ChemAgent: Enhancing LLMs for Chemistry and Materials Science through Tree-Search Based Tool Learning

Anonymous ACL submission

Abstract

Large language models (LLMs) have recently demonstrated promising capabilities in chemistry tasks while still facing challenges due to outdated pretraining knowledge and the difficulty of incorporating specialized chemical expertise. To address these issues, we propose an LLM-based agent that synergistically integrates 137 external chemical tools created ranging from basic information retrieval to complex reaction predictions, and a dataset curation pipeline to generate the dataset ChemToolBench that facilitates both effective tool selection and precise parameter filling during fine-tuning and evaluation. We introduce a Hierarchical Evolutionary Monte Carlo Tree Search (HE-MCTS) framework, enabling independent optimization of tool planning and execution. By leveraging self-generated data, our approach supports step-level fine-tuning (FT) of the policy model and training task-adaptive PRM and ORM that surpass GPT-4o. Experimental evaluations demonstrate that our approach significantly improves performance in Chemistry QA and discovery tasks, offering a robust solution to integrate specialized tools with LLMs for advanced chemical applications. All datasets and code will be available at https://github.com.

1 Introduction

In recent years, Large Language Models (LLMs) have shown considerable promise in tackling chemistry-related tasks (Xue et al., 2020; Zhang et al., 2024b; Mirza et al., 2024), such as molecule generation and reaction prediction. However, the expert chemistry knowledge embedded in pre-trained models may become outdated and face challenges when applied to real-world scenarios. One potential solution is the development of LLM-based agents that integrate language models with external, specialized tools to utilize the latest chemistry knowledge.

Developing LLM-based agents for chemistry has shown significant potential in recent years but there still exists several challenges. First, existing chemical toolkits rely on specialized cheminformatics software, which is difficult to develop and deploy. As a result, the number of available tools is limited, which restricts their use in a wider range of chemical tasks. Additionally, current datasets suffer from poor quality and lack proper evaluation settings. Even when tools are available, agents struggle with both selecting the right tools and generating accurate parameters due to the specialized knowledge required in chemistry. These limitations hinder the effectiveness of chemistry-focused LLM agents.

To address these challenges, we collect a large and diverse set of chemical tools to provide more available tools for LLMs. The new toolkit supports a variety of tasks, from simple information queries to complex reaction predictions, which broadens the potential applications of intelligent agents in chemistry. The code implementation of tools is also in a clear format that is easy to follow, which means more tools can be added to toolpool easily.

A high-quality, diverse meta-dataset ChemToolBench with above tools is then created for fine-tuning the model and serving as the benchmark. To construct the comprehensive dataset, we have designed a dataset curation pipeline for self-instruct chemistry Tool Learning data generation. The dataset includes difficult examples for both tool selection and parameter filling-in, which helps train the model to perform better to call chemistry domain tools.

For better tool calling, we introduce an efficient Hierarchical Evolutionary Monte Carlo Tree Search (HE-MCTS) framework. The high-level policy model iteratively explores and refines the tool selection sequence, while the fine-tuned low-level execution model iteratively reflects on execution feedbacks to enhance accuracy. Additionally, we leverage self-generated HE-MCTS data

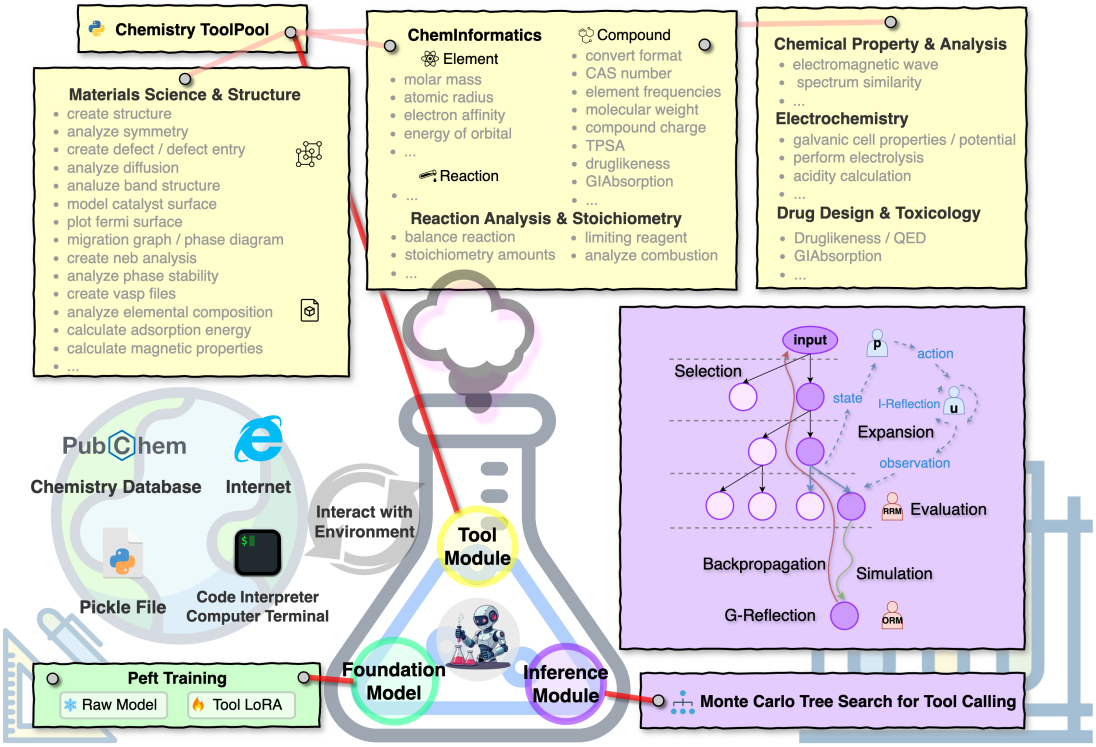

Figure 1: Overview of our ChemAgent.

alongside the meta-dataset to perform step-level fine-tuning on the policy model, and train task-adaptive PRM and ORM as alternatives to GPT-4o. Crucially, this training process requires no manual annotation or curation. The self-evolving agent, guided by HE-MCTS, autonomously optimizes its performance, demonstrating superior reasoning and execution capabilities.

Our contributions are listed as follows:

(1) We introduce the largest tool pool in the Chemistry and Materials domain, consisting of 137 tools. An agent augmented with this pool demonstrates superior performance in Chemistry-related QA and discovery tasks.

(2) We design a dataset curation pipeline tailored for domain-specific tool learning, enabling efficient data generation for fine-tuning. This pipeline supports the construction of the new dataset **ChemToolBench** for detailed benchmarking.

