AFMOMO: AN AGENTIC FRAMEWORK FOR EFFICIENT MULTI-OBJECTIVE MOLECULAR OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-objective molecular optimization requires a single candidate to satisfy several property and structural constraints simultaneously. Existing search methods often rely heavily on random exploration, with limited integration of chemical prior knowledge and limited control over which constraints an edit should address. This can waste oracle evaluations when property assessment is expensive. We present **AFMOMO**, an agentic framework that couples objective-aware planning, heuristic molecular genetic search, and feedback-driven memory. First, an LLM planner uses objective satisfaction states to select parents, editing operators, and generation-budget allocations. Second, a staged genetic search executes these plans through chemically annotated mutation rules and recombination operators for local refinement, crossover, and broader exploration. Finally, search feedback updates candidate memory and operator weights and enables conditional branch rollback when a search branch regresses and stalls. Across four tasks and two LLM backbones, AFMOMO improves joint success rates over MOMO and MOMO-QMO. With Qwen, it reduces the task-averaged recorded oracle count by 81.13% relative to MOMO on leads successfully optimized by both methods.

1 INTRODUCTION

Multi-objective molecular optimization seeks a molecule that simultaneously meets several property and structural requirements (Nicolaou & Brown, 2013). In lead optimization, these requirements often include retaining similarity to a starting molecule. The objectives can conflict: an edit that improves one property may degrade another (Segall, 2012). Success therefore depends on coordinating structural changes so that all constraints hold for the *same* candidate. With a limited oracle budget, an optimizer must decide both which constraint to address next and which molecular edit is worth evaluating.

Evolutionary search provides mutation, recombination, and selection mechanisms for exploring chemical space. MOMO, for example, combines latent-space perturbations with Pareto-based selection (Xia et al., 2024). Such selection is informed by objective scores, but the perturbations themselves offer limited control over which unmet constraint to target or which chemical transformation to apply. Search can consequently spend evaluations improving already adequate properties or undoing previous progress. This motivates integrating chemical prior knowledge into proposal generation, alongside the emphasis on sample efficiency in molecular optimization benchmarks (Gao et al., 2022).

LLMs offer a way to interpret molecular structures, reason about competing objectives, and select editing tools (Wang et al., 2025; Zhou et al., 2026; Li et al., 2026; Gao et al., 2026). Their usefulness depends on connecting these decisions to executable transformations and subsequent feedback. Direct molecular rewriting can produce invalid structures (Zhou et al., 2026); tool execution must also be coordinated across parents and edit scales. The central design question is how to turn objective-aware reasoning into a sequence of controlled genetic edits that can be revised as search progresses.

We address this question with **AFMOMO**, an agentic framework built around three coupled components. *Objective-aware planning* exposes each candidate's satisfied constraints and remaining gaps to an LLM, which selects parents, target objectives, operators, and generation-budget allocations. *Heuristic molecular genetic search* executes this plan using an annotated operator library spanning local mutation, fragment-level editing, and recombination. Our operator design combines 26 chemical transformation rules with BRICS and SELFIES operations in a four-stage procedure, allowing the

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

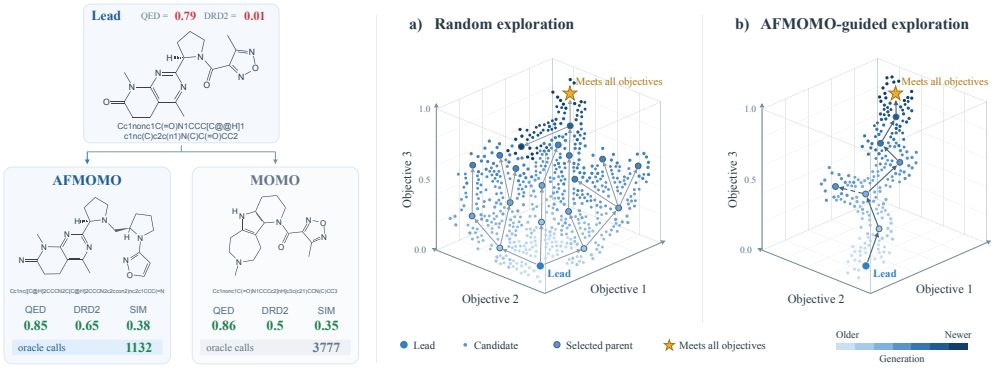

Figure 1: Objective-aware molecular search with AFMOMO. Left: An illustrative comparison starting from the same lead molecule. Right: AFMOMO uses objective satisfaction states to guide molecular edits toward joint constraint satisfaction.

agent to vary the type and scale of an edit. *Feedback-driven memory* retains candidate outcomes and lineage, adjusts operator weights, and enables a change of branch when regression is followed by stagnation. Together, these components let the agent observe, plan, act, and revise within an oracle-limited search.

Across four tasks, AFMOMO achieves average joint success rates of 92.49% with DeepSeek and 93.41% with Qwen, compared with 64.60% for MOMO and 74.76% for MOMO-QMO. Replacing LLM planning with randomized plans lowers average success by 9.41–10.33 percentage points, with the largest decreases on the activity-constrained tasks. On leads solved by both AFMOMO–Qwen and MOMO, the task-averaged recorded oracle count falls by 81.13%. These results support the value of coupling agent planning with structured molecular editing.

Our contributions are threefold:

- An **objective-aware molecular optimization agent** that coordinates parent selection, editing actions, and generation budgets using constraint status and search feedback.
- A **heuristic molecular genetic search procedure** that connects chemically annotated mutation rules, recombination, and edits at multiple scales through staged execution and adaptive operator weighting.
- An evaluation across four tasks showing improved joint success and lower recorded oracle costs, with a randomized-planning comparison that tests the contribution of agent decisions within the editing framework.

2 Related Work

Multi-objective molecular optimization. Property-driven molecular search includes reinforcement learning (Olivecrona et al., 2017), optimization in learned latent spaces (Gómez-Bombarelli et al., 2018), and graph-based mutation and crossover (Jensen, 2019). MOMO couples implicit-space evolution with Pareto-based selection (Xia et al., 2024). CMOMO combines latent-vector fragmentation with two-stage dynamic constraint handling for constrained multi-property optimization (Xia et al., 2025). GuacaMol benchmarks distribution learning and goal-directed generation (Brown et al., 2019), while PMO emphasizes the number of oracle evaluations needed for optimization (Gao et al., 2022).

LLM-guided editing and molecular agents. LLM-based methods assign different roles to language models. MOLLEO uses LLMs to redesign mutation and crossover within evolutionary search (Wang et al., 2025). ToolMol implements an agentic genetic operator through RDKit-backed editing tools (Zhou et al., 2026). MT-Mol coordinates specialist analysts, a molecular generator, a verifier, and a reviewer (Kim et al., 2025). TRACE formulates tool selection as sequential decisions over molecular optimization trajectories, accounting for how earlier edits influence later refinement (Li et al., 2026).

