j(m)|}, \tag{9}$$

where $\text{sgn}(w_j) = +1$ for properties to maximize (QED, plogP, JNK3, DRD2) and $\text{sgn}(w_j) = -1$ for properties to minimize (SA). The absolute value in the denominator handles properties that may take negative values (e.g., plogP). If optimization fails (i.e., no feasible molecule is found within the budget), we set RI to 0 for that instance.

D Details of Supervised Fine-Tuning

This section describes the supervised fine-tuning (SFT) stage used to initialize the policy in `MolMem`.

D.1 Training Data Construction

We follow the task setup of GeLLM^{3}O (Dey et al., 2025) and build SFT pairs from the molecule-pair dataset of Chen et al. (Chen et al., 2021). Each example is a pair (M_x, M_y) that corresponds to a local, single-fragment modification. We filter pairs to encourage similarity-preserving edits by retaining only those with Tanimoto similarity ≥ 0.6. For each retained pair, we compute task-relevant properties (QED, plogP, JNK3, DRD2, SA) and keep examples that show a meaningful improvement for the specified objective. Each pair is then converted into an instruction–response example by using M_x as the input molecule and M_y as the target output.

D.2 Fine-Tuning Setup

We fine-tune **Qwen2.5-1.5B-Instruct** using LoRA (Hu et al., 2022) for parameter-efficient adaptation. Unless otherwise stated, we use LoRA rank r=16 and α=32, and train for **10 epochs**. The resulting model is used as the SFT initialization for the policy π_θ in `MolMem`.

D.3 Prompt Template

We use the unified prompt template below for SFT data formatting and for LLM-based baselines to ensure consistent task specification.

Unified Prompt Template

```
You are an expert medicinal chemist specializing in
molecular optimization. You understand how structural
modifications affect key molecular properties including
drug-likeness, lipophilicity, synthetic accessibility, and
target inhibition activities.

Your task is to modify the given molecule to adjust the
specified molecular properties while keeping structural
changes as minimal as possible. The modified molecule
should maintain a structural similarity of at least 0.6
with the original molecule.

Input molecule: <SMILES> {input_smiles} </SMILES>
Requested modifications: {property_description}

Please provide the optimized molecule in SMILES format,
wrapped in <SMILES> </SMILES> tags.
```

The `{property_description}` field is instantiated according to the task objective:

- QED: increase drug-likeness (QED)
- LogP: increase lipophilicity (LogP)
- JNK3/DRD2: increase inhibition probability
- SA: decrease synthetic accessibility score (lower is better)
- Multi-property: combine the above objectives with conjunctions

D.4 Why high-similarity pairs for SFT.

In Stage I, our goal is to teach the LLM the SMILES syntax and to perform controlled, local edits rather than large scaffold jumps. We therefore construct SFT examples from molecule pairs with high structural overlap (Tanimoto similarity ≥ 0.6). This design encourages the model to learn "small but valid" modifications that preserve the core structure. As a consequence, the **SFT-only** baseline tends to produce outputs with consistently high similarity in evaluation: the model is trained to stay close to the input molecule, which naturally raises Sim even when property improvements are limited.

E Environment and Reward Details

E.1 Environment Interface and Termination

We formulate molecular optimization as a multi-turn interaction between an LLM agent and a molecule-editing environment. At turn t, the agent proposes a candidate molecule m_{t+1} in SMILES form; the environment parses and validates it, evaluates oracle properties, and returns a scalar reward r_t together with textual feedback for the next turn.

Algorithm 1 Step-wise reward computation.