(3) We propose **HE-MCTS**, the Hierarchical Evolutionary Monte Carlo Tree Search framework, that decouples tool planning and execution into separate models. Our framework enables autonomous optimization without manual annotation by leveraging self-generated HE-MCTS data to adopt enhanced step-level FT for the policy model and train the PRM and ORM that surpass GPT-4o in domain-specific task.

2 ChemAgent

Inspired by the success of LLM agents in general scenarios, we attempt to construct an agent for chemistry from scratch. The foundation LLM of our agent could retrieve and call external tools, and do deep reasoning on complex domain questions.

2.1 Tools Integration

This section introduces how to construct executable chemistry toolpools as shown in Figure 6. For convenience in agent deployment and evaluation, we hope the tool mainly executes in the local environment and requires slight free online services. The procedure can be divided into 3 steps as follows.

2.1.1 Collect Tools from the Internet

We conduct a survey on former works about chemistry agents / tools(Bran et al., 2023; McNaughton et al., 2024; Ong et al., 2013) and also investigate relevant repositories in Github[1]. Finally we collect tools from 5 sources listed in Table 1: ChemCrow, CACTUS, chemlib, pymatgen, and Chemistry Tools.

[1] https://github.com

Source	Amount
ChemCrow [2]	8
CACTUS [3]	10
chemlib [4]	24
pymatgen [5]	82
Chemistry Tools [6]	13
In total:	137

Table 1: Chemistry Domain Tools Source: The number of tools is counted after organization and rewriting in Sections 2.1.2 and 2.1.3.

2.1.2 Organize Tools in Uniform Format

To make the tool learning module of the agent extendable, we design a uniform file format for loading python tool packages. We count all the functions or methods in each package that can be used as tools. Then we list them in a new JSON file called "tools.json" in each package like in Figure 6. The code path for the implementation of the tool is also given in that file. With the uniform format of each packge, the agent can easily know which tools it has and where to call them. In the future, more and more chemistry tool packages can be added to our agent without refining the agent framework for compatibility issues as soon as they use the same package organization format as we do.

2.1.3 Write Documentation & Refine Code

To make the chemistry toolpool reliable, we also write tool documentation and refine code in the final step of tool integration like shown in Figure 6.

We write documentation for each chemistry tool so that the agent can better understand the purpose of tools and how to use them. Besides, chemistry usually contains a variety of compounds, reactions, and other specialized knowledge, with which large models may not be familiar. So we summarize the input parameters of all the tools with uniform naming.

In Addition, we refine the code implementation of tools to make them easy to use for the agent. Many tools rely on instances of classes defined in their original Python packages as inputs so it is difficult for the agent to only call the specific tool without declaring other classes. In order to decouple the tools from their original packages, we adopt two approaches. **(1)** For inputs that can be represented with common data types in Python, we convert the original parameters into their corresponding types. **(2)** For those can not be easily represented, we read and write them using the pickle file format.

2.2 Dataset Construction

A high-quality dataset is the prerequisite for agent fine-tuning and evaluation. In this section we talk about how to construct the chemistry domain Tool Learning dataset ChemToolBench, trying to design corresponding construction methods by incorporating the characteristics of the chemistry discipline.

2.2.1 Preparation

In order to better construct data, we generate cases for each kind of parameters. All parameter names have been standardized in Section 2.1.3 so that they can be easily categorized.

In our preliminary attempts, we find that LLMs are not good at making up diverse input parameters. Since the large amount of data constructed by the requirements, the cases returned by the large model over multiple inputs inevitably fall into homogenization. Besides some parameters involve the user personal privacy, and due to RLHF, the LLMs will simply refuse to return the results, even if they are ordered to generate some virtual examples.

To improve the situation, we find the way to provide some examples of input parameters in prompt like in Figure 2. We use 3 approaches to generate examples for these parameters. **(1)** For those chemistry-related concepts, we get examples from the online chemistry database like PubChem[7]. **(2)** For general parameters, we let LLMs to generate as many examples as possible. **(3)** For those parameters involving personal privacy like api-key or password, we write code to construct examples.

2.2.2 Single-Tool-Calling Data

For cases which only need to call single tool, it is relatively easy to generate. We provide the LLM with the tool and examples of input parameters then the LLM makes up the tool calling as in Figure 2. For tools in packages ChemCrow, CACTUS, chemlib and Chemistry Tools, we try to execute these tool callings to examine the correctness. For

[2] ChemCrow: https://github.com/ur-whitelab/chemcrow-public https://github.com/ur-whitelab/chemcrow-runs

[3] CACTUS: https://github.com/pnnl/cactus

[4] chemlib: https://github.com/harirakul/chemlib

[5] pymatgen: https://github.com/materialsproject/pymatgen

[6] Chemistry Tools: https://github.com/domdfcoding/chemistry_tools

[7] https://pubchem.ncbi.nlm.nih.gov

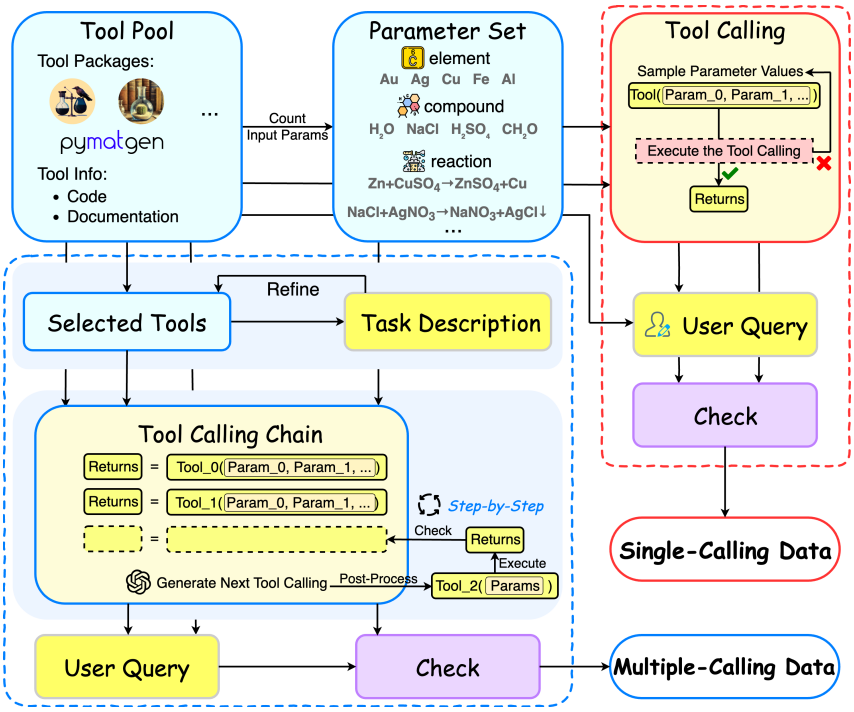

Figure 2: Domain-specific Tool Learning dataset construction pipeline.

tools in the package pymatgen, we do an exhaustive manual examination after generation. The same is true for tool calling chains in the next section 2.2.3. Then we fill in the tool calling and tool information in the prompt to let the LLM generate the user query. After final manual check, the generation of single-tool-calling data is completed.

