Advancing Tool-Augmented Large Language Models via Meta-Verification and Reflection Learning

Zhiyuan Ma
State Key Laboratory of Cognitive Intelligence, University of Science and Technology of China
Hefei, China
zhyma@mail.ustc.edu.cn

Jiayu Liu
State Key Laboratory of Cognitive Intelligence, University of Science and Technology of China
Hefei, China
jy251198@mail.ustc.edu.cn

Xianzhen Luo
Research Center for Social Computing and Interactive Robotics, Harbin Institute of Technology
Harbin, China
xzluo@ir.hit.edu.cn

Zhenya Huang*
State Key Laboratory of Cognitive Intelligence, University of Science and Technology of China
Hefei, China
huangzhy@ustc.edu.cn

Qingfu Zhu
Research Center for Social Computing and Interactive Robotics, Harbin Institute of Technology
Harbin, China
qfzhu@ir.hit.edu.cn

Wanxiang Che
Research Center for Social Computing and Interactive Robotics, Harbin Institute of Technology
Harbin, China
car@ir.hit.edu.cn

Abstract

Empowering large language models (LLMs) with effective tool utilization capabilities is crucial for enabling AI agents to solve complex problems. However, current models face two major limitations: (1) unreliable tool planning and invocation due to low-quality instruction datasets (e.g., widespread hallucinated API calls), and (2) weak tool reflection abilities (over 90% of errors cannot be corrected) resulting from static imitation learning. To address these critical limitations, we propose **Tool-MVR**, a novel Tool-Augmented LLM that achieves comprehensive System 2 reasoning through two key innovations. Specifically, we first introduce **M**ulti-**A**gent **M**eta-**V**erification (**MAMV**), a systematic pipeline that rigorously validates APIs, queries, and reasoning trajectories to construct ToolBench-V, a new high-quality instruction dataset that addresses the limitation of unreliable tool planning and invocation. Second, we propose **Explo**ration-based **Re**flection Learning (**EXPLORE**), which enhances tool reflection capabilities by leveraging tool feedback through a dynamic "Error → Reflection → Correction" learning paradigm, resulting in our reflection dataset ToolBench-R and addressing the critical weakness in tool reflection. Finally, we obtain Tool-MVR by finetuning open-source LLMs (e.g., Qwen-7B) on both ToolBench-V and ToolBench-R. Our experiments demonstrate that Tool-MVR achieves state-of-the-art performance on StableToolBench, surpassing both ToolLLM (by 23.9%) and GPT-4 (by 15.3%) while reducing API calls by 31.4%, with strong generalization capabilities across unseen tools and scenarios. Additionally, on our proposed RefineToolBench, the first benchmark specifically designed to evaluate tool reflection capabilities. Tool-MVR achieves a 58.9% error correction rate, significantly outperforming ToolLLM's 9.1%. [1]

CCS Concepts

• **Computing methodologies** → **Knowledge representation and reasoning**; **Natural language processing**.

Keywords

Large Language Models, Tool learning, Self-Reflection

ACM Reference Format:
Zhiyuan Ma, Jiayu Liu, Xianzhen Luo, Zhenya Huang, Qingfu Zhu, and Wanxiang Che. 2025. Advancing Tool-Augmented Large Language Models via Meta-Verification and Reflection Learning. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3711896.3736835

1 Introduction

Large language models (LLMs) have shown impressive capabilities in various tasks, from natural language understanding to complex reasoning [7, 12, 25, 43, 57, 64–66, 69]. However, general-purpose LLMs face inherent limitations in accessing real-time data and specialized expertise [26, 38, 52, 60]. Similar to how humans leverage tools for problem-solving, the tool learning task enables LLMs to utilize external tools (e.g., weather forecast APIs, airline reservation APIs) for accessing real-time information and interacting with the physical world, expanding their capabilities [36].

Tool learning requires multi-turn interactions between LLMs and external tools. As shown in the green trajectory of Figure 1, for a weather query "What's the weather …December 25, 2024?", the model involves three key capabilities: (1) **Tool planning**: The LLM correctly identifies the need to use the weather forecast API. (2) **Tool invocation**: It then generates precise API calls with proper parameters (city="London", date="2024-12-25"). (3) **Tool reflection**: It corrects errors through feedback when tool planning or invocation

*Zhenya Huang is the corresponding author.

KDD '25, Toronto, ON, Canada.

ACM ISBN 979-8-4007-1454-2/25/08
https://doi.org/10.1145/3711896.3736835

[1] Codes and data are available at: https://github.com/zhymma/Tool-MVR

fails [17, 19, 31, 35, 47, 68]. Such a complex tool learning process requires System 2 reasoning - a deliberate, step-by-step approach that humans use for sophisticated problem-solving [11, 22, 24, 50, 55].

Recent studies have explored two main approaches to tackle tool learning task [38, 52]. The first approach focuses on building Tool-Augmented agents using powerful closed-source LLMs through prompt engineering [4, 13], from ReAct's "thought-action-observation" format [59] to more sophisticated frameworks like ToolChain's A* search [71]. The second approach aims to enhance open-source LLMs' tool utilization capabilities through instruction tuning. ToolBench [37], currently the most widely-used dataset, leverages large-scale REST API collections and constructs extensive tool instruction data through depth-first search annotation. This dataset has enabled the development of various Tool-Augmented LLMs [3, 8, 61]. For example, the instruction-tuned ToolLLM achieves performance comparable to ChatGPT [33].

However, current tool learning research faces two critical limitations. First, mainstream tool instruction datasets (e.g., ToolBench) used for training Tool-Augmented LLMs suffer from severe quality issues. Recent analysis [18] reveals that in ToolBench [37], 57.3% of queries contain unsolvable requests or incomplete information, and more critically, 74.4% of API call trajectories exhibit hallucination behaviors (detailed in Section 3.2). As illustrated by the blue trajectory in Figure 1, which shows a typical hallucinated API call in ToolBench: (1) omitting the required date parameter and only calling the API with "city: London"; and (2) incorrectly using current weather data (December 20) to answer a query about December 25. These poor-quality instructions severely impair the model's System 2 reasoning capabilities in both tool planning and tool invocation [23].

Furthermore, current Tool-Augmented LLMs lack tool reflection capabilities - a critical weakness is given that API call errors are inevitable in complex real-world scenarios [20, 44]. The ability to correct errors based on execution feedback is essential for robust tool utilization. However, we observe that ToolLLM abandons over 25% of tasks because it falls into local traps when encountering errors, unable to correct wrong actions from multi-turn interactions. As shown in the red trajectory of Figure 1, when encountering an incorrect date format error, the model fails to make the simple correction despite receiving explicit API feedback about the required YYYY-MM-DD format, and abandons the task. To further emphasize this phenomenon, we introduce RefineToolBench, the first benchmark designed to measure tool reflection capabilities (detailed in Section 4.1.2), which shows that ToolLLM successfully corrects errors in only 9.1% of cases. This lack of adaptive tool reflection capability reveals that current Tool-Augmented LLMs have not yet achieved genuine System 2 reasoning [2].

To address these limitations, we develop a novel two-stage approach: First, unlike previous instruction construction methods that lack reliable verification mechanisms, we construct a new high-quality tool instruction dataset ToolBench-V through **M**ulti-**A**gent **M**eta-**V**erification (**MAMV**), a systematic pipeline that validates: (1) APIs and queries for tool planning - ensuring reliable API functionality and complete user requirements; and (2) API-call trajectories for tool invocation - guaranteeing accurate API call trajectories. This MAMV pipeline effectively addresses the data quality issues, establishing solid foundations for tool planning and tool invocation. Building upon this foundation, we introduce **Explor**ation-based **Re**flection Learning (EXPLORE) to enhance models' tool reflection capabilities. In contrast to previous training methods that focus solely on imitating successful expert trajectories while ignoring error cases [49, 56], EXPLORE actively explores error scenarios to identify model weaknesses and leverages execution feedback to construct reflection data. This forms a dynamic "Error → Reflection → Correction" learning paradigm, resulting in our reflection dataset ToolBench-R. Through this process, models learn to reflect and optimize wrong actions based on tool feedback, addressing the critical weakness in tool reflection capabilities.

Finally, by finetuning open-source LLMs (e.g., Qwen-2.5-7B [58]) on both ToolBench-V and ToolBench-R, we obtain Tool-MVR that achieves sophisticated System 2 reasoning with deliberate tool planning, accurate tool invocation, and adaptive tool reflection. The main contributions of this work are as follows:

- We propose Tool-MVR, achieving comprehensive System 2 reasoning capabilities with significantly improved performance, efficiency and generalization - surpassing GPT-4 by 15.3% on StableToolBench while reducing API calls by 31.4%, and maintaining strong performance across diverse unseen tools and scenarios.
- We develop MAMV for systematic instruction data verification, achieving substantial improvements in both query validity (from 52.7% to 98.8%) and trajectory accuracy (from 25.6% to 81.3%) in ToolBench-V.
- We introduce EXPLORE learning algorithm, enabling Tool-MVR to achieve a 58.9% error correction rate on RefineToolBench, significantly outperforming ToolLLM's 9.1%.