Feedback and adaptive search. Evolutionary search adapts variation through operator selection and restarts. Bandit-based mechanisms allocate operators using observed fitness improvements (Fialho et al., 2010); IPOP-CMA-ES enlarges the population across restarts to broaden exploration (Auger & Hansen, 2005). Molecular agents also reuse search history: F2R retrieves past actions, rationales, and domain feedback to revise molecular optimization reasoning (Gao et al., 2026). ATOM maintains alternative optimization trajectories through tree-structured agent coordination and global memory (Zhang et al., 2026).

3 AFMOMO

3.1 Problem Formulation

Given a lead molecule x_0, property functions $f_1, \ldots, f_K$, thresholds $\tau_1, \ldots, \tau_K$, and an oracle budget B, we seek a structurally valid candidate that satisfies all task constraints. For inclusive constraints, joint success is

$$d_i\big[f_i(x) - \tau_i\big] \geq 0, \qquad \forall i \in \{1, \ldots, K\}, \tag{1}$$

where $d_i = +1$ for maximization and $d_i = -1$ for minimization. Strict constraints use $>$; Table 2 specifies the experimental criteria. Similarity is measured relative to x_0. Success is determined from the complete archive of evaluated candidates, even when some candidates later become inactive.

The oracle budget B limits property evaluations, whereas the round budget Q_t controls candidate generation. One scored candidate counts as one oracle call for all task properties; lead evaluation, candidate generation, and LLM requests are excluded. Repeated requests use an evaluation cache. Section 4.3 states the counting conventions used in the reported comparisons.

3.2 Framework Overview

AFMOMO implements an iterative *observe–plan–edit–evaluate–update* loop (Figure 2). Its three components have distinct responsibilities. The **LLM planner** decides which parents to expand, which constraints to target, and how to allocate edits. The **genetic search executor** realizes these decisions through permitted molecular operators in four stages. The **memory manager** retains evaluated structures, operator outcomes, and branch history for subsequent decisions. The evaluator supplies property scores, while the main loop enforces the oracle budget and stopping policy.

At round t, active memory provides a candidate pool $\mathcal{C}_t$. The agent observes candidate structures and objective states, the annotated operator catalog, previous operator outcomes, and any rollback context. It returns a structured plan π_t containing parents, target constraints, operator weights, and stage ratios. After validation, the executor allocates Q_t, reweights the approved operators using historical feedback, and generates candidates. The main loop selects candidates within the remaining oracle budget and evaluates them. Memory then updates the available parents, operator statistics, and checkpoint state. In Algorithm 1, $\mathcal{M}$ denotes archive and search memory, $\mathcal{S}_t$ the agent observation, Θ the task configuration, b consumed oracle calls, and $\perp$ an invalid plan after repair.

Algorithm 1 AFMOMO: agent-guided genetic search

Require: Lead x_0, task Θ, oracle budget B, round limit T
$\mathcal{M} \leftarrow \textsc{Initialize}(x_0, \Theta)$; $b \leftarrow 0$
for $t = 1, \ldots, T$ **do**
  if $b \geq B$ or $\textsc{Stop}(\mathcal{M}, \Theta)$ **then break**
  $(\mathcal{C}_t, \mathcal{S}_t) \leftarrow \textsc{Observe}(\mathcal{M}, \Theta, Q_t)$
  $\pi_t \leftarrow \textsc{PlanAndValidate}(\mathcal{S}_t)$; **if** $\pi_t = \perp$ **then continue**
  $\mathcal{X}_t \leftarrow \textsc{StagedEdit}(\mathcal{C}_t, \pi_t, Q_t, \mathcal{M})$
  $\mathcal{Y}_t \leftarrow \textsc{Select}(\mathcal{X}_t, \pi_t, B - b)$
  $(\mathcal{E}_t, \Delta b) \leftarrow \textsc{Evaluate}(\mathcal{Y}_t, \mathcal{M}, \Theta)$
  $b \leftarrow b + \Delta b$; $\mathcal{M} \leftarrow \textsc{UpdateAndRollback}(\mathcal{M}, \mathcal{E}_t, B - b)$
end for
return $\textsc{Archive}(\mathcal{M})$

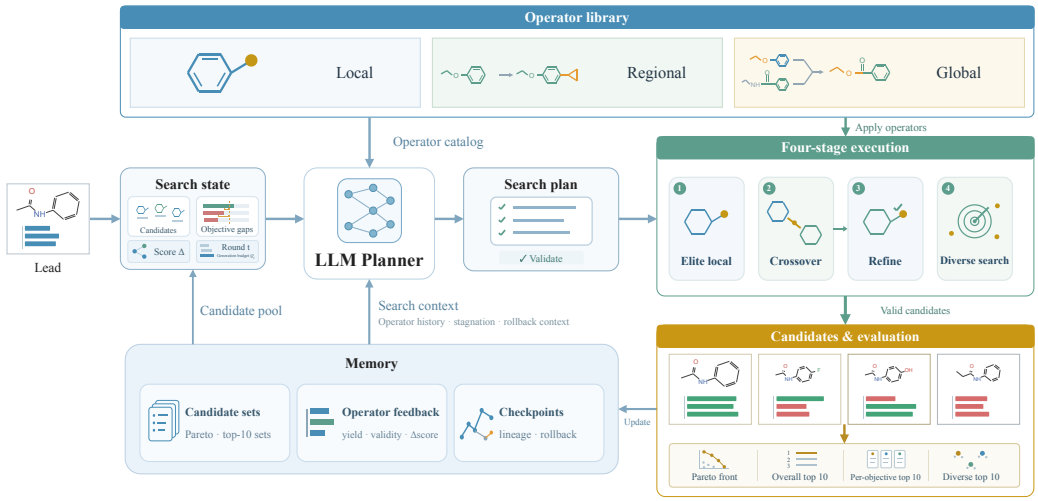

Figure 2: AFMOMO's planning–execution–feedback loop. The LLM converts molecular states and search history into a structured plan. The executor applies the selected genetic operators, and evaluation supplies feedback to candidate, operator, and checkpoint memory. Conditional rollback restricts the next parent choice to eligible alternatives.

3.3 Objective-Aware Planning

Observing constraint status. The planner needs to distinguish a remaining deficit from surplus improvement on an already satisfied objective. For each objective, we compute a normalized margin, capped attainment, and nonnegative gap:

$$m_i(x) = \frac{d_i[f_i(x) - \tau_i]}{s_i}, \qquad a_i(x) = \mathrm{clip}(1 + m_i(x), 0, 1), \qquad g_i(x) = \max(0, -m_i(x)), \quad (2)$$

where $s_i > 0$ is a task-specific scale. Attainment reaches one at the threshold, while the gap measures the shortfall. A separate satisfaction indicator applies the exact strict or inclusive comparison. The observation pairs these quantities with molecular structures, provenance, operator descriptions, generation yield, historical property changes, stagnation information, and Q_t.

Choosing an executable search plan. The LLM selects a primary parent and a nonempty subset of its unmet constraints, when any remain. It then specifies parents or parent pairs, stage budget ratios, and weighted primary and backup operators. This allows it to direct local edits toward a remaining deficit, select two parents for recombination, or allocate more attempts to broader structural changes. Operator annotations supply chemical priors about edit scale and anticipated effects; actual property changes are established by evaluation. The executor constructs molecules from the selected actions, so the plan itself contains parent references and operator choices rather than new molecular strings.