Require: current molecule m_t; proposed molecule (SMILES) $\tilde{m}_{t+1}$; similarity threshold γ; target property oracle F with direction $\text{sgn}(w_F) \in \{+1, -1\}$

Ensure: reward r_t

Parse $\tilde{m}_{t+1}$ into a molecule m_{t+1}; if parsing fails, return $r_t \leftarrow -0.5$
Canonicalize m_t and m_{t+1}; if $m_{t+1} = m_t$ (no-op), return $r_t \leftarrow -0.3$
Compute similarity $\text{sim} \leftarrow \text{sim}(m_0, m_{t+1})$; if $\text{sim} < \gamma$, return $r_t \leftarrow -2(\gamma - \text{sim})$
Compute property change $\Delta F \leftarrow F(m_{t+1}) - F(m_t)$
if $\text{sgn}(w_F) \cdot \Delta F > 0$ **then**
    $r_t \leftarrow 5|\Delta F|$ ▷ property improves
else
    $r_t \leftarrow -|\Delta F|$ ▷ property degrades
end if

A rollout terminates when any of the following conditions is met: (1) the maximum trajectory length T is reached; (2) a molecule satisfying the task-specific success criterion is found (under the similarity constraint).

E.2 Reward Computation

The reward is designed to (i) discourage invalid or trivial edits, (ii) enforce the similarity constraint, and (iii) provide dense, step-wise learning signals from oracle feedback. Algorithm 1 summarizes the computation.

We use asymmetric scaling so that genuine improvements receive a stronger learning signal than small degradations, which helps avoid overly conservative behavior under a strict similarity constraint.

E.3 Implementation Details.

Tanimoto similarity is computed using Morgan fingerprints (radius $= 2$, 2048 bits). When the task involves minimizing a property (e.g., SA), we set $\text{sgn}(w_F) = -1$ so that decreases count as improvements. All oracle evaluations are performed only after passing validity and similarity checks. We count one oracle call whenever a *valid* candidate molecule passes parsing/canonicalization and is evaluated by the property oracle(s). Invalid SMILES and similarity-violating candidates are rejected before oracle evaluation and therefore do not consume the oracle-call budget.

F Hyperparameter Settings

F.1 Training Configuration

We train `MolMem` in two stages. We first apply supervised fine-tuning (SFT) to Qwen2.5-1.5B-Instruct with LoRA (r=16, α=32) for 10 epochs to teach valid, similarity-preserving SMILES edits. Starting from the SFT checkpoint, we further optimize the multi-turn policy using PPO for 100 update steps with learning rate 5×10^{-5}, mini-batch size 32, and standard clipping/GAE settings (ϵ=0.2, γ_{rl}=0.99, λ=0.95). Each PPO iteration collects rollouts with horizon T=5 from 128 training leads, with 16 rollouts per lead and a similarity constraint γ=0.4. All experiments are run on 2×H100 GPUs with maximum sequence length 4096.

F.2 Evaluation Configuration

At test time, each lead molecule is optimized under a fixed oracle-call budget of B=500 and the same similarity threshold γ=0.4. We run a search procedure for G=20 iterations. In each iteration, the agent samples N=32 rollouts with horizon T=5 to propose candidate molecules, and we keep the best feasible molecule seen so far as the current incumbent. To encourage diversity when improvements slow down, we use a temperature schedule across iterations: $\tau_g = \min(\tau_0 + g\Delta\tau,\ \tau_{\max})$ with τ_0=0.9, $\Delta\tau$=0.1, and $\tau_{\max}$=2.0.

G Static Exemplar Memory Implementation

Database Construction. We build the static exemplar bank from ChEMBL (Zdrazil et al., 2024) (2.8M molecules). For each molecule, we precompute and store (i) oracle-relevant properties (QED, LogP, SA, and target activity scores such as JNK3/DRD2), (ii) an ECFP4 fingerprint (radius = 2, 2048-bit) normalized for FAISS L2 search, and (iii) a binary fingerprint for fast Tanimoto computation. We store the metadata in SQLite and build a FAISS IVF index (nlist= 1689), with an index size of ∼22GB.