2 Related Works

2.1 Tool Learning

Tool learning has emerged as a crucial task for expanding LLMs' problem-solving abilities by enabling interaction with external tools to overcome inherent limitations such as outdated information and lack of real-time data access [43, 66]. Prior works have explored two main approaches. The first approach, focusing on prompt engineering with closed-source LLMs [4, 13, 46, 53, 62], has evolved from ReAct's [59] basic thought-action format to more sophisticated frameworks. Chameleon [30] enhanced compositional reasoning by orchestrating multiple tools with GPT-4, while ToolChain [71] improved efficiency through A* search in action space. The second approach aims to enhance open-source LLMs through instruction tuning [5, 15, 46, 54, 63]. Toolformer [45] pioneered this direction by embedding API calls into text generation. ToolBench [37] made significant progress by establishing the largest tool learning dataset to date, encompassing 16,464 real-world APIs, and introduced ToolLLM with its depth-first search decision tree strategy (DFSDT) for trajectory annotation. This work provided a foundation for systematic tool learning research but was limited by dataset quality issues. Building upon ToolBench, TP-LLaMA [3] introduced preference learning to leverage failed explorations through DPO [39, 40].

Most recently, works such as APIGen [29] and Tool-Ace [27] have introduced automated data generation frameworks aimed at

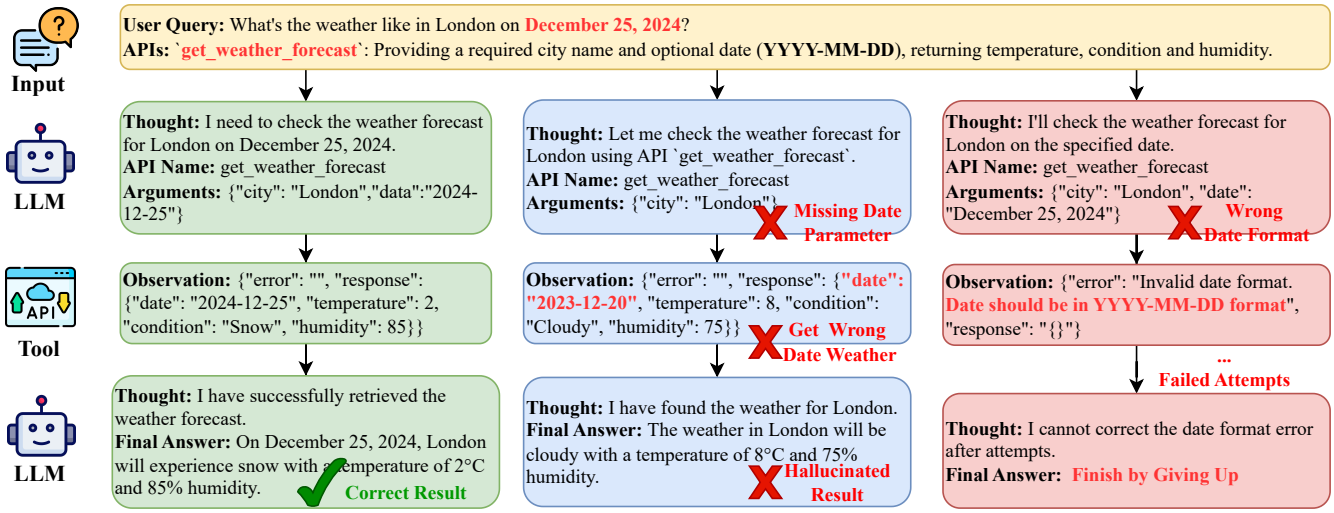

Figure 1: Tool learning examples: given user queries and APIs, models generate API calls for answers. The green trajectory shows correct API calls, the blue trajectory shows hallucinated API calls, and the red trajectory demonstrates failed self-reflection.

improving tool utilization capabilities through methods like multi-stage API validation and diverse query styles. However, these approaches primarily focus on single-API instructions. In contrast, our work aims to enhance tool using and reflexion in complex multi-tool scenarios.

2.2 Self-Reflection

Recent work has explored feedback-based approaches for LLM self-reflection through various mechanisms [20, 44, 67]. Self-Refine [32] uses the same LLM for output generation and feedback-based improvements, while Reflexion [48] adds memory mechanisms to prevent recurring errors [34]. For precise reasoning tasks, LeDex [21] enhances code generation through explanation-based debugging, and CRITIC [14] leverages external verification for mathematical reasoning. However, in tool learning, self-reflection remains limited. Existing approaches like DFSDT [37] and AnyTool [9] handle errors through simple backtracking or complete restarts, leading to inefficient reasoning. Our work advances feedback-driven reflection through an exploration-based learning algorithm, enabling direct error correction from tool feedback.

3 Method

In this section, we first provide a formal definition of tool learning task in Section 3.1. Then we present our approach to enhance Tool-Augmented LLMs through two key innovations: (1) developing MAMV to construct a new high-quality dataset ToolBench-V with 11,765 multi-turn tool interaction instances (Section 3.2), and (2) developing EXPLORE, an exploration-based reflection learning algorithm that enables models to learn from tool feedback and correct errors in tool interaction (Section 3.3).

3.1 Preliminaries

The tool learning task requires models to effectively utilize external tools through multi-turn interactions [38, 52]. As shown in Figure 1, given a user query Q (e.g., "What's the weather ...?") and a tool subset $T = \{t_1, \ldots, t_n\}$ (e.g., weather forecast APIs), the agent's objective is to utilize appropriate tools to acquire information and derive an answer. Let $\mathcal{T}$ denote the tool learning process:

$$\mathcal{T} : Q \times T \rightarrow (S, r), \tag{1}$$

where S represents the reasoning trajectory and r the final answer. The model gradually solves problems through multiple rounds of interaction. Each round i consists of: (1) action $a_i = (t_i, p_i)$: selecting tool $t_i \in T$ and generating calling parameters p_i (e.g., selecting weather API and specifying parameters like city="London", date="2024-12-25"); (2) observation $o_i = t_i(p_i)$: executing the call and obtaining API feedback information o_i (e.g., receiving weather forecast data or error messages about incorrect date format). Over the course of interaction, the model forms a reasoning trajectory and generates the final answer:

$$(S, r) = ((a_1, o_1, \ldots, a_t, o_t), r). \tag{2}$$

Based on the API responses in trajectory S, the model synthesizes a final answer r (e.g., "The weather will be ..."). The effectiveness of solution is evaluated by determining whether trajectory S and answer r adequately address query Q (details in Section 4.1).

Unlike System 1's fast, intuitive thinking process, tool learning requires sophisticated System 2 reasoning - a deliberate, step-by-step problem solving approach [11, 17, 35]. Specifically, this involves three key capabilities: (1) tool planning: analyzing query requirements and available APIs to determine the necessary tools; (2) tool invocation: constructing API calls with appropriate parameters (e.g., date="2024-12-25"); and (3) tool reflection: adjusting tool planning and invocation based on execution feedback (e.g., modifying date format or selecting alternative APIs). In this work, we aim to enhance open-source LLMs' System 2 reasoning capabilities through these three aspects, enabling them to achieve the tool utilization capability of powerful closed-source models like GPT-4 [1].

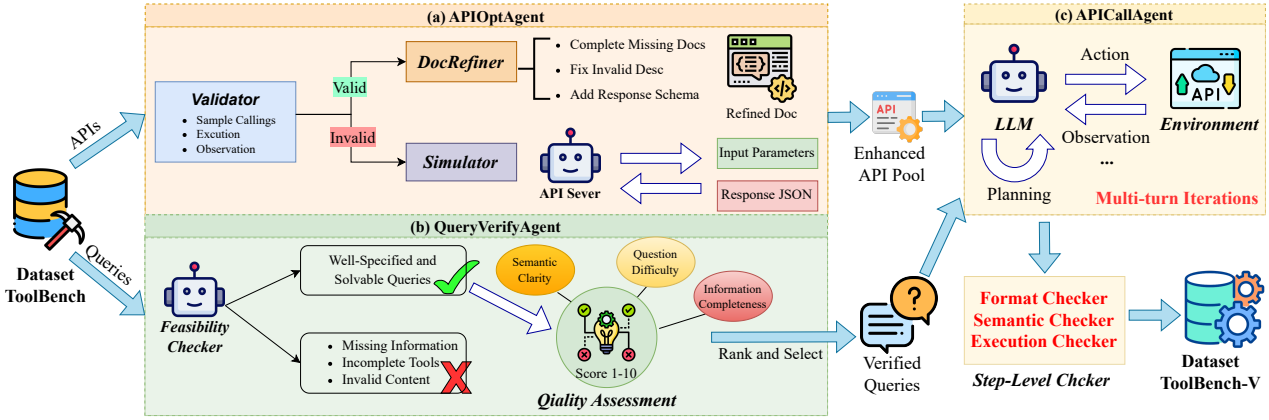

Figure 2: Overview of Multi-Agent Meta-Verification (MAMV) framework. MAMV creates a new high-quality tool instruction dataset ToolBench-V via (a) APIOptAgent for API verification and optimization, (b) QueryVerifyAgent for query assessment and filtering, and (c) APICallAgent for step-level accurate API call trajectory generation.

