LAM SIMULATOR: Advancing Data Generation for Large Action Models Trainings via Online Exploration and Feedback Simulation

Anonymous ACL submission

Abstract

Large Action Models (LAMs) for AI Agents offer incredible potential but face challenges due to the need for high-quality training data, especially for multi-steps tasks that involve planning, executing tool calls, and responding to feedback. To address these issues, we present LAM SIMULATOR, a comprehensive framework designed for online exploration of agentic tasks with high-quality feedback. Our framework features a dynamic task query generator, an extensive collection of tools, and an interactive environment where Large Language Model (LLM) Agents can call tools and receive real-time feedback. This setup enables LLM Agents to explore and solve tasks independently and potentially come up with multiple approaches to tackle any given task. Generated data are then used to create high-quality training datasets for LAMs.

Our research shows that LAM SIMULATOR enables LLM Agents to autonomously solve tasks while automating the creation of high-quality training data. Models trained with these self-generated datasets demonstrated significant performance gains, showing up to a 49.3% improvement over their own baselines. This was especially evident in experiments conducted with the ToolBench and CRMArena environments. The process requires minimal human input during dataset creation, highlighting the LAM SIMULATOR's efficiency and effectiveness in speeding up AI agents' development.

1 Introduction

Large Action Models (LAMs) (Zhang et al., 2024b; Xu et al., 2024; Liu et al., 2024b) are an advanced type of Large Language Model, specifically optimized for tool usage, reasoning, and function calling. Recent advancements have propelled their capabilities, making them integral to applications such as AI agents and task automation. While strong closed-world commercial models like Claude-3 (Anthropic, 2024) and GPT-4 (Achiam et al., 2023) can also perform complex agent tasks, LAMs benefit from specialized training for enhanced performance in agent applications and offer more open-source options and developments for the community. As use cases grow, the demand for more accurate models will continue to increase.

Current approaches for creating open-sourced LAMs include prompt engineering, incorporating additional contextual information into prompts, Supervised Fine-Tuning (SFT), Reinforcement Learning from Human Feedback (RLHF) (Ouyang et al., 2022), among others. Most of these methods, however, rely heavily on manual data curation, a process that is both time-consuming and expensive.

To address these challenges, utilizing LLM Agents to explore environments autonomously has emerged as a promising method to reduce the need for human labeling and annotation in agent model development. Recent studies, including those by ToolTalk (Farn and Shin, 2023), WebArena (Zhou et al., 2023), APIGen (Liu et al., 2024b), and Learn-by-Interact (Su et al., 2025), have demonstrated the ability to generate data for agent learning and evaluation through automated means. However, ToolTalk is limited to specific tasks that are curated or filtered by humans; WebArena offers a very limited action space within Web domain; APIGen and Learn-by-Interact, while showing considerable potential in generating agentic data, is limited to the heavy usage of LLMs to assess data quality, thus introduces a notable amount of uncertainty, which is an important issue to consider.

Given these limitations, we introduce LAM SIMULATOR, a comprehensive framework designed to enhance data generation for agent learning through exploration. As shown in Figure 1, LAM SIMULATOR employs a template-filling strategy to dynamically create queries. Users only need to develop a series of query templates and description for query parameters, along with logic to compute task an-

swers using these parameters. A large language model (LLM) then creates these variables in real-time and populates the templates to form real user queries. These generated queries are then given to LLM Agents, which explore solutions using a provided set of tools that might differ from those used to generate the ground-truth answers. As the agent tackles each task by making a series of function calls, LAM SIMULATOR provides immediate feedback. This enables a smooth interaction between the agent and its environment, allowing agents to freely explore problem-solving and rectify their actions through real-time feedback. Once the agent finishes its task, we apply thorough filtering to the generated trajectories, utilizing both the agent's data and pre-computed ground-truth answers. This process facilitates the creation of diverse datasets suitable for LAM training.

Our testing on well-known agentic benchmarks, including ToolBench (Qin et al., 2023) and CRMArena (Huang et al., 2025), demonstrates the high quality of data produced using LAM SIMULATOR. When fine-tuning top-performing models with data they generated themselves, we observed a pass rate increase of 4.1% for `gpt-4o` on ToolBench, and an 24.1% increase on CRMArena. In addition, models with lower performance showed significant improvement, with `mixtral-8x7b-inst` achieving over a 19.3% increase on ToolBench, and `gpt-4o-mini` improving by 49.3% on CRMArena. These results clearly demonstrate substantial performance enhancements resulting from exploration through LAM SIMULATOR, underscoring the effectiveness of our framework across diverse environments.

Our experiments demonstrate the remarkable effectiveness of the LAM SIMULATOR in improving model performance and identifying and addressing model weaknesses in an automated manner.

2 Related Work

With the rapid evolution of Large Language Models (LLMs), there has been a significant increase in their application to tool-use and function-calling scenarios. Enhancing the capabilities of LLMs (Achiam et al., 2023; Anthropic, 2024; Dubey et al., 2024; Zhang et al., 2024b) with external tools allows them to go beyond the limitations of their static parametric knowledge and text-based input-output interfaces. This extension enables them to access real-time information, leverage external reasoning systems, and perform meaningful actions in dynamic environments.

