ORACLE-FREE AUTONOMOUS POST-TRAINING FOR CHEMISTRY VIA DUAL-STREAM DATA SYNTHESIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models have shown strong potential in chemistry, but employing them for long-tail chemistry tasks still requires domain adaptation, a process that remains labor-intensive. Recent AI-for-AI agents have demonstrated success in automating this process across domains such as programming and mathematics. However, these methods rely on rich training data and ready-made evaluation oracles, which long-tail chemistry rarely offers. To this end, we propose AutoChem, an oracle-free agentic system that starts with a single chemistry task description, builds its own training data and evaluation oracle, and post-trains a model to meet the task's requirements. At its core is a dual-stream data synthesis stage, in which two agents independently build two data synthesis engines, so that either stream can train the model while the other grades it. AutoChem then post-trains on the samples from one stream and evaluates on the other, so that gains reflect learned chemistry rather than leakage from the training engine into the test set. Extensive experiments across three scientific discovery tasks and eleven tasks from four chemistry benchmarks demonstrate AutoChem's effectiveness, generalization, and applicability.

1 INTRODUCTION

Large language models (LLMs) demonstrate broad general-purpose capabilities. However, their deployment in specialized domains (e.g., medicine, finance, and chemistry) typically necessitates domain-specific adaptation [41, 46, 49]. Post-training remains the predominant approach for endowing LLMs with domain expertise, but it is frequently labor-intensive. For instance, practitioners must manually curate data, configure training protocols, and conduct task-specific evaluations for each new application. To mitigate these costs, AI-for-AI (AI4AI) has been proposed as a paradigm that leverages agents to automate the post-training workflow [9, 18]. For example, LaMDAgent [48] explores a predefined search space of datasets and training recipes to produce a fine-tuned model that maximizes an oracle-defined performance score, whereas PostTrainBench [37] investigates the extent to which LLM agents can perform post-training autonomously across multiple domains.

Despite the success of AI4AI in data-rich domains such as programming and mathematics, empowering LLMs with chemistry capabilities remains challenging due to the **data scarcity of both training and evaluation for scientific discovery in chemistry**. Unlike well-mined areas, where large-scale databases such as the PDB [4] and ChEMBL [15] have enabled model training at scale, the long tail of discovery tasks, *e.g.*, supramolecular host–guest recognition, lacks resources of comparable size [40]. Specifically, these tasks require high-quality data from wet-lab experiments rather than human annotation, making them expensive to construct and rarely available for long-tail chemistry tasks. For instance, cucurbit[7]uril (CB[7]), a cornerstone host molecule in drug delivery and biosensing [22], has merely 231 guests with measured binding affinities over two decades [5, 20]. Due to this scarcity, it is hard to equip a model with the domain knowledge a task demands, and even harder to evaluate whether the model can facilitate scientific discovery in practice.

A natural direction is data synthesis, where a strong teacher model generates question-answer pairs for student models [19, 24, 45], and AI4AI agents can in principle run this synthesis end to end. However, a task description alone does not tell a synthesizer which chemistry capabilities the task demands, so the synthesized samples may drift from the task or collapse onto a handful of scaffolds. Moreover, without evaluation oracles, agents may struggle to identify the misalignment between

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.

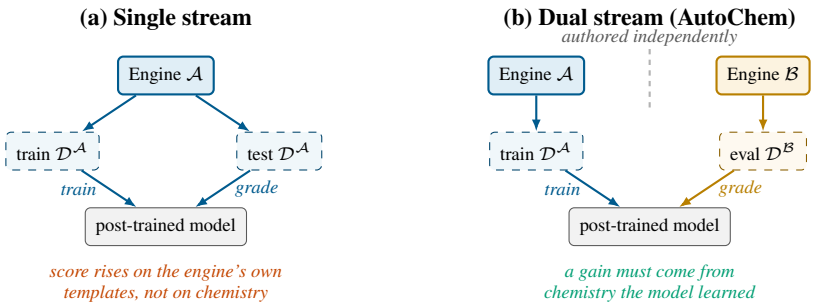


Figure 1: **Why two streams.** (a) With no evaluation oracle, a single synthesis engine supplies both the training and the test data, so a post-trained model depends purely on the single synthesis engine. (b) AutoChem employs agents to build two engines independently, and each serves as training or evaluation interchangeably, so the score reflects the chemistry the model has learned.

the required capabilities and the synthesized data. This raises a natural question: *Can an agent automatically manufacture both training and evaluation data for long-tail chemistry tasks to post-train and evaluate a model?*

To address these challenges, we propose AutoChem, an oracle-free agentic system that post-trains a model from a task description without a training pool or evaluation oracles. AutoChem first profiles the task with a research agent, which identifies the chemistry capabilities the task requires and the public sources valid for synthesis, *e.g.*, PubChem and ChEBI, so every engine is built toward the required capabilities. At its core is a dual-stream data synthesis stage in which two synthesizer agents independently author two synthesis engines, *i.e.*, executable programs that generate and annotate samples, and a verifier agent admits an engine only if it reproduces the given solved examples. We then score each admitted engine for sample coverage and diversity, and stratify samples by difficulty so the synthesized data spans the task's required capabilities rather than the engine's favorite scaffolds. Since the two engines are authored independently, a model cannot pass the other stream's questions by reproducing the templates and scaffolds, which turns the missing evaluation oracle into a by-product of data synthesis and exposes any misalignment between the two streams, as illustrated in Figure 1. Afterward, a search loop post-trains the model on one stream, evaluates it on the other, and varies the data mixture, the response format, and the training recipe across rounds, so gains reflect learned chemistry rather than leakage from the training engine into the test set.

We evaluate AutoChem on three chemistry discovery tasks and eleven chemistry tasks from four benchmarks. Across four base models, AutoChem achieves the best performance in 30 of 44 benchmark settings. Our ablations also demonstrate that removing dual-stream synthesis produces the largest performance drop among the evaluated components on every backbone, supporting its role in the system. Our contributions are threefold:

- We formalize the problem as *oracle-free autonomous post-training for chemistry*, in which the agent merely receives a chemistry task description and delivers a model tuned toward the capabilities the task requires.
- We propose AutoChem, an oracle-free agentic system that post-trains a model from a task description, where a research agent profiles the required capabilities, two synthesizer agents author verified data synthesis engines, and a search loop trains on one stream and evaluates on the other without any human-provided data or oracle.
- Across three chemistry discovery tasks and eleven chemistry tasks from four benchmarks, AutoChem outperforms baseline methods, including a general coding agent and both general and chemistry-specific data synthesis methods, and stays on par with the strongest general method on PostTrainBench beyond chemistry.

2 Methodology

Unlike existing agentic post-training benchmarks [26, 37, 48], where training and evaluation sets are provided or accessible online, we aim to mimic real-world scenarios of scientific discovery in

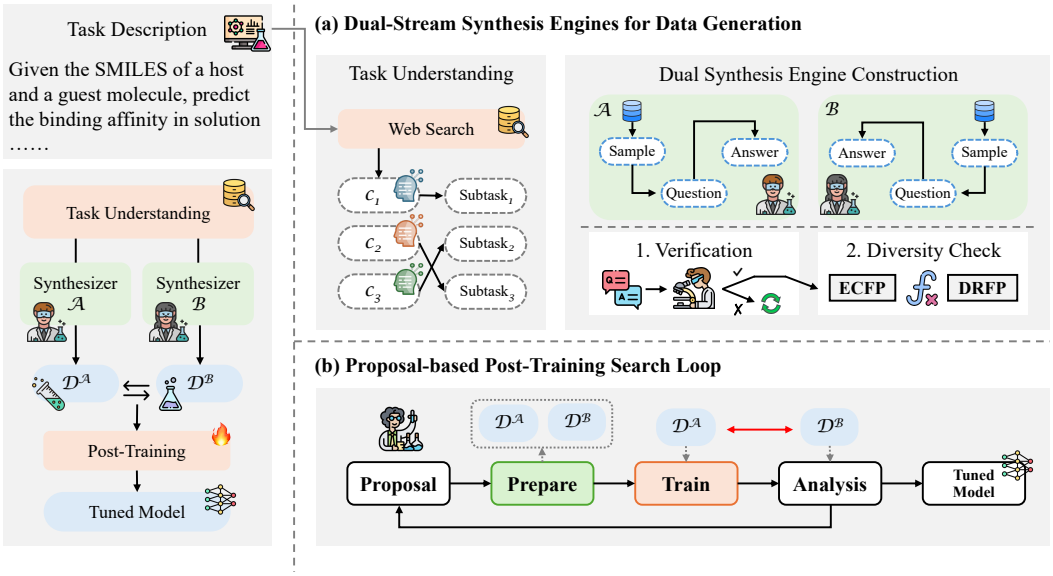


Figure 2: **The overall framework of AutoChem.** Our proposed method merely takes a task description with few-shot examples as input, and post-trains the model over the synthesized training data and evaluates it on the constructed evaluation oracle to find the optimal model.

chemistry, where (i) only a handful of solved examples with gold labels are available, and (ii) no evaluation oracle exists, *i.e.*, neither evaluation samples nor a protocol to assess the model performance. In light of this, we formulate the problem as follows:

Problem 1 (Oracle-Free Autonomous Post-Training for Chemistry)**.** *Given a chemistry task $\mathcal{T} = (T, \mathcal{E}, M_\phi)$, where T is a natural-language task description, $\mathcal{E} = \{(x_j, y_j)\}_{j=1}^{k}$ is a set of k solved examples, and M_ϕ is a base model with parameters ϕ, the objective is to design a method F that synthesizes the training and evaluation data and post-trains the base model, which is formulated as:*

$$F(\mathcal{T}) = \big(M_{\phi^*},\ \mathcal{D}_{syn},\ \mathcal{D}_{eval}\big), \qquad \phi^* = \arg\max_{\phi'}\ s\big(M_{\phi'};\ \mathcal{D}_{eval}\big), \tag{1}$$

where $\mathcal{D}_{syn}$ and $\mathcal{D}_{eval}$ are the synthesized training and evaluation sets with $\mathcal{D}_{syn} \cap \mathcal{D}_{eval} = \emptyset$, and $s(\cdot\,; \mathcal{D}_{eval}) \in \mathbb{R}$ is a scorer.

2.1 Overview

In this section, we present the details of AutoChem, a multi-agent system that contains an orchestrator agent to organize the workflow over subagents, *i.e.*, a research agent for task understanding, two synthesizers and a verifier for synthesis engine construction, and a proposer, a trainer, and an evaluator for the proposal-based post-training search loop. The workflow includes two key stages: (i) the dual-stream data synthesis, where the two synthesizers independently author synthesis engines for training and evaluation sample construction, and (ii) the proposal-based post-training search loop, which post-trains the base model over the dual-stream data synthesis engines. Figure 2 illustrates the overall framework.

2.2 Dual-Stream Synthesis Engines for Data Construction

Given a complex chemistry task $\mathcal{T} = (T, \mathcal{E}, M_\phi)$, AutoChem conducts analysis to understand the model capabilities required for the task, and then forks synthesis into two independent streams, with each delivering a set of *synthesis engines* E for training and evaluation sample generation.

Chemistry Task Understanding. A complex chemistry task commonly demands model capabilities in different directions [17, 25]. To identify the required capabilities $\mathcal{C}$ and facilitate data synthesis for model training, a research agent first conducts research on the task $\mathcal{T}$ using search tools, and decomposes the capabilities $\mathcal{C}$ into m fine-grained subtasks $\{\tau_i : \tau_i = (c_i, \mathcal{S}_i, \mathcal{U}_i)\}_{i=1}^{m}$ with capability

description $c_i \in \mathcal{C}$, source chemistry databases $\mathcal{S}_i \subseteq \mathcal{S}$ for data synthesis, and a set of evaluation metrics $\mathcal{U}_i$, *e.g.*, accuracy, F1, or BLEU.