3.2 Multi-Agent Meta-Verification Pipeline

While instruction quality is crucial for model performance in specialized domains, existing mainstream tool instruction datasets primarily focus on quantity over quality [23, 28]. Recent analysis from Quality Matters [18] reveals systematic issues in current datasets' construction methodology. These issues arise primarily from the lack of systematic verification in data collection process. Specifically, ToolBench [37] relies heavily on ChatGPT [33] for data generation and employs a depth-first search strategy for trajectory annotation, without incorporating reliable verification mechanisms. This leads to the propagation of model hallucinations and annotator biases into the instruction data.

Specifically, we first analyze the representative dataset ToolBench [37], following the evaluation protocol in Quality Matters [18]. As shown in Table 1, our analysis reveals significant quality issues: For query quality, approximately 47.3% of queries exhibit three critical issues: (1) information completeness: queries frequently omitting essential parameters (e.g., "Recommend romantic venues" missing location specification); (2) logical consistency: requirements often containing contradictions (e.g., booking past dates); and (3) API feasibility: queries requesting operations beyond API capabilities. For API-call quality, about 74.4% of trajectories contain three systematic problems: (1) parameter accuracy: 47.2% of API calls containing incorrect formats or values; (2) solution completeness: 33.1% of trajectories missing critical API calls; and (3) trajectory minimality: redundant API calls appearing as repeated queries. Such hallucination-prone instruction datasets like ToolBench lead to ToolLLM learning incorrect tool reasoning patterns, fundamentally limiting its tool planning and invocation capabilities.

To address the severe quality issues in existing tool learning datasets, we propose **M**ulti-**A**gent **M**eta-**V**erification (**MAMV**), a novel pipeline that systematically validates three meta-components in tool learning: APIs, user queries, and API call trajectories. Through collaborative verification, MAMV performs: (1) API verification (for tool planning - ensuring reliable API functionality and clear documentation), (2) query verification (for tool planning - ensuring query completeness and feasibility for tool selection), and (3) trajectory verification (for tool invocation - ensuring accurate parameter generation and efficient execution flow). Using MAMV, we improve ToolBench to ToolBench-V. Formally, each instance in ToolBench-V can be represented as:

$$V = \{(Q, T, S)\}, \tag{3}$$

where Q represents the verified queries from ToolBench, T the enhanced API pool, and S the newly constructed API call trajectory with step-wise verification. MAMV employs multiple agents powered by GPT-4. Each agent focuses on a specific verification task, collaboratively ensuring comprehensive quality control.

Table 1: Evaluation of instruction dataset quality scores (%).

Query Quality	Spec.	Coh.	Solv.	Overall
ToolBench	79.6	77.9	81.8	52.7
ToolBench-V	**99.9**	**99.0**	**99.9**	**98.8**
API-Call Quality	**Align.**	**Suff.**	**Min.**	**Overall**
ToolBench	52.1	66.4	54.9	25.6
ToolBench-V	**90.6**	**96.5**	**81.9**	**81.3**

3.2.1 API Verification. We first focus on verifying and enhancing both API functionality and documentation. Leveraging RapidAPI Hub [42], the world's largest API marketplace, we selected over 12,000 real-world APIs spanning 49 core domains (e.g., social media, travel services). These professionally maintained APIs support diverse practical applications, denoted as $T_{all} = \{t_1, \ldots, t_n\}$. However, these third-party APIs often present two major challenges: service instability and inconsistent documentation quality with incomplete parameter descriptions. To address these challenges, we develop **APIOptAgent** as part of our MAMV pipeline (Figure 2(a)). Specifically, we first employ GPT-4 to generate and execute diverse request samples for feedback collection. The **Validator** then assesses API validity based on these results, identifying common failure cases

(e.g., "API doesn't exist", "ACCESS_DENIED"). For invalid APIs, our **Simulator** serves as a virtual API server, responding to requests with semantically appropriate JSON responses containing simulated data derived from API documentation to ensure continuous API functionality. For valid APIs, **DocRefiner** enhances documentation by adding missing parameter specifications (types, defaults, ranges) and standardizing descriptions based on API response analysis. Through this process, we transformed 11,302 original APIs into an optimized pool: validating 6,770 functional APIs, enhancing 6,334 API documents, and providing simulation alternatives for 4,500+ unavailable APIs. This verification ensures both API functionality and clear documentation for effective tool planning (details in Appendix B.1).

3.2.2 Query Verification. User queries are crucial for accurate user intent analysis and appropriate tool selection. In ToolBench [37], approximately 80,000 initial queries were generated by ChatGPT [33], covering various API usage scenarios. For each query Q, we select a relevant subset $T \subseteq T_{all}$ to address it. However, these ChatGPT-generated queries often contain unsolvable cases that hinder proper requirement analysis and tool mapping. For instance, queries like "I'm interested in a famous actor's career. Could you provide detailed information about their filmography?" lack essential information (actor name specification), making it impossible to map to appropriate APIs and parameters. Such incomplete queries lead to incorrect tool selection and flawed reasoning patterns, fundamentally limiting the model's tool planning capabilities. We tackle these limitations by developing **QueryVerifyAgent** with two key components, as illustrated in Figure 2(b). The **Feasibility Checker** utilizes GPT-4 to filter out unsolvable queries due to missing information, logical contradictions, or lack of necessary tools. The **Quality Assessor** then evaluates remaining queries across three dimensions: semantic clarity, information completeness, and reasoning complexity, using a 1-10 scoring scale. We retain only high-quality queries (scores 8-10) to ensure query solvability, complexity, and completeness (see Appendix B.2 in Appendix). This rigorous filtering process yielded 23,903 high-quality queries from an initial pool of 62,147 (detailed implementation in Appendix B.2).

3.2.3 Trajectory Verification. The quality of API call trajectories directly impacts models' tool invocation capabilities. The previous DFSDT approach, which combines ChatGPT with DFS algorithm for API sequence search, results in over 47% parameter errors, 33% incomplete solutions, and 45.1% redundant trajectories. By focusing solely on final answer accuracy while ignoring intermediate reasoning processes, DFSDT generates numerous intermediate errors and hallucinated behaviors.

These challenges stem from the absence of verification in existing methods. To overcome these limitations, we develop **APICallAgent**, a multi-turn interactive framework with step-wise verification mechanism. As shown in Figure 2(c), APICallAgent operates through multi-round interactions among user queries, LLM reasoning (GPT-4), and API environment feedback. The agent follows a systematic cycle of planning (chain-of-thought reasoning for tool selection), action (parameter generation), and observation (execution feedback processing) to ensure logical consistency throughout the reasoning process. Inspired by CodeAct [51], we adopt Python function calls instead of JSON format to better leverage LLMs' inherent coding capabilities, thereby improving API calling accuracy.

To ensure reliable API call trajectories, we designed a step-wise verification mechanism with three components: (1) **Format Checker** ensures final answer generation through syntax parsing, (2) **Semantic Checker** validates comprehensive problem resolution through constraint verification, and (3) **Execution Checker** examines each execution round to ensure positive contribution while eliminating redundant calls and hallucinated behaviors. This comprehensive verification ensures high-quality API call trajectories (detailed implementation in Appendix B.3).

3.2.4 ToolBench-V. Hence, our MAMV pipeline creates ToolBench-V, a high-quality dataset of 11,765 multi-turn tool interaction instances. Through systematic verification, we effectively address the quality issues in existing datasets - achieving significant improvements in both query validity (from 52.7% to 98.8%) and trajectory accuracy (from 25.6% to 81.3%). The well-verified queries enhance models' ability to understand user intent and select appropriate tools, while accurate trajectories guide proper API parameter generation. This comprehensive meta-verification approach establishes a solid foundation for System 2 reasoning by ensuring reliable tool planning and efficient tool invocation capabilities.

3.3 Exploration-based Reflection Learning

Currently, existing Tool-Augmented LLMs rely solely on static imitation learning with correct expert trajectories, inevitably leading to brittle models that struggle with error recovery [49, 56]. Our analysis reveals that ToolLLM abandons 25% of tasks when encountering execution failures, failing to recover and continue reasoning. To systematically verify this limitation, we introduce RefineToolBench, the first benchmark for evaluating models' tool reflection capabilities, where ToolLLM achieves only a 9.1% error correction rate (details in Section 4.6). This poor tool reflection ability significantly limits models' practical applications in complex scenarios.

3.3.1 EXPLORE. In response to these limitations, we introduce EXPLORE, an exploration-based reflection learning algorithm enabling systematic error correction through a dynamic "Error → Reflection → Correction" learning paradigm. As illustrated in Figure 3, EXPLORE actively samples step-level error actions online to identify model weaknesses and leverages tool feedback to construct reflection data using GPT-4. We categorize tool learning errors into two types: (1) API Calling errors (execution failures due to missing parameters, type mismatches, or incorrect API names); and (2) API Planning errors (successful but task-irrelevant executions due to incorrect tool selection or parameter content). Unlike previous approaches that discard failed cases [37], EXPLORE utilizes these errors as valuable learning signals, enabling models to reflect on mistakes and improve multi-turn interaction capabilities.