Recently, open-sourced research has focused increasingly on enhancing the efficiency of LLMs in tool-use contexts (Qin et al., 2023; Chen et al., 2023; Liu et al., 2024a; Zhang et al., 2024a), while also exploring various prompting and training strategies to improve their performance in agentic tasks. Prominent prompting techniques like Chain of Thought (CoT) (Wei et al., 2022), Reflection (Shinn et al., 2024), and ReACT (Yao et al., 2023) have garnered attention. While initial efforts centered on In-Context Learning (ICL)—where pre-trained LLMs were prompted with API specifications and tool-use examples—current approaches are increasingly incorporating fine-tuning methods to enhance model accuracy.

Moreover, popular agent environments such as Webshop (Yao et al., 2022), AgentBench (Liu et al., 2023), WebArena (Zhou et al., 2023), OSworld (Xie et al., 2024), AgentBoard (Ma et al., 2024), BFCL (Yan et al., 2024), and τ-bench (Yao et al., 2024) facilitate agent interactions and evaluations within various scenarios such as web navigation, shopping, games, and computer environments. Specifically, AgentBoard designs two multi-turn environments for tool query and operations, but they only contain 100 user queries in total and do not include real-time feedback. τ-bench simulates conversations between a user and a language agent, providing API tools and policy guidelines, but it only includes two domains: retail and airline.

The development of data generation pipelines for agentic learning has also seen significant advancements in recent years. Notable contributions include ToolBench (Qin et al., 2023), APIGen (Liu et al., 2024b), and the more recent Learn-by-Interact framework (Su et al., 2025). ToolBench has established an extensive collection of tasks and tools for agent learning, leveraging LLMs to generate these tasks. APIGen is a data generation pipeline with features similar to ToolBench, but it includes additional engines to verify generations for improved quality control. Learn-by-Interact innovates by utilizing a backward construction mechanism to align LLM-generated trajectories with instruction queries effectively.

Despite the reliance of the aforementioned works on LLMs for assessing the quality of query and trajectory pairs, our proposed framework, LAM SIMULATOR, diverges by eliminating LLMs from the quality assessment process. Instead, LAM SIMU-

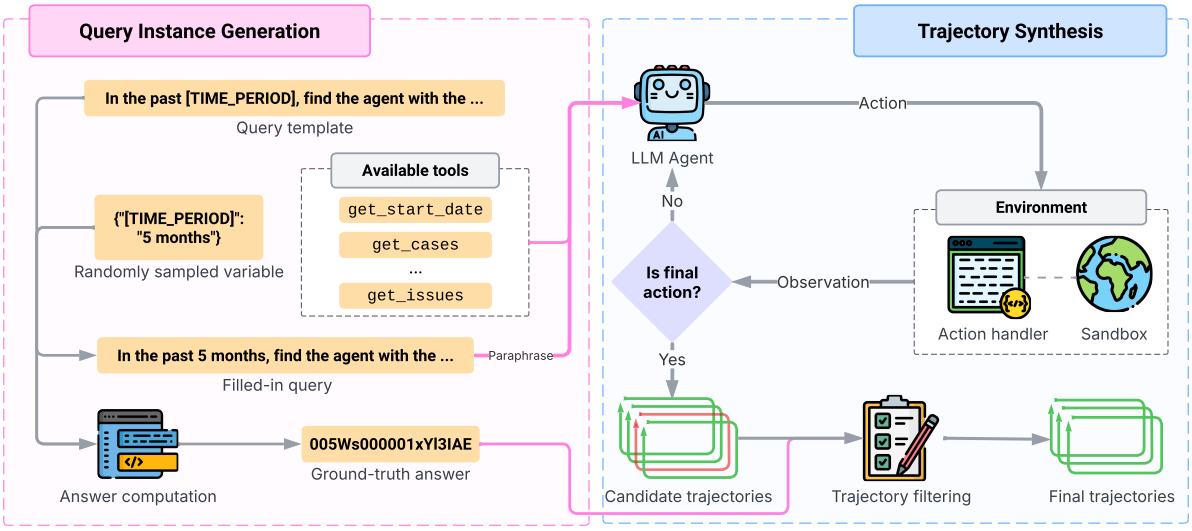

Figure 1: Overview of the LAM SIMULATOR. This figure illustrates the framework's components and their interactions, highlighting the simulator's capabilities in generating tool-use data, executing functions, and evaluating results.

Framework	Multi-turn	Open Action	Automated Data Gen	Programmatic Evals
ToolBench (Qin et al., 2023)	✔	✔	✔	✗
APIGen (Liu et al., 2024b)	✗	✔	✔	✗
Learn-by-Interact (Su et al., 2025)	✔	✔	✔	✗
LAM SIMULATOR (ours)	✔	✔	✔	✔

Table 1: Comparison of Prior Frameworks and Our LAM SIMULATOR. **Multi-turn** assesses support for multi-turn settings, **Open Action** assesses if agent's actions space are predefined or open, **Automated Data Gen** assesses automated training data generation capabilitiesm, and **Programmatic Evals** assesses if ALL evaluators in the framework are using a programmatic approach without using LLMs.

LATOR employs a programmatic approach to evaluating the quality of outputs from LLM Agents. This is achieved through dynamic task creation utilizing a template-filling strategy, thereby enhancing data quality without depending on LLMs as quality judges. Table 1 presents a comparison between our framework and other prominent systems developed for AI agent training.

3 LAM SIMULATOR

To enable LLM Agents to autonomously explore and enhance their problem-solving skills, we propose LAM SIMULATOR. We first construct query instances, each of which details the goals the LLM Agents should accomplish and the available tools (§3.2). Based on the given query instance, LLM Agents self-synthesize trajectories by iteratively interacting with the environment until the final state is reached (§3.3). Finally, the generated trajectories are filtered based on ground-truth answers, and subsequently used for training LLM Agents. Figure 1 shows an overview of our framework. We illustrate how each component works in the following subsections.