Dual-Stream Data Synthesis Engines. Samples from one synthesis method may share molecule scaffolds and question patterns, which models can memorize without learning the chemistry capabilities. Therefore, AutoChem builds two synthesis engine sets, denoted as $\mathcal{A}$ and $\mathcal{B}$, to separate training set $\mathcal{D}_{\text{syn}}$ from evaluation set $\mathcal{D}_{\text{eval}}$ at the chemistry level, *i.e.*, $(\mathcal{D}_{\text{syn}}, \mathcal{D}_{\text{eval}}) \in \{(\mathcal{D}^{\mathcal{A}}, \mathcal{D}^{\mathcal{B}}), (\mathcal{D}^{\mathcal{B}}, \mathcal{D}^{\mathcal{A}}) : \mathcal{D}^{\mathcal{A}} \sim \mathcal{A}, \mathcal{D}^{\mathcal{B}} \sim \mathcal{B}\}$. Specifically, we feed each subtask $\tau_i = (c_i, \mathcal{S}_i, \mathcal{U}_i)$ into two synthesizer agents to build the synthesis engines independently. To prevent hacking, for each synthesizer, the orchestrator splits the examples into a shown subset $\mathcal{E}_i^{\text{s}}$ and a hidden subset $\mathcal{E}_i^{\text{h}}$. Afterward, the synthesizer authors a data engine $e_i = (\pi_i, g_i, f_i)$, encompassing a sampler π_i to draw a valid input $z \in \mathcal{S}_i$, a question generator g_i that formalizes the input z into a question x, and an answer function f_i that generates the corresponding reference answer y. Mathematically, an engine composes its three functions into a single sampling procedure, *i.e.*,

$$e_i(\mathcal{S}_i) = \big(g_i \circ \pi_i(\mathcal{S}_i),\ \ f_i \circ g_i \circ \pi_i(\mathcal{S}_i)\big) = (x, y), \tag{2}$$

and the three stages are computed as

$$z = \pi_i(\mathcal{S}_i), \qquad x = g_i(z), \qquad y = f_i(x) = \begin{cases} \text{prog}_i(x), & \text{if answer is computable,} \\ \text{LLM}(\rho_i, x, \mathcal{F}_i), & \text{otherwise,} \end{cases} \tag{3}$$

where prog_i is a program backed by chemistry packages [23], ρ_i is a prompt instruction for LLMs to answer the question that is not directly computable by programs, equipped with a set of tools $\mathcal{F}_i$ that the LLM can leverage to solve the problem. As a result, the two synthesizers deliver two sets of synthesis engines, *i.e.*, $\mathcal{A} = \{e_i^{\mathcal{A}}\}_{i=1}^m$ and $\mathcal{B} = \{e_i^{\mathcal{B}}\}_{i=1}^m$, to sample data $\mathcal{D}^{\mathcal{A}}$ and $\mathcal{D}^{\mathcal{B}}$ that serve as training and evaluation sets interchangeably.

Synthesis Engine Verification. With the synthesis engines, we introduce a verifier agent to review whether these engines are chemically valid and leakage-free. Moreover, the verifier executes each question-answer function pair (g_i, f_i) on the corresponding hidden samples to ensure the correctness of the functions. An engine is verified only if the answer function passes every evaluation metric of the subtask within tolerance ϵ:

$$\mu\big(f_i(x),\, y\big) \ \geq\ 1 - \epsilon, \quad \forall\, (x, y) \in \mathcal{E}_i,\ \forall\, \mu \in \mathcal{U}_i, \tag{4}$$

where the tolerance is $\epsilon = 0$ under exact metrics, and a threshold proposed by the research agent otherwise. Synthesizers further revise synthesis engines that fail verification.

Diversity Score. Even though a verified engine successfully reproduces the few-shot examples, it may still drift away from the task or collapse onto a handful of skeletons [7, 33, 50]. In light of this, we draw a pilot batch $\mathcal{P}_i = \{z_t\}_{t=1}^{n_i}$ from the sampler π_i of each engine and compute two scores over this batch, *i.e.*, a coverage score cov_i of the shown examples and a diversity score div_i of the batch. Given two question-answer pairs p and p', and a similarity threshold θ, we have:

$$\text{cov}_i = \frac{1}{|\mathcal{E}_i^{\text{s}}|} \sum_{p \in \mathcal{E}_i^{\text{s}}} \mathbf{1}\Big[\max_{t \leq n_i} \kappa(p, p_t) \geq \theta\Big], \qquad \text{div}_i = 1 - \frac{2}{n_i(n_i - 1)} \sum_{t < t'} \kappa(p_t, p_{t'}), \tag{5}$$

where the kernel function $\kappa(\cdot)$ is computed as

$$\kappa(p, p') = \text{Mean}\Big(\text{Tan}\big(\text{ECFP}(p),\, \text{ECFP}(p')\big),\ \text{Tan}\big(\text{DRFP}(p),\, \text{DRFP}(p')\big)\Big), \tag{6}$$

where $\text{ECFP}(\cdot)$ is the extended-connectivity fingerprint [38], $\text{DRFP}(\cdot)$ is the differential reaction fingerprint [34], and $\text{Tan}(\cdot, \cdot) \in [0, 1]$ is the Tanimoto coefficient [36]. The algorithms of $\text{ECFP}(\cdot)$ and $\text{DRFP}(\cdot)$ are detailed in Appendix F.

Intuitively, a low cov_i indicates that part of the shown examples is unmatched by the batch, while a low div_i indicates that the batch concentrates on a few molecule skeletons. In either case, the engine is sent back to its synthesizer together with the two scores.

Data Difficulty. We compute a difficulty score $d(x, y)$ for each generated pair, which combines the number of tool operations $d^{\text{tool}}(x)$ executed by the answer function f_i, *i.e.*, the operations in prog_i or the calls to the tools in $\mathcal{F}_i$, with the per-token negative log-likelihood of the reference under the base model:

$$d(x, y) = \text{Mean}\Big(\text{rank}_i\big(d^{\text{tool}}(x)\big),\ \text{rank}_i\big(-\tfrac{1}{|y|} \log P_\phi(y \mid x)\big)\Big), \tag{7}$$

where rank_i is the within-subtask quantile rank, which assigns each pair to one of L_d difficulty strata. With the difficulty strata, a round in the search loop realizes its mixture by generating a pool and retaining samples from each subtask and stratum in the requested proportion.

2.3 Why Dual Streams?

As discussed in Section 2.2, AutoChem post-trains a model on $\mathcal{D}_{\text{syn}} \in \{\mathcal{D}^{\mathcal{A}}, \mathcal{D}^{\mathcal{B}}\}$ and evaluates it on $\mathcal{D}_{\text{eval}} = \{\mathcal{D}^{\mathcal{A}}, \mathcal{D}^{\mathcal{B}}\} \setminus \{\mathcal{D}_{\text{syn}}\}$. We enforce $\mathcal{D}_{\text{syn}} \cap \mathcal{D}_{\text{eval}} = \emptyset$ to remove the overlapping pairs at the chemistry level [3, 44, 47]. The key insight here is that, with a single data synthesis stream, the model would learn the patterns the engine generates rather than the chemistry. Specifically, an engine reuses its own scaffolds and question templates, so a model post-trained on its samples answers the same engine's questions partly by reproducing those habits rather than by solving the chemistry. In contrast, a separate data synthesis engine over different examples and sources removes this concern. Therefore, a gain measured across the two streams reflects the chemistry the model has learned, which the search loop in Section 2.4 needs for post-training.

2.4 Post-Training via Search Loop

The dual-stream engines supply training samples from one stream and an evaluation oracle from the other, while leaving the sample mixture drawn from the pool and the training recipe open. To search for both, we introduce a search loop with three agents, *i.e.*, a proposer, a trainer, and an evaluator, which trains on dual-stream synthesis engines $(\mathcal{A}, \mathcal{B})$. Intuitively, we do not force the agents to use a specific synthesis engine as the training source, and they are free to use the two engines in either direction. In the remainder of this section, we use $\mathcal{A}$ for training and $\mathcal{B}$ for evaluation.

Round Proposal. The loop runs at most R rounds, each producing one candidate checkpoint $h_r \in \mathcal{H}$. From the past proposals and outcomes, the proposer changes exactly one factor per round, *i.e.*, the data mixture over subtasks and difficulty strata, the response format, or the training recipe.

Cross-Stream Training and Evaluation. The trainer generates a sample pool with the engines of stream $\mathcal{A}$, keeps the proposed mixture, and removes any sample that overlaps the evaluation set. It then fine-tunes the base model on the training set by SFT [32] or by reward-based optimization [39] for non-computable answers. Every round starts from the base model, so the rounds stay comparable. The evaluator scores the checkpoint on the evaluation set, drawn from the synthesis engine $\mathcal{B}$. Since the two streams are authored independently, this score reflects the chemistry the checkpoint has learned rather than the templates it has memorized.

Acceptance and Termination. The base model M_ϕ is the initial best checkpoint with score s_{best}. A round r is accepted only if $s_r - s_{\text{best}} > \sigma$, where s_r and s_{best} are the scores of the round and the current best checkpoint, and σ is an acceptance margin proposed by the proposer agent from a pilot draw disjoint from the evaluation set, so that a gain within the noise of that draw is not accepted. An accepted checkpoint becomes the new best checkpoint, and the ledger records the changed factor, the coverage and diversity of its samples, and the margin. The loop stops after a fixed number of consecutive rejections or at round R, and returns the best checkpoint as M_{ϕ^*}.

3 Experiments

3.1 Experiments for Real-world Applications

To validate the applicability of AutoChem, we conduct experiments on data-scarce chemistry discovery tasks, *i.e.*, host–guest binding affinity, hydration free energy, and toluene/water partitioning. Table 1 reports the performance comparison over the three tasks, and the details of the datasets are provided in Appendix B.1. According to the table, we find that: (i) All post-training methods outperform the base model on tasks SupraBank and FreeSolv, indicating that **data synthesis methods can contribute to scientific discovery in chemistry** in tasks where rich and high-quality samples are hard to obtain, *e.g.*, from wet-lab experiment results. (ii) Claude Code is the strongest baseline and ranks second in most settings, but it trails AutoChem on most benchmarks. This indicates that **the gain of AutoChem comes from its design rather than from the capability of its agents**, since Claude Code is built on similar reasoning and tool-use abilities. (iii) AutoChem achieves the best

Table 1: **Performance comparison over chemistry scientific discovery tasks.** In each group, **bold** highlights the best and underline the second best.

	Method	SUPRABANK			FREESOLV			SAMPL9		
		MAE ↓	RMSE ↓	Acc@1 ↑	MAE ↓	RMSE ↓	Acc@1 ↑	MAE ↓	RMSE ↓	Acc@1 ↑
Qwen3-4B	Base	4.59	5.78	16.83	3.35	4.15	11.63	1.95	**2.19**	25.00
	Claude Code	1.51	2.12	48.24	**2.11**	3.10	35.66	**1.73**	2.23	**37.50**
	ChemOrch	2.31	3.29	38.69	2.59	3.51	26.36	1.79	2.92	**37.50**
	ANDES	2.84	3.62	34.67	2.50	3.45	27.91	1.82	2.22	31.25
	AutoChem	**1.35**	**1.93**	**50.25**	2.20	**2.92**	**37.21**	2.27	2.73	**37.50**
Llama3-8B	Base	6.75	8.13	8.54	5.47	6.92	6.98	**1.67**	**2.11**	18.75
	Claude Code	1.59	2.16	41.46	**2.02**	**3.01**	40.31	2.66	3.72	31.25
	ChemOrch	2.39	3.08	36.43	2.41	3.65	34.88	2.56	3.60	31.25
	ANDES	3.50	4.36	29.15	3.59	4.72	25.58	2.23	3.04	25.00
	AutoChem	**1.42**	**2.01**	**44.47**	2.15	3.17	**41.86**	2.47	2.96	**37.50**

performance in 11 out of 18 settings, and ranks first on every Acc@1 column. This demonstrates **the effectiveness of AutoChem in autonomous post-training for chemistry tasks**.