Validation checks parent availability, stage permissions, positive weights, and execution dependencies. In particular, local refinement must include the primary parent, crossover requires distinct parents, and post-crossover refinement requires crossover products. An invalid plan receives one repair attempt; otherwise the round produces no candidates. After rollback, the primary parent must be selected from the checkpoint's eligible alternatives. These checks turn the agent's decisions into a constrained, executable plan.

3.4 Heuristic Molecular Genetic Search

An operator interface grounded in chemical transformations. The operator library maps plans to edits on molecules represented as SMILES (Weininger, 1988). It contains 18 operators: seven local, eight regional, and three global. Local operators include terminal pruning, heteroatom replacement, and functional-group or bioisosteric replacement. The latter two use 26 annotated transformation

Table 1: Four-stage execution of an agent plan. Operator scale (local, regional, global) and execution stage describe different aspects of the search.

Stage	Input and operators	Role in the search
Elite local refinement	Selected pool parents; local mutation	Refine existing candidates, including the primary parent
Crossover	Selected parent pairs; BRICS recombination	Combine fragments from distinct parents
Post-crossover refinement	Crossover offspring; local mutation	Refine newly recombined structures
Diverse exploration	Selected pool parents; regional/global single-parent mutation	Explore broader structural alternatives

rules: 18 functional-group rules and eight bioisosteric rules. Their annotations describe edit scale, expected heavy-atom changes, scaffold preservation, and anticipated property effects, giving the planner information for selecting a transformation.

Regional operators act on fragments, linkers, and rings, including fragment growth, linker replacement, and phenyl-to-heteroaryl replacement. Global operators provide larger SELFIES mutations, mixed mutation, and two-parent BRICS recombination (Krenn et al., 2020; Degen et al., 2008). The design combines these established mutation and recombination mechanisms with chemical rules in a common planning interface. Edit scale specifies the transformation's scope; it does not guarantee a property gain or a hard similarity bound.

Four stages with complementary roles. Table 1 shows how the executor organizes the operators. Local refinement searches around selected parents, crossover combines parent fragments, post-crossover refinement edits the newly formed offspring, and broader exploration generates alternative structures. The planner controls how much generation effort each stage receives. This makes parent choice, transformation type, and search breadth explicit decisions within the genetic search.

Stage ratios are converted to integer generation budgets summing to Q_t. Feedback adjusts weights only among operators approved by the plan. If primary operators leave a shortfall, approved backup operators can supplement the same stage and parent set; unused generation quotas are not transferred between stages. Post-crossover refinement acts on the actual crossover products, preserving the dependency between recombination and subsequent mutation.

Validation before oracle evaluation. Generated candidates undergo RDKit parsing, canonicalization, parent exclusion, and deduplication (Landrum et al., 2013); previously seen and inactive structures are also excluded. If candidates exceed the remaining oracle budget, stage-stratified selection uses the planned ratios to allocate evaluation slots. Property evaluation then determines whether an edit improves its targets while preserving other constraints. Scores and parent–operator–child relations provide feedback for the next round.

3.5 Feedback-Driven Memory

Candidate memory: retaining progress and alternatives. The complete archive stores all evaluated candidates and their provenance. Active memory maintains constraint-ranked, per-objective, diverse, and nondominated collections, from which the next candidate pool is assembled. Ranking first favors more satisfied objectives, then higher weakest attainment, then smaller total gap:

$$R(x) = \left(N_{\mathrm{sat}}(x),\ \min_i a_i(x),\ -\sum_i g_i(x),\ P(x),\ D(x),\ \kappa(x) \right). \tag{3}$$

The key is maximized lexicographically. Here P indicates Pareto membership, D is a diversity term used in diversity selection, and κ is a stable canonical-string tie-break key. Ordinary best-candidate selection uses $D = 0$. This ordering prioritizes joint feasibility, while the additional collections retain parents with different property profiles.

Operator memory: adapting how a plan is executed. For each lead, history records operator invocation counts, generation yield, and mean evaluated parent–child changes $\overline{\Delta}_i(a)$. The LLM observes this history, and the executor also uses it to adjust operator weights. Let x^* be the constraint-ranked reference candidate, $\mathcal{I}_-$ its unmet objectives, and $\mathcal{I}_+$ its satisfied objectives. With gap weights $w_i = g_i(x^*)/\sum_{j\in\mathcal{I}_-} g_j(x^*)$, define

$$E(a) = \sum_{i\in\mathcal{I}_-} w_i \frac{d_i\overline{\Delta}_i(a)}{s_i}, \tag{4}$$

$$L(a) = \sum_{i\in\mathcal{I}_+} \max\left(0, -m_i(x^*) - \frac{d_i\overline{\Delta}_i(a)}{s_i}\right), \tag{5}$$

$$U(a) = E(a) - \lambda L(a) + \frac{\beta}{\sqrt{n_a+1}}, \qquad \widetilde{p}_a \propto p_a \exp\big(\mathrm{clip}(U(a), -2, 2)\big). \tag{6}$$

Thus, utility rewards changes aligned with remaining gaps, penalizes projected violations of satisfied constraints, and gives less-tried operators an exploration bonus. We set $w_i = 0$ when the denominator is zero and use zero mean changes for unseen operators. Effective weights are normalized within each stage's primary or backup group. The rule preserves the agent's operator choices and stage ratios while adapting their relative execution weights. The reference x^* can differ from the LLM's primary parent; utility uses all unmet objectives of x^*, while the plan may focus on a subset for its selected parent.

Checkpoint memory: revising a stalled branch. AFMOMO saves the active best candidate and candidate pool before editing. It creates a checkpoint when the new active best loses a satisfied constraint or suffers a sufficiently large margin decrease. Rollback becomes eligible if a compatible descendant branch stalls with exactly one unmet constraint near its threshold, and that constraint was protected by the checkpoint. An alternative parent must come from the saved pool, remain active, satisfy the protected constraints, be untried at that checkpoint, and lie outside the failed branch.

When these conditions and the remaining-budget requirement hold, AFMOMO deactivates the failed branch and its descendants, rebuilds active memory, and asks the next plan to choose an eligible alternative. The archive, cache, operator history, and consumed budget are retained. Rollback is limited and is disabled once the archive contains a successful candidate. This gives the agent a way to revise its search trajectory while retaining what earlier evaluations revealed.

4 Experiments

We evaluate whether AFMOMO improves joint constraint satisfaction, reduces oracle cost, and benefits from LLM-guided decisions within the genetic editing framework.

4.1 Experimental Setup

Tasks and data. We use four fixed lead sets distributed with MOMO, containing 800, 800, 780, and 801 molecules, respectively (Xia et al., 2024). Table 2 summarizes their constraints. Tasks 1, 3, and 4 correspond to the published benchmark's Tasks 1, 3, and 6; our Task 2 uses its code-distributed pLogP dataset. All methods use the same lead sets without resampling.