2.2.3 Multiple-Tool-Calling Data

For cases which need to call multiple tools, we break down the goal into three steps to construct the data. The quality of the data obtained by letting LLM generate it directly is poor. The format of the output is often wrong, not to mention the logic of the tool calling chain. Splitting and subdividing that data generation task as much as possible facilitates better LLMs.

STEP 1: Candidate Tool Selection

The first step is to select several tools from the whole tool pool. The tool documentation is put into prompt in a disorganized order, and the LLM picks the tools that are relevant from the prompt and generates a rough task description.

STEP 2: Tool Calling Chain Generation

Given candidate tool, input parameter examples and rough task description generated in the last step, the model is asked to generate the tool calling chain step-by-step like in Figure 2.

STEP 3: User Query Generation

The third step is to generate the user query according to the tool calling chain and tool documentation. Finally we do a manual check to examine the correctness and logical soundness.

2.2.4 Dataset Analysis

To the best of our knowledge, we construct the largest and most comprehensive Chemistry Tool Learning dataset. Our dataset ChemToolBench contains two main splits:

Comprehensive Chemistry split: It has 10441 single-calling data (8353/1044/1044 for train/dev/test) and 2003 multiple-calling data (1623/200/200 for train/dev/test).

Materials Science split: It has 15742 single-calling data (14102/820/820 for train/dev/test) and 1623 multiple-calling data (1187/436 for train/test).

2.3 The HE-MCTS Framework

Our approach, HE-MCTS, is outlined in Figure 3 and developed using four main components.

• Policy Model, which treats tools as aids and integrates tool invocation into a coherent decision process, and Execution Model, which generates specific parameters for each tool invocation, jointly generate step-by-step solutions for each task.

• Hierarchical MCTS, which performs efficiently under the guidance of PRM and ORM.

• Process Reward Model (PRM), which evaluates the quality of any reasoning step, and Outcome Reward Model (ORM), which assesses the quality of the final answer, jointly guide HE-MCTS.

• LLM Self-Training, which leverages HE-MCTS to collect decision trajectories, trains Policy Model on enhanced positive samples, and trains both PRM and ORM on all generated trajectories.

2.3.1 Policy Model and Execution Model

Existing tool agents (Chen et al., 2024c; Schick et al., 2024) typically use a single model for both tool planning and execution, though these tasks are inherently different. Tool planning, guided by Tool-Augmented Learning (Parisi et al., 2022), requires high-level tasks and tool understanding, while tool execution, guided by Tool-Oriented Learning (Qin et al., 2024), demands precise operational knowledge. To address this, we decouple tool selection and execution into two components: Policy Model p and Execution Model u.

At step i, Policy Model generates k actions $a_i^j \sim p(a_i|s_{i-1}^p)$, for $j = 1, \ldots, k$. The state s_{i-1}^p denotes a partial trajectory $s_{i-1}^p = [x^p, a_1, o_1, \ldots, a_{i-1}, o_{i-1}]$. The input x^p comprises tool selection task prompt, task examples, query q. The action a_i comprises thought and tool invocation at step i. The valid action space is defined as $A = \{a_i \mid a_i \in T \cup A_n\}$, where $T = \{t_1, t_2, \ldots, t_m\}$ denotes the set of available tools, and A_n denotes an aggregated response derived from prefix trajectory.

Given an action a_i^j, the execution result o_i^j is obtained via $o_i^j = u(a_i^j, s_{i-1}^u)$, where the state s_{i-1}^u is defined as $s_{i-1}^u = [x^u, a_1, o_1, \ldots, a_{i-1}, o_{i-1}]$, the input x^u comprises tool execution task prompt, task examples, query q. Execution Model is independently fine-tuned on the dataset D^u, which derived from meta-dataset. Only the log probability of parameter_token is computed.

2.3.2 Search-Based Hierarchical Reasoning

In our hierarchical evolutionary framework, we integrate Monte Carlo Tree Search (MCTS) into Policy Model, where each node denotes s_{i-1}^p.

Uniqueness Enforcement: Unlike Alphazero(Wan et al., 2024), which promotes diversity through clustering, we enforce uniqueness among sibling nodes by directly filtering out identical execution results, as each result is uniquely determined by the tool and its parameters.

Explicit Promotion of Diversity: Instead of relying on temperature adjustments (Zhang et al., 2024a; Song et al., 2024; Chen et al., 2024b), we enhance exploration by tracking historical sibling nodes and incorporating diversity prompts.

Prioritization of Unexplored Branches: Following CPO(Zhang et al., 2024c), we prioritize non-terminal nodes to encourage further exploration of unfinished branches during selection.

Adaptive Pruning for Efficient Exploration: Building on AlphaLLM(Tian et al., 2024), we introduce an more adaptive pruning mechanism, which dynamically evaluates nodes using score $I(s_i^p)$ and incorporates Hierarchical Pruning, Soft Pruning, and Fast Recovery to balance search quality and stability. Details are provided in the appendix.

Additionally, we integrate fast-rollout and Global Reflection (Policy-Level), which refines Policy Model by incorporating feedback across multiple search iterations.

Execution Model is directly invoked by Policy Model during the expansion and simulation of the H-MCTS. Upon execution failure, Execution Model refines through **(Tool-Level) Immediate Reflection**, incorporating real-time execution error feedback. Once the iterative self-corrective reflection process concludes, the final execution results are returned to Policy Model, which then proceeds with the HE-MCTS evaluation.

2.3.3 Enhanced Self-Step-FT for Policy Model

Based on meta-dataset, we construct a step-level dataset for tool selection, denoted as $D^p = \{(s_{i-1}^p, a_i)\}$. Additionally, we construct an enhanced dataset $\tilde{D}^p = \{(s_{i-1}^p, a_i^j)\}$ by two strategies.