3.2 EXPERIMENTS OVER CHEMISTRY BENCHMARKS

Table 2 reports the performance comparison of baseline methods and AutoChem over eleven chemistry tasks. Implementation details are provided in Appendix B. According to the table, we find that: (i) All post-training methods outperform the base models in most settings. This result indicates **the necessity of domain adaptation for general-purpose LLMs on chemistry tasks**. (ii) The chemistry-specific baseline, *i.e.*, ChemOrch, outperforms the general data synthesis methods, *i.e.*, LLM2LLM, Prismatic, and ANDES, in most settings, particularly on CHEMCOTBENCH. This demonstrates that **general-purpose data synthesis methods fall short on chemistry-specific tasks**, since they synthesize samples without chemistry tools to ensure that the samples are chemically valid. (iii) Claude Code generally outperforms the above data synthesis methods. This indicates **the potential of agentic systems for post-training**, where a general agent with reasoning, coding, and tool-use abilities synthesizes more task-aligned samples than fixed synthesis pipelines. (iv) AutoChem achieves the best performance in 30 out of 44 tasks and ranks within the top two in 35 tasks. Particularly, on the out-of-distribution split of MUMOINSTRUCT with `Qwen3-4B`, AutoChem achieves 45.57, compared with 30.36 for the best baseline method. This demonstrates **the effectiveness of AutoChem in oracle-free autonomous post-training for chemistry tasks**, where the verified dual-stream engines provide reliable training samples and the cross-stream evaluation guides the search loop toward real improvements.

3.3 RESULTS OVER POSTTRAINBENCH

To assess the capability of AutoChem beyond chemistry-specific tasks, we conduct experiments over POSTTRAINBENCH [37], following the settings in ANDES [51] using backbone `Qwen3-4B-Base`. According to Figure 3, all methods improve the base model on all three tasks. AutoChem performs on par with ANDES, the strongest general data synthesis method, and outperforms Claude Code on GSM8K and HUMANEVAL. This demonstrates **the applicability of AutoChem beyond chemistry**, where its verified engines and cross-stream evaluation remain effective on general-purpose tasks without any chemistry-specific component.

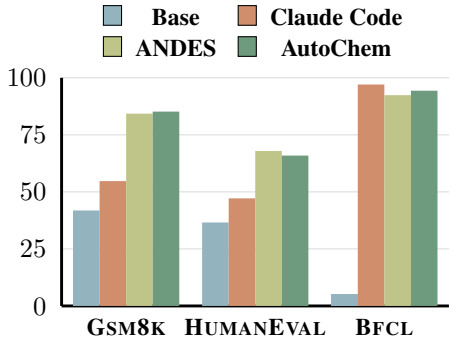


Figure 3: **Results on POSTTRAINBENCH with backbone `Qwen3-4B-Base`.**

Table 2: **Performance comparison between baseline methods and AutoChem.** In each group, **bold** highlights the best and underline the second best.

	Method	ChemCoTBench				MuMOInstruct		SMolInstruct			MoleculeQA		Rank ↓
		MolEdit ↑	MolOpt ↑	MolUnd ↑	RxnPred ↑	IND ↑	OOD ↑	Cap ↑	Gen ↑	BBBP ↑	Src ↑	Use ↑	
Qwen3-4B	Base	22.00	19.67	34.06	8.21	3.28	3.45	5.26	0.04	30.96	38.79	24.71	6.55
	Claude Code	61.00	60.00	49.06	10.11	29.92	15.41	16.19	0.56	82.74	44.08	30.22	3.00
	ChemOrch	37.00	60.67	55.94	9.47	11.76	21.74	10.89	0.32	67.51	36.26	**31.39**	3.73
	LLM2LLM	34.00	30.50	40.62	8.21	15.46	30.36	20.77	0.32	59.90	38.79	27.21	4.45
	Prismatic	38.00	37.50	41.25	8.84	14.28	26.12	17.62	0.60	60.41	34.40	29.22	4.09
	ANDES	29.00	43.00	53.75	8.63	6.68	18.15	16.47	**1.00**	69.54	35.82	27.05	4.45
	AutoChem	**66.00**	**63.00**	**59.38**	**10.53**	**31.06**	**45.57**	**21.75**	0.40	**87.31**	**47.73**	30.05	**1.45**
Qwen3-8B	Base	32.00	26.67	45.62	9.47	6.00	9.22	10.26	0.16	30.96	36.49	30.05	5.91
	Claude Code	66.00	61.33	56.88	9.05	**34.38**	11.10	16.03	0.56	84.26	43.56	38.90	3.27
	ChemOrch	51.00	56.33	61.56	10.32	14.46	31.39	13.23	0.64	77.16	34.33	33.72	3.18
	LLM2LLM	42.00	17.50	50.31	9.26	10.96	26.48	20.58	0.24	31.47	36.49	28.88	5.09
	Prismatic	47.00	19.83	51.88	9.26	11.60	25.20	17.85	0.52	52.28	31.72	31.39	4.64
	ANDES	26.00	43.50	61.56	9.47	10.50	26.05	16.26	**0.92**	35.53	38.72	29.55	4.09
	AutoChem	**68.00**	**68.67**	**64.38**	**11.16**	32.52	**40.89**	**21.37**	0.48	**90.86**	**49.59**	**39.57**	**1.45**
Llama3-3B	Base	0.00	6.67	0.31	2.95	0.24	0.60	6.37	0.04	31.47	32.39	28.71	6.36
	Claude Code	52.00	46.67	47.81	**6.95**	19.12	7.90	16.58	0.20	80.71	36.71	31.89	2.64
	ChemOrch	32.00	49.00	58.13	**6.95**	4.38	16.98	12.13	0.04	69.04	33.13	**33.06**	3.18
	LLM2LLM	28.00	21.00	42.81	5.26	5.80	13.77	18.83	0.08	43.65	32.54	28.05	4.64
	Prismatic	28.00	27.33	43.44	5.47	9.98	**17.33**	11.17	0.00	72.08	24.35	28.21	4.45
	ANDES	28.00	27.33	**73.44**	5.05	5.14	15.77	12.27	0.16	34.01	29.64	26.04	4.45
	AutoChem	**61.00**	**54.83**	70.31	5.68	**23.52**	14.66	**20.38**	**0.28**	**82.74**	**39.84**	30.55	**1.73**
Llama3-8B	Base	5.00	28.00	0.31	5.26	0.00	0.00	6.57	0.32	31.47	35.07	29.05	6.64
	Claude Code	51.00	58.33	44.69	**9.47**	**29.82**	9.61	16.06	0.96	65.48	41.92	34.22	3.18
	ChemOrch	56.00	**73.33**	60.31	8.42	9.00	24.98	13.58	1.08	59.90	40.21	33.56	3.18
	LLM2LLM	41.00	29.33	50.00	6.32	17.44	25.55	17.71	0.36	65.99	34.33	29.22	4.00
	Prismatic	39.00	31.33	47.81	6.11	14.36	23.56	16.42	0.84	**71.57**	30.23	30.05	4.36
	ANDES	29.00	50.50	**67.19**	8.00	10.18	18.65	10.05	**1.20**	46.19	29.04	27.88	4.64
	AutoChem	**58.00**	71.83	65.31	7.58	29.68	**29.82**	**24.56**	0.88	65.99	**45.94**	**37.40**	**1.91**

3.4 Data Synthesis Analysis

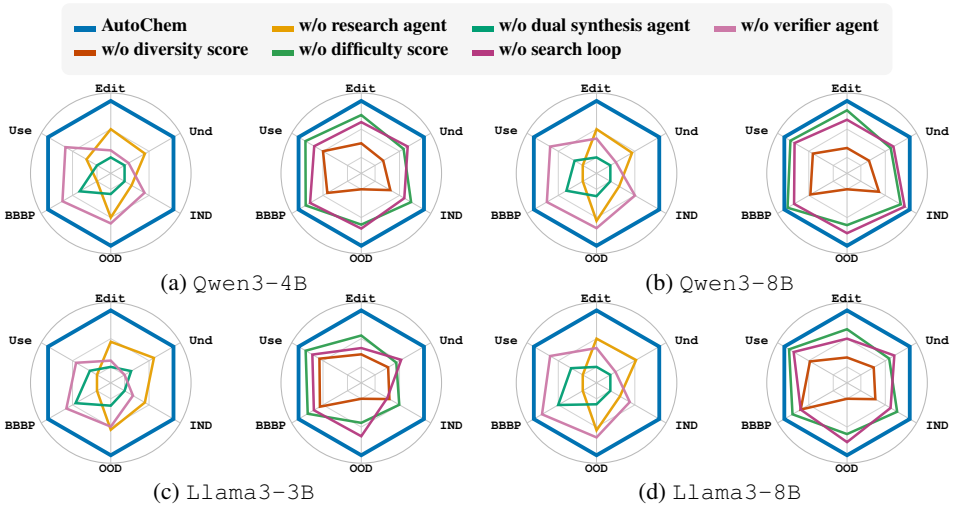
Question-Answer Pair Quality. Domain experts and LLM-as-Judge, *i.e.*, `Fable-5.1`, evaluate 100 synthesized data points for each task. According to Table 3, the two judges agree on the quality of the synthesized data, where they disagree on at most 3 of the 100 samples for every task and rate every task above 93. This indicates that **AutoChem synthesizes high-quality question-answer pairs**.

Table 3: **Analysis for data synthesis.** Expert and Fable are the judge ratings over 100 samples generated by AutoChem. Engine and Base denote the performance of the synthesis engine and the base model, respectively.

Task	Expert	Fable	Engine	Base
`MolEdit`	98.00	97.00	91.00	22.00
`MolOpt`	99.00	96.00	89.33	19.67
`MolUnd`	97.00	96.00	96.88	34.06
`Cap`	95.00	93.00	91.15	5.26
`Gen`	93.00	94.00	96.43	0.04
`Src`	96.00	97.00	96.43	38.79
`Use`	95.00	95.00	91.32	24.71

Analysis for Synthesis Engine. We evaluate the quality of the answer function inside the synthesis engine the same way we evaluate a model. As illustrated in Table 3, each engine answers the test questions and achieves a score higher than 90 in six of the seven tasks against the gold labels, indicating that **the synthesis engines generate high-quality labels**, and every base or tuned model in Table 2 sits far below it.

Dual Synthesis Engine Comparison. Table 8 (Appendix C.2) scores the final `Qwen3-4B` checkpoints on a fresh draw from the engine that trained them, on draws from the other engine, and on the benchmark test set. According to the table, we find that: (i) On average, a checkpoint scores 25 to 29 points higher on its own training engine than on the other engine, in either direction. This confirms that **a model post-trained on one engine partly reproduces that engine's regularities rather than the chemistry**. (ii) The cross-stream score falls about 9 points from the benchmark test score on average and preserves the task ordering, whereas the same-stream score deviates by 28 points and inverts it. This validates that **the cross-stream score is a faithful proxy for the missing oracle**, while a same-stream score would overstate every round. (iii) The inflation is largest on tasks where the engine itself decides what counts as a correct answer, *i.e.*, it writes the answer options in `Src` and `Use` and judges property improvement in `IND` and `OOD`, and smallest on tasks where the answer is fixed by chemistry tools or databases, *e.g.*, `MolEdit`, `MolUnd`, and `BBBP`. This suggests the model picks up the habits of the engine that grades it rather than the molecules it samples, which our dual-stream design mitigates by grading the model with an engine it was not trained on.



(a) `Qwen3-4B` (b) `Qwen3-8B`

(c) `Llama3-3B` (d) `Llama3-8B`

Figure 4: **Ablation studies over six chemistry tasks and four backbones.** For each backbone, the left radar removes an agent and the right radar removes a mechanism.