Table 2: Lead sets and joint success criteria. Every condition must hold for the same candidate. Task 2 tracks both similarity milestones and reports a single SR counting only leads that reach both.

Task	Leads	Success conditions
1	800	QED ≥ 0.9, SIM ≥ 0.4
2	800	ΔpLogP > 0, SIM $> \delta$, for both $\delta \in \{0.4, 0.6\}$
3	780	QED ≥ 0.8, DRD2 ≥ 0.4, SIM ≥ 0.3
4	801	QED ≥ 0.7, GSK3$\beta \geq 0.4$, $\mathrm{SA}_{\mathrm{norm}} \geq 0.7$, SIM ≥ 0.2

Table 3: Joint success rate (%, higher is better) at the end of each method's runs. **Bold** and underlined values mark the best and second-best distinct results in each column; ties share the same style.

Method	Task 1	Task 2	Task 3	Task 4	Mean
MOMO	65.00	95.88	58.59	38.95	64.60
MOMO-QMO	90.25	<u>96.13</u>	73.72	38.95	74.76
AFMOMO (Random)	<u>91.50</u>	**100.00**	60.90	79.90	83.08
AFMOMO (DeepSeek)	**96.38**	**100.00**	<u>78.33</u>	<u>95.26</u>	<u>92.49</u>
AFMOMO (Qwen)	**96.38**	**100.00**	**79.74**	**97.50**	**93.41**

Mean averages the four tasks. QMO seeds are used only in Tasks 1–3.

Property evaluation. QED is computed with RDKit (Bickerton et al., 2012; Landrum et al., 2013); SIM is Tanimoto similarity to the lead using radius-2, 2048-bit Morgan fingerprints (Rogers & Hahn, 2010). The implemented pLogP score is $\mathrm{MolLogP} - \mathrm{SA}_{\mathrm{raw}}$, and $\Delta\mathrm{pLogP}$ is its change from the lead. Normalized synthetic accessibility is $(10 - \mathrm{SA}_{\mathrm{raw}})/9$ (Ertl & Schuffenhauer, 2009). DRD2 and GSK3β scores use the fixed MOMO predictors (Xia et al., 2024). These scores are computational optimization targets.

Methods and metrics. We use DeepSeek-V4-Flash and Qwen3.8-27B (DeepSeek-AI, 2026; Qwen Team, 2026), abbreviated DeepSeek and Qwen, with at most 5,000 oracle calls and 100 rounds per lead. MOMO uses original-lead initialization; MOMO-QMO uses seeds from QMO (Hoffman et al., 2022) for Tasks 1–3, falling back to the lead when unavailable. Both baselines share the original-lead configuration for Task 4. The oracle-budget comparison uses MOMO without QMO seeds to match starting leads. Success rate (SR) is the fraction of leads for which the archive contains a jointly successful candidate, including failures in the denominator. Task 2 counts a lead as successful only when both similarity milestones have been reached with positive pLogP improvement; overall SR is the arithmetic mean across the four tasks. Final SR uses each method's completed runs, whereas finite-budget curves filter by recorded oracle counts.

4.2 Joint Optimization Performance

Both LLM configurations outperform MOMO and MOMO-QMO on all four tasks (Table 3). AFMOMO–Qwen achieves 93.41% average SR, exceeding the reported baseline averages by 28.80 and 18.64 percentage points. DeepSeek achieves 92.49%, showing that the improvement is shared by both tested backbones.

The pattern varies with the constraint combination. Task 2 is nearly saturated for the baselines and reaches 100% with AFMOMO, leaving little room for final-SR improvement. Task 1 reaches 96.38%, indicating that high QED can often be achieved while retaining the required similarity. Task 4 exhibits the largest gain: Qwen reaches 97.50% versus 38.95% for both baselines, a 58.55-point increase under four simultaneous constraints. In contrast, Task 3 remains the hardest for AFMOMO (78.33–79.74%), despite having fewer constraints. This suggests that difficulty depends on the particular property combination and thresholds, rather than objective count alone.

QMO initialization improves MOMO substantially on Tasks 1 and 3, yet AFMOMO still achieves higher SR from the original leads. The comparison therefore supports the effectiveness of its iterative search beyond the advantage of an improved starting population. The final-SR table uses different stopping rules; the following analysis examines recorded evaluation cost explicitly.

4.3 Oracle Efficiency

Success under finite budgets. Figure 3 reports SR as the oracle cutoff increases. Both AFMOMO backbones reach higher success over broad budget ranges on Tasks 1, 3, and 4. Task 2 is informative despite its saturated final SR: AFMOMO reaches full success at lower recorded budgets, so its benefit appears primarily as reduced evaluation effort. Together, the curves show that the advantage extends beyond final outcomes to search with limited property evaluations.

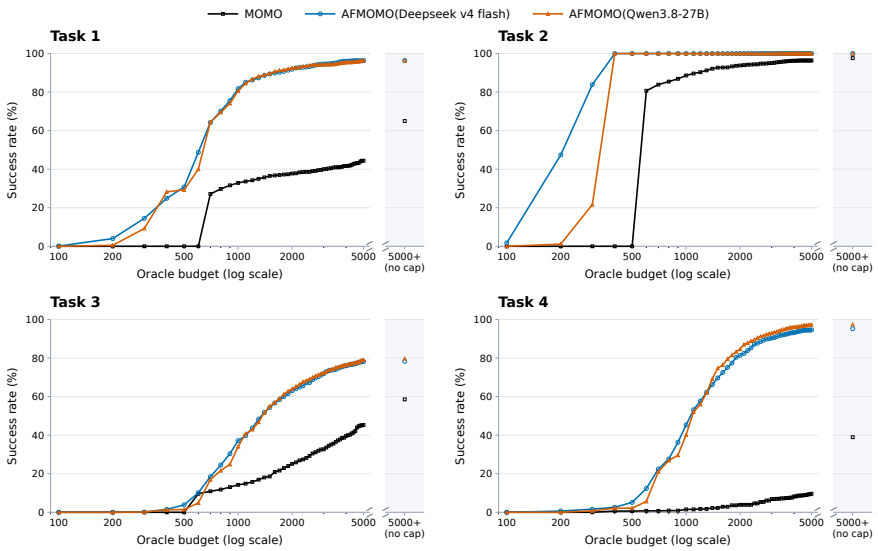

Figure 3: Joint SR versus recorded oracle budget. Finite cutoffs range from 100 to 5,000 in steps of 100 and are displayed on a logarithmic axis. A lead contributes if it succeeds and its recorded count is strictly below the cutoff. Separate "5000+" points report final SR without a cutoff.

Table 4: Recorded oracle cost on the same jointly solved leads. Lower mean cost is **bold**; the last row averages the four per-task means equally.