3.1 Preliminaries

The task defined by each query instance can be conceptualized as a Partially Observable Markov Decision Process (POMDP), defined by the tuple $(\mathcal{U}, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T})$. Here, $\mathcal{U}$ denotes the user query space, $\mathcal{S}$ represents the state space, $\mathcal{A}$ is the action space, $\mathcal{O}$ refers to the observation space, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is the state transition function.

3.2 Query Instance Generation

Query instances form the starting point for trajectory synthesis, comprising three elements: a user query $u \in \mathcal{U}$, a set of available tools $\mathcal{F}$, and a ground-truth answer y. The user query and tools initiate agents' iterative self-exploration, while the ground-truth answer is used to verify the validity of the resulting trajectory, as detailed in §3.3. Below, we illustrate the details of each element.

User Query Construction. User queries are natural-language questions that specify the objectives the agent must achieve. We begin constructing user queries by manually creating query templates with placeholders. It is sufficient to have one or multiple query templates for each objective, as the variation in language style can be later managed through the use of LLMs for paraphrasing. Once the query templates are in place, we utilize LLMs to sample values for these placeholders. The queries that result from filling in the placeholders are paraphrased and finalized as user queries.[1] A detailed example for a Query Instance is covered in Appendix A.1.

Ground-truth Answer Computation. In developing query templates, each template is linked to a sequence of tool usages aimed at deriving a ground-truth answer. This programmatic approach ensures the accurate production of answers. Here, the ground-truth-answer will later be used to compare with agent's answer during trajectory filtering step. Nonetheless, the tools used within this sequence are unlikely to align with those later made accessible to LLM Agents for exploration. This discrepancy is intentional, as we encourage agents to engage in exploratory problem-solving with different available tools, rather than merely replicating a predefined sequence of tool calls. Consequently, a successful strategy may either align with these hidden solution paths or comprise an alternative series of actions that achieve the same objective.

Available Tools. The available tools for each user query are dependent on the benchmark dataset. We describe the details of tools in §4.

3.3 Self-synthesized Trajectories

Iterative Self-exploration. Given a user query $u \in \mathcal{U}$, for each time step t at state $s_t \in \mathcal{S}$, the agent selects an action $a_t = (f, p) \in \mathcal{A}$ by choosing an appropriate tool from the available set of tools $f \in \mathcal{F}$ and corresponding tool-call arguments p. This tool is used to interact with the environment, resulting in an observation $o_t \in \mathcal{O}$ after the function is executed. This process continues iteratively until a final state is reached. The final state is achieved under one of two conditions: (1) the agent performs an action that returns a result to the user, such as the `submit` function in CRMArena (Huang et al., 2025); or (2) the trajectory of actions exceeds the predefined maximum number of steps.

To guarantee the proper execution of tool calls and enable agents to learn from significant feedback from the environment, we introduced an action handler. This handler checks the structure, syntax, and validity of the tool call to ensure responses are in the correct required format, and also to avoid hallucinations like fabricated tool names or malformatted tool-call arguments. Second, it retrieves the error message from the sandbox and sends it back to the agent for correction, while also maintaining a record of the error history for the trajectory filtering stage.

Trajectory Filtering. Trajectory filtering guarantees that the resulting paths are both valid and useful for training. We accomplish this by using string matching to compare the final response y' of each trajectory with the actual answer y. Any discrepancies ($y \neq y'$) result in exclusion from the selected set. Furthermore, to ensure the quality of our training dataset, we selectively include trajectories that meet the following criteria: (1) the LLM Agent completes the process without any errors, or (2) if any errors occur during tool usage, they are rectified in the subsequent action. For example, if the agent incorrectly applies a parameter to a tool at action a_t, we required that it is corrected in the next action, a_{t+1}.

Programmatic Evaluation. Our framework includes the Action Handler for assessing actions and Trajectory Filtering for monitoring trajectory quality. This ensures the selection of trajectories that demonstrate effective tool use and accuracy, enhancing our confidence in the data quality.

Agent Training. After the filtering process, we train our LLM Agents on these self-generated trajectories. This approach offers two advantages over using only the gold-standard trajectories from §3.2. First, training on agent-explored trajectories, which may include errors and corrections in subsequent iterations, allows the model to explicitly learn error recovery strategies. This is crucial for real-world deployments where unexpected inputs or tool states are common. Second, our approach allows the model to encounter and adapt to a wider range of scenarios, including potential interactions with previously unseen tools or alternative, valid solution paths. As empirically demonstrated in §5, exposing the model to this broader range

[1] In this work, we employ `gpt-4o` to generate appropriate placeholder values and paraphrase texts.

of interactions during training notably enhances its generalization abilities and its robustness in utilizing tools and completing agent-based tasks.

3.4 Generalizability

The proposed LAM SIMULATOR framework is easily generalizable to a wide range of agentic environments and tasks. To support a new set of tasks and environments, one only needs to design a set of query templates reflecting the specific task goals and establish the associated mappings from these templates to sequences of tool calls that yield the ground-truth answers. The capacity to rapidly adapt to novel tasks not only underscores the generalizability of LAM SIMULATOR but also highlights its potential as a universal tool for enhancing problem-solving capabilities of AI Agents.