3.5 Agent Backbone Variants

Besides `Claude-Sonnet-5`, the agent backbone in Section 3, we further conduct experiments with various agent backbones, including `GPT-5.6-luna`, `MiniMax-M3`, `GLM-5.3`, `DeepSeek-v4-pro`, and `Qwen3.8-27B`, and report the results in Table 7 (Appendix C.1). According to the table, we find that (i) the performance varies substantially across agent backbones, indicating that **the performance of the tuned model largely depends on the capabilities of the agent backbones**. (ii) **AutoChem consistently outperforms the baseline methods regardless of the agent backbone**. AutoChem achieves the best performance in 75 out of the 96 settings, and its average over the four tasks and four base models exceeds ChemOrch by 7 to 10 points and ANDES by 20 to 23 points under every agent. The remaining settings are mostly on `MolUnd` and `Use`, where the margins among all methods are small.

3.6 Ablation Studies

We further ablate our design to validate the contribution of each component. Specifically, we remove three agents, *i.e.*, (i) the research agent, where the synthesizers build their engines without the instruction of required capability for the task; (ii) the dual synthesis agents, where a single synthesizer builds the synthesis engine for both training and evaluation data; and (iii) the verifier agent, where an engine is admitted without verification. Moreover, we remove three mechanisms, *i.e.*, (i) the diversity score, where we do not require revision due to low diversity scores; (ii) the difficulty score, where the trainer retains its samples without the difficulty strata; and (iii) the search loop, where we only post-train a model in a single round. We report the results over four backbones in Figure 4, and we find that: (i) Removing the dual synthesis agents causes the largest performance drop on every backbone, indicating that **the dual-stream design is the single component that contributes the most**. (ii) Removing the diversity score causes the largest drop on the OOD task of MuMOInstruct on every backbone. This validates that **the coverage and diversity of the synthesized samples carry the generalization**. (iii) Every variant falls below AutoChem, which demonstrates **the contribution of every component in AutoChem**.

3.7 Cost Analysis

We further conduct cost analysis of AutoChem against two baseline methods, *i.e.*, ChemOrch and Claude Code, and the results are listed in Table 9 (Appendix C.3). According to the table, we find that: (i) AutoChem consumes 3.5× fewer tokens than ChemOrch and ranks within the top two on every task, indicating that **verified synthesis engines are cheaper than per-sample tool grounding**, since an engine is authored once and then samples for free, whereas ChemOrch invokes

tools and recovers from execution errors for every sample. (ii) AutoChem stays within $1.4\times$ of Claude Code in total, and consumes fewer tokens on five tasks, even though Claude Code synthesizes a single training set in one pass while AutoChem additionally authors two synthesis engines. This validates that **the dual-stream design and the search loop come at a modest cost**.

4 Related Works

Agentic Post-Training. AI4AI agents automate data preparation, recipe selection, and result analysis [12, 18], but rely on evaluation data supplied by the environment: MLE-bench [9], MLE-Dojo [35], and AIDE [18] score agents on a test set fixed by the competition, and AlphaEvolve [31] evolves programs against a given evaluator [27]. Agents that post-train a model end-to-end inherit the same dependence: LaMDAgent [48] and FT-Dojo [26] search a prepared data pool under a provided evaluator, PostTrainBench [37] and EvoMaster [52] take the evaluation set as an external input, and Pioneer Agent [2] builds its own evaluation set but never verifies it against gold labels, so nothing stops the agent from rewarding its own synthesis habits. In long-tail chemistry, gold labels are a handful of wet-lab measurements, creating a data-scarcity issue for training and evaluation. Therefore, both the training and the evaluation data must be synthesized, and an agent that scores itself on its own synthesized data cannot tell learned chemistry from memorized templates. To this end, AutoChem manufactures both the training and the evaluation data from the task description, verifies the evaluation engine against the few solved examples, and grades the model with an engine authored independently of the one that trained it.

LLMs for Chemistry. Chemistry demands representations and operations that general corpora cover thinly [8, 16, 49]. Tool-augmented agents such as ChemCrow [6] keep the LLM frozen, so the model learns nothing about a new task, while specialists such as LlaSMol [49] and GeLLM3O [10] tune on curated corpora and are evaluated on curated suites such as ChemCoTBench [25] and MoleculeQA [28]. Even where a corpus exists, its fixed recipe cannot follow a new capability, so adapting a model to an unseen task restarts the curation. Both lines stop at the boundary of curated data: for host–guest recognition, every label is a wet-lab measurement [5, 22, 40], no corpus or benchmark exists, and prior work falls back to DFT surrogates [43]. AutoChem takes neither a training corpus nor an evaluation set as input and builds executable engines over public sources, so a model can be adapted to a task that has never been curated.

Data Synthesis for Post-Training. Synthetic data is the standard remedy for scarcity: Self-Instruct [45] bootstraps from a seed set, LLM2LLM [24] targets student errors, Prismatic Synthesis [19] retains sparse gradient clusters, ANDES [51] evolves a topic tree against fixed benchmarks, and ChemOrch [17] grounds each sample in chemistry tools but uses its evaluation set only for diagnosis. Self-consistency and execution feedback [11, 21] filter some errors, yet the checker shares the generator's blind spots and never confronts a gold label. These methods share two limitations: labels come from a teacher or the student itself and are never verified against the task, and training and evaluation data come from the same generator, so a student can score by reproducing its templates. AutoChem verifies an executable answer function against hidden examples before synthesizing a sample, and trains on one engine while grading on the other, accepting a round only when its cross-stream gain exceeds the margin proposed by the agent.

5 Conclusion

In this work, we present AutoChem, an oracle-free agentic system that post-trains a chemistry model from only a task description and few-shot examples, where neither a training corpus nor an evaluation protocol is given. To achieve this, we design (i) a dual-stream data synthesis stage, in which a research agent decomposes the task into subtasks with their capabilities and metrics, two synthesis agents independently author an executable engine per subtask, and a verifier agent admits an engine only after it reproduces the few-shot examples, and (ii) a search loop that post-trains on the samples of one stream and evaluates on the other, such that a round is accepted only when its gain cannot be explained by the regularities of the engine that trained the model. We further compare AutoChem with general and chemistry-specific data synthesis methods over eleven tasks from four chemistry benchmarks, three scientific discovery tasks, and a general-purpose benchmark, demonstrating the effectiveness, the generalization, and the applicability of our proposed method.

THE USE OF LARGE LANGUAGE MODELS

Large language models are the subject of this work and also a tool in its implementation, and we discuss both uses. As a subject, every agent role in AutoChem is backed by an LLM, and the training and evaluation samples in our experiments are synthesized by the engines that these agents author, as described in Section 2 and Appendix B.5. The reference answers of the synthesized samples are produced by executable programs wherever the answer is computable, and by an LLM equipped with chemistry tools otherwise. As a tool, we used LLM-based coding agents to assist code implementation and to polish this manuscript. All experiments were designed, run, and interpreted by the authors, and every LLM-assisted passage of text and code was reviewed and verified by the authors, who take full responsibility for the content of this paper.

ETHICS STATEMENT

This work involves no human subjects or personal data. All benchmark datasets and chemistry databases we use, *i.e.*, CHEMCOTBENCH, MUMOINSTRUCT, SMOLINSTRUCT, MOLECULEQA, SUPRABANK, FREESOLV, SAMPL9, PubChem, and ChEBI, are publicly available under their respective licenses, and we use them for research only. AutoChem post-trains models for molecular property prediction, editing, and optimization; like any tool that improves the chemistry capabilities of language models, such models could in principle be misused to assist in the design of harmful compounds. The tasks in this paper concern benchmark and chemistry properties, *e.g.*, binding affinity, hydration free energy, and partition coefficients, and we do not train on or release data related to hazardous substances. We believe the benefit of lowering the cost of adapting models to data-scarce, long-tail chemistry outweighs this risk, and we encourage users to apply the usual safeguards when deploying models tuned by AutoChem.

REPRODUCIBILITY STATEMENT

The problem setting is stated in Problem 1, and the complete method is described in Section 2, with the synthesis engine, verification, coverage, diversity, and difficulty definitions given as equations. Appendix B.5 lists the compute environment, the agent backbone, the number of solved examples, the student models, the evaluation harness, and decoding settings shared by every run, and Appendix B.6 lists the hyperparameters of the synthesis stage and the search loop. Appendix B.1 and Table 5 describe the datasets, their splits, and the evaluation metrics; Appendix B.4 describes the baseline configurations. We will release the source code of AutoChem, the run configurations, and the synthesized engines and samples upon publication.

REFERENCES

[1] Mohammad A Alnajjar, Werner M Nau, and Andreas Hennig. A reference scale of cucurbit [7] uril binding affinities. *Organic & biomolecular chemistry*, 19(39):8521–8529, 2021.

[2] Dhruv Atreja, Julia White, Nikhil Nayak, Kelton Zhang, Henrijs Princis, George Hurn-Maloney, Ash Lewis, and Urchade Zaratiana. Pioneer agent: Continual improvement of small language models in production. *arXiv preprint arXiv:2604.09791*, 2026.

[3] Guy W. Bemis and Mark A. Murcko. The properties of known drugs. 1. molecular frameworks. *Journal of Medicinal Chemistry*, 39(15):2887–2893, 1996.

[4] Helen M Berman, John Westbrook, Zukang Feng, Gary Gilliland, Talapady N Bhat, Helge Weissig, Ilya N Shindyalov, and Philip E Bourne. The protein data bank. *Nucleic acids research*, 28(1):235–242, 2000.

[5] Frank Biedermann and SupraBank Team. SupraBank: An open resource for intermolecular interactions. `https://suprabank.org`, 2020.

[6] Andres M Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. Chemcrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*, 2023.

[7] 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.

[8] Sébastien Bubeck, Varun Chadrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4, 2023.

[9] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. In *ICLR*, volume 2025, 2025.

[10] Vishal Dey, Xiao Hu, and Xia Ning. Large language models for controllable multi-property multi-objective molecule optimization. In *Findings of EMNLP*, 2025.

[11] Guanting Dong, Keming Lu, Chengpeng Li, Tingyu Xia, Bowen Yu, Chang Zhou, and Jingren Zhou. Self-play with execution feedback: Improving instruction-following capabilities of large language models. In *International Conference on Learning Representations*, volume 2025, pp. 39286–39313, 2025.

[12] Yaxin Du, Xiyuan Yang, Zhifan Zhou, Wanxu Liu, Zixing Lei, Zimeng Chen, Fenyi Liu, Haotian Wu, Yuzhu Cai, Zexi Liu, et al. Datamaster: Data-centric autonomous ai research. *arXiv preprint arXiv:2605.10906*, 2026.

[13] Sebastian Ehlert, Marcel Stahn, Sebastian Spicher, and Stefan Grimme. Robust and efficient implicit solvation model for fast semiempirical methods. *Journal of Chemical Theory and Computation*, 17(7):4250–4261, 2021.

[14] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL `https://zenodo.org/records/12608602`.

[15] Anna Gaulton, Louisa J Bellis, A Patricia Bento, Jon Chambers, Mark Davies, Anne Hersey, Yvonne Light, Shaun McGlinchey, David Michalovich, Bissan Al-Lazikani, et al. Chembl: a large-scale bioactivity database for drug discovery. *Nucleic acids research*, 40(D1):D1100–D1107, 2012.

[16] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[17] Yue Huang, Zhengzhe Jiang, Xiaonan Luo, Kehan Guo, Haomin Zhuang, Yujun Zhou, Zhengqing Yuan, Xiaoqi Sun, Jules Schleinitz, Yanbo Wang, Shuhao Zhang, Mihir Surve, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. ChemOrch: Empowering LLMs with chemical intelligence via synthetic instructions. In *NeurIPS*, 2025.