		Mean oracle calls ↓		
Task	**Shared leads**	**MOMO**	**AFMOMO (Qwen)**	**Saving (%)**
Task 1	513	4,566.69	**611.64**	86.61
Task 2	767	882.79	**323.47**	63.36
Task 3	412	3,116.37	**1,199.12**	61.52
Task 4	307	9,259.96	**1,228.71**	86.73
Task mean	—	4,456.45	**840.74**	**81.13**

Cost on the same successfully optimized leads. Table 4 compares AFMOMO(Qwen) and MOMO on the intersection of their successful leads within each task. Equal averaging of the four per-task mean counts gives 4,456.45 calls for MOMO and 840.74 for AFMOMO, a reduction of $1 - 840.74/4456.45 = 81.13\%$. The reduction is largest on Task 4 (86.73%), where MOMO requires 9,259.96 calls on average for the jointly solved leads, compared with 1,228.71 for AFMOMO. Task 3 also benefits, but retains the highest difficulty in final SR and a smaller cost reduction (61.52%). These complementary views distinguish improved coverage over all leads from lower cost on a common successful subset.

Interpretation of oracle counts. The comparisons use recorded computational evaluation counts and exclude LLM and generation costs. AFMOMO does not charge the initial lead evaluation; its evaluator increments after an original-string cache miss and before the canonical-cache check, with canonicalization in the search loop limiting duplicate charges. Tasks 1, 3, and 4 use total-run counts, which can include the remainder of the successful round. Task 2 uses a shared batch-equivalent count for AFMOMO's two milestones, whereas MOMO uses threshold-specific counts. Accordingly, these results establish lower recorded oracle cost under the stated conventions, rather than an exact comparison of first-success or wall-clock cost.

4.4 Contribution of Agent Planning

Randomized-planning comparison. AFMOMO (Random) replaces LLM decisions with randomized primary and exploration parents, crossover pairs, and local/exploration operators. Stage ratios are

sampled from two predefined templates, with a separate single-parent allocation. It retains candidate memory, validation, evaluators, rollback, and history-based selection for post-crossover refinement. Each randomized primary or backup operator group contains one operator. This comparison tests LLM-guided plan construction within the framework; it changes the planning policy and the breadth of its operator choices together.

Random achieves 83.08% average SR, already above MOMO's 64.60% (Table 3). Thus, the genetic editing and feedback framework is effective even without LLM-generated plans. The full LLM-guided configurations outperform the tested randomized configuration by approximately 9.4–10.3 percentage points in average SR. The gains are concentrated on Tasks 3 and 4: Qwen improves over Random by 18.84 and 17.60 points, respectively, compared with 4.88 points on Task 1 and no final-SR gain on Task 2. This pattern supports the value of selecting parents, edit types, and search effort using the current constraint state when several property requirements must be coordinated.

Random retains the operator library and memory, so its advantage over MOMO cannot be assigned to any single operator. The comparison with the LLM-guided configuration changes both the planning policy and operator-choice breadth; the observed gain therefore cannot be attributed solely to LLM planning. Capacity-matched controls and separate ablations are needed to isolate LLM planning and its individual decision components.

This comparison changes both planning policy and operator-choice breadth; capacity-matched controls are needed to isolate the contribution of LLM planning.

Direct-LLM ablation. We compare AFMOMO with a Direct-LLM variant that uses DeepSeek v4 flash to propose candidate SMILES directly instead of using the structured genetic search. This small-scale comparison uses the first 10 leads from each task. SR is computed over all 10 leads, whereas mean oracle counts are compared only on leads successfully optimized by both methods. As shown in Table 5, AFMOMO achieves an average SR of 92.50%, compared with 72.50% for Direct-LLM. On the shared-success subsets, its task-averaged oracle count is 24.85% lower. These results support the structured search on this subset; the limited sample size warrants caution in generalizing the findings.

Table 5: Direct-LLM comparison using DeepSeek v4 flash on the first 10 leads per task. SR includes all leads; mean oracle counts use only the shared-success subset (n).The final row reports unweighted averages across tasks.

	SR (%)			Mean oracle calls	
Task	Direct-LLM	AFMOMO	n	Direct-LLM	AFMOMO
1	90.00	100.00	9	874.22	585.00
2	100.00	100.00	10	192.70	244.10
3	60.00	70.00	5	813.80	967.20
4	40.00	100.00	4	1708.00	900.50
Average	72.50	92.50	–	897.18	674.20

Feedback and branch revision. Search logs provide a complementary view of the agent's ability to revise a trajectory. Among task–lead instances with rollback, 91 of 189 for DeepSeek and 51 of 128 for Qwen subsequently reach joint success (48.15% and 39.84%). Most rollback-exposed leads occur in Task 3, while Task 2 has none, consistent with their relative difficulty in the success-rate results. These cases show that successful optimization can follow branch rollback, allowing the search to continue from alternative parents.

5 Conclusion

AFMOMO integrates objective-aware planning, chemically annotated genetic operators, and feedback-driven memory. The agent selects parents, target constraints, editing actions, and generation budgets; staged execution and evaluation feedback turn these decisions into adaptive search. Both tested backbones improve joint success over MOMO and MOMO-QMO. Randomized planning lowers average SR by 9.41–10.33 percentage points, while AFMOMO–Qwen reduces task-averaged recorded oracle cost by 81.13% on jointly solved leads. These results support coupling agent decisions with structured editing. Future work should isolate operator and memory contributions, quantify variability, and validate candidates beyond computational proxies.

AI USE STATEMENT

We used large language models (LLMs) for coding assistance and language polishing. All AI-assisted outputs were carefully reviewed and verified by the authors. The authors take full responsibility for the accuracy, integrity, and content of this work.

REFERENCES

Anne Auger and Nikolaus Hansen. A restart cma evolution strategy with increasing population size. In *2005 IEEE congress on evolutionary computation*, volume 2, pp. 1769–1776. IEEE, 2005.

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

Nathan Brown, Marco Fiscato, Marwin HS Segler, and Alain C Vaucher. Guacamol: benchmarking models for de novo molecular design. *Journal of chemical information and modeling*, 59(3): 1096–1108, 2019.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. arXiv preprint arXiv:2606.19348, 2026. URL https://arxiv.org/abs/2606.19348.

Jorg Degen, Christof Wegscheid-Gerlach, Andrea Zaliani, and Matthias Rarey. On the art of compiling and using'drug-like'chemical fragment spaces. *ChemMedChem*, 3(10):1503, 2008.

Peter Ertl and Ansgar Schuffenhauer. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of cheminformatics*, 1(1):8, 2009.

Álvaro Fialho, Luis Da Costa, Marc Schoenauer, and Michele Sebag. Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence*, 60(1): 25–64, 2010.

Wenhan Gao, Xiran Fan, Chin-Chia Michael Yeh, Jiarui Sun, Yuzhong Chen, Menghai Pan, Mahashweta Das, and Yi Liu. Feedback to reasoning: Llm-assisted molecular optimization with domain feedback and historical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 12735–12754, 2026.

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

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268, 2018.

Samuel C. Hoffman, Vijil Chenthamarakshan, Kahini Wadhawan, Pin-Yu Chen, and Payel Das. Optimizing molecules using efficient queries from property evaluations. *Nature Machine Intelligence*, 4(1):21–31, 2022. doi: 10.1038/s42256-021-00422-y.

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

Hyomin Kim, Yunhui Jang, and Sungsoo Ahn. Mt-mol: Multi agent system with tool-based reasoning for molecular optimization. In *EMNLP (Findings)*, pp. 11544–11573, 2025.