Building upon the foundational capabilities from the first stage, EXPLORE uses verified ToolBench-V trajectories as references for self-improvement. For a trajectory $S = (a_1, o_1, \ldots, a_n, o_n)$, we randomly select steps o_t as exploration points. As shown in Figure 3, for a weather query, the verified trajectory S demonstrates correct API usage. Through multiple sampling, we construct an exploration tree where successful explorations (blue) lead to task completion,

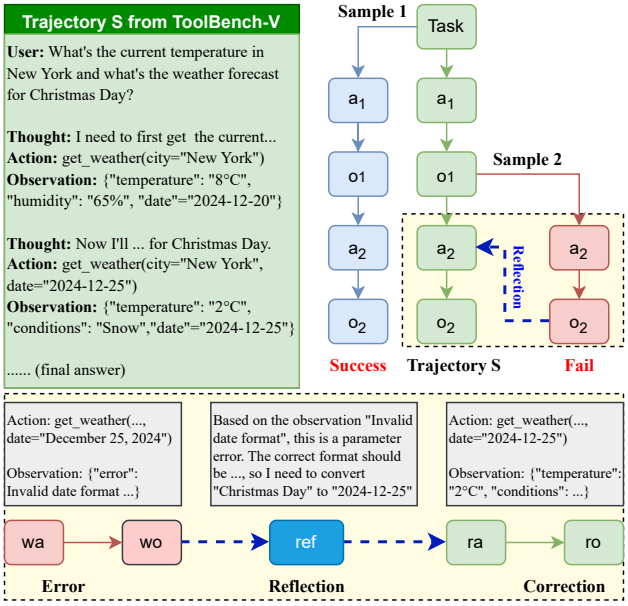

Figure 3: Exploration-based Reflection Learning (EXPLORE): performs step-level exploration with sampling, enabling "Error→Reflection→Correction" learning paradigm.

while failed attempts (red) trigger reflection generation. For failed attempts, the exploration process with o_t as the starting point can be formalized as:

$$E_t = \{(\text{wa}, \text{wo}, \text{ra}) | \text{ra} = a_{t+1}\}, \tag{4}$$

where wa represents wrong actions, wo represents the feedback, and ra $= a_{t+1}$ the corresponding correct action. This reflection process is crucial, enabling models to learn from feedback and adjust tool planning or invocation.

3.3.2 ToolBench-R. Through this dynamic exploration process, we construct ToolBench-R, a reflection-oriented dataset:

$$R = \{(Q, T, S_{<t}, \text{wa}, \text{wo}, (\text{ref}, \text{ra})) \mid (\text{wa}, \text{wo}, \text{ra}) \in Et\}, \tag{5}$$

where each instance captures a complete reflection process with context: the query Q, tools T, and historical interactions $S_{<t}$.

ToolBench-R is constructed by leveraging GPT-4 to generate structured reflection content for each error case. This involves analyzing execution feedback for issues, developing correction strategies, and formulating precise execution plans. For example, when encountering date format errors in weather queries, GPT-4 analyzes the tool feedback to generate structured reflections that guide parameter correction from "December 25, 2024" to "2024-12-25". This structured reflection content forms the basis of ToolBench-R.

The reflection process emphasizes adaptive tool reflection through active processing and learning from real-time feedback. Each instance in ToolBench-R includes analytical (error analysis, correction strategy) and practical (concrete execution plans) components, enabling comprehensive tool reflection capabilities. This systematic approach results in ToolBench-R having 3,625 reflection instances covering diverse error types, robustly handling errors in complex real-world scenarios (further details in Appendix Appendix B.4).

3.4 Training

We train our model through Supervised Fine-Tuning (SFT) to develop three System 2 reasoning capabilities. The training objective consists of two components corresponding to tool planning, invocation, and reflection abilities.

For ToolBench-V, which focuses on tool planning and invocation capabilities, we optimize the action prediction loss at each interaction step t in sequence S:

$$\mathcal{L}_V = -\sum_{x \in V} \sum_{t=1}^{|S|} \log p(a_t | Q, T, a_{<t}, o_{<t}), \tag{6}$$

where $x = (Q, T, S)$ denotes instances from the meta-verified dataset.

For ToolBench-R, which enhances tool reflection capabilities through reflection learning, we optimize the joint prediction of reflection and corrected action:

$$\mathcal{L}_R = -\sum_{y \in R} \log p((\text{ref}, \text{ra}) | Q, T, S_{<t}, \text{wa}, \text{wo}), \tag{7}$$

where $y = (Q, T, S_{<t}, \text{wa}, \text{wo}, (\text{ref}, \text{ra}))$ denotes instances from the reflection dataset. The final training objective combines both losses:

$$\mathcal{L} = \mathcal{L}_V + \lambda \mathcal{L}_R, \tag{8}$$

where λ is a hyperparameter balancing the importance between meta-verified instruction learning and reflection learning.

By jointly optimizing these objectives during SFT, we train ToolMVR to master all three System 2 capabilities: (1) reliable tool planning through high-quality API and query verification, (2) accurate tool invocation through verified trajectories, and (3) effective tool reflection through feedback-based error correction.

4 Experiments

4.1 Benchmark and Evaluation Metrics

4.1.1 StableToolBench. We evaluate model performance on StableToolBench [16], an enhanced and more stable test set derived from ToolBench [37]. It contains 765 carefully curated tasks that systematically evaluate different aspects of tool learning capabilities. We examine six test scenarios: G1-Ins., G1-Tool, G1-Cat., G2-Ins., G2-Cat., and G3-Ins. These scenarios combine task complexity (**G1**: single-tool instructions, **G2**: intra-category multi-tool instructions, **G3**: cross-category multi-tool instructions) with generalization levels (**Ins.**: unseen instructions, **Tool**: unseen tools from seen categories, **Cat.**: unseen tool categories). Performance is measured using two key metrics: **Pass Rate** and **Win Rate**, evaluating task completion success and reasoning quality respectively.

4.1.2 RefineToolBench. While existing benchmarks focus primarily on basic tool usage capabilities, they lack systematic assessment of error recovery - a crucial aspect of System 2 reasoning. To address this gap, we introduce RefineToolBench, the first benchmark specifically designed to evaluate tool reflection capabilities. To ensure comprehensive coverage of error scenarios, the benchmark includes both API-level errors in single-tool scenarios ($I1$, $I2$) and complex reasoning failures in multi-tool interactions ($I3$), with carefully designed test cases that require models to recover from

errors through reflection. The final dataset contains 344 $I1$ cases (testing parameter and API name errors with available APIs), 388 $I2$ cases (evaluating error handling with simulator-required APIs), and 178 $I3$ cases (assessing complex tool selection and interaction errors). We evaluate using two key metrics: **Error Recognition Rate** (**ERR**) measuring error identification capability, and **Error Correction Rate** (**ECR**) assessing successful error resolution.

The construction process for RefineToolBench varies by scenario type. For single-tool scenarios (I1 and I2), we use GPT-4 to generate query-action pairs with deliberate API-level errors, ensuring coverage of various error types. For multi-tool scenarios (I3), we first obtain complete reasoning trajectories from StableToolBench using our Stage 1 multi-agent system, then use GPT-4 to introduce errors in tool selection and parameter content at randomly selected steps. All cases undergo rigorous filtering to ensure solvability and meaningful feedback availability, resulting in a comprehensive benchmark for evaluating models' error handling and reflection capabilities.

Comprehensive and detailed evaluation protocols for both benchmarks are provided in Appendix Appendix A.

4.2 Baselines and Settings

We compare our Tool-MVR with the following baselines: **GPT-3.5** [33], a widely-adopted closed-source LLM that demonstrates excellent performance in tool utilization. **GPT-4** [1], the current state-of-the-art closed-source model, exhibits exceptional capabilities in complex reasoning and tool invocation. In the open-source domain, we select two powerful models: **LLaMA-3.1-8B-Instruct** [10] and **Qwen-2.5-7B-Instruct** [58]. Using these as backbone models, **ToolLLM** [37] was developed through supervised fine-tuning on the ToolBench dataset. Subsequently, **TP-LLaMA** [3] represents a strong DPO-optimized baseline that enhances inference trajectories by leveraging preference data extracted from decision trees. **StepTool** [61] introduces an effective PPO framework that pioneers step-grained reinforcement learning with reward shaping to improve tool learning capabilities. Note that all baselines here are combined with DFSDT for inference.

4.3 Implementation Details

We adopt LLaMA-3.1-8B-instruct and Qwen-2.5-7B-instruct as our Tool-MVR backbone and develop our training pipeline based on the LLaMA-Factory [70] framework. For training loss, we set $\lambda = 1$ and maintain a 10:1 ratio between ToolBench-V and ToolBench-R to balance three System 2 reasoning capabilities. The models are optimized using full-parameter fine-tuning with a learning rate of 2e-5 and warmup ratio of 0.04. We leverage DeepSpeed ZeRO-3 [41] optimization and Flash Attention [6] for training acceleration. All experiments are conducted on a machine equipped with 8 NVIDIA A800 GPUs (80GB memory each).