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

[19] Jaehun Jung, Seungju Han, Ximing Lu, Skyler Hallinan, David Acuna, Shrimai Prabhumoye, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, and Yejin Choi. Prismatic synthesis: Gradient-based data diversification boosts generalization in llm reasoning. In *NeurIPS*, 2026.

[20] Jaheon Kim, In-Sun Jung, Soo-Young Kim, Eunsung Lee, Jin-Koo Kang, Shigeru Sakamoto, Kentaro Yamaguchi, and Kimoon Kim. New cucurbituril homologues: syntheses, isolation, characterization, and x-ray crystal structures of cucurbit [n] uril (n= 5, 7, and 8). *Journal of the American Chemical Society*, 122(3):540–541, 2000.

[21] Lecheng Kong, Xiyuan Wang, Yixin Chen, and Muhan Zhang. Round-trip reinforcement learning: Self-consistent training for better chemical llms. *arXiv preprint arXiv:2510.01527*, 2025.

[22] Kit Ieng Kuok, Shengke Li, Ian W Wyman, and Ruibing Wang. Cucurbit [7] uril: an emerging candidate for pharmaceutical excipients. *Annals of the New York Academy of Sciences*, 1398 (1):108–119, 2017.

[23] Greg Landrum et al. RDKit: Open-source cheminformatics. `https://www.rdkit.org`, 2026. Release 2026.03.6.

[24] Nicholas Lee, Thanakul Wattanawong, Sehoon Kim, Karttikeya Mangalam, Sheng Shen, Gopala Anumanchipalli, Michael Mahoney, Kurt Keutzer, and Amir Gholami. Llm2llm: Boosting llms with novel iterative data enhancement. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 6498–6526, 2024.

[25] Hao Li, He Cao, Bin Feng, Yanjun Shao, Xiangru Tang, Zhiyuan Yan, Li Yuan, Yonghong Tian, and Yu Li. Beyond chemical qa: Evaluating llm's chemical reasoning with modular chemical operations. In *NeurIPS*, 2025.

[26] Qizheng Li, Yifei Zhang, Xiao Yang, Xu Yang, Zhuo Wang, Weiqing Liu, and Jiang Bian. Ft-dojo: Towards autonomous llm fine-tuning with language agents. In *ICML*, 2026.

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

[28] Xingyu Lu, He Cao, Zijing Liu, Shengyuan Bai, Leqing Chen, Yuan Yao, Hai-Tao Zheng, and Yu Li. Moleculeqa: A dataset to evaluate factual accuracy in molecular comprehension. In *Findings of EMNLP*, 2024.

[29] Tianyi Ma, Yijun Ma, Zehong Wang, Weixiang Sun, Ziming Li, Connor R Schmidt, Chuxu Zhang, Matthew J Webber, and Yanfang Ye. Suprabench: A benchmark for supramolecular chemistry. *arXiv preprint arXiv:2606.13477*, 2026.

[30] David L Mobley and J Peter Guthrie. Freesolv: a database of experimental and calculated hydration free energies, with input files. *Journal of computer-aided molecular design*, 28(7): 711–720, 2014.

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

[32] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744, 2022.

[33] Daniil Polykovskiy, Alexander Zhebrak, Benjamin Sanchez-Lengeling, Sergey Golovanov, Oktai Tatanov, Stanislav Belyaev, Rauf Kurbanov, Aleksey Artamonov, Vladimir Aladinskiy, Mark Veselov, et al. Molecular sets (moses): a benchmarking platform for molecular generation models. *Frontiers in pharmacology*, 11:565644, 2020.

[34] Daniel Probst, Philippe Schwaller, and Jean-Louis Reymond. Reaction classification and yield prediction using the differential reaction fingerprint drfp. *Digital discovery*, 1(2):91–97, 2022.

[35] Rushi Qiang, Yuchen Zhuang, Yinghao Li, Dingu Sagar VK, Rongzhi Zhang, Changhao Li, Ian Wong, Sherry Yang, Percy Liang, Chao Zhang, et al. Mle-dojo: Interactive environments for empowering llm agents in machine learning engineering. In *NeurIPS*, 2026.

[36] Liva Ralaivola, Sanjay J Swamidass, Hiroto Saigo, and Pierre Baldi. Graph kernels for chemical informatics. *Neural networks*, 18(8):1093–1110, 2005.

[37] Ben Rank, Hardik Bhatnagar, Ameya Prabhu, Shira Eisenberg, Karina Nguyen, Matthias Bethge, and Maksym Andriushchenko. Posttrainbench: Can llm agents automate llm post-training? *arXiv preprint arXiv:2603.08640*, 2026.

[38] David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of chemical information and modeling*, 50(5):742–754, 2010.

[39] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[40] Alok Shaurya, Amir Hassan Bagherzadeh Mostaghimi, David R Turnbull, Fraser Hof, and Jeffrey F Van Humbeck. Synthesis and machine learning techniques to enable data-driven investigation of supramolecular host–guest interactions. *Chemical Science*, 17(26):12884–12894, 2026.

[41] Karan Singhal, Tao Tu, Juraj Gottweis, Rory Sayres, Ellery Wulczyn, Mohamed Amin, Le Hou, Kevin Clark, Stephen R Pfohl, Heather Cole-Lewis, et al. Toward expert-level medical question answering with large language models. *Nature medicine*, 31(3):943–950, 2025.

[42] Miriam Sprick and Gabriele Raabe. Prediction of toluene/water partition coefficients of sampl9 compounds: comparison of the molecular dynamics force fields gaff/resp and gaff/ipolq-mod+ lj-fit. *Physical Chemistry Chemical Physics*, 26(4):3126–3138, 2024.

[43] Anthony Tabet, Thomas Gebhart, Guanglu Wu, Charlie Readman, Merrick Pierson Smela, Vijay K Rana, Cole Baker, Harry Bulstrode, Polina Anikeeva, David H Rowitch, et al. Applying support-vector machine learning algorithms toward predicting host–guest interactions with cucurbit [7] uril. *Physical Chemistry Chemical Physics*, 22(26):14976–14982, 2020.

[44] Izhar Wallach and Abraham Heifets. Most ligand-based classification benchmarks reward memorization rather than generalization. *Journal of Chemical Information and Modeling*, 58 (5):916–932, 2018.

[45] Y Wang, Y Kordi, S Mishra, A Liu, NA Smith, D Khashabi, and H Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. arxiv 2022. *arXiv preprint arXiv:2212.10560*, 2022.

[46] Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564*, 2023.

[47] Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. Moleculenet: A benchmark for molecular machine learning. *arXiv preprint arXiv:1703.00564*, 2017.

[48] Taro Yano, Yoichi Ishibashi, and Masafumi Oyamada. Lamdagent: An autonomous framework for post-training pipeline optimization via llm agents. In *EMNLP*, 2025.

[49] Botao Yu, Frazier N. Baker, Ziqi Chen, Xia Ning, and Huan Sun. LlaSMol: Advancing large language models for chemistry with a large-scale, comprehensive, high-quality instruction tuning dataset. In *COLM*, 2024.

[50] Yue Yu, Yuchen Zhuang, Jieyu Zhang, Yu Meng, Alexander J Ratner, Ranjay Krishna, Jiaming Shen, and Chao Zhang. Large language model as attributed training data generator: A tale of diversity and bias. *Advances in neural information processing systems*, 36:55734–55784, 2023.

[51] Zhengyang Zhao, Shengjie Ye, Lu Ma, Hao Liang, Hengyi Feng, and Wentao Zhang. Andes: Agent native data evolving synthesis tool for autonomous instruction alignment. *arXiv preprint arXiv:2606.01279*, 2026.

[52] Xinyu Zhu, Yuzhu Cai, Zexi Liu, Cheng Wang, Fengyang Li, Wenkai Jin, Wanxu Liu, Zehao Bing, Bingyang Zheng, Jingyi Chai, et al. Evomaster: A foundational evolving agent framework for agentic science at scale. *arXiv preprint arXiv:2604.17406*, 2026.

APPENDIX CONTENTS

A Notation **15**

B Implementation Details **16**

B.1 Datasets for Real-World Applications 16

B.2 Benchmark Datasets 16

B.3 Evaluation Metrics 17

B.4 Baseline Methods 18

B.5 Experimental Setup 18

B.6 Hyperparameters 19

C Additional Experiments **19**

C.1 Experiments for Agent Backbones 19

C.2 Dual Synthesis Engine Comparison 19

C.3 Cost Analysis 19

D Examples of Synthesis Engines **21**

E Examples of Synthesized Samples **22**

F Fingerprint Algorithms **24**

A NOTATION

Table 4 lists the symbols used in the paper, grouped by the part of the paper that introduces them.

Table 4: **Notation.** Symbols of the problem setting and the method (Section 2) and of the fingerprint algorithms (Appendix F).

Symbol	Meaning
Problem setting	
$\mathcal{T} = (T, \mathcal{E}, M_\phi)$	A chemistry task
T	Natural-language task description.
$\mathcal{E} = \{(x_j, y_j)\}_{j=1}^{k}$, k	Set of k solved examples with gold labels ($k = 18$ in our experiments).
M_ϕ, ϕ	Base model and its parameters.
M_{ϕ^*}, ϕ^*	Post-trained model returned by AutoChem and its parameters.
F	The method that maps a task $\mathcal{T}$ to $(M_{\phi^*}, \mathcal{D}_{\text{syn}}, \mathcal{D}_{\text{eval}})$.
$\mathcal{D}_{\text{syn}}$, $\mathcal{D}_{\text{eval}}$	Synthesized training set and evaluation set, with $\mathcal{D}_{\text{syn}} \cap \mathcal{D}_{\text{eval}} = \emptyset$.
$\mathcal{D}_{\text{test}}$	Benchmark test set, used only once for the terminal evaluation.
$s(\cdot\,; \mathcal{D}_{\text{eval}})$	Scorer of a model on the evaluation set.
Task understanding	
$\mathcal{C}$, c_i	Set of capabilities the task requires, and capability i's description.
m	Number of subtasks the research agent decouples the task into.
$\tau_i = (c_i, \mathcal{S}_i, \mathcal{U}_i)$	The i-th subtask.
$\mathcal{S}$, $\mathcal{S}_i$	Public chemistry databases valid for synthesis, and the subset for τ_i.
$\mathcal{U}_i$, μ	Evaluation metrics of subtask i, and one such metric with values in $[0, 1]$.
Dual-stream synthesis engines	
$\mathcal{A}$, $\mathcal{B}$	The two streams, each a set of m synthesis engines.
E	A generic stream, $E \in \{\mathcal{A}, \mathcal{B}\}$.
$\mathcal{D}^{\mathcal{A}}$, $\mathcal{D}^{\mathcal{B}}$	Samples drawn from the engines of stream $\mathcal{A}$ and $\mathcal{B}$, respectively.
$\mathcal{E}_i^{\text{s}}$, $\mathcal{E}_i^{\text{h}}$	Shown and hidden subsets of the solved examples of subtask τ_i.
$e_i = (\pi_i, g_i, f_i)$, $e_i^{\mathcal{A}}$, $e_i^{\mathcal{B}}$	Synthesis engine of subtask i, and its instance in each stream.
π_i	Sampler that draws a valid input z from the source databases $\mathcal{S}_i$.
g_i	Question generator that formalizes an input z into a question x.
f_i	Answer function that produces the reference answer y of a question x.
z, x, y	Sampled input, question, and reference answer.
prog_i	Program backed by chemistry packages that computes the answer.
ρ_i, $\mathcal{F}_i$	Prompt and tool set used to answer a non-computable question.
ϵ	Tolerance of engine verification ($\epsilon = 0$ under exact metrics).
Coverage, diversity, and difficulty	
$\mathcal{P}_i = \{z_t\}_{t=1}^{n_i}$, n_i	Pilot batch drawn from the sampler of engine i, and its size.
p, p', p_t	Question-answer pairs; p_t is the pair generated from the t-th pilot input.
cov_i, div_i	Coverage score of the shown examples and diversity score.
θ	Similarity threshold for a shown example to count as covered.
$\kappa(p, p')$	Similarity kernel between two pairs.
$\text{ECFP}(\cdot)$, $\text{DRFP}(\cdot)$	Extended-connectivity and differential reaction fingerprints of a pair.
$\text{Tan}(\cdot, \cdot)$	Tanimoto coefficient between two fingerprints, with values in $[0, 1]$.
$d(x, y)$	Difficulty score of a generated pair.
$d^{\text{tool}}(x)$	Number of tool operations executed by the answer function on x.
$P_\phi(y \mid x)$	Likelihood of the reference answer under the base model.
rank_i	Within-subtask quantile rank.
L_d	Number of difficulty strata.
Search loop	
R, r	Maximum number of rounds, and the index of a round.
$\mathcal{H}$, h_r	Set of candidate checkpoints, and the checkpoint produced at round r.
s_r, s_{best}	Evaluation score of round r and of the current best checkpoint.
σ	Acceptance margin proposed by the proposer agent.