Retrieval Pipeline. At turn t, we retrieve exemplars based on the current molecule m_t, while enforcing the similarity constraint with respect to the original lead m_0:

1. **ANN recall.** Query FAISS with the normalized ECFP4 fingerprint of m_t to obtain a candidate pool.
2. **Lead-based filtering.** Compute Tanimoto similarity between each candidate and the lead m_0 (using binary fingerprints) and keep only those satisfying the lead similarity constraint.
3. **Objective-aware ranking.** Rank the remaining candidates by the target objective and return the top-K molecules as exemplars.

This setup uses m_t to stay in a relevant neighborhood, and uses m_0 to respect the lead-preserving constraint.

When retrieval is triggered. We do not retrieve at every turn. Instead, retrieval is triggered only when the optimization stalls: if the agent fails to improve the target objective for *two consecutive turns*, we query the exemplar bank and provide a small set of high-scoring, lead-similar references.

How exemplars are presented to the agent. Retrieved exemplars are appended to the observation as a compact reference block:

Retrieved Template

```
=== SIMILAR HIGH-SCORING MOLECULES FOR REFERENCE ===
Here are K similar molecules with high target scores
(higher is better):

1. SMILES: CC1=CC=C(C=C1)NC(=O)C2=CC=CC=C2
    target score: 0.892
    Similarity to original lead: 0.654

2. SMILES:
    ...
    ...

Learn from structural patterns, but do not copy directly.
```

Efficiency Notes. In our optimized implementation, retrieving the nearest-neighbor candidate set for a given query molecule takes about **4 ms** (measured with the FAISS index resident on GPU). To keep retrieval fast, we (i) load the FAISS index on GPU, (ii) cache fingerprints to avoid recomputation, (iii) compute Tanimoto similarity in batch with precomputed binary fingerprints, and (iv) tune IVF search parameters (e.g., `nprobe`) to balance latency and recall.

H Evolving Skill Memory Implementation

Skill Representation. We store each learned strategy as a *skill card*. A card contains one short, reusable instruction in natural language, together with lightweight evidence so it can be retrieved and trusted later. Concretely, each card records:

- **Skill text**: one actionable sentence (Sec. 4.1), written in the form *[Action] [What] [Where (if clear)] to [Effect]*.

Table 5: Training hyperparameters for `MolMem`.

Category	Parameter	Value
Backbone & SFT	Base model	Qwen2.5-1.5B-Instruct
	SFT epochs	10
	LoRA rank r	16
	LoRA α	32
PPO	Training steps	100
	PPO learning rate	5×10^{-5}
	PPO minibatch size	32
	Clip ratio ϵ	0.2
	Discount factor γ_{rl}	0.99
	GAE λ	0.95
	Micro batch size / GPU	2
	Max sequence length	4096
Rollouts & Env	Max turns per rollout T	5
	Training leads per iteration	128
	Rollouts per lead	16
	Similarity threshold γ	0.4
	GPUs	2×H100

Table 6: Inference hyperparameters for `MolMem`.

Category	Parameter	Value
Budget & constraints	Oracle-call budget per lead B	500
	Similarity threshold γ	0.4
	Search iterations (generations) G	20
Candidate generation	Rollouts per iteration N	32
	Max turns per rollout T	5
Decoding diversity	Base temperature τ_0	0.9
	Temperature increment $\Delta\tau$	0.1
	Max temperature $\tau_{\max}$	2.0

- **Source edit**: the SMILES pair (m_t, m_{t+1}) and the observed improvement signal.
- **Edit summary**: an MCS-based decomposition of what changed (added/removed/replaced fragments), plus scaffold and functional-group changes detected by RDKit.
- **Retrieval keys**: an ECFP4 fingerprint and a functional-group tag set for the *source* molecule.

Skill acquisition from rollouts. During training, we harvest skills from multi-turn rollouts by keeping only *meaningful improving edits*. For each step, we compare the oracle score before and after the edit and keep transitions that yield a clear improvement. For bookkeeping, we also merge duplicates (e.g., identical SMILES pairs or near-identical edit patterns) and retain the best instance.