Mario Krenn, Florian Häse, AkshatKumar Nigam, Pascal Friederich, and Alan Aspuru-Guzik. Self-referencing embedded strings (selfies): A 100% robust molecular string representation. *Machine Learning: Science and Technology*, 1(4):045024, 2020.

Greg Landrum et al. Rdkit documentation. *Release*, 1(1-79):4, 2013.

Lingxiao Li, Haobo Zhang, Ruohao Fan, Bin Chen, and Jiayu Zhou. Molecular lead optimization via agentic tool planning. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 11328–11338, 2026.

Christos A Nicolaou and Nathan Brown. Multi-objective optimization methods in drug design. *Drug Discovery Today: Technologies*, 10(3):e427–e435, 2013.

Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. Molecular de-novo design through deep reinforcement learning. *Journal of cheminformatics*, 9(1):48, 2017.

Qwen Team. Qwen3.8-27B: Official model card, 2026. URL https://huggingface.co/Qwen/Qwen3.8-27B. Accessed September 26, 2026.

David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling*, 50(5):742–754, 2010. doi: 10.1021/ci100050t.

Matthew D. Segall. Multi-parameter optimization: identifying high quality compounds with a balance of properties. *Current pharmaceutical design*, 18(9):1292–1310, 2012.

Haorui Wang, Marta Skreta, Cher-Tian Ser, Wenhao Gao, Lingkai Kong, Felix Strieth-Kalthoff, Chenru Duan, Yuchen Zhuang, Yue Yu, Yanqiao Zhu, et al. Efficient evolutionary search over chemical space with large language models. In *International Conference on Learning Representations*, volume 2025, pp. 51694–51727, 2025.

David Weininger. SMILES, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36, 1988. doi: 10.1021/ci00057a005.

Xin Xia, Yiping Liu, Chunhou Zheng, Xingyi Zhang, Qingwen Wu, Xin Gao, Xiangxiang Zeng, and Yansen Su. Evolutionary multiobjective molecule optimization in an implicit chemical space. *Journal of Chemical Information and Modeling*, 64(13):5161, 2024.

Xin Xia, Yajie Zhang, Xiangxiang Zeng, Xingyi Zhang, Chunhou Zheng, and Yansen Su. Cmomo: a deep multi-objective optimization framework for constrained molecular multi-property optimization. *Briefings in Bioinformatics*, 26(4):bbaf335, 2025.

Jia Zhang, Tengfei Ma, Tianle Li, Daojian Zeng, Xieping Gao, and Xiangxiang Zeng. Agents on a tree: Pathwise coordination for multi-objective molecular optimization. arXiv preprint arXiv:2606.00008, 2026. URL https://arxiv.org/abs/2606.00008.

Andrew Y Zhou, Sharvaree Vadgama, Sumanth Varambally, Peter Eckmann, Michael K Gilson, and Rose Yu. Toolmol: Evolutionary agentic framework for multi-objective drug discovery. *arXiv preprint arXiv:2605.12784*, 2026.

A METHOD DETAILS

This section follows the three components of AFMOMO: the agent observation and plan interface, molecular genetic operators and their staged execution, and feedback memory with branch revision. The search loop and governing equations are given in Section 3.

A.1 NOTATION

Table 6: Notation used throughout the paper.

Symbol	Meaning
x_0, x, x^*	lead molecule, candidate, and utility reference candidate; x^* need not be the primary parent
f_i, d_i, τ_i, s_i	Objective function, direction, threshold, and normalization scale
m_i, a_i, g_i	Normalized margin, capped attainment, and nonnegative gap
B, T, Q_t	Evaluation budget, round limit, and round-t generation budget
$\Theta, \mathcal{M}$	Task configuration; archive and search memory in Algorithm 1
$\mathcal{C}_t, \mathcal{S}_t, \pi_t$	Candidate pool, agent observation, and validated plan at round t
$\mathcal{X}_t, \mathcal{Y}_t, \mathcal{E}_t$	Generated candidates, selected candidates, and evaluation records
$b, \Delta b, \perp$	Consumed oracle calls, their increment, and an invalid plan after repair
$\overline{\Delta}_i(a), n_a, U(a)$	Mean observed change, operator invocation count, and operator utility
$\lambda, \beta, p_a, \widetilde{p}_a$	Regression penalty coefficient, exploration coefficient, plan weight, and effective weight
SR	Fraction of lead molecules for which the search finds a jointly successful candidate

A.2 AGENT OBSERVATIONS AND PLAN VALIDATION

The agent receives the candidate pool together with objective states, operator annotations, and search history. The interface below specifies the available parent records, the returned plan, and the checks required before execution.

Candidate pool. Active memory contains the nondominated set, the top 10 candidates under constraint ranking, per-objective top-10 collections, and a diverse top-10 collection. The pool includes the available lead record and samples these collections in round-robin order with deduplication, up to 20 candidates. Canonical SMILES provide persistent molecular identities; candidate IDs are local to a planning round.

Diversity selection first considers the top 120 constraint-ranked candidates. It then uses the minimum Tanimoto distance to the already selected set as a later ranking term, preserving the priority of joint feasibility. Pareto dominance is computed using raw objective values and their optimization directions.

Structured plan. A plan contains `primary_parent_id`, `focus_objectives`, and `stages`. Each non-crossover stage specifies a `budget_ratio`, primary and backup operator weights, and a `rationale`. Parent references use `parent_ids`; crossover instead uses `parent_pairs` and the fixed `brics_recombine` operator. The executor supplies the parents for `post_crossover_local` and derives integer stage budgets from the ratios.

Validation. The validator checks allowed fields, parent references, stage permissions, positive weights, ratio bounds, and stage dependencies. Elite local refinement must include the primary parent. Non-crossover stages allow at most three primary and three backup operators, with disjoint groups. When the primary parent has unmet constraints, the focus set must be a nonempty subset of them. Fields containing `smiles` are rejected because the planner selects actions rather than supplying new structures. One semantic repair is allowed; a plan that remains invalid generates no candidates in that round.

A.3 Molecular Genetic Operators and Staged Execution

A.3.1 Operator Catalog and Chemical Rules

The catalog exposes the following 18 registered operators. Their identifiers specify callable actions, while the accompanying annotations describe their chemical scope.

Local (7)
`halogen_exchange`, `methylation`, `terminal_prune`, `heteroatom_replace`, `selfies_mutate_local`, `functional_group_replace`, and `bioisostere_replace`.

Regional (8)
`polar_group_addition`, `fragment_growth`, `linker_replace`, `ring_simplification`, `phenyl_to_heteroaryl`, `amine_modification`, `fragment_replace`, and `selfies_mutate_medium`.

Global (3)
`selfies_mutate_large`, `brics_recombine`, and `mixed_random_mutation`.

Functional-group replacement has 18 annotated transformation rules and bioisosteric replacement has eight. The annotations cover edit scale, expected heavy-atom changes, scaffold preservation, and anticipated property effects. They inform the LLM's operator selection; they do not enter the historical utility calculation in Eq. (6). BRICS and SELFIES provide the underlying recombination and string-mutation mechanisms (Degen et al., 2008; Krenn et al., 2020).