Table 4 (continued)

Symbol	Meaning
Fingerprint algorithms	
L	Fingerprint width in bits ($L = 2048$).
r	Radius of the substructures a fingerprint retains.
H	Hash function.
u, v	ECFP and DRFP fingerprints in $\{0,1\}^L$.
$\mathrm{mol}(\cdot)$	Molecules contained in a question or an answer.
G, a, a'	Molecular graph of a pair, and atoms of the graph.
$\mathrm{inv}(a)$	Initial atom invariants of atom a.
$\mathrm{id}^{(t)}(a)$	Identifier of atom a after t iterations.
$\mathcal{N}(a)$, $b_{aa'}$	Neighbors of atom a, and the bond type between a and a'.
Λ	Sorted multiset of neighbor bond types and identifiers.
S, S_{q}, S_{a}	Substructure identifier sets of a pair, of its question, and of its answer.

B Implementation Details

B.1 Datasets for Real-World Applications

We employ the following datasets to evaluate AutoChem on chemistry scientific discovery tasks, in which every label is an experimental measurement rather than an annotation.

- **SupraBank** [5] is an open repository of peer-reviewed supramolecular interactions, from which we follow the binding affinity prediction task of SupraBench [29], where a model predicts the association constant $\log K_a$ of a host–guest pair under standard aqueous conditions. Host–guest recognition underpins drug delivery, sensing, and separation, while its affinities are measured by titration and thus remain scarce even for the most studied hosts, *e.g.*, cucurbit[7]uril carries affinities for merely 231 guests over two decades, and the records are scattered across inconsistent conditions [1].
- **FreeSolv** [30] collects 642 experimental hydration free energies ΔG_{hyd} for neutral small molecules, measured by calorimetry or by solubility experiments. The hydration free energy governs solubility and partitioning, and it is the first quantity a chemist estimates when a candidate has to survive an aqueous environment. We hold out a random fifth of the set for the terminal evaluation, and AutoChem labels its training samples with a semiempirical solvation calculation, *i.e.*, GFN2-xTB with the ALPB implicit solvent [13], which costs seconds per molecule and carries roughly 1.4 kcal/mol of error against experiment.
- **SAMPL9** [42] is a blind challenge that reports toluene/water partition coefficients $\log P$ for 16 drug molecules, and we further curate a collection of 251 molecules as the extended split, which covers the entire experimental literature for this system. Partitioning into a non-hydrogen-bonding aromatic solvent probes a desolvation that the correlations fitted to octanol/water data do not capture, and thus the widely used octanol/water estimators predict this quantity poorly.

B.2 Benchmark Datasets

We employ the following benchmarks to evaluate the performance of AutoChem, whose statistics are summarized in Table 5.

- **ChemCoTBench** [25] is a large-scale benchmark for step-wise reasoning on complex chemical problems, which contains a suite of four core tasks, *i.e.*, Molecule Editing (`MolEdit`), Molecule Optimization (`MolOpt`), molecular understanding (`MolUnd`), and Reaction Prediction (`RxnPred`). Specifically, `MolEdit` contains tasks to modify molecular structures by adding, deleting, or substituting functional groups, `MolOpt` requires models to optimize molecules for target properties, while preserving scaffolds. `MolUnd` examines molecular understanding by predicting molecular properties from SMILES, *e.g.*, functional group counting, scaffold extraction, and equivalence, *etc.* `RxnPred` requires a model to predict reaction products, recommend conditions, and select plausible reaction mechanisms.
- **MuMOInstruct** [10] is an instruction-tuning benchmark for multi-property molecule optimization, in which a model must rewrite a given molecule so that several properties (*e.g.*, `BBBP`,

Table 5: **Benchmark statistics.**

Benchmark	Task	Test	Train	Metric
ChemCoTBench	MolEdit	100	4,497	Edit correctness
	MolOpt	600	5,025	Success rate
	MolUnd	320	7,690	Accuracy
	RxnPred	475	10,282	Exact match
MuMOInstruct	IND	5,000	14,804	Success rate
	OOD	2,810	5,037	Success rate
SMolInstruct	Cap	2,538	56,498	METEOR
	Gen	2,493	56,498	Canonical SMILES
	BBBP	197	1,569	Accuracy
MoleculeQA	Src	1,343	11,061	Accuracy
	Use	599	1,916	Accuracy
Scientific Discovery	SupraBank	398	1,985	MAE, RMSE, Acc@1
	FreeSolv	129	513	MAE, RMSE, Acc@1
	SAMPL9	16	251	MAE, RMSE, Acc@1

DRD2, HIA, mutagenicity, plogP, and QED) improve simultaneously. We report its two evaluation slices, *i.e.*, in-distribution (IND) and out-of-distribution (OOD). Specifically, IND contains property combinations that also appear in the few-shot examples, while OOD contains combinations that are not present in the examples entirely. Therefore, it measures whether a method generalizes beyond the combinations it has seen.

- **SMolInstruct** [49] is a large-scale, multi-task instruction-tuning dataset for chemistry, which covers name conversion, molecule description, reaction, and property prediction tasks. We report three tasks, including Molecule Captioning (Cap), Molecule Generation (Gen), and Blood-Brain Barrier Penetration (BBBP). Specifically, Cap requires a model to generate a natural-language description of a molecule from its SMILES, Gen is its inverse, generating a SMILES from a textual description, while BBBP is a binary property prediction task that asks whether a molecule can penetrate the blood-brain barrier.
- **MoleculeQA** [28] is a question-answering benchmark that evaluates the factual accuracy of molecular comprehension, where each item is a four-way multiple-choice question about a given molecule. We adopt two tasks, *i.e.*, Source (Src) and Usage (Use), where Src asks about the origin of a molecule, *e.g.*, the organism or precursor it is derived from, while Use asks about its application, *e.g.*, its therapeutic role or how it is administered.

B.3 Evaluation Metrics

We adopt the same evaluation metrics in the original benchmarks, as listed in Table 5.

- **Exact match** compares the normalized prediction string against the reference and is used where the answer has a single written form.
- **Accuracy** scores a discrete label, *e.g.*, a count, a scaffold, a Yes/No decision, or a multiple-choice option.
- **Canonical SMILES** parses both the predicted and the reference SMILES with RDKit and compares their canonical forms, so that two spellings of the same molecule agree. An unparseable prediction is scored as incorrect, since failing to produce a valid molecule is a failure of the task.
- **Edit correctness** scores a molecule editing task by its outcome rather than its text: the requested functional-group change is applied to the input molecule and the result is compared with the prediction, so that a molecule which achieves the edit by a different route still counts.
- **Success rate** scores a molecule optimization task by its outcome: an item counts as successful only if the predicted molecule is valid and improves *every* requested property over the input molecule. Improving most, but not all, of the properties scores zero, which is what makes the multi-property setting harder than the sum of its parts.
- **METEOR** scores a generated description against the reference by aligning their words, allowing stems and WordNet synonyms to match, and combining unigram precision and recall with a penalty for fragmented alignments; the reported number is the mean over items, scaled to $[0, 100]$ like the other columns.

- **MAE** and **RMSE** score a scientific discovery task, where the answer is a single measured quantity, by the mean absolute error and the root mean squared error between the predicted and the experimental values, in the unit of the task, *i.e.*, $\log K_a$ for SUPRABANK, kcal/mol for FREESOLV, and $\log P$ for SAMPL9. RMSE penalizes large deviations more heavily than MAE, so the gap between the two indicates whether the error is spread evenly or concentrated on a few samples.
- **Acc@1** is the fraction of samples whose predicted value falls within one unit of the experimental measurement, *i.e.*, one $\log K_a$ unit, one kcal/mol, or one $\log P$ unit, scaled to $[0, 100]$. A reply from which no number can be read counts as incorrect for Acc@1 and is excluded from MAE and RMSE, so that a malformed reply neither dominates the error nor lowers it by abstaining.

B.4 BASELINE METHODS

We compare AutoChem against the following methods:

- **Base** is the released instruction-tuned model evaluated directly on the test set without any post-training. It measures how much of the final score comes from the base model alone.
- **Claude Code** is a general-purpose coding agent, and it represents what a capable general agent achieves without a chemistry-specific pipeline.
- **LLM2LLM** [24] fine-tunes a student, identifies its incorrect predictions, and generates additional examples targeting those mistakes.
- **Prismatic Synthesis** (Prismatic) [19] clusters examples using loss gradients from a proxy model and preferentially retains new examples assigned to sparse clusters. It builds its initial pool from the solved examples and their synthetic expansions, and tests the contribution of gradient-based diversity without directing generation by student errors.
- **ANDES** [51] is an agent-native data synthesis tool, in which a teacher model writes questions and answers around a self-evolving topic tree, and then critiques and rewrites them before returning the samples.
- **ChemOrch** [17] is a chemistry-specific instruction synthesis framework that grounds each generated instruction in chemistry tools by selecting relevant tools, generating code to invoke them, and recovering from execution errors.

B.5 EXPERIMENTAL SETUP

Compute Environment. All experiments run on a single node with $8\times$ NVIDIA A100 40GB GPUs, and each run of AutoChem occupies one GPU. Following existing works [26, 37, 51], every run is bounded by a 12-hour wall clock.

Agent Backbone. Unless otherwise stated, every agent role in AutoChem, *i.e.*, the orchestrator, the research agent, the two synthesizers, the verifier, the proposer, the trainer, and the evaluator, is served by `Claude-Sonnet-5` with at most 32,768 output tokens per call. The baseline agents, *i.e.*, Claude Code, ChemOrch, and ANDES, use the same backbone, and Appendix C.1 reports the results over five alternative backbones.

Task Inputs. Each task provides a natural-language description and $k = 18$ solved examples drawn from the training split of the benchmark with a fixed seed. The agents cannot access any other training or evaluation data, and the benchmark test set is used only once, during terminal evaluation of the returned model checkpoint.

Training Models. We employ four models, including `Qwen3-4B-Instruct-2507`, `Qwen3-8B`, `Llama-3.2-3B-Instruct`, and `Llama-3.1-8B-Instruct`. Baseline methods share the same base models, as described in Appendix B.4.

Evaluation. Checkpoints are evaluated with the LM Evaluation Harness [14] under greedy decoding, zero-shot, with the chat template applied and at most 4,096 generated tokens, except MUMOINSTRUCT, which follows the sampling protocol of the benchmark with a single generation per query. Note that the synthesis engines in AutoChem draw from public databases (e.g., PubChem and DeepChem releases), from which several benchmarks are constructed, so a synthesized sample may coincide with a test item. In light of this, before every training round, we employ a filter outside the agents to remove any sample whose molecule matches a test molecule under canonical SMILES after salt stripping or under the InChIKey skeleton block.

B.6 HYPERPARAMETERS

Table 6 lists the hyperparameters of the dual-stream synthesis stage (Section 2.2) and the search loop (Section 2.4), which are shared by every task, engine, and base model.