Turning edits into a reusable sentence. The structured edit information above is useful for retrieval, but it is not convenient for an LLM agent to apply directly. We therefore use an external summarizer (GPT-4o) to rewrite each retained edit into *one* strategy sentence. The summarizer is given the before/after molecules and the detected fragment / functional-group changes, and is asked to produce a concise, actionable rule. We use the following prompt to convert each structured edit card into one reusable strategy sentence.

Prompt Template of Summarizer Agent

```
SUMMARIZER_PROMPT = """Analyze this molecular
transformation for {task} optimization:

=== Molecules ===
Before: {before_smiles}
After:  {after_smiles}
Score:  {score_before:.3f} -> {score_after:.3f}
({score_delta:+.3f})

=== MCS Analysis ===
Modification: {modification_type}
- Removed: {removed_fragment}
- Added:   {added_fragment}

=== Scaffold Analysis ===
Before Scaffold: {before_scaffold}
After Scaffold:  {after_scaffold}
Scaffold Type: {scaffold_type}

=== Functional Group Changes ===
- Removed: {fg_removed}
- Added:   {fg_added}

=== Property Changes ===
- MW: {mw_change:+.1f} Da | Rings: {ring_changes:+d}
- PSA: {psa_change:+.1f} A2 | HBD: {hbd_change:+d} | HBA:
{hba_change:+d}
```

```
Result: {result}

=== Task ===
Generate ONE actionable strategy sentence following this
format:

CONSTRAINTS:
1. Focus on the 1-2 MOST IMPORTANT functional group changes
2. Format: "[Action] [What] [Where (if clear)] to [Effect]"
3. If scaffold_type is "scaffold_hop", mention the core
change (e.g., "Replace benzene with pyridine").
4. Use the Removed/Added Fragment from MCS for precise
description.
5. If location is clear from MCS, specify it (e.g., "on the
aromatic ring").

EXAMPLES:
- "Replace benzene core with pyridine to improve water
solubility and {task}." (scaffold_hop)
- "Add fluorine (-F) to the aromatic ring to enhance
metabolic stability." (addition)
- "Remove the sulfonamide group from aromatic ring to
reduce polar surface area." (removal)
- "Replace methoxy (-OCH3) with fluorine (-F) to decrease
MW and improve {task}." (replacement)

Focus on: WHAT changed, WHERE (if clear from MCS), and WHY
it improves {task}:"""
```

Example outputs look like:

- "Replace an aromatic methoxy with fluorine to reduce MW and improve QED."
- "Remove a tertiary amine side chain to reduce polarity while keeping the core scaffold."

Skill Retrieval. At test time, we retrieve skills that are relevant to the current molecule in two complementary ways: (1) *structure-based* retrieval using fingerprint similarity, and (2) *feature-based* retrieval using overlap in functional-group tags. We then take a small set of top matches and present them as high-level hints.

When skills are used. Similar to exemplar retrieval, we do not inject skills at every turn. If the agent makes no progress for *two consecutive turns*, we retrieve a few relevant skills and add them to the next observation.

How retrieved skills are presented. Retrieved skills are appended as a compact hint block:

Retrieved Skills

```
=== Potential Useful Strategies for qed ===
1. Replace benzene core with pyridine to improve water
solubility and qed.
2. Add fluorine (-F) to the aromatic ring to enhance
metabolic stability.
3. Remove the sulfonamide group from aromatic ring to
reduce polar surface area.
```

Capacity Control. We cap the skill bank at 1,000 entries to keep it actively refreshed. When new skills are added beyond this limit, we apply a simple survival-of-the-fittest rule: we rank candidate skill cards by their improvement magnitude (score delta Δ) and retain the top 1,000, discarding the rest. This favors high-impact skills while continuously removing weaker or redundant ones.

Summarizer Cost. Skill summarizer uses GPT-4o and incurs additional inference overhead. In our runs, summarizing skills for a single optimization task costs on the order of $10 in API usage.