Multi-Path Reasoning and Noise Filtering Strategy: For multi-step tool invocation tasks, LLMs can exhibit multiple valid reasoning paths. It is natural to apply a reward-based mechanism that incorporates estimated values to select paths (Chen et al., 2024b; Zhang et al., 2024a; Chen et al., 2024a) to extract multiple reasoning paths from HE-MCTS trees for fine-tuned. However, such mechanisms do not eliminate noisy actions, potentially leading to errors in credit assignment. To address this, we filter reasoning paths with meta-dataset, enforcing consistency between each node and the standard invocation chain, ensuring noise-free training labels. Analysis of search trees reveals that multiplicity stems from the parallel execution of certain tools, with dependencies and interchangeability naturally forming a directed acyclic

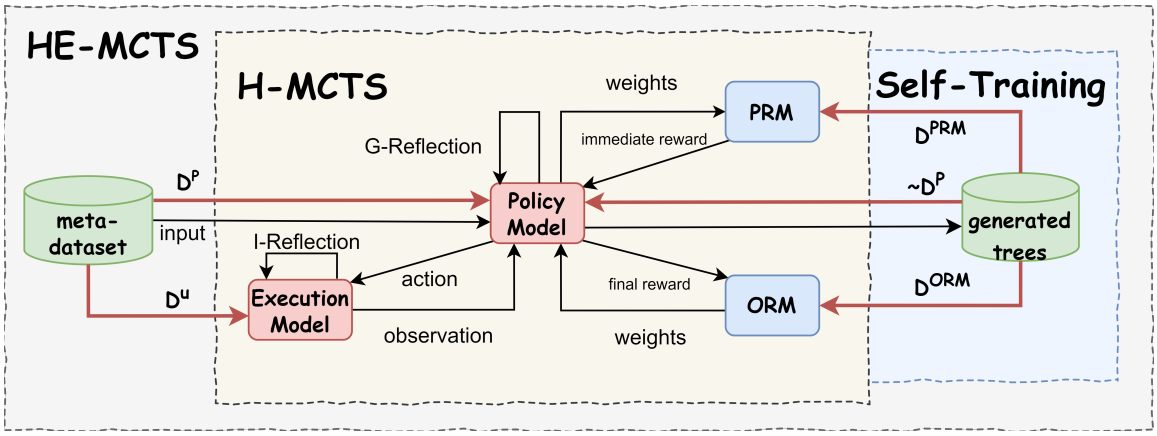

Figure 3: HE-MCTS pipeline.The left part presents the process of Search-Based Hierarchical inferring process. The right part denotes the self-training.

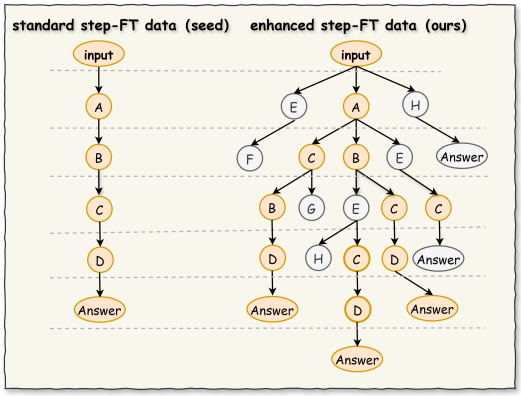

Figure 4: contrast of D^p and $\tilde{D}^p$

graph(DAG). Leveraging this structure, an alternative approach is reordering interchangeable tools in meta-dataset, while using GPT to ensure coherent reasoning within each invocation chain.

Robustness Reasoning and Noise Retention Strategy: In real-world scenarios, the Policy Model iteratively generates and corrects errors. Discarding paths with incorrect steps or final answers (Zhang et al., 2024c; Song et al., 2024; Tian et al., 2024) wastes valuable trajectories and weakens its robustness to real-world error patterns. To address this, we extract nodes from the HE-MCTS tree that follow the correct tool selection strategy, even if their reasoning paths are incomplete or contain errors. Specifically, s^p_{i-1} may contain incorrect tool invocations, erroneous execution results or perturbed reasoning, while a^j_i remain correct. Guided by this option, a complementary and more efficient approach perturbs D^p via rule-based modifications while using GPT to generate corresponding thoughts, observations, or answers.

The comparison between D^p and $\tilde{D}^p$ is presented in Figure 4, where each highlighted node can be used to construct a step-FT training sample. The loss function for fine-tuning policy_model is: $\tilde{\mathcal{L}}p = \mathbb{E}_{(s^p_{i-1},a^j_i)\sim\tilde{\mathcal{D}}^p\cup D^p}\left[\log p(a^j_i|s^p_{i-1})\right]$.

2.3.4 Self-Training for PRM and ORM

We train two types of self-improving critic models to guide the search process. Both PRM and ORM are initialized using Policy Model, and their weights remain fixed throughout the HE-MCTS iterations.

PRM The dataset for PRM is constructed as $D^{PRM} = \{(s^p_i, v_i)\}$, where s^p_i is sampled from nodes in HD-MCTS trees or the augmented synthetic data. v_i is determined on the correctness of a_i rather than the calibrated value of node s^p_i (Chen et al., 2024a; Zhang et al., 2024a; Chen et al., 2024b). Specifically, if a_i aligns with the standard tool invocation chain, v_i is 1; otherwise, v_i is 0. The loss function is: $\mathcal{L}_{PRM} = -\mathbb{E}_{(s^p_i,v_i)\sim D^{PRM}}\left(V(s^p_i) - v_i\right)^2$.

ORM The dataset for ORM is formulated as $D^{ORM} = \{([q, a_L], r_L)\}$, where q and a_L originate from the terminal nodes $s^p_L = [q, a_1, o_1, \ldots, a_{L-1}, o_{L-1}, a_L](a_L \in An)$, sampled from nodes in HD-MCTS trees or the augmented synthetic data. r_L is a weighted average of two scores: (1) r^1_L, obtained by prompting GPT to assess a_L, (2) r^2_L, derived from rule-based correctness evaluation of the sequence $[q, a_1, o_1, \ldots, a_{L-1}, o_{L-1}]$ using meta-dataset. The loss function is: $\mathcal{L}_{ORM} = -\mathbb{E}_{([q,a_L],r_L)\sim D^{ORM}}\left(R_L - r_L\right)^2$.

Model	Format	Tool			Param			Return			Pass Rate
		P	R	F1	P	R	F1	P	R	F1	
GPT-4o-mini	99.83	83.22	78.86	80.98	76.04	72.19	74.06	77.05	73.13	75.04	58.00
Claude-3.5-S	97.42	83.64	85.37	84.50	76.03	77.53	76.77	74.96	78.54	76.71	57.00
ChemLLM*	98.10	76.84	88.08	82.07	68.22	80.20	73.72	68.84	80.45	74.19	53.00
Qwen-2.5-7B-I	51.25	64.15	41.81	50.63	56.48	37.36	44.97	29.25	37.20	32.75	22.50
Llama-3.1-8B-I	75.99	59.95	38.31	46.75	52.98	33.71	41.20	40.83	34.34	37.31	15.00
Llama-3.1-8B-I*	99.20	93.10	92.21	92.65	83.85	83.15	83.50	85.03	84.90	84.96	55.00
GPT-4o-mini-M	/	85.06	83.57	84.31	88.47	/	/	75.97	74.64	75.30	62.30
Claude-3.5-S-M	/	89.80	86.27	88.00	86.36	/	/	77.55	74.51	76.00	57.06
Qwen-2.5-7B-M3	/	91.14	90.45	90.80	89.32	/	/	81.41	80.79	81.10	67.32
Llama-3.1-8B-M0	/	75.45	78.45	76.92	85.25	/	/	65.09	67.68	66.36	29.19
Llama-3.1-8B-M1	/	87.74	87.32	87.53	85.92	/	/	75.81	75.44	75.62	69.50
Llama-3.1-8B-M2	/	93.18	88.39	90.79	95.12	/	/	88.64	84.07	86.36	72.30
Llama-3.1-8B-M3	/	93.22	91.73	92.47	92.36	/	/	86.09	84.72	85.41	72.20

Table 2: Main Results on the Multiple-Tool-Calling Comprehensive Chemistry Benchmark. * represents the model fine-tuned with ChemToolBench Comprehensive Chemistry split.