4 Applications

LAM SIMULATOR is engineered to be seamlessly adaptable to a wide range of scenarios, showcasing its potential in advancing contemporary models. To demonstrate the effectiveness of LAM SIMULATOR, we supported its application to two prominent environments for agentic tasks: ToolBench (Qin et al., 2023) and CRMArena (Huang et al., 2025). ToolBench emphasizes generic tool-use capability, while CRMArena delves into complex and realistic Customer Relationship Management (CRM) scenarios. Our goal is to exhibit LLM Agent's self-improvement capability via explorative processes through LAM SIMULATOR.

4.1 ToolBench

4.1.1 Tools collection creation

ToolBench's Tools Collection: We leveraged ToolBench (Qin et al., 2023)'s extensive repository, which encompasses 16,464 REST APIs sourced from the RapidAPI Hub. Although this collection is notably extensive, we encountered numerous entries that were either non-functional or inadequately documented. To enhance the quality and reliability of our exploration, we conducted a thorough clean-up process as detailed in Appendix A.3. This refinement led us to a more manageable and useful collection comprising 3,420 effective tools.

In-house designed tools: Given that the ToolBench's tools collection necessitated the use of tools from various providers, availability issues could arise during exploration. To address these challenges and ensure stability in exploration, our team constructed a suite of 57 tools. These tools encompass critical domains, such as Data, Science, Entertainment, and Tool Usage. This strategic approach aims to reduce reliance on external providers, thereby enhancing the reliability and consistency of tool availability during exploration.

4.1.2 Query Instances

We devised 30 high-level tasks based on instances from the ToolBench training dataset, each addressing objectives such as retrieving movie details or housing property searches. For these tasks, we developed sequences of tool calls to generate solutions utilizing our in-house tools, as detailed in §4.1.1. This process yielded 400 unique query instances, with each instance consists of a paraphrased fill-in query with parameters produced by LLMs, a pre-determined ground-truth answer, and a set of tools for exploration. These tools include either those used to compute the solutions or alternative options, along with supplementary tools intended to challenge the decision-making capabilities of agents.

4.2 CRMArena

4.2.1 Tools collection creation

To facilitate exploration in solving the tasks in §4.2.2, relevant tools necessary for the four supported tasks were extracted, representing 15 out of the 25 tools available in CRMArena (Huang et al., 2025). The remaining 10 tools were deliberately left unmodified to rigorously test the system's adaptability in out-of-domain scenarios.

4.2.2 Query Instances

From the six finely crafted tasks derived from their framework: New Case Routing (NCR), Handle Time Understanding (HTU), Monthly Trend Analysis (MTA), Best Region Identification (BRI), Transfer Count Understanding (TCU), and Top Issue Identification (TII), we selected the first four tasks—NCR, HTU, MTA, and BRI—for exploration, leaving TCU and TII for rigorous out-of-domain testing. We follow the same procedure indicated in §4.1.2 to generate 400 query instances, all formatted consistently.

5 Experiments

With the integration of environments into LAM SIMULATOR detailed in §4, we conducted experiments to demonstrate our framework's effectiveness on ToolBench and CRMArena benchmarks.

5.1 Experiment Setups

5.1.1 Evaluation datasets and metrics

ToolBench Evaluation. To comprehensively assess ToolBench's performance, we employed three distinct test sets with 600 instances tailored to examine different scenarios. The first test set, **Unseen Instruction**, or `G1_inst`, is designed to measure how well the model performs when presented with new instructions for potentially familiar tools. It is important to highlight that due to the small number of tasks and tools selected for exploratory purposes, as detailed in §4.1.2, the likelihood of encountering similar instructions or tools in this set is exceedingly low. The second test set, **Unseen Tools**, or `G1_tool`, evaluates the model's ability to manage tools it has never encountered before. Finally, the **Unseen Tools & Unseen Categories**, or `G1_cat`, test set presents the most challenging scenario by testing the model on tasks from entirely new categories, requiring the use of unfamiliar tools.

CRMArena. In evaluating CRMArena, we utilized public test sets corresponding to the six tasks detailed in §4.2.2. Each task contains 130 entirely new instances to the exploration data. The NCR, HTU, MTA, and BRI tasks, while sharing similar scenarios or tools encountered during exploration, are distinguished by their novel use cases. In contrast, the TCU and TII tasks are entirely distinct from any previously explored domain, ensuring these tests are entirely out-of-domain.

5.1.2 Agent LLM

ToolBench. In our study, we focused on the two leading models in the ToolBench benchmark: `gpt-4o` (Achiam et al., 2023) and `xlam-8x7b` (Zhang et al., 2024b). For a more comprehensive analysis, we also included their more compact counterparts, `gpt-4o-mini` and `xlam-7b-r`, in our exploration and fine-tuning processes. To assess a broader range of performance capabilities, we additionally experimented with the lower-performing model, `mixtral-8x7b-inst` (Jiang et al., 2024).

CRMArena. In the CRMArena environment, which demands advanced planning and the complex use of tools, we chose the best-performing model, `gpt-4o`, along with its compact version, `gpt-4o-mini`, as our baseline LLM Agents for exploration and fine-tuning. Given these requirements, we observed that lower-performing models such as `mixtral-8x7b-inst` struggled to produce effective trajectories. Thus, constrained by time and resources, we opted not to experiment with models failing to meet the environment's high-performance demands.