I Computational Cost Analysis

Table 7 summarizes the required GPU time of representative baselines and `MolMem` on 2×H100. A main difference is what can be reused across different molecules. Offline instruction tuning is typically a one-time cost, while online RL methods (like Reinvent 4) must be re-trained for each new lead molecule.

For `MolMem`, training has two parts. We first run a one-time SFT stage (10 hours) that teaches similarity-preserving SMILES edits and is reused across all tasks. We then run task-specific policy optimization (4 hours) for each objective. Evaluation on 200 test leads with an oracle-call budget of 500 per lead takes about 3 hours per task with batched rollouts.

Table 7: **Computational cost comparison.** GPU hours are reported on 2×H100. *Online RL baseline.

Method	Model Size	One-time Training	Per-task Training	Per-task Inference
Reinvent 4*	–	–	14h (online)	–
GeLLM^{3}O	8B	48h (SFT)	–	1h
`MolMem`	1.5B	10h (SFT)	4h	3h

Overall, `MolMem` keeps per-task cost modest: after a reusable SFT checkpoint, task adaptation requires a short policy-optimization phase and batched multi-turn inference. Using a 1.5B backbone also lowers the hardware barrier compared to 7–8B LLM baselines.

J Case Studies

We present two types of case studies. First, Fig. 6 shows representative *skill cards* distilled from successful transitions. These examples highlight that the skill bank captures insightful, actionable edit patterns rather than memorizing entire molecules. Second, we include a full multi-turn rollout example to illustrate how `MolMem` uses retrieved skills and exemplars.

DRD2 Skill 1

Skill: Replace the imidazole tail with a propyl–fluorophenyl group to reduce polarity and improve DRD2 affinity.

Before:

`N#Cc1c(F)cccc1N1CCN(CC2=CN=CNC2)CC1`

After:

`N#Cc1c(F)cccc1N1CCN(CCCc2ccc(F)cc2)CC1`

Score: 0.01 → 0.99

Note: Hydrophobic tailoring: A polar heterocycle is replaced by a more hydrophobic fluorophenyl tail, which is consistent with improved fit to hydrophobic regions in DRD2 ligands.

DRD2 Skill

Skill: Replace the small dimethylamine group with a fluorophenethyl–piperazine tail to introduce a DRD2-relevant pharmacophore.

Before:

`CN(C)Cc1ccc(N=Nc2ccc3c(c2)CCCN3C)cc1`

After:

`Cc1ccc(N=Nc2ccc3c(c2)CCCN3C)cc1N1CCN(CCc2ccc(F)cc2)CC1`

Score: 0.03 → 1.00

Note: Pharmacophore grafting: Adding a fluorophenethyl–piperazine tail introduces a stronger hydrophobic/basic motif, which is consistent with the score jump observed for DRD2.

QED Skill 1

Skill: Replace the furan ring with a cyclopropane group to reduce polar surface area and improve QED.

Before:

`Cc1ccoc1C(=O)NCc1nc(-c2ccccc2)n[nH]1`

After:

`CC1(C)CC1C(=O)NCc1nc(-c2ccccc2)n[nH]1`

Score: 0.769 → 0.894

Note: Bioisosteric swap: Replacing a heteroaromatic furan with an sp^3 cyclopropane reduces PSA (-13.14) while keeping the rest of the scaffold unchanged.

QED Skill 2

Skill: Replace the fluorobenzene (-C6H4F) group with an isopropyl group (-CH(CH3)2) to reduce molecular weight.

Before:

`CCn1c(=O)n(CC(=O)NCc2ccc(F)c(C)c2)c2ccccc21`

After:

`CCn1c(=O)n(CC(=O)NCC(C)C)c2ccccc21`

Score: 0.775 → 0.901

Note: Side-chain simplification: Removing an aromatic side chain reduces molecular weight (-66 Da), a change that often correlates with higher QED and is reflected in the score increase here.