Table 6: **Hyperparameters of the dual-stream synthesis stage and the search loop.**

Hyperparameter	Symbol	Value
Dual-stream synthesis		
Pilot batch size, per subtask and per engine	n_i	256
Similarity threshold for coverage	θ	0.5
Minimum coverage score	cov_i	0.8
Minimum diversity score	div_i	0.5
Fingerprint width in bits	L	2,048
ECFP radius	r	2
DRFP radius	r	3
Number of difficulty strata (easy, medium, hard)	L_d	3
Initial mixture over difficulty strata	–	equal proportions
Search loop		
Maximum number of rounds	R	10
Consecutive rejections before termination	–	3
Wall-clock budget per run	–	12 hours
Acceptance margin	σ	proposed by the agent

C ADDITIONAL EXPERIMENTS

C.1 EXPERIMENTS FOR AGENT BACKBONES

Table 7 reports the results of AutoChem and the baseline methods under five agent backbones besides `Claude-Sonnet-5`, *i.e.*, `GPT-5.6-luna`, `MiniMax-M3`, `GLM-5.3`, `DeepSeek-v4-pro`, and `Qwen3.8-27B`, and the observations are discussed in Section 3.

C.2 DUAL SYNTHESIS ENGINE COMPARISON

Table 8 reports the scores of the final `Qwen3-4B` checkpoints on a fresh draw from the engine that trained them, on draws from the other engine, and on the benchmark test set, and the observations are discussed in Section 3.

C.3 COST ANALYSIS

Table 9 reports the agent tokens consumed by AutoChem and the two strongest baseline methods, *i.e.*, ChemOrch and Claude Code, and the observations are discussed in Section 3. All methods employ `Claude-Sonnet-5`, so their token counts are directly comparable.

Table 7: **Performance comparison over agent backbones.**

(a) `Claude-Sonnet-5`

Setting	Method	MolEdit	MolUnd	BBBP	Use
Qwen3-4B	Base	22.00	34.06	30.96	24.71
	ChemOrch	37.00	55.94	67.51	**31.39**
	ANDES	29.00	53.75	69.54	27.05
	AutoChem	**66.00**	**59.38**	**87.31**	30.05
Qwen3-8B	Base	32.00	45.62	30.96	30.05
	ChemOrch	51.00	61.56	77.16	33.72
	ANDES	26.00	61.56	35.53	29.55
	AutoChem	**68.00**	**64.38**	**90.86**	**39.57**
Llama3-3B	Base	0.00	0.31	31.47	28.71
	ChemOrch	32.00	58.13	69.04	**33.06**
	ANDES	28.00	**73.44**	34.01	26.04
	AutoChem	**61.00**	70.31	**82.74**	30.55
Llama3-8B	Base	5.00	0.31	31.47	29.05
	ChemOrch	56.00	60.31	59.90	33.56
	ANDES	29.00	**67.19**	46.19	27.88
	AutoChem	**58.00**	65.31	**65.99**	**37.40**

(b) `GPT-5.6-luna`

Setting	Method	MolEdit	MolUnd	BBBP	Use
Qwen3-4B	Base	22.00	34.06	30.96	24.71
	ChemOrch	38.89	51.47	59.90	**32.39**
	ANDES	28.00	50.31	56.85	25.54
	AutoChem	**63.00**	**58.44**	**78.17**	32.22
Qwen3-8B	Base	32.00	45.62	30.96	30.05
	ChemOrch	42.78	59.60	71.07	28.88
	ANDES	23.00	54.69	33.50	30.72
	AutoChem	**57.00**	**65.31**	**86.29**	**35.39**
Llama3-3B	Base	0.00	0.31	31.47	28.71
	ChemOrch	33.33	43.97	62.44	**34.72**
	ANDES	23.00	**57.81**	30.96	26.54
	AutoChem	**54.00**	50.62	**74.62**	31.72
Llama3-8B	Base	5.00	0.31	31.47	29.05
	ChemOrch	40.56	61.47	58.38	27.55
	ANDES	24.00	42.50	35.03	28.21
	AutoChem	**46.00**	**62.19**	**61.93**	**38.06**

(c) `MiniMax-M3`

Setting	Method	MolEdit	MolUnd	BBBP	Use
Qwen3-4B	Base	22.00	34.06	30.96	24.71
	ChemOrch	25.00	49.38	61.42	29.88
	ANDES	23.00	45.62	36.04	26.38
	AutoChem	**47.00**	**57.81**	**70.56**	**31.72**
Qwen3-8B	Base	32.00	45.62	30.96	30.05
	ChemOrch	27.00	**51.25**	48.22	29.38
	ANDES	15.00	44.06	30.96	30.22
	AutoChem	**50.00**	49.69	**57.87**	**33.22**
Llama3-3B	Base	0.00	0.31	31.47	**28.71**
	ChemOrch	22.00	61.25	65.48	27.71
	ANDES	16.00	44.06	30.96	26.71
	AutoChem	**44.00**	**66.88**	**71.57**	24.71
Llama3-8B	Base	5.00	0.31	31.47	29.05
	ChemOrch	**34.00**	**58.13**	68.53	28.55
	ANDES	20.00	19.38	16.24	29.22
	AutoChem	31.00	51.56	**78.68**	**35.06**

(d) `GLM-5.3`

Setting	Method	MolEdit	MolUnd	BBBP	Use
Qwen3-4B	Base	22.00	34.06	30.96	24.71
	ChemOrch	24.00	**46.88**	61.93	27.38
	ANDES	22.00	43.12	35.03	26.21
	AutoChem	**58.00**	45.94	**74.62**	**27.88**
Qwen3-8B	Base	32.00	45.62	30.96	30.05
	ChemOrch	26.00	53.13	37.56	29.88
	ANDES	15.00	45.00	31.98	29.22
	AutoChem	**39.00**	**61.25**	**44.16**	**38.73**
Llama3-3B	Base	0.00	0.31	31.47	**28.71**
	ChemOrch	18.00	46.88	67.01	27.38
	ANDES	15.00	42.50	30.96	26.88
	AutoChem	**41.00**	**52.50**	**78.68**	23.87
Llama3-8B	Base	5.00	0.31	31.47	29.05
	ChemOrch	36.00	**42.50**	69.54	29.05
	ANDES	23.00	20.00	17.26	28.38
	AutoChem	**45.00**	36.25	**81.22**	**31.05**

(e) `DeepSeek-v4-pro`

Setting	Method	MolEdit	MolUnd	BBBP	Use
Qwen3-4B	Base	22.00	34.06	30.96	24.71
	ChemOrch	31.00	52.50	70.05	**27.38**
	ANDES	22.00	45.00	34.52	25.88
	AutoChem	**55.00**	**60.31**	**85.79**	25.88
Qwen3-8B	Base	32.00	45.62	30.96	30.05
	ChemOrch	28.00	**50.31**	43.15	34.56
	ANDES	15.00	43.75	32.99	30.05
	AutoChem	**49.00**	47.81	**59.39**	**39.40**
Llama3-3B	Base	0.00	0.31	31.47	28.71
	ChemOrch	20.00	58.75	**72.59**	25.71
	ANDES	18.00	46.87	29.44	25.54
	AutoChem	**43.00**	**63.44**	68.02	**29.55**
Llama3-8B	Base	5.00	0.31	31.47	29.05
	ChemOrch	28.00	**60.94**	60.41	**31.72**
	ANDES	21.00	20.31	14.21	30.38
	AutoChem	**34.00**	55.00	**82.74**	30.22

(f) `Qwen3.8-27B`

Setting	Method	MolEdit	MolUnd	BBBP	Use
Qwen3-4B	Base	22.00	34.06	30.96	24.71
	ChemOrch	42.22	45.67	74.11	28.88
	ANDES	23.00	45.94	46.19	25.71
	AutoChem	**73.00**	**47.19**	**88.32**	**30.05**
Qwen3-8B	Base	32.00	45.62	30.96	30.05
	ChemOrch	47.22	53.30	49.75	33.06
	ANDES	17.00	47.81	30.46	30.88
	AutoChem	**58.00**	**60.00**	**63.45**	**41.07**
Llama3-3B	Base	0.00	0.31	31.47	28.71
	ChemOrch	23.33	51.57	70.56	**29.38**
	ANDES	21.00	50.62	33.50	25.88
	AutoChem	**48.00**	**56.88**	**76.14**	24.71
Llama3-8B	Base	5.00	0.31	31.47	29.05
	ChemOrch	38.89	**44.40**	57.36	33.22
	ANDES	21.00	30.62	25.89	28.55
	AutoChem	**44.00**	39.06	**69.04**	**35.89**

Table 8: **Dual synthesis engine comparison.** Scores of the final `Qwen3-4B` checkpoints. $\mathcal{D}^{\mathcal{A}} \rightarrow \mathcal{D}^{\mathcal{B}}$ denotes a model post-trained on samples from stream $\mathcal{A}$ and evaluated on samples from stream $\mathcal{B}$, and $\mathcal{D}^{\mathcal{A}} \rightarrow \mathcal{D}^{\mathcal{A}}$ denotes the same model evaluated on a fresh draw from stream $\mathcal{A}$ that is disjoint from its training samples. $\mathcal{D}^{\mathcal{A}} \rightarrow \mathcal{D}_{\text{test}}$ is the checkpoint of Table 2 on the benchmark test set.

Source	ChemCoTBench				MuMOInstruct		SMolInstruct			MoleculeQA	
	MolEdit	MolOpt	MolUnd	RxnPred	IND	OOD	Cap	Gen	BBBP	Src	Use
$\mathcal{D}^{\mathcal{A}} \rightarrow \mathcal{D}^{\mathcal{A}}$	95.63	78.19	87.16	30.84	86.32	79.80	33.10	5.21	92.28	83.53	97.33
$\mathcal{D}^{\mathcal{A}} \rightarrow \mathcal{D}^{\mathcal{B}}$	79.35	52.67	70.58	5.44	25.21	31.64	15.01	0.83	75.20	51.65	46.40
$\mathcal{D}^{\mathcal{B}} \rightarrow \mathcal{D}^{\mathcal{B}}$	92.47	82.40	71.56	35.34	91.15	72.87	27.35	9.59	87.30	88.92	86.40
$\mathcal{D}^{\mathcal{B}} \rightarrow \mathcal{D}^{\mathcal{A}}$	72.80	71.74	52.63	17.52	23.70	43.76	28.33	1.84	79.67	41.37	37.48
$\mathcal{D}^{\mathcal{A}} \rightarrow \mathcal{D}_{\text{test}}$	66.00	63.00	59.38	10.53	31.06	45.57	21.75	0.40	87.31	47.73	30.05

Table 9: **Agent tokens consumed for baseline methods and AutoChem in millions.** All methods run on `Claude-Sonnet-5`.