5.1.3 Training Datasets

We incorporated the Query Instances from Section §4 for exploration. We let the Agent LLM to continuously explore the given tasks with temperature 1.0, and collected all trajectories that passed our evaluators, until it reached 500 trajectories for training. The filtered trajectories then directly being used to finetune the same base model.

5.2 Main Results and Discussions

5.2.1 ToolBench

The evaluation of the ToolBench datasets, as presented in Table 2, demonstrates substantial performance improvements in our fine-tuned models relative to their baselines. The `gpt-4o-ls` model shows a marked improvement, increasing its performance from 47.4% to 51.5% compared to the baseline `gpt-4o`. Similarly, the `xlam-8x7b-ls` model exhibits a notable enhancement, rising from 43.5% to 49.8% over its baseline, `xlam-8x7b-r`. Among compact models, `gpt-4o-mini-ls` and `xlam-7b-ls` achieved performance gains of 4.5% and 2.1%, respectively, when compared to their baselines. These improvements are particularly impressive because these models already rank among the best in the benchmark. Notably, low-performing models benefited even more from our approach. The `mixtral-8x7b-inst` model, initially achieving an 11.7% pass rate, improved significantly to 31.0% after fine-tuning to the `mixtral-8x7b-ls` version. This demonstrates the effectiveness of self-generated data in enhancing model performance.

When analyzing the performance across different test sets, we observed substantial gains in out-of-domain tasks. This is evident in the improvements on the G1_cat (Unseen Tools in Unseen Category) and G1_tool (Unseen Tool) datasets. For instance, top-performing model `xlam-8x7b-ls` gained 4% on G1_cat and 12.5% on G1_tool compared to its baseline. Similarly, the lower-performing `mixtral-8x7b-ls` model recorded more than double gains on both G1_cat and G1_tool over its baseline. These results highlight the efficacy of our framework in producing high-quality data for enhancing agentic learning.

Model Name	ALL	G1_inst	G1_cat	G1_tool
gpt-4o-ls	0.515	0.465	0.555	0.525
gpt-4o	0.474	0.437	0.519	0.466
xlam-8x7b-ls	0.498	0.440	0.530	0.525
xlam-8x7b-r	0.435	0.415	0.490	0.400
gpt-4o-mini-ls	0.497	0.475	0.540	0.475
gpt-4o-mini	0.452	0.415	0.505	0.435
xlam-7b-ls	0.413	0.400	0.460	0.380
xlam-7b-r	0.392	0.355	0.425	0.395
mixtral-8x7b-ls	0.310	0.280	0.385	0.265
mixtral-8x7b-inst	0.117	0.085	0.160	0.105

Table 2: Pass Rate (%) on three distinct ToolBench test sets. "ALL" denotes the average performance across all test sets. Models are categorized into baseline versions and their fine-tuned counterparts, with models trained using self-generated data through LAM SIMULATOR highlighted in cyan.

Model Name	ALL	NCR	HTU	MTI	BRI	TCU	TII
gpt-4o-ls	0.864	0.677	0.808	0.985	0.869	0.862	0.985
gpt-4o	0.623	0.600	0.477	0.277	0.592	0.815	0.977
gpt-4o-mini-ls	0.678	0.262	0.715	0.762	0.485	0.877	0.969
gpt-4o-mini	0.185	0.080	0.108	0.000	0.215	0.108	0.600

Table 3: Pass Rate (%) on six distinct CRMArena test sets. "ALL" denotes the average performance across all test sets. Models are categorized into baseline versions and their fine-tuned counterparts, with models trained using self-generated data through LAM SIMULATOR highlighted in cyan.

5.2.2 CRMArena

For CRMArena test sets, which require advanced problem-solving abilities in complex, realistic CRM environments, our fine-tuned model, gpt-4o-ls, demonstrates a marked improvement. It achieves an overall pass rate of 86.4%, representing a significant increase of 24.1% over its baseline version, gpt-4o, which scores 62.3%. This enhancement is even more pronounced in the weaker model, where our framework enhances gpt-4o-mini's performance nearly fourfold, elevating the accuracy from 18.5% to 67.8%. Such a transformation turns a previously inadequate model into one that is highly effective for these tasks.

Crucially, these performance improvements extend beyond just in-domain tasks to also significantly impact out-of-domain tasks. gpt-4o-mini-ls shows notable gains, achieving a 76.9% increase on the TCU task and a 36.9% increase on the TII task. These outcomes illustrate a substantial enhancement in the model's understanding of CRM tools and associated tasks.

5.3 Ablation Studies

5.3.1 Tools usage errors reduction

Besides discussing the effectiveness of LAM SIMULATOR in enhancing agents' overall problem-solving capabilities, we also examined its utility in improving agents' tool usage. Our study involved models xlam-8x7b-ls and xlam-7b-ls, compared against their baseline counterparts xlam-8x7b-r and xlam-7b-r. We randomly selected 200 states, each with preceding steps generated by one of these four models, tasked with solving tasks from the ToolBench test sets. Each model's subsequent actions were assessed for errors using our Action Handler.

Model Name	ALL	Structure	Toolname	Arguments
xlam-8x7b-ls	16.67	25	1	24
xlam-8x7b-r	25.33	30	6	40
xlam-7b-ls	13.67	17	12	12
xlam-7b-r	35.00	42	8	55

Table 4: Number of errors (lower is better) for actions generated from 200 sampled states across ToolBench test sets, comparing different models. Error types include structural errors (Structure), hallucinated tool calls (Toolname), and incorrect tool argument usage (Arguments). "ALL" indicates overall average number of errors. Models fine-tuned via LAM SIMULATOR are highlighted in cyan.