Figure 6: **Case studies of learned skills.** (Top) DRD2 examples highlighting tail substitution and motif augmentation. (Bottom) QED examples highlighting bioisosteric replacement and side-chain simplification. Each card shows the distilled strategy, the before/after edit, and the corresponding score change.

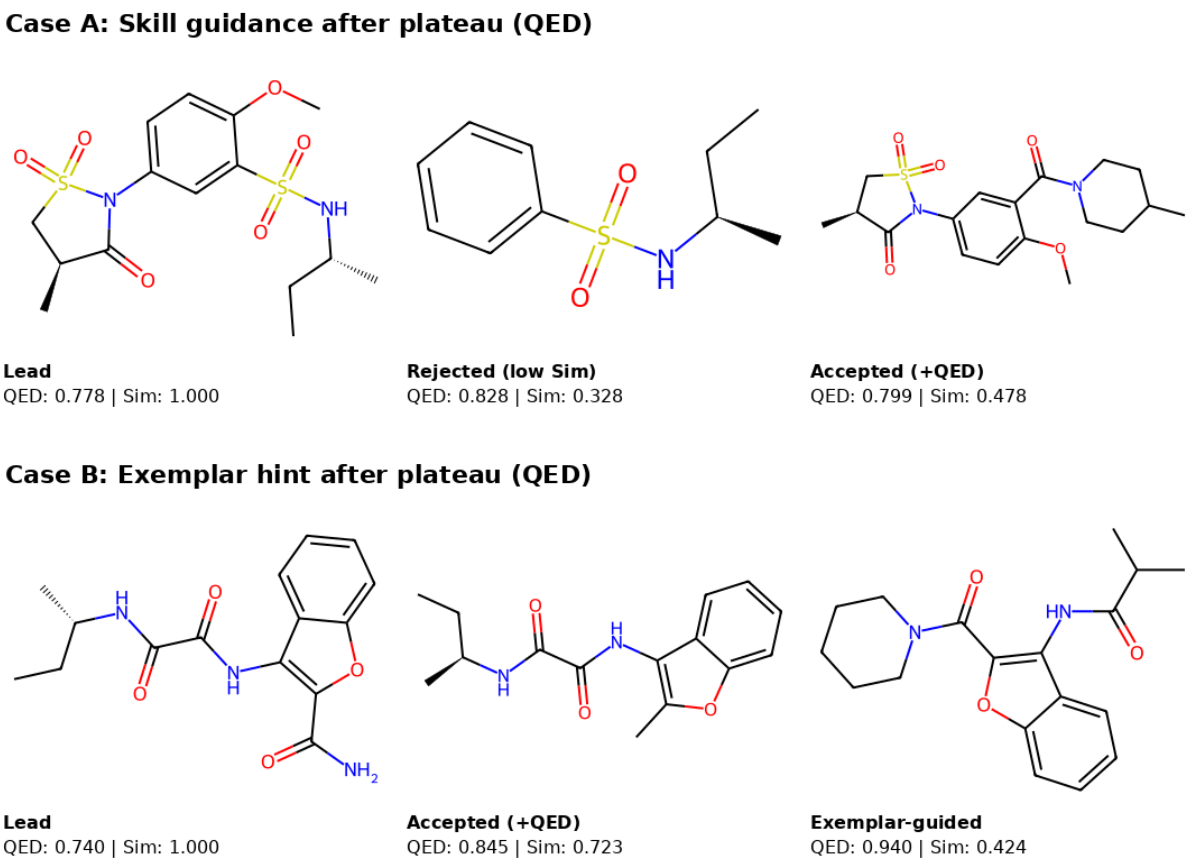


Figure 7: **Memory-triggered case studies.** (Top) A QED run where an early proposal violates the similarity constraint; after the agent plateaus, retrieved guidance leads to a feasible edit that improves QED. (Bottom) A QED run where exemplar hints provide a viable direction for subsequent edits. Each panel reports the lead molecule, representative intermediate proposals, and the resulting QED and similarity to the lead.