Method	ChemCoTBench				MuMOInstruct		SMolInstruct			MoleculeQA		Total
	MolEdit	MolOpt	MolUnd	RxnPred	IND	OOD	Cap	Gen	BBBP	Src	Use	
ChemOrch	14.62	17.69	20.10	15.43	78.05	78.51	14.28	16.68	23.23	14.87	14.96	308.41
Claude Code	**2.15**	5.40	6.31	**5.74**	**3.49**	7.80	8.72	**3.88**	7.94	**3.94**	**8.13**	**63.50**
AutoChem	8.21	**2.12**	**5.25**	9.73	12.61	**5.18**	**7.54**	15.62	**6.83**	5.23	10.69	89.01

D Examples of Synthesis Engines

Listing 1: `MolUnd`: the answer is computed.

```
def answer(query: str, scoring_context: dict | None = None) -> str:
   kind = classify(query) # one of six question shapes
   if kind == "equivalence": # are these the same molecule?
      m = _RE_TWO_MOLS.search(query)
      same = same_molecule(_strip_smiles(m.group(1)),
                           _strip_smiles(m.group(2)))
      return json.dumps({"output": "Yes" if same else "No"})
   if kind == "fg_count": # how many of this group?
      m = _RE_MOL_FRAG.search(query)
      n = count_fragment(_strip_smiles(m.group(1)), m.group(2))
      return json.dumps({"count": str(n if n is not None else 1)})
   if kind == "ring_count":
      m = _RE_MOL_RING.search(query)
      n = count_ring(_strip_smiles(m.group(1)), _strip_smiles(m.group(2)))
      return json.dumps({"count": str(n if n is not None else 0)})
   ... # three further shapes
   return ""
def count_fragment(smiles, fragment_name):
   mol, patt = _mol(smiles), _fg_pattern(fragment_name)
   if mol is None or patt is None:
      return None
   return len(mol.GetSubstructMatches(patt, uniquify=True))
```

Listing 2: `MolOpt`: the target is enumerated and scored, and the answer is the retained target.

```
def variants(m, task):
   out = []
   for i in eligible(m)[:12]: # carbons with a spare hydrogen
      if task == 'logp': specs = [(6, False), (9, False), (17, False)]
      elif task == 'solubility': specs = [(8, False), (7, False), (8, True)]
      elif task == 'drd': specs = [(7, True), (7, False)] # amine handle
      else: specs = [(6, False), (8, False), (7, False), (9, False)]
      for z, chain in specs: # add one atom, or a -CH2-X, at carbon i
         n = attach(m, i, z, chain); s = canonical(n)
         if s: out.append((s, n))
   return out

def score(m, task):
   return {'logp': Crippen.MolLogP, 'qed': QED.qed,
         'solubility': solubility_proxy, 'drd': drd2_proxy}[task](m)

def make_examples(count, seed, bases, blocked_targets=()):
   rng = random.Random(seed); rows = []
   for task in ('drd', 'logp', 'qed', 'solubility'):
      for source in candidates[task]:
         m = Chem.MolFromSmiles(source); old = score(m, task)
         scored = [(score(n, task), s) for s, n in variants(m, task)]
         eps = .002 if task == 'qed' else .01
         good = [x for x in scored if x[0] > old + eps and x[1] not in blocked_targets]
         if not good: continue # no strict improvement: no sample
         target = good[rng.randrange(len(good))][1]
         rows.append({'subtask': task, 'source_smiles': source, 'target_smiles': target})
   return rows

def answer(sample):
   return {'Final Target Molecule': sample['target_smiles']}
```

E EXAMPLES OF SYNTHESIZED SAMPLES

CHEMCOTBENCH.

Listing 3: `MolEdit`, add a functional group.

```
Q: You are a chemical assistent. Given the SMILES structural formula of a molecule, help me
     add a specified functional group and output the improved SMILES sequence of the molecule
     . Input: Molecule SMILES string, Functional Group Name. Output: Modified Molecule SMILES
      string.
Your response must be directly parsable JSON format:
{
    "output": "Modified Molecule SMILES"
}

Input Molecule: COc1cnc(OCC(=O)Nc2ccc(C)cc2)nc1N1CCOCC1, Functional Group to add: thiol.
A: {"output":"COc1cnc(OCC(=O)Nc2ccc(CS)cc2)nc1N1CCOCC1"}
```

Listing 4: `MolOpt`, improve drug-likeness.

```
Q: You are a chemical assistent, Optimize the Source Molecule to improve the QED property (
     Drug-likeness) while following a structured intermediate optimization process.

Source Molecule: COc1ccc(N)cc1.
Your response must be directly parsable JSON format:
{
    "Final Target Molecule": "SMILES",
}
A: {"Final Target Molecule":"COc1ccc(N)cc1C"}
```

Listing 5: `MolUnd`, count a ring system.

```
Q: You are a chemistry expert. Help me count the number of ring structure in the Input
     Molecule.
Input Molecule: CCCCCCCCc1cscc1, Ring Structure: c1cscc1.
Your response must be directly parsable JSON format:
{
    "count": "Your Answer Number"
}
A: {"count":"1"}
```

Listing 6: `RxnPred`, forward synthesis.

```
Q: Given the reactants SMILES and reagents SMILES, your task is to predict the main product
     SMILES using reaction prediction knowledge.

Input:
Reactants SMILES separated by '.'
Reagents SMILES separated by '.'

Question:
Reactants: CCCCCC(C)=O
Reagents: [BH4-].[Na+]

Return compact strict JSON only: {"Major Product":"SMILES"}.
A: {"Major Product":"CCCCCC(C)O"}
```

MUMOINSTRUCT.

Listing 7: `IND`, three properties at once.

```
Q: Modify the given molecule with minimal structural changes to achieve the requested
     property adjustments.
%%% Input : <SMILES> COc1ccccc1N1CCN(CCCNC(=O)c2ccccc2OCc2ccccc2)CC1 </SMILES>
%%% Adjust: increase BBB permeability, increase DRD2 inhibition and increase QED
Your response should only contain a valid SMILES representation of the modified molecule
     enclosed with <SMILES> </SMILES> tag.
A: <SMILES> COc1ccccc1N1CCN(CCCNC(=O)c2ccccc2-c2ccccc2)CC1 </SMILES>
```

SMOLINSTRUCT.

Listing 8: `Cap`, molecule captioning.

```
Q: How would you describe this molecule in a few words? <SMILES> O=C1CN[C@H](C(=O)O)C1 </
     SMILES> Answer in plain English.
A: The L-enantiomer of 4-oxoproline.
```

Listing 9: `Gen`, molecule generation.

```
Q: Could you please return a molecule that adheres to this description? The molecule is
     Ugonin C is a chemical entity. It is classified as ether and flavonoids. Answer with the
      SMILES enclosed in <SMILES> and </SMILES> tags.
A: <SMILES>COc1c(-c2ccc(O)cc2)oc2c3c(cc(O)c2c1=O)OC(C)C3(C)C</SMILES>
```

Listing 10: `BBBP`, blood–brain barrier penetration.

```
Q: Assess BBB penetration for <SMILES> O=C1NCCNC1c1cccs1 </SMILES>. Reply with exactly one
     token sequence: <BOOLEAN>Yes</BOOLEAN> or <BOOLEAN>No</BOOLEAN>.
A: <BOOLEAN>Yes</BOOLEAN>
```

MOLECULEQA.

Listing 11: `Src`, origin of a molecule.

```
Q: Please complete the following question answering task: You are given a SMILES of a
     molecule and a question about it with several options, please analysis the structure of
     the molecule and choose the right answer for the question from given options.
Molecule SMILES: CC1OC(=O)Cc2cc(O)cc(O)c21
Question about this molecule: How does the cached NP Atlas source record classify this
     compound's origin?
Option A: The cached NP Atlas source record classifies its origin as plant.
Option B: The cached NP Atlas source record classifies its origin as bacterium.
Option C: The cached NP Atlas source record classifies its origin as animal.
Option D: The cached NP Atlas source record classifies its origin as fungus.

Your answer should be one of A/B/C/D. Reply in the form "Answer: X" and nothing else.
A: Answer: D
```

Listing 12: `Use`, usage of a molecule.

```
Q: Please complete the following question answering task: You are given a SMILES of a
     molecule and a question about it with several options, please analysis the structure of
     the molecule and choose the right answer for the question from given options.
Molecule SMILES: O=C(O)CNCP(=O)(O)O
Question about this molecule: Which documented use, mechanism, or status statement is correct
     ?
Option A: ChEMBL records a clinical indication for hypertension.
Option B: It is a selective herbicide.
Option C: ChEMBL flags it as withdrawn.
Option D: It is a systemic herbicide used to control weeds.

Your answer should be one of A/B/C/D. Reply in the form "Answer: X" and nothing else.
A: Answer: D
```

Scientific discovery.

Listing 13: SUPRABANK, binding affinity of a cucurbit[7]uril complex.

```
Q: Predict the association constant log Ka of the host-guest complex below, under standard
     aqueous conditions.
%%% Host SMILES: O=C1N2CN3C(=O)N4CN5C(=O)N6CN7C(=O)N8CN9C(=O)N%10CN%11C(=O)N%12CN%13C(=O)N%14
     CN1C1C2N2CN%15C(=O)N(CN%16C(=O)N(CN%17C(=O)N(CN%18C(=O)N(CN%19C(=O)N(CN%20C(=O)N(CN1C2=O
     )C%14C%13%20)C%12C%11%19)C%10C9%18)C8C7%17)C6C5%16)C4C3%15
%%% Guest SMILES: C[NH+]1CCN(CCC(=O)Nc2cc(Br)ccc2Sc2cccc(NC(=O)CCCC(=O)NCC3CCCCC3)c2)CC1
Answer with the number only, enclosed in <VALUE> and </VALUE> tags, and nothing else.
A: <VALUE>8.0521</VALUE>
```

F FINGERPRINT ALGORITHMS

Both fingerprints used in Section 2.2 map a query-answer pair to a binary vector in $\{0,1\}^L$ with $L = 2048$, and differ in the substructures they retain. Algorithm 1 encodes the molecules a pair involves, Algorithm 2 encodes the transformation it performs. Two fingerprints u and v are compared by the Tanimoto coefficient $\mathrm{Tan}(u,v) = |u \wedge v|/|u \vee v|$ with the convention $\mathrm{Tan}(\mathbf{0},\mathbf{0}) = 1$.

Algorithm 1 ECFP(x, y), the extended-connectivity fingerprint of a pair [38]

Require: query-answer pair (x, y), radius r, width L, hash H
Ensure: fingerprint $u \in \{0,1\}^L$

$G \leftarrow$ the molecular graph of $\mathrm{mol}(x) \cup \mathrm{mol}(y)$, treated as one graph with several components
for each atom a of G **do**
  $\mathrm{id}^{(0)}(a) \leftarrow H(\mathrm{inv}(a))$ ▷ $\mathrm{inv}(a)$ collects atomic number, degree, valence, charge, attached hydrogens, ring membership
end for
$S \leftarrow \{\mathrm{id}^{(0)}(a) : a \in G\}$
for $t = 0$ to $r - 1$ **do**
  for each atom a of G **do**
    $\Lambda \leftarrow$ the multiset $\{(b_{aa'}, \mathrm{id}^{(t)}(a')) : a' \in \mathcal{N}(a)\}$ sorted in canonical order
    $\mathrm{id}^{(t+1)}(a) \leftarrow H(\mathrm{id}^{(t)}(a), \Lambda)$ ▷ sorting makes the identifier independent of atom numbering
  end for
  $S \leftarrow S \cup \{\mathrm{id}^{(t+1)}(a) : a \in G\}$
end for
$u \leftarrow \mathbf{0} \in \{0,1\}^L$
for each $s \in S$ **do**
  $u[s \bmod L] \leftarrow 1$ ▷ fold the identifier set into L bits
end for
return u

Algorithm 2 DRFP(x, y), the differential reaction fingerprint of a pair [34]

Require: query-answer pair (x, y), radius r, width L, hash H
Ensure: fingerprint $v \in \{0,1\}^L$

$S_\mathrm{q} \leftarrow$ the substructure set of $\mathrm{mol}(x)$ at radius r ▷ lines 2–11 of Algorithm 1
$S_\mathrm{a} \leftarrow$ the substructure set of $\mathrm{mol}(y)$ at radius r
if $\mathrm{mol}(y) = \emptyset$ **then**
  $S_\mathrm{a} \leftarrow \emptyset$ ▷ an answer carrying no molecule leaves the query substructures
end if
$S \leftarrow (S_\mathrm{q} \cup S_\mathrm{a}) \setminus (S_\mathrm{q} \cap S_\mathrm{a})$ ▷ substructures shared by query and answer cancel
$v \leftarrow \mathbf{0} \in \{0,1\}^L$
for each $s \in S$ **do**
  $v[H(s) \bmod L] \leftarrow 1$
end for
return v