Our analysis focused on three layers of errors: 1) structural errors, where actions are unparseable; 2) toolname errors, where actions are parseable but the tool names are hallucinated; and 3) arguments errors, where both parsing and tool names are correct, but arguments are misused.

The results in Table 4 illustrates that LAM SIMULATOR substantially reduces errors, nearly halving them on average. For the compact model xlam-7b-r, substantial reductions were observed particularly in structural errors and incorrect tool argument errors. Though there's a slight increase in tool hallucination, it can be attributed to the baseline model primarily committing structural errors,

resulting in previously obscured toolname errors being revealed. Even among top-performing models, notable reductions in argument errors were observed. This suggests that LAM SIMULATOR effectively enhances agents' tool usage capabilities.

5.3.2 Impact of Core Monitoring Components

We conducted an ablation study to evaluate the impact of core monitoring components in our LAM SIMULATOR framework. Specifically, we sought to understand the contributions of **Action handler**, which monitors the LLM Agent's ability to perform correct actions through tool usage, and **Trajectory filtering**, which examines the agent's overall task-solving ability by comparing its trajectory against a ground-truth solution.

We used the `mixtral-8x7b-inst` as our baseline LLM Agent. We utilized LAM SIMULATOR with the LLM Agent as it interacted with tasks from ToolBench, as specified in §4.1.2, similar to the primary experiment. However, we made a critical modification by disabling the Action Handler when collecting training trajectories. That means, the Agent's actions, particularly those involving tool calls, were not monitored. We then fine-tuned the baseline model with the self-generated data and evaluated on the three ToolBench test sets used in the main experiment, enabling us to analyze the impact of **"Not monitor Action"**.

Similarly, we started with the same baseline LLM Agent and pass through LAM SIMULATOR for generating data while disabling ground-truth comparison within Trajectory filtering. In this approach, there was no quality control for trajectories misaligned with the task requirements. Subsequently, we fine-tuned the baseline model and performed evaluations on the same three ToolBench test sets. This allowed us to assess the impact of **"Not monitor Trajectory"**.

Our ablation study, depicted in Figure 2, underscores the essential role of both action-level and trajectory-level monitoring in our model's performance. The removal of action-level monitoring ("Not monitor Action" in Figure 2) results in a significant decline in pass rates across all test sets. This suggests that without proper regulation of tool usage, errors accumulate, which the model is unable to rectify autonomously. Similarly, the absence of trajectory-level monitoring ("Not monitor trajectory") hinders the overall effectiveness of the model by failing to ensure alignment with task requirements. This shortcoming is particularly evident in the out-of-domain G1_cat setting, where the pass rate drastically decreases from 38.5% to 19.0%. These findings underscores the importance of monitoring the actions and outcomes of LLM Agents in scenarios where we aim to apply self-improvement to agentic models, thus, clearly demonstrate the importance of incorporating all monitoring components in our LAM SIMULATOR.

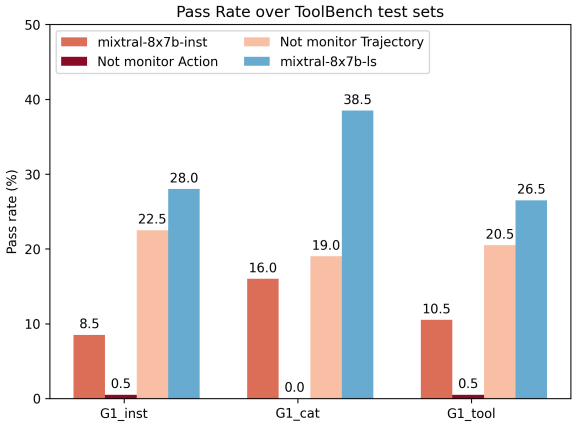

Figure 2: Ablation study on monitoring components across three ToolBench test sets. `mixtral-8x7b-inst` shows baseline model performance. "Not monitor Action" indicates performance when fine-tuning with a self-exploration dataset via LAM SIMULATOR without action monitoring, while "Not monitor Trajectory" shows results without trajectory monitoring. `mixtral-8x7b-ls` demonstrates performance with both action and trajectory monitoring enabled.

6 Conclusion

In this paper, we presented LAM SIMULATOR, a comprehensive framework designed to advance the development of Large Action Models (LAMs) by enabling self-learning through online exploration and automated feedback. Our system effectively addresses the limitations of traditional supervised learning and manual data curation, offering a scalable solution that enhances both agentic performance and training efficiency. LAM SIMULATOR provides real-time interactions, multi-turn task processing, and high-quality feedback, contributing to significant improvements in model training performance across various benchmarks, such as ToolBench and CRMArena, where models trained with self-generated data via LAM SIMULATOR gained a significant improvements and potentially outperformed other leading models. Our framework accelerates the learning and adaptation process of LAMs with minimal human intervention, demonstrating its potential as a pivotal tool for future research and development in AI agents.

7 Limitations

LAM SIMULATOR still has some limitations, its current implementation focuses on predefined tasks and tools, which may limit its adaptability in more dynamic or unstructured environments. In future work, we aim to expand the framework's generalization capabilities by incorporating a wider range of tasks and tool integrations, as well as exploring methods for better handling ambiguous or incomplete task specifications. Furthermore, we plan to investigate the scalability of the system in environments with more complex action spaces and interdependencies, pushing the boundaries of autonomous agent learning.