The executor validates requests and parents, invokes the selected action, and applies RDKit parsing, canonicalization, parent exclusion, and within-batch deduplication. The main loop also excludes previously seen and inactive molecules. Under the default configuration, generation imposes no required substructure, minimum similarity, heavy-atom-change bound, or scaffold-preservation requirement. Similarity is measured during evaluation, so annotations about locality or scaffold preservation should be read as descriptions of the transformations rather than universal generation constraints.

A.3.2 Stage Allocation and Candidate Generation

Table 7: Execution stages in `STAGE_ORDER`. Ratios and derived integer budgets are both validated.

Stage identifier	Parents and permitted actions	Ratio bounds
`elite_local`	Candidate-pool parents; local primary and backup operators	20–70%
`complementary_crossover`	Distinct parent pairs; fixed `brics_recombine`	0–40%
`post_crossover_local`	Actual crossover products; local operators	0–20%; budget no greater than crossover
`diverse_exploration`	Candidate-pool parents; regional or global single-parent operators	15–60%

Table 7 specifies the stage permissions and ratio bounds. The executor normalizes ratios and uses largest-remainder allocation to obtain integer budgets summing to Q_t. With a single parent, crossover and post-crossover refinement receive zero budget. Post-crossover refinement uses transient IDs for actual crossover products. The identifier `complementary_crossover` denotes the crossover stage; validation requires distinct valid parents, without a numerical complementarity threshold.

Within a non-crossover stage, attempts are allocated according to effective primary weights. Productive primary operators receive one supplementation pass. If a shortfall remains, execution uses the approved backup operators and then gives productive backups one supplementation pass. Unfilled quotas are discarded. Supplementation retains the same stage and parent set. The registered `mixed_random_mutation` action can combine lower-level edits internally, so operator permissions apply at the registered interface.

Generation quotas and evaluation slots serve different purposes. When generated candidates exceed the remaining oracle budget, stage-stratified selection uses the planned ratios and can redistribute evaluation slots according to realized yields. Selection retains the crossover parent when retaining a post-crossover child. This does not transfer unused generation quotas between stages.

A.4 Feedback Memory and Branch Revision

A.4.1 Historical Operator Statistics

Operator statistics are maintained separately for each lead. The count n_a measures generation invocations, while $\overline{\Delta}_i(a)$ averages changes over evaluated children with available parent records. Changes are pooled across the molecular contexts visited during that lead's search; crossover changes are principally attributed relative to the first recorded parent. These pooled statistics are heuristic guidance, not calibrated predictions for a specific parent.

The utility reference x^* is selected by satisfied-objective count, weakest attainment, and negative total gap, with candidate ID breaking ties. It can differ from the planner's primary parent. Equations (4)–(6) apply all unmet objectives of this reference, whereas the plan's focus set refers to its selected primary parent. A projected decrease contributes to the regression penalty only if it crosses a currently satisfied threshold. Operators without history have zero mean changes and retain the exploration bonus. Effective weights are normalized independently within each stage's primary and backup groups; the defaults are $\lambda = 2.0$ and $\beta = 0.1$.

A.4.2 Checkpoint Creation and Branch Rollback

Detecting regression. Before generation, AFMOMO saves the active joint-best candidate and candidate pool. A checkpoint is created after evaluation if the new active best loses a satisfied objective or any normalized margin falls by more than 0.25. It records the protected objectives, earlier pool, new branch root, and parents already tried. Regression is measured between the active best candidates before and after the round.

Detecting stagnation near feasibility. Tracking starts when exactly one objective remains unmet and its normalized gap is at most 0.20. The initial stagnation count is zero. A cumulative gap reduction of at least 0.02 from the last significant-improvement point resets the count; otherwise it increments. Rollback is considered after three subsequent valid-plan rounds without sufficient progress. Valid plans with zero output count toward stagnation; invalid plans do not. A change of unmet objective or compatible branch resets tracking.

Selecting an alternative. The current best must belong to the checkpoint's branch, and the unmet objective must be protected by that checkpoint. Alternatives come from its saved pool and must remain active, satisfy all protected objectives, be untried for the checkpoint, and lie outside the failed branch. Ranking uses satisfied-objective count, total gap on the originally unmet objectives, diversity, and a stable tie-break key. Up to three alternatives are supplied to the planner. An earlier compatible checkpoint can be used if a newer one has no eligible alternative. Without an alternative, no branch is deactivated.

Executing rollback. Lineage is registered before evaluation trimming using canonical identities and parent relations. BRICS records both parents, and post-crossover edits record the actual intermediate. Rollback deactivates the failed root and its transitive descendants; a crossover descendant is inactive if either parent is inactive. Active memory is rebuilt, while the full archive, global deduplication state, cache, operator history, and consumed budget remain intact.

The next plan must select its primary parent from the supplied alternatives and include it in elite local refinement. Its focus set is validated against that parent's current unmet objectives. The rollback context persists through an unsuccessful plan repair and is consumed only after a valid plan executes. The default limit is three rollbacks per lead. Rollback is disabled when the remaining budget is insufficient or the archive already contains a successful candidate, including when continued search is enabled.

B Experimental Protocol

B.1 Data Sources and Property Predictors

Table 8: MOMO input files and correspondence to the published benchmark.

Task	Leads	MOMO data file	Benchmark correspondence
Task 1	800	`qed_test.csv`	Task 1: QED and similarity
Task 2	800	`logp_test.csv`	Code-distributed pLogP task; distinct from published Task 2
Task 3	780	`qeddrd_test.csv`	Task 3: QED, DRD2, and similarity
Task 4	801	`gsk3_test.csv`	Task 6: QED, GSK3β, normalized SA, and similarity

Table 8 identifies the fixed input files from MOMO (Xia et al., 2024); no resampling is applied. Our Task 2 uses the code-distributed pLogP task, whereas the published benchmark's Task 2 optimizes ligand efficiency and similarity. The two LLM configurations use matching lead IDs and starting structures within each task.

DRD2 and GSK3β use the fixed predictors inherited from MOMO (Xia et al., 2024). DRD2 uses radius-3 feature Morgan count fingerprints folded to 2048 dimensions; GSK3β uses radius-2, 2048-bit Morgan fingerprints. Both activity scores are the second column of the respective class-probability output. The implemented pLogP score subtracts raw SA from RDKit MolLogP, and improvement is measured relative to the original lead. Normalized SA is $(10 - \mathrm{SA}_{\mathrm{raw}})/9$, with larger values preferred.

B.2 Run Settings and Baseline Configurations

Table 9 lists the default search-control parameters. The normalization scales s_i, in the objective order of Table 2, are $(0.9, 0.4)$ for Task 1, $(1.0, 0.6)$ for the primary Task 2 configuration, $(0.8, 0.4, 0.3)$ for Task 3, and $(0.7, 0.4, 0.7, 0.2)$ for Task 4. These scales define the state representation; they do not add success constraints.

Table 9: Default hyperparameters.