For dataset construction in our ToolBench-V and ToolBench-R, we employ a combination of "gpt-4-turbo", "gpt-4o", and "gpt-4" models. In our baseline comparisons, GPT-4 refers to "gpt-4-turbo", while GPT-3.5 refers to "gpt-3.5-turbo". All evaluation metrics are computed using "gpt-4-turbo" with temperature set to 0.

4.4 Data Quality Comparison

As discussed in Section 3.2, ToolBench suffers from severe quality issues across two dimensions: query quality and API-call quality, both of which are evaluated using three key metrics, following the evaluation protocol in Quality Matters [18]. To validate the effectiveness of our MAMV pipeline in addressing these limitations, we conducted a comprehensive quality assessment comparing ToolBench-V against the ToolBench dataset, as shown in Table 1. The results demonstrate substantial improvements with our MAMV approach: for user queries, MAMV increases the overall quality score from 52.7% to 98.8%, validating our QueryVerifyAgent's effectiveness in filtering out problematic queries. Similarly, for API-call trajectories, our method improves the overall accuracy from 25.6% to 81.3% through APICallAgent's comprehensive verification mechanism. For instance, the Execution Checker effectively eliminates redundant API calls and trial-and-error attempts, raising the trajectory minimality score from 54.9% to 81.9%. Through these systematic improvements, MAMV successfully addresses the critical limitations in existing datasets: poor query quality, hallucinated API usage, and establishes a strong foundation for tool learning."

4.5 Main Results on StableToolBench

We evaluate Tool-MVR against various baselines. First, we conduct comprehensive performance comparisons using Pass Rate and Win Rate metrics on StableToolBench, with results presented in Tables 2 and 3. Second, we perform ablation studies to analyze the contribution of each component in our framework, which further highlights the superior tool planning and invocation capabilities of Tool-MVR.

4.5.1 Pass Rate Analysis. As shown in Tables 2, we first evaluate models' performance using Pass Rate, which measures their ability to successfully complete tool learning tasks. Our Tool-MVR models, implemented on both LLaMA-3.1-8B (Tool-MVR (L)) and Qwen-2.5-7B (Tool-MVR (Q)), achieve substantial improvements over existing baselines. Tool-MVR (Q) attains an average Pass Rate of 83.8%, surpassing GPT-4 (by 15.3%), GPT-3.5 (by 39.9%), and ToolLLM (Q) (by 23.9%). Similarly, Tool-MVR (L) achieves 80.5%, exceeding all LLaMA-based baselines and GPT-4 by 12%. The superior performance of Tool-MVR can be attributed to two key innovations. First, MAMV's systematic verification ensures high-quality instruction data, which directly improves tool planning and invocation capabilities. Second, EXPLORE's reflection learning mechanism enables models to effectively adapt to and recover from errors. Beyond performance improvements, our experimental results also demonstrate Tool-MVR's advantages in several key aspects: 1) Generalization Performance: Tool-MVR (Q) achieves its highest Pass Rate of 86.7% on G1-Tool scenarios with unseen tools from seen categories. 2) Less is More: Tool-MVR achieves these results with only 15,390 training examples, compared to ToolLLM's 73,423, demonstrating that high-quality instruction data matters more than data quantity. 3) Data Quality Impact: While recent approaches like TP-LLaMA and StepTool focus on advanced optimization techniques (DPO and PPO), their modest gains over SFT baselines stem from ToolBench's low-quality data significantly limiting models' basic capabilities. This highlights that improving instruction data quality for SFT is fundamental for enhancing tool learning capabilities.

Table 2: Pass Rate (%) comparison in StableToolBench. Bold values indicate the highest scores, underlined values indicate the second highest. L and Q represent using LLaMA-3.1-8B-Instruct and Qwen-2.5-7B-Instruct as backbone, respectively.

Type	Model	G1-Ins.	G1-Tool	G1-Cat.	G2-Ins.	G2-Cat.	G3-Ins.	Average
Closed	GPT-3.5	49.1	49.4	54.2	29.2	26.6	50.8	43.9
	GPT-4	74.2	72.8	60.8	63.2	66.1	75.4	68.5
Base	LLaMA-3.1-8B-Instruct	53.4	53.2	54.2	42.5	48.4	49.2	50.8
	Qwen-2.5-7B-Instruct	57.7	57.0	58.2	46.2	51.6	54.1	54.8
SFT	ToolLLM(L)	60.1	60.1	67.3	50.0	46.8	70.5	58.8
	ToolLLM(Q)	63.2	60.1	67.3	51.9	57.3	50.8	59.9
DPO	TP-LLaMA(L)	54.0	65.2	78.4	59.4	65.3	60.7	64.3
PPO	StepTool(L)	63.8	62.7	66.7	60.4	62.9	54.1	62.7
	StepTool(Q)	68.7	72.8	67.3	58.5	65.3	70.5	67.5
Ours	Tool-MVR(L)	79.1	82.9	79.1	**79.2**	79.0	**86.9**	80.5
	w/o Stage 2(L)	76.1	78.5	80.4	**79.2**	80.6	78.7	78.8
	w/o Stage 1&2(L)	60.7	65.2	60.8	57.5	49.2	70.5	60.1
	Tool-MVR(Q)	**83.4**	**86.7**	**83.0**	**79.2**	**85.5**	83.6	**83.8**
	w/o Stage 2(Q)	76.1	75.3	74.5	**79.2**	74.2	82.0	76.2
	w/o Stage 1&2(Q)	46.6	43.7	54.2	47.2	47.6	41.0	47.3

Table 3: Win Rate (%) comparison in StableToolBench.

Model	Reference	G1	G2	G3	Average
GPT-4	GPT-3.5	77.8	83.5	77.0	79.5
ToolLLM(L)	GPT-3.5	73.4	77.8	78.7	75.2
ToolLLM(Q)	GPT-3.5	72.6	78.3	60.7	73.3
TP-LLaMA(L)	GPT-3.5	75.5	80.9	78.7	77.4
StepTool(L)	GPT-3.5	75.1	80.0	70.5	76.2
StepTool(Q)	GPT-3.5	77.4	83.0	77.0	79.1
Tool-MVR(L)	GPT-3.5	84.8	88.7	86.9	86.1
Tool-MVR(Q)	GPT-3.5	**86.3**	**89.1**	**90.2**	**87.5**
Tool-MVR(L)	GPT-4	75.5	78.7	78.7	76.7
Tool-MVR(L)	ToolLLM	80.4	83.0	80.3	81.2
Tool-MVR(L)	StepTool	78.1	80.4	78.7	78.8

4.5.2 Win Rate Analysis. To further evaluate the quality of tool reasoning beyond simple task completion, we analyze models' Win Rate performance. As shown in Table 3, Tool-MVR (Q) and Tool-MVR (L) demonstrate superior capabilities with Win Rates of 87.5% and 86.1% against GPT-3.5 respectively. Notably, Tool-MVR (L) maintains strong performance even against GPT-4 (76.7%), ToolLLM (81.2%), and StepTool (78.8%). These results confirm that our systematic approach to developing tool capabilities - combining high-quality instruction data with reflection learning - not only improves task completion rates but also significantly enhances the quality and completeness of intermediate reasoning processes characteristic of System 2 Reasoning.

4.5.3 Ablation Study. To analyze how different components contribute to System 2 reasoning capabilities, we conduct ablation experiments with three configurations. As shown in Table 2, (1) Tool-MVR (L/Q) represents our complete framework incorporating both meta-verification and reflection learning. (2) "w/o Stage 2 (L/Q)", which uses only ToolBench-V without reflection learning, and (3) "w/o Stage 1&2 (L/Q)", which employs the base model with only APICallAgent inference. Results demonstrate the complementary nature of our two-stage approach. Specifically, (1) the significant performance improvement of "w/o Stage 2 (Q)" models over ToolLLM (Q) (76.2% vs. 59.9%) validates MAMV's effectiveness in developing tool planning and invocation abilities; (2) the further enhancement achieved by the complete Tool-MVR (Q) (83.8%) highlights EXPLORE's crucial role in strengthening tool reflection capabilities; (3) the substantially lower performance of "w/o Stage 1&2 (L)" (47.3%) confirms that both components are essential for comprehensive System 2 reasoning in tool learning.

4.6 Tool Reflection Results on RefineToolBench

To systematically evaluate models' tool reflection capabilities, we conduct experiments on RefineToolBench across three error scenarios: single API errors with available APIs ($I1$), single API errors requiring simulators ($I2$), and multi-API interaction errors ($I3$). As shown in Table 4, we assess both Error Recognition Rate (ERR) and Error Correction Rate (ECR) to measure models' ability to identify and rectify mistakes.

4.6.1 Limitations of Existing Approaches. Our analysis reveals a critical limitation in existing Tool-Augmented LLMs: their poor tool reflection capabilities. ToolLLM(L) and ToolLLM(Q) achieve only 7.8% and 9.1% Error Correction Rate (ECR) respectively. These models repeatedly make the same mistakes without adaptation when encountering errors. Notably, base models (LLaMA-3.1-8B-Instruct and Qwen-2.5-7B-Instruct) show better error handling (29.7% and 35.4% ECR), indicating that static imitation learning on ToolBench actually impairs models' inherent error recovery capabilities by focusing solely on successful examples while ignoring valuable learning signals from failures.