References

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*.

AI Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku. *Claude-3 Model Card*.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*.

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*.

Nicholas Farn and Richard Shin. 2023. Tooltalk: Evaluating tool-usage in a conversational setting. *arXiv preprint arXiv:2311.10775*.

Kung-Hsiang Huang, Akshara Prabhakar, Sidharth Dhawan, Yixin Mao, Huan Wang, Silvio Savarese, Caiming Xiong, Philippe Laban, and Chien-Sheng Wu. 2025. Crmarena: Understanding the capacity of llm agents to perform professional crm tasks in realistic environments. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

Zhiwei Liu, Weiran Yao, Jianguo Zhang, Liangwei Yang, Zuxin Liu, Juntao Tan, Prafulla K Choubey, Tian Lan, Jason Wu, Huan Wang, et al. 2024a. Agentlite: A lightweight library for building and advancing task-oriented llm agent system. *arXiv preprint arXiv:2402.15538*.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. 2024b. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *arXiv preprint*.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024. Agentboard: An analytical evaluation board of multi-turn llm agents. *arXiv preprint arXiv:2401.13178*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

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*.

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.

Hongjin Su, Ruoxi Sun, Jinsung Yoon, Pengcheng Yin, Tao Yu, and Sercan Ö. Arık. 2025. Learn-by-interact: A data-centric framework for self-adaptive agents in realistic environments. *Preprint*, arXiv:2501.10893.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*.

Yiheng Xu, Hongjin Su, Chen Xing, Boyu Mi, Qian Liu, Weijia Shi, Binyuan Hui, Fan Zhou, Yitao Liu, Tianbao Xie, et al. 2024. Lemur: Harmonizing natural language and code for language agents. *ICLR*.

Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. Berkeley function calling

leaderboard. https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. tau-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Jianguo Zhang, Tian Lan, Rithesh Murthy, Zhiwei Liu, Weiran Yao, Juntao Tan, Thai Hoang, Liangwei Yang, Yihao Feng, Zuxin Liu, et al. 2024a. Agentohana: Design unified data and training pipeline for effective agent learning. *arXiv preprint arXiv:2402.15506*.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, et al. 2024b. xlam: A family of large action models to empower ai agent systems. *arXiv preprint arXiv:2409.03215*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

A Appendix

A.1 Examples of Query Instance

Query template Here, we are providing an example for query templates. As we can see from Figure 3, a query template contains natural text portions and placeholders to-be-filled.

```
"query_template": "I've been looking up {
    movie_detail} about the movie {movie_name}. Fun
     fact: the set of {movie_name} was built inside
     a massive warehouse to create a surreal
    atmosphere!"
```

Figure 3: Example of Query Templates

Placeholders description We then provide example for query template's placeholders' descriptions (Figure 4).

```
"placeholders_metadata": {
    "movie_name": {
        "type": str,
        "description": "The name of the movie to
            search for.",
    },
    "movie_detail":  {
        "type": str,
        "description": "The detail of the movie to
            search for.",
    }
}
```

Figure 4: Example of Placeholders description

Generated Placeholders Given the query template and its placeholders' descriptions, we can use Large Language Models (LLMs) for generating dynamic values for the placeholders. An example of a generated placeholder is shown in Figure 5.

```
"placeholders": {
    "movie_name": "The Dark Knight",
    "movie_detail": "genres"
}
```

Figure 5: Example of generated Placeholders

Filled-in query With Query Template (3), and Generated Placeholders (5), we can fill in the value of the placeholder into the query template to create Filled-in query. An example is showned at 6.

Answer computation We also give an example of how we can compute answer for the generated query 6. As illustrated in Figure 7, we pre-define a solution path for any task that can be formed with the query template. This solution path will then being used by any generated task with the query template 3 for ground-truth answer computation.

```
"filled_in_query": "I've been looking up genres
    about the movie The Dark Knight. Fun fact: the
    set of The Dark Knight was built inside a
    massive warehouse to create a surreal
    atmosphere!"
```

Figure 6: Example of Filled-in Query with Query template 3 and generated Placeholders 5. Note that after this, the query can be further paraphrased with LLM for diversity purpose.

```
"solution_paths": [
    {
        "tool_call": "
            get_search_movie_for_movie_tools",
        "arguments": {
          "movie_name": null
        }
    },
    {
        "tool_call": "
            get_movie_details_for_movie_tools",
        "arguments": {
          "id": null
        }
    }
]
```

Figure 7: Example of a solution path for the task 6. The arguments would be searched among 1) placeholder values and 2) objects generated during execution. In this example, `movie_name` can be extracted directly from the placeholder value (The Dark Night), from 5, while `id` is a new field can be retrieved from the execution response of `get_search_movie`.

Available tools We then provide an example of a set of available tools provided to LLM Agents for exploration. As we can see from 8, the tools set does not include `get_search_movie_for_movie_tools`, but instead include an alternative version `search_movie_for_imdb`, which does the similar objective with `get_search_movie`, but it is a different tool with different way to use. In addition, there are extra tools provided here to, where the LLM Agent is expected to decide what is the right tool to use at a given time step.