Parameter	Default	Scope
Candidate-pool limit	20	Rebuilt each round
Primary/backup operators	At most 3 each	Disjoint groups in non-crossover stages
Regression penalty coefficient, λ	2.0	Operator utility
Exploration coefficient, β	0.1	Operator utility
Near-threshold gap	0.20	Normalized objective gap
Minimum progress threshold	0.02	Accumulated gap reduction
Stagnation patience	3	Subsequent valid-plan rounds
Regression threshold	0.25	Decrease in normalized margin
Maximum number of rollbacks	3	Executed rollbacks per lead molecule
Semantic plan repairs	1	Separate from transport retries

Baselines. MOMO starts from the original lead without QMO seeds, followed by Gaussian latent-space perturbations and evolutionary search. MOMO-QMO uses QMO initialization (Hoffman et al., 2022) for Tasks 1–3, falling back to the original lead when no seed is available. Both configurations use identical original-lead Gaussian initialization for Task 4 and therefore share its per-lead results. The baseline configuration uses a population of 100, 200 offspring, 300 disturbance candidates, and at most 100 generations.

AFMOMO and reproducibility. The recorded LLM identifiers are `deepseek/deepseek-v4-flash` and `qwen/qwen3.8-27b`. AFMOMO limits each lead to 5,000 oracle calls and 100 rounds. Per-lead seeds are derived from a stable hash of the base seed and lead identifier, followed

by round-level execution seeds. Remote model outputs and service versions introduce additional variability. Batches partition the input leads within a configuration; they are not independent repeated runs. The reported results do not include repeated-run uncertainty estimates.

B.3 Ablation Configurations

Randomized planning. AFMOMO (Random) constructs plans without an LLM. It randomly selects the primary parent, exploration parent, crossover pairs, and one primary and one backup operator for elite local refinement and diverse exploration. For multiple parents, stage ratios are sampled from $(0.40, 0.25, 0.15, 0.20)$ and $(0.25, 0.20, 0.10, 0.45)$; for a single parent they are $(0.70, 0, 0, 0.30)$, in execution order.

All unmet constraints of the selected primary parent become the focus set. Post-crossover refinement retains history-based operator selection from the rule-based planner; crossover uses BRICS. Each primary or backup group contains one operator with unit weight. Candidate memory, validation, evaluation, and checkpoint mechanisms remain active, including restrictions on eligible parents after rollback. This configuration evaluates LLM-guided plan construction within the shared editing and feedback framework. Results are reported in Table 3 and discussed in Section 4.4.

Direct LLM generation. The Direct-LLM variant uses DeepSeek v4 flash to propose candidate SMILES directly in place of structured genetic search. We evaluate it on the first 10 leads from each task. SR includes all 10 leads, whereas mean oracle counts are compared only on the subset successfully optimized by both Direct-LLM and AFMOMO. Task 2 uses batch-equivalent oracle counts for both methods. Results are reported in Table 5 and discussed in Section 4.4.

B.4 Oracle Accounting and Result Aggregation

Evaluation counter. The AFMOMO counter measures computational evaluation calls. Initial lead scoring is free. The evaluator checks an original-string cache, increments the counter on a miss, and then checks the canonical cache. Consequently, a different string for an already cached structure can incur a count. Main-loop canonicalization and deduplication reduce this discrepancy. All task properties for one candidate count together as one call; LLM requests and generation are excluded.

Task 2 milestones. Task 2 tracks two similarity milestones, $\mathrm{SIM} > 0.4$ and $\mathrm{SIM} > 0.6$, both requiring positive pLogP improvement. A lead contributes to the single reported SR only when both milestone flags are true. Separate first-success counters record when each milestone is reached and persist across rollback. A high-threshold hit also satisfies the lower threshold. Once both milestones are reached, evaluation stops immediately unless continued search is enabled.

Completed-round counts. For Tasks 1, 3, and 4, AFMOMO checks success at round entry. A completed round may therefore include evaluations after the first successful candidate. Figure 3 uses recorded total-run counts for these tasks rather than reconstructed first-success counts. Some MOMO runs exceed AFMOMO's 5,000-call cap; final SR includes their outcomes regardless of call count.

Cumulative success curves. At a finite cutoff b, SR counts successful leads whose recorded calls are strictly below b, divided by all leads. Denominators are 800, 800, 780, and 801 for Tasks 1–4. Cutoffs range from 100 to 5,000 in increments of 100. The separate "5000+" point includes every successful lead and denotes final SR, not an additional numerical budget.

For Task 2, success requires both milestone flags to be true. AFMOMO uses `batch_equivalent_oracle_calls`, defined as the actual oracle calls plus the candidates left unevaluated in the success-reaching batch after early stopping. This charges the entire batch, including generated candidates that were not evaluated, rather than only the calls made before success. MOMO uses the larger of `oracle_calls_04` and `oracle_calls_06`, representing the cumulative count when both milestones have been reached. For each method, a successful lead contributes to the curve only when its corresponding count is strictly below the oracle cutoff.

Costs on jointly solved leads. Table 4 restricts each task to leads solved by both MOMO and AFMOMO–Qwen and computes each method's mean recorded calls on that same set. The task

average gives each of the four tasks equal weight. The reported 81.13% reduction is one minus the ratio of these task-averaged counts; it is neither a lead-weighted reduction nor the mean of the four percentage reductions. This completed-run analysis complements the all-lead success curves. Percentages are reported to two decimal places, with exact ties rounded up.

C Supplementary Analysis of Branch Recovery

Definition and counting unit. A post-rollback recovery is a task–lead instance with at least one logged rollback whose first observed successful round occurs after its last rollback. Each lead is counted once, regardless of the number of rollback events. The recovery fraction divides recovered leads by all leads exposed to rollback. Linking rollback records, round-level success counts, and final outcomes identifies these cases; every counted recovery has zero successful candidates in its rollback rounds.

Table 10: Post-rollback recovery. Leads counts distinct task–lead instances with rollback; Recovered counts those subsequently reaching joint success. The fraction is Recovered/Leads. No rollback occurs in Task 2, so its fraction is undefined.

Task	Events	Leads	Recovered	Fraction
AFMOMO / DeepSeek v4 flash				
Task 1	3	3	2	66.67%
Task 2	0	0	0	—
Task 3	150	128	49	38.28%
Task 4	72	58	40	68.97%
Pooled	225	189	**91**	**48.15%**
AFMOMO / Qwen3.8-27B				
Task 1	6	5	3	60.00%
Task 2	0	0	0	—
Task 3	110	95	30	31.58%
Task 4	32	28	18	64.29%
Pooled	148	128	**51**	**39.84%**

Task-level interpretation. Table 10 separates rollback events from distinct affected leads. Task 3 accounts for 128 of 189 affected leads with DeepSeek and 95 of 128 with Qwen, and has lower recovery fractions than Task 4 for both backbones. Together with its lower overall success rate in Table 3, this pattern shows that Task 3 remains challenging both overall and within the subset of leads exposed to rollback. These observations describe trajectories under the full system. Recovery fractions do not establish a causal rollback benefit or rank the backbones, because the affected lead sets differ and no matched no-rollback control is reported.