Table 4: Error Recognition Rate (%) and Correction Rate (%) across different error scenarios in RefineToolBench. Note: LLaMA-3.1-8B and Qwen-2.5-7B refer to instruction versions.

Method	Error Recognition				Error Correction			
	I1	I2	I3	Avg.	I1	I2	I3	Avg.
GPT-3.5	42.2	45.9	66.3	48.5	39.0	44.6	60.1	45.5
GPT-4	50.0	45.4	81.5	54.2	48.0	44.1	72.5	51.1
LLaMA-3.1-8B	30.8	37.1	24.7	32.3	28.8	34.8	20.2	29.7
Qwen-2.5-7B	46.2	48.2	28.1	43.5	36.3	41.0	21.3	35.4
ToolLLM (L)	6.1	4.1	34.8	10.9	4.9	2.8	24.2	7.8
ToolLLM (Q)	5.2	4.9	36.5	11.2	5.2	4.9	25.8	9.1
Tool-MVR (L)	59.9	57.7	88.8	64.6	51.2	52.3	**81.5**	57.6
w/o Stage 2 (L)	51.2	50.0	78.1	55.9	43.3	45.9	68.0	49.2
Tool-MVR (Q)	**63.7**	**60.3**	**90.4**	**67.5**	**53.2**	**54.4**	79.8	**58.9**
w/o Stage 2 (Q)	56.1	53.9	82.6	60.3	46.2	46.6	66.9	50.4

4.6.2 Performance Improvements. Tool-MVR demonstrates substantial improvements in reflection capabilities across all scenarios. Tool-MVR(Q) achieves an average ERR of 67.5% and ECR of 58.9%, significantly outperforming GPT-4 (54.2% ERR, 51.1% ECR), while Tool-MVR(L) attains 64.6% ERR and 57.6% ECR. These results validate the effectiveness of our dynamic "Error → Reflection → Correction" learning paradigm, which enables models to systematically learn from and adapt to execution feedback.

4.6.3 Ablation Analysis. Ablation studies further validate the importance of reflection learning. As shown in Table 4, removing Stage 2 (EXPLORE) leads to significant performance drops: ERR decreases from 67.5% to 60.3%, and ECR drops from 58.9% to 50.4%, demonstrating EXPLORE's crucial role in error handling. The benefits of reflection capabilities extend beyond RefineToolBench: on StableToolBench, the pass rate improves from 76.2% to 83.8% through enhanced tool reflection abilities from Stage 2 (EXPLORE).

4.6.4 Performance Analysis Across Different Scenarios. Tool-MVR shows varying performance across error scenarios: lower ECR on single-tool cases (53.2% and 54.4% for $I1$ and $I2$) compared to multi-tool scenarios (79.8% for $I3$). This pattern aligns with the training data distribution, as $I3$ cases from StableToolBench contain familiar multi-tool tasks, while single-tool scenarios present entirely novel challenges. Notably, despite this domain gap, Tool-MVR(Q) maintains strong generalization ability, achieving substantial improvements over GPT-4 (48.0%, 44.1%, 72.5%) across all scenarios.

4.7 Efficiency Analysis

As shown in Figure 4, Tool-MVR achieves superior efficiency in StableToolBench, requiring only 3.48-3.99 API calls per task across scenarios. This represents a 37.4% reduction compared to ToolLLM and 31.4% compared to GPT-4, while maintaining state-of-the-art performance. This significant reduction in API calls substantially decreases user interaction latency and reduces operational costs in commercial applications. Our MAMV pipeline achieves this efficiency improvement by significantly enhancing training data quality, reaching a minimality score of 81.9% compared to ToolBench's 54.9%, as shown in Table 1. High-quality training examples with minimal and accurate API calls enable Tool-MVR to develop efficient tool usage patterns, eliminating redundant operations during inference. This combination of improved efficiency and effectiveness demonstrates Tool-MVR's enhanced capabilities in deliberate tool planning and precise tool invocation.

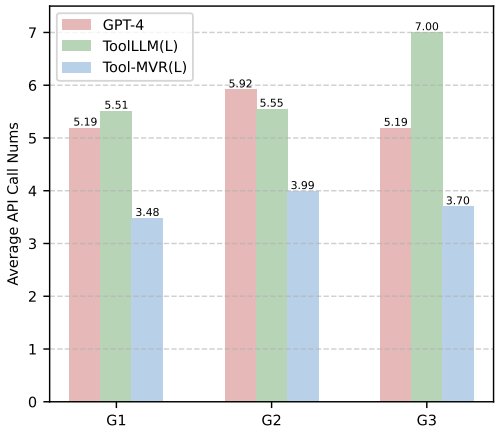

Figure 4: Average number of API calls required by different models across test scenarios in StableToolBench.

5 Conclusions

In this paper, we introduced Tool-MVR, which successfully enhanced open-source LLMs' System 2 reasoning capabilities and achieved state-of-the-art performance in tool learning through two key innovations. First, we developed MAMV to establish high-quality instruction data through meta-verification, addressing the critical limitation of instruction dataset quality and establishing strong foundations for deliberate tool planning and accurate tool invocation. Second, we proposed EXPLORE to enable systematic error correction through reflection learning, addressing a critical gap in existing models' reflection capabilities that resulted from static imitation learning. Our experiments demonstrated that Tool-MVR achieved state-of-the-art performance on StableToolBench, surpassing both open-source and closed-source baselines while requiring significantly fewer API calls, and exhibited strong generalization capabilities across unseen tools and scenarios. Additionally, on RefineToolBench - the first benchmark for evaluating tool reflection capabilities - Tool-MVR achieved the highest error recognition and correction rates, demonstrating unprecedented adaptive tool reflection capabilities.

Acknowledgments

This research was partially supported by the National Natural Science Foundation of China (Grants No.62477044), Anhui Provincial Natural Science Foundation (No. 2308085QF229), the Fundamental Research Funds for the Central Universities (No.WK2150110038). Zhenya Huang gratefully acknowledges the support of the Young Elite Scientists Sponsorship Program by CAST (No. 2024QNRC001).

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Erkan Başar, Xin Sun, Iris Hendrickx, Jan de Wit, Tibor Bosse, Gert-Jan De Bruijn, Jos A Bosch, and Emiel Krahmer. 2025. How Well Can Large Language Models Reflect? A Human Evaluation of LLM-generated Reflections for Motivational Interviewing Dialogues. In *Proceedings of the 31st International Conference on Computational Linguistics*. 1964–1982.

[3] Sijia Chen, Yibo Wang, Yi-Feng Wu, Qing-Guo Chen, Zhao Xu, Weihua Luo, Kaifu Zhang, and Lijun Zhang. 2024. Advancing Tool-Augmented Large Language Models: Integrating Insights from Errors in Inference Trees. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=ZIpdu0cHYu

[4] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *Transactions on Machine Learning Research* (2023).

[5] Zhi-Yuan Chen, Shiqi Shen, Guangyao Shen, Gong Zhi, Xu Chen, and Yankai Lin. 2024. Towards Tool Use Alignment of Large Language Models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. 1382–1400.

[6] Tri Dao. 2024. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. In *International Conference on Learning Representations (ICLR)*.

[7] Hao Ding, Ziwei Fan, Ingo Guehring, Gaurav Gupta, Wooseok Ha, Jun Huan, Linbo Liu, Behrooz Omidvar-Tehrani, Shiqi Wang, and Hao Zhou. 2024. Reasoning and planning with large language models in code development. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 6480–6490.

[8] Ling Ding, Chao Li, Di Jin, and Shifei Ding. 2024. Survey of spectral clustering based on graph theory. *Pattern Recognition* 151 (2024), 110366. doi:10.1016/j.patcog.2024.110366

[9] Yu Du, Fangyun Wei, and Hongyang Zhang. [n. d.]. AnyTool: Self-Reflective, Hierarchical Agents for Large-Scale API Calls. In *Forty-first International Conference on Machine Learning*.

[10] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[11] Jonathan St BT Evans. 2003. In two minds: dual-process accounts of reasoning. *Trends in cognitive sciences* 7, 10 (2003), 454–459.

[12] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 6491–6501.

[13] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*. PMLR, 10764–10799.

[14] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Nan Duan, Weizhu Chen, et al. [n. d.]. CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing. In *The Twelfth International Conference on Learning Representations*.

[15] Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2024. ToRA: A Tool-Integrated Reasoning Agent for Mathematical Problem Solving. In *The Twelfth International Conference on Learning Representations*.

[16] Zhicheng Guo, Sijie Cheng, Hao Wang, and et al. 2024. StableToolBench: Towards Stable Large-Scale Benchmarking on Tool Learning of Large Language Models. In *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics, 11143–11156. doi:10.18653/v1/2024.findings-acl.664

[17] Ruixin Hong, Xinyu Pang, and Changshui Zhang. 2024. Advances in reasoning by prompting large language models: A survey. *Cybernetics and Intelligence* (2024).