3 Experiments

3.1 Experimental Setup

To evaluate the reasoning capabilities of our tool agent in the field of chemistry, we conduct experiments on the ChemToolBench. Given a user query, the tool retriever would first search relevant tools from the whole tool pool. Then the LLM judges whether to call these candidate tools. With the tool calling executed, the LLM takes all return values into consideration and generates the final answer. For multi-tool calling tasks, we evaluate the performance of the agent under both the Chain-of-Thought (CoT) paradigm and the HE-MCTS paradigm(-M). Since the -M agent essentially operates as a multi-agents system, we provide detailed training configurations of each model in Appendix Table 5. For LLMs, we evaluate both commercial models, such as GPT-4o-mini and Claude-3.5-Sonnet, as well as open-source models, including Qwen-2.5, ChemLLM[8] and the Llama series. For tool retriever, we take dense retrievers like all-MiniLM-L6-v2[9] and NV-Embed v2[10].

3.2 Evaluation Metric

We conduct a comprehensive evaluation of the agent's process reasoning accuracy and result accuracy. Process reasoning accuracy is assessed in terms of tool selection and tool execution (parameter generation). To provide a fine-grained analysis of the agent's reasoning capability, we compute **Precision, Recall**, and **F1-score** of tool selection and parameter filling-in. Result accuracy is measured using the **Pass Rate**, which, in the context of question-answering tasks, denotes the proportion of final answers generated by the agent that GPT-4o deems consistent with the reference answers.

3.3 Results & Discussion

3.3.1 Main Results

Table 2 presents the main experimental results. For the HE-MCTS, GPT-4o-mini-M and Claude-3.5-S-M exhibit superior than GPT-4o-mini and Claude-3.5-S, primarily due to enhanced tool selection capabilities. Our empirical evaluations reveal key advantages of the proposed decoupled hierarchical framework: (1) **Enhanced Tool Execution Capability**: Compared to end-to-end inference, independently optimizing the Execution Model significantly improves parameter generation accuracy. We attribute this improvement to the substantial reduction in action space enabled by the decoupled framework, allowing the model to focus on specific tasks without unnecessary reasoning over an excessively large search space. (2) **Positive Impact of Tool Selection on Execution**: As the performance of the Policy Model improves, we observe a minor yet consistent enhancement in the Execution Model. This suggests that more precise tool selection provides a more reliable context for parameter generation, ultimately leading to better execution.

3.3.2 Ablation Analysis

For the Policy Model, tool selection capabilities of **-M0, -M1**, and **-M2** models exhibit a consistent upward trend, with the **-M2** models surpassing

[8] https://huggingface.co/AI4Chem/ChemLLM-20B-Chat-DPO

[9] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

[10] https://huggingface.co/nvidia/NV-Embed-v2

GPT-4o-mini-M and Claude-3.5-S-M. This result validates the effectiveness of the method we proposed in Section 2.3.3.

For **PRM** and **ORM**, **-M3** models outperform **-M2** models, indicating that the PRM and ORM models we trained surpass GPT-4o and Policy Model. This advantage is primarily attributed to the improvement of tool execution, as the more specialized critic models reduce redundant sampling in erroneous exploration regions.

3.3.3 Generalization Verification

Model	Format	Tool ACC	Param P	Param R	Param F1
GPT-4o-mini	99.75	88.41	93.80	90.75	92.25
Qwen-2.5-7B-I	54.53	43.17	85.54	44.00	58.11
Llama-3.1-8B-I	98.91	75.37	88.89	81.94	85.27
Llama-3.1-8B-I*	97.93	79.51	91.32	87.50	89.37

Table 3: Single-Calling Results on the Materials Science Benchmark. * represents the model fine-tuned with ChemToolBench Comprehensive Chemistry split.

Model	Format	Tool P	Tool R	Tool F1	Param P	Param R	Param F1
GPT-4o-mini	99.90	61.37	41.31	49.38	55.91	40.21	46.78
Qwen-2.5-7B-I	87.37	47.91	33.02	39.10	40.31	33.10	36.35
Llama-3.1-8B-I	60.52	64.35	19.18	29.56	60.34	19.51	29.49
Llama-3.1-8B-I*	94.25	76.91	73.26	75.04	71.60	65.02	68.15

Table 4: Multiple-Calling Results on the Materials Science Benchmark. * is the same as in Table 3.

As shown in Table 3 and 4, we also evaluate LLMs on the Materials Science split. The LLM trained with Comprehensive Chemistry split is also compatible with the other split. It may suggest that LLMs can learn general chemistry tool knowledge from our dataset ChemToolBench.

4 Related Works

The LLM agent with equipped tools has become a hit because it fully extends the application scenarios of the LLMs like science discovery and embodied intelligence. Tool Learning is one of the important components in an agent.

Several studies focus on constructing tools and datasets for tool learning. ToolLLM (Qin et al., 2023) collects APIs from RapidAPI Hub [11] and employs bottom-up instruction generation, releasing dataset ToolBench. API-Bank (Li et al., 2023) sets various types of evaluation settings and explores the self-instruct method to construct the dataset. Seal-Tools (Wu et al., 2024a) tries to generate tools and datasets with LLM from scratch to test the scaling law of tool learning. ToolACE (Liu et al., 2024) introduces a self-evolving API synthesis method and a multi-agent interaction-driven data generation approach, producing 26,507 APIs. ToolPreference (Chen et al., 2024c) trains models using DPO enhances tool usage proficiency.

In general scenarios, many Tool Learning works have gained success in recent years. Toolformer (Schick et al., 2024) demonstrates that LLMs can autonomously learn to use external tools. HuggingGPT (Shen et al., 2024) takes domain-specific language models from Huggingface Hub as tools to solve professional problems. ToolkenGPT (Hao et al., 2024) encodes tools as special tokens in the LLM to decide whether to call a tool during generation. ToolPlanner (Wu et al., 2024b) simulates real-world user behaviors through multi-granularity instructions and optimizes via path planning.