A.2 Example of LLM Agent system prompt

We also include an example of a system prompt for LLM Agent that we derived from ToolBench's (Qin et al., 2023) and xLAM's (Zhang et al., 2024b) system prompt, as shown in 10.

```
"task_available_tools": [
  "get_movie_details_for_movie_tools",
  "search_movie_for_imdb",
  "get_movie_production_companies_for_movie_tools",
  "get_current_temp_for_weather_tools",
  "Finish"
]
```

Figure 8: Example set of available tools to be provided for LLM Agent for exploration. Note that this tools set does not include `get_search_movie_for_movie_tools`, but instead include an alternative version `search_movie_for_imdb`, which does the similar objective with `get_search_movie`, but it is a different tool with different way to use. In addition, there are extra tools provided here to, where the LLM Agent is expected to decide what is the right tool to use at a given time step.

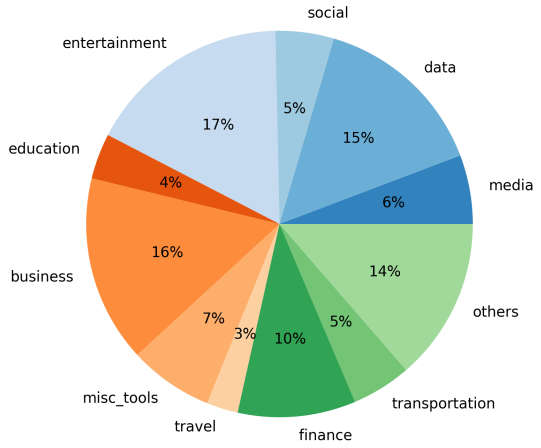

Figure 9: Tools distribution for ToolBench environment.

A.3 Clean-up Toolbench tools

We leveraged ToolLLM's extensive work under Apache-2.0 license, which collated over 16,464 REST APIs across 49 categories from RapidAPI Hub. Despite the breadth, the quality and documentation of these tools were inconsistent due to their mass collection approach, leading to many non-functional or poorly documented entries. To rectify this, critical elements such as tool names, parameters, execution code, and related metadata were extracted. Large language models (LLMs) played a crucial role in refining the tool descriptions and docstrings to ensure clarity and coherence. This process involved integrating necessary Python code components, conducting validations for code executability, and leveraging LLMs to assess quality. Moreover, rule-based techniques and LLM prompting were used to eliminate duplicate or similar tools, enhancing the collection's integrity. In Figure 9, we display the distribution of our tools collection for ToolBench after cleaning up and rewriting documentations.

Note that for now, we are only utilizing a small subset of this collection for exploration. In the future, we are going to scale up to try to use all of the processed tools.

B Licenses

Here, we discussed about the licenses of the artifacts we used in our work.

For tools (code logic), we used tools from ToolBench and CRMArena. ToolBench is under Apache-2.0 License, and CRMArena is under Creative Commons Attribution 4.0 License (CC BY).

For evaluation datasets, we also used the datasets from ToolBench and CRMArena, in which licenses are mentioned above.

For models we used for exploration (data generation), finetuning, and evaluating, we used `gpt-4o`, `gpt-4o-mini`, `xlam-8x7b-r`, `xlam-7b-r`, `mixtral-8x7b-inst`. Here:

- `gpt-4o` and `gpt-4o-mini` (Achiam et al., 2023) is under MIT License.
- `xlam-8x7b-r` and `xlam-7b-r` (Zhang et al., 2024b) is under Apache 2.0 License.
- `mixtral-8x7b-inst` (Jiang et al., 2024) is under Apache 2.0 License.

.

All of the licenses above enable us to perform research experiments.

C Experimental Detail

Our data generation and trainings of `mixtral-8x7b-inst` (56B parameters), `xlam-8x7b-r` (56B parameters), and `xlam-7b-r` (7B parameters) are performed with 4*H100s machines. Each of the exploration iteration to generate data is limited to 8 hours, and corresponding training time is limited by 4 hours. The hyperparameters of training all instances are at $5e-6$ for 3 epochs.

For the data generations and trainings on `gpt-4o` and `gpt-4o-mini`, we used OpenAI's endpoint with the same time limit. For trainings, we used default hyperparameters and number of epochs suggested by OpenAI, which is at between 1 and 2 for LR multiplier and for 3 epochs.

For evaluations, we configured the generative temperature to be 0.0. This allows us to have deterministic results for the presented ones in Section 5.

```
[BEGIN OF TASK INSTRUCTION]
You are an expert in agentic task. You will be given a task, and you can use many tools sequentially to
    solve the task. At each time step, you will call exactly 1 tool, and based on the environment feedback,
     you will be able to decide your next step. Keep repeating this action until you gather enough
    information to solve the task. By that time, call the special function "Finish" given to use to return
    the final answer in the exact format.

Remember:
1. MOST IMPORTANT, in your response of the "Finish" step, you MUST strictly follow the response format of
    what to be written inside "final_answer".
2. The state change is irreversible, you can't go back to one of the former state.
3. All the thought is short, at most in 5 sentences.
4. Your action must be calling one of the given tools (functions).
5. Your action input must be in json format, where action inputs must be realistic and from the user. Never
    generate any action input by yourself or copy the input description. Do not add unrelated parameters if
     not needed. Do not add optional parameters when it is not required or when these information is not
    needed.
6. You can do more then one trys, so if your plan is to continusly try some conditions, you can do one of
    the conditions per try.

Task description:
You should use functions to help handle the real time user querys. Remember:
1. ALWAYS call "Finish" function at the end of the task. And the final answer should contain enough
    information to show to the user.
[END OF TASK INSTRUCTION]
```

Figure 10: Example of system prompt for LLM Agent

D Others

When constructing this paper, we used `gpt-4o` (Achiam et al., 2023) for several paraphrasing.