[18] Shadi Iskander, Sofia Tolmach, Ori Shapira, Nachshon Cohen, and Zohar Karnin. 2024. Quality Matters: Evaluating Synthetic Data for Tool-Using LLMs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. 4958–4976.

[19] Yixin Ji, Juntao Li, Hai Ye, Kaixin Wu, Jia Xu, Linjian Mo, and Min Zhang. 2025. Test-time Computing: from System-1 Thinking to System-2 Thinking. *arXiv preprint arXiv:2501.02497* (2025).

[20] Ziwei Ji, Tiezheng Yu, Yan Xu, Nayeon Lee, Etsuko Ishii, and Pascale Fung. 2023. Towards mitigating LLM hallucination via self reflection. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. 1827–1843.

[21] Nan Jiang, Xiaopeng Li, Shiqi Wang, Qiang Zhou, Soneya Binta Hossain, Baishakhi Ray, Varun Kumar, Xiaofei Ma, and Anoop Deoras. 2024. LeDex: Training LLMs to Better Self-Debug and Explain Code. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=d1XrZ4EINV

[22] Daniel Kahneman. 2011. *Thinking, fast and slow*. Macmillan.

[23] Ming Li, Yong Zhang, Zhitao Li, Jiuhai Chen, Lichang Chen, Ning Cheng, Jianzong Wang, Tianyi Zhou, and Jing Xiao. 2024. From Quantity to Quality: Boosting LLM Performance with Self-Guided Data Selection for Instruction Tuning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. 7595–7628.

[24] Jiayu Liu, Zhenya Huang, Qi Liu, Zhiyuan Ma, Chengxiang Zhai, and Enhong Chen. 2025. Knowledge-Centered Dual-Process Reasoning for Math Word Problems With Large Language Models. *IEEE Transactions on Knowledge and Data Engineering* 37, 6 (2025), 3457–3471. doi:10.1109/TKDE.2025.3556367

[25] Jiayu Liu, Zhenya Huang, Zhiyuan Ma, Qi Liu, Enhong Chen, Tianhuang Su, and Haifeng Liu. 2023. Guiding Mathematical Reasoning via Mastering Commonsense Formula Knowledge. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1477–1488.

[26] Jiayu Liu, Zhenya Huang, Tong Xiao, Jing Sha, Jinze Wu, Qi Liu, Shijin Wang, and Enhong Chen. 2024. SocraticLM: Exploring Socratic Personalized Teaching with Large Language Models. In *Advances in Neural Information Processing Systems*, Vol. 37.

[27] Weiwen Liu, Xu Huang, Xingshan Zeng, xinlong hao, Shuai Yu, Dexun Li, et al. 2025. ToolACE: Winning the Points of LLM Function Calling. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=8EB8k6DdCU

[28] Yilun Liu, Shimin Tao, Xiaofeng Zhao, Ming Zhu, Min Ma, et al. 2024. Coachlm: Automatic instruction revisions improve the data quality in llm instruction tuning. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 5184–5197.

[29] Zuxin Liu, Thai Quoc Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh R N, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. 2024. APIGen: Automated PIpeline for Generating Verifiable and Diverse Function-Calling Datasets. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. https://openreview.net/forum?id=Jfg3vw2bjx

[30] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2024. Chameleon: Plug-and-play compositional reasoning with large language models. *Advances in Neural Information Processing Systems* 36 (2024).

[31] Zhiyuan Ma, Zhenya Huang, Jiayu Liu, Minmao Wang, Hongke Zhao, and Xin Li. 2025. Automated Creation of Reusable and Diverse Toolsets for Enhancing LLM Reasoning. *Proceedings of the AAAI Conference on Artificial Intelligence* 39, 23 (Apr. 2025), 24821–24830. doi:10.1609/aaai.v39i23.34664

[32] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems* 36 (2024).

[33] OpenAI. 2022. ChatGPT: Optimizing Language Models for Dialogue. https://openai.com/blog/chatgpt. Accessed: 2024-01-30.

[34] Hongbin Pei, Yuheng Xiong, Pinghui Wang, Jing Tao, Jialun Liu, Huiqi Deng, Jie Ma, and Xiaohong Guan. 2024. Memory disagreement: A pseudo-labeling measure from training dynamics for semi-supervised graph learning. In *Proceedings of the ACM Web Conference 2024*. 434–445.

[35] Aske Plaat, Annie Wong, Suzan Verberne, Joost Broekens, Niki van Stein, and Thomas Back. 2024. Reasoning with large language models, a survey. *arXiv preprint arXiv:2407.11511* (2024).

[36] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, et al. 2024. Tool Learning with Foundation Models. *ACM Comput. Surv.* 57, 4, Article 101 (Dec. 2024), 40 pages. doi:10.1145/3704435

[37] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. [n. d.]. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *The Twelfth International Conference on Learning Representations*.

[38] Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. Tool Learning with Large Language Models: A Survey. *arXiv preprint arXiv:2405.17935* (2024).

[39] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In *Thirty-seventh Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=HPuSIXJaa9

[40] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems* 36 (2024).

[41] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–16.

[42] RapidAPI. 2024. RapidAPI Hub - The World's Largest API Hub. https://rapidapi.com/hub. Accessed: 2024-01-30.

[43] Xubin Ren, Jiabin Tang, Dawei Yin, Nitesh Chawla, and Chao Huang. 2024. A survey of large language models for graphs. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 6616–6626.
[44] Matthew Renze and Erhan Guven. 2024. Self-Reflection in LLM Agents: Effects on Problem-Solving Performance. *arXiv preprint arXiv:2405.06682* (2024).
[45] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems* 36 (2023), 68539–68551.
[46] Zhengliang Shi, Shen Gao, Xiuyi Chen, Yue Feng, Lingyong Yan, Haibo Shi, Dawei Yin, Pengjie Ren, Suzan Verberne, and Zhaochun Ren. 2024. Learning to Use Tools via Cooperative and Interactive Agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. 10642–10657.
[47] Zhengliang Shi, Shen Gao, Lingyong Yan, Yue Feng, Xiuyi Chen, Zhumin Chen, Dawei Yin, Suzan Verberne, and Zhaochun Ren. 2025. Tool learning in the wild: Empowering language models as automatic tool agents. In *Proceedings of the ACM on Web Conference 2025*. 2222–2237.
[48] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems* 36 (2024).
[49] Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and Error: Exploration-Based Trajectory Optimization of LLM Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 7584–7600. doi:10.18653/v1/2024.acl-long.409
[50] Keith E Stanovich and Richard F West. 2000. Individual differences in reasoning: Implications for the rationality debate? *Behavioral and brain sciences* 23, 5 (2000), 645–665.
[51] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. [n. d.]. Executable Code Actions Elicit Better LLM Agents. In *Forty-first International Conference on Machine Learning*.
[52] Zhiruo Wang, Zhoujun Cheng, Hao Zhu, Daniel Fried, and Graham Neubig. 2024. What are tools anyway? a survey from the language model perspective. *arXiv preprint arXiv:2403.15452* (2024).
[53] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Jiale Li, et al. [n. d.]. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.
[54] Qinzhuo Wu, Wei Liu, Jian Luan, and Bin Wang. 2024. ToolPlanner: A Tool Augmented LLM for Multi Granularity Instructions with Path Planning and Feedback. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. 18315–18339.
[55] Tong Xiao, Jiayu Liu, Zhenya Huang, Jinze Wu, Jing Sha, Shijin Wang, and Enhong Chen. 2024. Learning to Solve Geometry Problems via Simulating Human Dual-Reasoning Process. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, Kate Larson (Ed.). International Joint Conferences on Artificial Intelligence Organization, 6559–6568. doi:10.24963/ijcai.2024/725 Main Track.
[56] Weimin Xiong, Yifan Song, Xiutian Zhao, Wenhao Wu, Xun Wang, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. 2024. Watch Every Step! LLM Agent Learning via Iterative Step-level Process Refinement. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 1556–1572. doi:10.18653/v1/2024.emnlp-main.93
[57] Shangzi Xue, Zhenya Huang, Jiayu Liu, Xin Lin, Yuting Ning, Binbin Jin, Xin Li, and Qi Liu. 2024. Decompose, Analyze and Rethink: Solving Intricate Problems with Human-like Reasoning Cycle. In *Advances in Neural Information Processing Systems*, Vol. 37.
[58] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Zheng, et al. 2024. Qwen2. 5 Technical Report. *arXiv preprint arXiv:2412.15115* (2024).
[59] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. [n. d.]. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*.
[60] Mingjia Yin, Hao Wang, Wei Guo, Yong Liu, Suojuan Zhang, Sirui Zhao, Defu Lian, and Enhong Chen. 2024. Dataset Regeneration for Sequential Recommendation. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Barcelona, Spain) *(KDD '24)*. Association for Computing Machinery, New York, NY, USA, 3954–3965. doi:10.1145/3637528.3671841
[61] Yuanqing Yu, Zhefan Wang, Weizhi Ma, Zhicheng Guo, Jingtao Zhan, Shuai Wang, Chuhan Wu, Zhiqiang Guo, and Min Zhang. 2024. StepTool: A Step-grained Reinforcement Learning Framework for Tool Learning in LLMs. *arXiv preprint arXiv:2410.07745* (2024).
[62] Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Kan Ren, Dongsheng Li, and Deqing Yang. [n. d.]. EASYTOOL: Enhancing LLM-based Agents with Concise Tool Instruction. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.
[63] Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2024. MAmmoTH: Building Math Generalist Models through Hybrid Instruction Tuning. In *The Twelfth International Conference on Learning Representations*.
[64] Weixu Zhang, Yifei Wang, Yuanfeng Song, Victor Junqiu Wei, Yuxing Tian, Yiyan Qi, Jonathan H. Chan, Raymond Chi-Wing Wong, and Haiqin Yang. 2024. Natural Language Interfaces for Tabular Data Querying and Visualization: A Survey. *IEEE Trans. on Knowl. and Data Eng.* 36, 11 (Nov. 2024), 6699–6718. doi:10.1109/TKDE.2024.3400824
[65] Hongke Zhao, Likang Wu, Yuqing Shan, Zonghan Jin, Yuanpei Sui, Zipeng Liu, Nan Feng, Minqiang Li, and Wei Zhang. 2024. A comprehensive survey of large language models in management: Applications, challenges, and opportunities. *Challenges, and Opportunities (August 14, 2024)* (2024).
[66] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).
[67] Yuze Zhao, Zhenya Huang, Yixiao Ma, Rui Li, Kai Zhang, Hao Jiang, Qi Liu, Linbo Zhu, and Yu Su. 2024. RePair: Automated Program Repair with Process-based Feedback. In *Findings of the Association for Computational Linguistics ACL 2024*. 16415–16429.
[68] Yuze Zhao, Tianyun Ji, Wenjun Feng, Zhenya Huang, Qi Liu, Zhiding Liu, Yixiao Ma, Kai Zhang, and Enhong Chen. 2025. Unveiling the Magic of Code Reasoning through Hypothesis Decomposition and Amendment. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=kN25ggeq1J
[69] Zihuai Zhao, Wenqi Fan, Jiatong Li, Yunqing Liu, Mei, et al. 2024. Recommender Systems in the Era of Large Language Models (LLMs). *IEEE Transactions on Knowledge and Data Engineering* 36, 11 (2024), 6889–6907. doi:10.1109/TKDE.2024.3392335
[70] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. 2024. LlamaFactory: Unified Efficient Fine-Tuning of 100+ Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Yixin Cao, Yang Feng, and Deyi Xiong (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 400–410. doi:10.18653/v1/2024.acl-demos.38
[71] Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A Rossi, Somdeb Sarkhel, and Chao Zhang. [n. d.]. ToolChain*: Efficient Action Space Navigation in Large Language Models with A* Search. In *The Twelfth International Conference on Learning Representations*.