In scientific scenarios, related explorations are just beginning. SciAgent (Ma et al., 2024) proposes the scientific reasoning method with domain tools and evaluates it on the new benchmark SciToolBench. Pymatgen (Ong et al., 2013) builds robust and fast python package for material analysis with many extensions. ChemCrow (Bran et al., 2023) integrates 18 expert-designed chemistry tools in the LLM engine to solve tasks like drug analysis and materials design. It performs better than GPT4 across a range of chemistry tasks while its tools and evaluation questions are limited in amount. CACTUS(McNaughton et al., 2024) integrates 10 cheminformatics tools to give precise answer in chemistry and molecular discovery questions.

5 Conclusion

In this work, we have presented a novel LLM-based agent specifically tailored for chemical applications by integrating a comprehensive tool pool, an innovative dataset curation pipeline, and an advanced reasoning framework. Our approach addresses two major challenges in applying large language models to the chemistry domain: incorporating specialized chemical knowledge and calling multiple tools to solve complex chemistry tasks. Furthermore, the introduction of HE-MCTS framework, guided by trained critic models and integrated with an enhanced STEP-FT paradigm, allows our agent to overcome the inherent limitations of the token-by-token decision process in LLMs.

[11] https://rapidapi.com/hub

Limitations

Despite the promising results and substantial improvements demonstrated by our approach, several limitations must be acknowledged:

• Computational Overhead: Although our HE-MCTS employs various mechanisms to enhance iterative accuracy and efficiency, it inevitably introduces additional computational complexity. This overhead can hinder real-time applications and may require further optimization to balance efficiency with decision-making accuracy.

• Reliance on Pretrained Knowledge: As with many large language models, our agent effectiveness is partly limited by the potential obsolescence of its pretraining knowledge. Continuous updates and domain-specific fine-tuning are necessary to mitigate this issue and maintain reliability over time.

Addressing these limitations in future research will be crucial for further refining the agent's performance, ensuring broader applicability, and advancing the integration of specialized tools with large language models in chemical research.

References

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

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. 2024a. Alphamath almost zero: process supervision without process. *arXiv preprint arXiv:2405.03553*.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. 2024b. Step-level value preference optimization for mathematical reasoning. *arXiv preprint arXiv:2406.10858*.

Sijia Chen, Yibo Wang, Yi-Feng Wu, Qing-Guo Chen, Zhao Xu, Weihua Luo, Kaifu Zhang, and Lijun Zhang. 2024c. Advancing tool-augmented large language models: Integrating insights from errors in inference trees. *arXiv preprint arXiv:2406.07115*.

Sylvain Gelly and David Silver. 2011. Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, 175(11):1856–1875.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2024. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *Advances in neural information processing systems*, 36.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, et al. 2024. Toolace: Winning the points of llm function calling. *arXiv preprint arXiv:2409.00920*.

Yubo Ma, Zhibin Gou, Junheng Hao, Ruochen Xu, Shuohang Wang, Liangming Pan, Yujiu Yang, Yixin Cao, Aixin Sun, Hany Awadalla, et al. 2024. Sciagent: Tool-augmented language models for scientific reasoning. *arXiv preprint arXiv:2402.11451*.

Andrew D McNaughton, Gautham Krishna Sankar Ramalaxmi, Agustin Kruel, Carter R Knutson, Rohith A Varikoti, and Neeraj Kumar. 2024. Cactus: Chemistry agent connecting tool usage to science. *ACS omega*, 9(46):46563–46573.

Adrian Mirza, Nawaf Alampara, Sreekanth Kunchapu, Martiño Ríos-García, Benedict Emoekabu, Aswanth Krishnan, Tanya Gupta, Mara Schilling-Wilhelmi, Macjonathan Okereke, Anagha Aneesh, et al. 2024. Are large language models superhuman chemists? *arXiv preprint arXiv:2404.01475*.

Shyue Ping Ong, William Davidson Richards, Anubhav Jain, Geoffroy Hautier, Michael Kocher, Shreyas Cholia, Dan Gunter, Vincent L Chevrier, Kristin A Persson, and Gerbrand Ceder. 2013. Python materials genomics (pymatgen): A robust, open-source python library for materials analysis. *Computational Materials Science*, 68:314–319.

Aaron Parisi, Yao Zhao, and Noah Fiedel. 2022. Talm: Tool augmented language models. *arXiv preprint arXiv:2205.12255*.

Judea Pearl. 1984. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, et al. 2024. Tool learning with foundation models. *ACM Computing Surveys*, 57(4):1–40.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2024. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and error: Exploration-based trajectory optimization for llm agents. *arXiv preprint arXiv:2403.02502*.

Peter Spirtes and Clark Glymour. 1991. An algorithm for fast recovery of sparse causal graphs. *Social science computer review*, 9(1):62–72.

Ye Tian, Baolin Peng, Linfeng Song, Lifeng Jin, Dian Yu, Haitao Mi, and Dong Yu. 2024. Toward self-improvement of llms via imagination, searching, and criticizing. *arXiv preprint arXiv:2404.12253*.

Ziyu Wan, Xidong Feng, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. 2024. Alphazero-like tree-search can guide large language model decoding and training. In *Forty-first International Conference on Machine Learning*.

Mengsong Wu, Tong Zhu, Han Han, Chuanyuan Tan, Xiang Zhang, and Wenliang Chen. 2024a. Seal-tools: Self-instruct tool learning dataset for agent tuning and detailed benchmark. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 372–384. Springer.

Qinzhuo Wu, Wei Liu, Jian Luan, and Bin Wang. 2024b. Toolplanner: A tool augmented llm for multi granularity instructions with path planning and feedback. *arXiv preprint arXiv:2409.14826*.

Dongyu Xue, Han Zhang, Dongling Xiao, Yukang Gong, Guohui Chuai, Yu Sun, Hao Tian, Hua Wu, Yukun Li, and Qi Liu. 2020. X-mol: large-scale pre-training for molecular understanding and diverse molecular analysis. *bioRxiv*, pages 2020–12.

Dan Zhang, Sining Zhoubian, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024a. Rest-mcts*: Llm self-training via process reward guided tree search. *arXiv preprint arXiv:2406.03816*.

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

Xuan Zhang, Chao Du, Tianyu Pang, Qian Liu, Wei Gao, and Min Lin. 2024c. Chain of preference optimization: Improving chain-of-thought reasoning in llms. *arXiv preprint arXiv:2406.09136*.