A Experimental Evaluation Details

We employ four comprehensive metrics to evaluate model performance, with standardized evaluation protocols implemented through GPT-4:

Pass Rate (PR) evaluates the overall task completion success by examining both final answers and execution chains. A Pass requires sufficient query resolution with successful API calls and verifiable information, while a Fail indicates execution errors or invalid solutions. Cases lacking sufficient validation data or complete reasoning processes are marked as Unsure:

$$PR = \frac{1}{N}\sum_{i=1}^{N} s_i, \text{ where } s_i = \begin{cases} 1.0 & \text{if Pass} \\ 0.5 & \text{if Unsure} \\ 0.0 & \text{if Fail} \end{cases} \tag{9}$$

Win Rate (WR) conducts pairwise comparisons between solutions using a comprehensive 100-point scoring system. The evaluation considers answer quality (60%) - assessing completeness of required information (20pts), accuracy with API responses (20pts), presentation clarity (10pts), and error documentation (10pts) - and execution efficiency (40%) - evaluating appropriate tool selection and usage (15pts), logical progression (15pts), and strategic optimization (10pts):

$$WR = \frac{\sum_{i=1}^{M} \mathbb{I}(C(A_i, B_i) = 0)}{M} \tag{10}$$

where $C(A_i, B_i)$ returns 0 if A_i is superior to B_i, with ties resolved in favor of higher answer quality scores.

Following the error handling evaluation protocol, we assess two complementary aspects:

Error Recognition Rate (ERR) measures the model's ability to explicitly identify API errors or invalid responses. A Pass requires clear error acknowledgment, while proceeding without recognizing errors results in a Fail:

$$ERR = \frac{\sum_{i=1}^{K} \mathbb{I}(r_i = \text{Pass})}{K} \tag{11}$$

Error Correction Rate (ECR) evaluates both error recognition and successful resolution, requiring valid subsequent API calls and proper error handling. This metric captures the complete error recovery process:

$$ECR = \frac{\sum_{i=1}^{K} \mathbb{I}(r_i = \text{Pass} \wedge c_i = \text{Pass})}{K} \tag{12}$$

Here, N denotes the total number of tasks, M the number of comparison pairs, and K the number of error cases. All evaluations require complete execution chains and API responses for comprehensive assessment.

B Implementation Details of Tool-MVR

B.1 APIOptAgent Details

APIOptAgent enhances API documentation through three components: Validator, Simulator, and DocRefiner. The Validator generates diverse API calls to obtain real execution observations, which serve as empirical evidence for API validation and documentation refinement. For invalid or unstable APIs, the Simulator creates standardized responses. The DocRefiner then improves documentation quality by leveraging both successful and failed API calls. The refined documentation follows OpenAI's function calling format, structured as a JSON object with standardized fields including "type", "function", "name", "description", and "parameters", ensuring compatibility with existing LLM tool-use frameworks.

This empirical refinement process effectively transforms vague documentation into accurate and actionable specifications based on observed API behaviors. This observation-driven approach significantly improves documentation quality by grounding specifications in empirical evidence, while maintaining strict adherence to OpenAI's standardized API documentation format.

B.2 QueryVerifyAgent Details

QueryVerifyAgent evaluates query solvability and quality through a systematic verification process based on four key rules: (1) queries containing invalid or nonsensical information, (2) queries missing essential API-required information, (3) queries unmappable to available tools, and (4) queries with sufficient valid information solvable by APIs. Each query is also assigned a quality score (1-10) considering solvability, semantic clarity, information completeness, reasoning difficulty, and tool compatibility. For instance, a query seeking transaction volume and webhook notifications might be deemed unsolvable if API setup requires missing transaction IDs.

Our analysis reveals substantial variations in query solvability across groups: G3 (50.7% of 14,361 queries), G1 (47.0% of 27,059 queries), and G2 (23.4% of 20,727 queries). To ensure high-quality instruction data, we retain only queries with quality scores between 8 and 10. This rigorous filtering process resulted in 11,776 queries from G1 (43.5% retention), 4,347 from G2 (21.0% retention), and 6,664 from G3 (46.4% retention).

The retained high-quality queries demonstrate clear intent, complete information, and strong alignment with available tools, ensuring our instruction dataset effectively develops models' tool planning capabilities by enabling accurate user intent understanding and systematic API selection.

B.3 APICallAgent Details

APICallAgent constructs high-quality API call trajectories through a two-stage process: trajectory construction and verification. The construction stage guides the agent to break down complex queries into sub-problems, solving them iteratively via Python function calls. This approach enables sophisticated operations like complex data processing, conditional branching, cascading calls, and adaptive parameter adjustment based on previous results, unlike traditional JSON formats. For each sub-problem, the agent provides its reasoning in a `<thought>` tag before making API calls in an `<execute>` block, ensuring transparent decision-making and flexible adaptation to API feedback.

The verification stage implements a triple-verification mechanism with three components. The Format Checker ensures the presence of a `<final_answer>` tag and filters out incomplete responses, ensuring each retained trajectory represents a complete solution attempt. The Semantic Checker evaluates whether the answer comprehensively resolves the query, verifying information completeness, logical consistency, and solution validity. Trajectories with partial or incorrect solutions are eliminated. The Execution Checker assesses the accuracy and necessity of each step, ensuring each API call contributes meaningfully to the final solution and eliminating redundant or trial-and-error steps, thereby ensuring trajectory optimality and proper error handling.

This rigorous verification process assigns each trajectory an answer status ("Pass", "Fail", or "Unsure") and validates step necessity. Only trajectories passing all verification stages are retained, ensuring the final dataset contains complete, accurate, and efficient solutions by eliminating common issues like incomplete answers, partial solutions, redundant API calls, and trial-and-error attempts.

Through this construction and verification pipeline, APICallAgent successfully generates reliable and efficient API call trajectories. This combination of Python-based flexibility and rigorous verification ensures that models learn to generate accurate parameter values and execute APIs in the correct format, leading to precise and efficient tool invocation during inference.

B.4 Reflection Details from EXPLORE

To systematically develop tool reflection capabilities characteristic of System 2 reasoning, we implement a structured reflection mechanism. This mechanism enables models to dynamically process execution feedback and adjust their reasoning strategies through three key steps: (1) **Error Analysis**: identifying specific issues in API selection or parameter generation from execution feedback. (2) **Reflection Generation**: developing correction strategies through deliberate analysis of API documentation and previous attempts. (3) **Action Planning**: formulating precise execution plans with corrected API calls.