A More Experimental Details

A.1 HE-MCTS Model Settings

Model	Policy Model	Execution Model	**PRM**	**ORM**
GPT-4o-mini-M	/	/	/	gpt-4o
Claude-3.5-S-M	/	/	/	Claude-3.5-S
Qwen-2.5-7B-M1	D^p	D^u	D^p	gpt-4o
Qwen-2.5-7B-M2	$\tilde{D}^p \cup D^p$	D^u	$\tilde{D}^p \cup D^p$	gpt-4o
Qwen-2.5-7B-M3	$\tilde{D}^p \cup D^p$	D^u	D^{PRM}	D^{ORM}
Llama-3.1-8B-M0	/	D^u	/	gpt-4o
Llama-3.1-8B-M1	D^p	D^u	D^p	gpt-4o
Llama-3.1-8B-M2	$\tilde{D}^p \cup D^p$	D^u	$\tilde{D}^p \cup D^p$	gpt-4o
Llama-3.1-8B-M3	$\tilde{D}^p \cup D^p$	D^u	D^{PRM}	D^{ORM}

Table 5: Training dataset for Different Models

A.2 Results of Single-Calling Dataset

Model	**Format**	**Tool**	**Param**			**Return**	**Pass Rate**
		ACC	**P**	**R**	**F1**	**ACC**	
GPT-4o-mini	100.00	93.20	96.24	94.74	95.48	90.71	86.88
Claude-3.5-S	98.08	92.34	97.24	95.08	96.15	90.52	80.27
ChemLLM-1*	93.97	88.31	97.06	89.02	92.86	88.12	70.98
Qwen-2.5-7B-I	64.08	56.70	91.15	61.79	73.65	54.50	49.23
Llama-2-7B-C	41.11	9.29	42.64	15.43	22.66	6.51	12.16
Llama-3.1-8B-I	98.80	83.62	93.60	87.67	90.54	81.23	63.22
Llama-3.1-8B-I*	97.89	93.87	98.08	92.79	95.36	94.54	69.25

Table 6: Single-Calling Results

A.3 Fine-tuning Results Comparison

Model	Format	Tool ACC	Param P	Param R	Param F1	Return ACC	Pass Rate
ChemLLM-1*							
- v1	93.97	88.31	97.06	89.02	92.86	88.12	70.98
- v2	86.21	79.89	97.49	83.69	90.07	79.98	64.37
- v3	88.79	82.66	97.22	84.97	90.69	82.47	63.98
Llama-3.1-8B-I*							
- v1	97.89	93.87	97.93	92.65	95.22	94.25	68.30
- v2	97.99	93.49	97.80	92.65	95.16	94.16	72.41
- v3	97.89	93.87	98.08	92.79	95.36	94.54	69.25

Table 7: Results of LLMs with different fine-tuning model settings on the single-calling benchmark.

Model	Format	Tool P	Tool R	Tool F1	Param P	Param R	Param F1	Return P	Return R	Return F1	Pass Rate
ChemLLM-1*											
- v1	98.50	56.98	89.51	69.64	50.43	81.60	62.34	51.74	82.51	63.60	46.00
- v2	86.46	68.19	89.98	77.59	59.42	81.04	68.57	53.85	82.19	65.07	46.50
- v3	98.10	76.84	88.08	82.07	68.22	80.20	73.72	68.84	80.45	74.19	53.00
Llama-3.1-8B-I*											
- v1	99.18	60.25	93.00	73.13	51.60	83.57	63.81	55.06	85.69	67.04	57.50
- v2	96.81	94.56	91.26	92.88	85.03	82.16	83.57	84.21	83.94	84.08	57.00
- v3	99.20	93.10	92.21	92.65	83.85	83.15	83.50	85.03	84.90	84.96	55.00

Table 8: Results of LLMs with different fine-tuning model settings on the multiple-calling benchmark.

v1 means the fine-tuning dataset contains no negative cases. **v2** means the fine-tuning dataset contains negative cases for both single and multiple callings. **v3** means the fine-tuning dataset contains negative cases for only multiple callings.

B Algorithm Details of H-MCTS

B.1 Process of H-MCTS

Hierarchical Monte Carlo Tree Search (H-MCTS) is a sampling-based search algorithm. It iteratively constructs a search tree by repeating six phases as illustrated in Figure 5: Expansion, Evaluation, Selection, Simulation, Reflection, and Backpropagation.

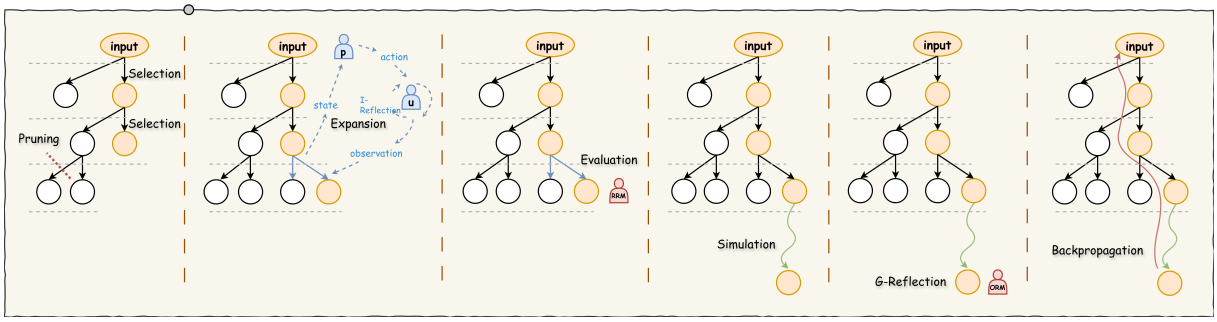

Figure 5: H-MCTS process

Expansion: Policy Model generates k child nodes. To enhance the efficiency of exploration, T is constrained to retrieved tools. To mitigate redundancy, uniqueness is enforced among child nodes, ensuring parent node does not generate duplicate children. Furthermore, to enhance the distinctions between sibling nodes, leverage diversity prompts.

Evaluation: The PRM initializes a scalar score V_i for newly expanded nodes:

$$V_i = V(s_i^p) = PRM(s_i^p) \tag{1}$$

This score is used in the Selection phase to compute the Upper Confidence Bound for Trees (UCT) value of nodes and serves as a reference for choosing starting points in the subsequent Simulation phase.

Selection: It recursively selects nodes from the root based on the Upper Confidence Bound(Kocsis and Szepesvári, 2006) (UCB) which allows the search to prioritize high-value nodes while still encouraging the discovery of new solutions:

$$UCT(s_{i-1}^p, a_i^j) = V_i^j + C \cdot \sqrt{\frac{\ln(N(s_{i-1}^p))}{1 + N(s_{i-1}^p, a_i^j)}} \tag{2}$$

where $N(s_{i-1}^p)$,$N(s_{i-1}^p, a_i^j)$ denote visit counts. V_i^j is initialized by PRM. The hyperparameter C is an exploration coefficient. To promotes exploration of unfinished branches, prioritize non-terminal nodes over terminal ones. To maintain efficient search space, nodes with low information gain and value are adaptively pruned before selection.

Simulation: The Policy Model predicts subsequent actions from selected leaf node until reaching a terminal node s_L^p, where $s_L^p = [q, a_1, o_1, \ldots, a_{L-1}, o_{L-1}, a_L](a_L \in A_n)$. The reward R_L is assigned by $ORM(q, a_L)$. To expedite trajectory simulation and expansion, a single node is sampled at this stage.

Global Reflection(Policy-Level): If agent fails to yield a correct answer, the Policy Model performs failure analysis and generates recommendations to guide subsequent iterations.

Backpropagation: Starting from s_L^p, updates propagate along the path back to s_0^p:

$$N(s_i^p) \leftarrow N(s_i^p) + 1 \tag{3}$$

$$V(s_i^p) \leftarrow V(s_i^p) + \frac{R_L - V(s_i^p)}{N(s_i^p)} \tag{4}$$

where final reward R_L can source heuristic rule or external reward function, like ORM:

$$R_L = ORM(q, a_L), \quad a_L \in A_n \tag{5}$$

B.2 Details of Adaptive Pruning mechanism

Scoring Mechanism The core of pruning is the evaluation of node importance. We compute a comprehensive node score:

$$I(s_i^p) = \alpha V(s_i^p) + \beta U(s_i^p) \tag{6}$$

where $V(s_i^p)$ is the value of node s_i, measuring the historical search gains, $U(s_i^p)$ is the uncertainty estimation of node s_i^p, based on Information Gain (Pearl, 1984)(IG), which quantifies the node's importance in the overall search strategy.

For a node s_i^p with visit count $N(s_i^p)$ and a child set $C(s_i^p)$, Information Gain (IG) is defined as:

$$U(s_i^p) = H(s_i^p) - \sum_{c \in C(s_i)} \frac{N(c)}{N(s_i^p)} H(c) \tag{7}$$

where $H(s_i^p)$ represents the entropy(Silver et al., 2016) of node s_i^p, computed based on search trajectory, indicating the uncertainty of decision-making at that node. A higher information gain suggests a greater impact on the search strategy.

Pruning is guided by an adaptive threshold $\tau(i)$, such that nodes with scores below the threshold are pruned.

Hierarchical Pruning The pruning threshold $\tau(i)$ dynamically adjusts based on search depth i:

• Shallow search($i < D_{\text{early}}$): A lower pruning threshold encourages broader exploration, reducing premature pruning effects

$$\tau(i) = \tau_0 \cdot \left(1 - \lambda \frac{i}{D_{\max}}\right) \tag{8}$$

• Deep search($i > D_{\text{early}}$): The pruning threshold increases, prioritizing high-value paths

$$\tau(i) = \tau_0 \cdot \left(1 + \lambda \frac{i}{D_{\max}}\right) \tag{9}$$

where τ_0 is the initial pruning threshold controlling overall pruning intensity, λ is a hyperparameter regulating threshold variation, $D_{\max}$ is the maximum search depth, ensuring progressive pruning refinement.

Soft Pruning To mitigate search loss from mispruning, Soft Pruning(Gelly and Silver, 2011) retains pruned nodes with a certain probability. If $I(s_i^p) < \tau(i)$, the node is retained with probability:

$$P_{\text{retain}} = e^{-\kappa(\tau(i) - I(s_i^p))} \tag{10}$$

where κ controls the pruning probability decay rate, allowing nodes close to the threshold to have a higher retention probability.

Fast Recovery To prevent excessive pruning from limiting search effectiveness, we introduce a Fast Recovery mechanism(Spirtes and Glymour, 1991) :

• Pruned Node Logging: Maintain records of pruned nodes, including Score $I(s_i^p)$, Pruning depth i, Visit count $N(s_i^p)$.

• Detect Search Degradation: If the search reward drops significantly compared to the best path:

$$\frac{V_{\text{best}} - V_{\text{current}}}{V_{\text{best}}} > \epsilon \tag{11}$$

where V_{best} is the average value of the best search path, V_{current} is the average value of the current search path, ϵ is the recovery threshold.

Restore recently pruned high-score nodes from history records to reintroduce potentially valuable search directions.

C Tool Integration Procedure

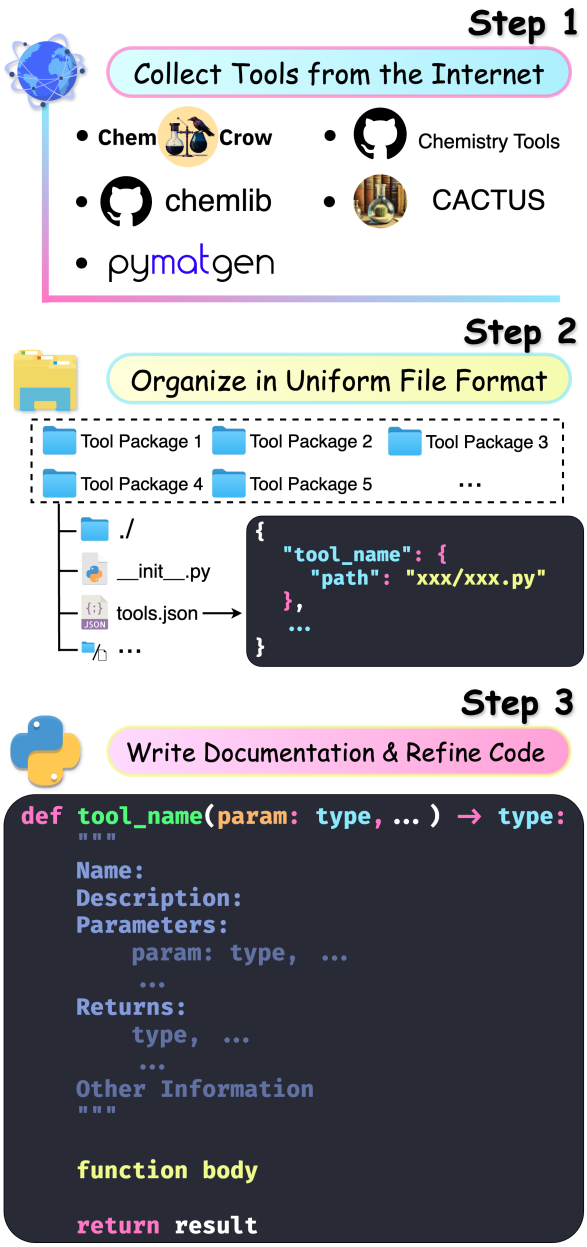

Figure 6: Tool Integration Procedure in 3 steps.