ActionStudio: An Open-Source Lightweight Framework for Agentic Data and Training of Large Action Models

Anonymous ACL submission

Abstract

Large Action models are essential for enabling autonomous agents to perform complex tasks. However, training such models remains challenging due to the diversity of agent environments and the complexity of noisy agentic data. Existing infrastructure offers limited support for scalable, agent-specific fine-tuning and standardized agent data processing. We introduce **ActionStudio**, a lightweight and extensible data and training framework designed for large action models. ActionStudio unifies diverse agent trajectories using our proposed Unified Format 2.0, supports a range of training workflows with optimized multi-node distributed setup, and integrates robust preprocessing and real-time verification tools. ActionStudio demonstrates up to 9× higher throughput compared to existing agentic training frameworks, and our trained models yield top performances across public and realistic agent benchmarks. To support the broader research community, we open-source the ActionStudio framework and release *actionstudio-98k*, a curated dataset of 98k high-quality trajectories.[1]

1 Introduction

Action models are becoming increasingly critical for enabling autonomous agents to operate effectively across complex, multi-step tasks in diverse environments-from personal productivity assistants to real-world industrial automation systems. While recent open-source initiatives have advanced the action models development (Zeng et al., 2023; Xu et al., 2023; Yin et al., 2023; Zhang et al., 2024a,b), infrastructure for efficient agentic data processing and model training remains underdeveloped.

A central challenge lies in the nature of agentic training data, which often comprises long-horizon trajectories with tool interactions, observations, and user feedback originating from varied environments. While prior work has attempted to address data standardization (Zhang et al., 2024a,b), existing solutions usually rely on instruction-following templates that abstract away tool use (Yin et al., 2023; Xi et al., 2024) or adopt fixed rigid formats that may not generalize across tasks. Moreover, the data conversion and quality control processes required to turn raw agent trajectories into training-ready datasets are seldom open-sourced, which limits its reproducibility and cross-task transfer.

On the training side, general-purpose frameworks such as Transformers (Wolf et al., 2020) and LLAMA-Factory (Zheng et al., 2024) have played an important role in Large language model (LLM) development. However, these frameworks are primarily designed and optimized for standard LLM workflows, requiring substantial customization and heavy modifications to support agent-specific data and training. Even high-performing open-source models like xLAM (Zhang et al., 2024b) have not fully released their implementation code. This creates barriers for researchers and developers aiming to build agentic systems in real-world settings.

To address these challenges, we introduce **ActionStudio**, an end-to-end, open-source framework for data processing and training of large action models. Designed for production-scale use, ActionStudio integrates a novel *critique-and-filter* pipeline, deterministic rule-based checks, and a real-time verifier for filtering and visualizing agent trajectories. The resulting dataset, ACTIONSTUDIO-98K, comprises 98,000 high-quality trajectories formatted using our proposed Unified Format 2.0. The framework features an extensible backend supporting supervised fine-tuning (SFT) and preference-based learning (e.g., DPO), with flexible configurations including LoRA, quantization, and near-linear multi-node scalability. Models trained with ActionStudio achieve new state-of-the-art results on NexusRaven and the CRM Agent Benchmark,

[1] https://anonymous.4open.science/r/actionstudio-28AB

outperforming both open-source agent models and commercial systems.

The key contributions of our work are:

- We present an open-source, lightweight, and efficient agentic training framework, supporting flexible workflows such as LoRA, full fine-tuning, and multi-node distributed training. Our framework achieves up to 9× higher throughput compared to popular agentic training frameworks.
- We propose a *critique-and-filter* pipeline and a real-time data verifier that automate the ingestion, filtering, and conversion of diverse agent trajectories. We also release *actionstudio-98k*, a high-quality dataset of 97,755 trajectories spanning over 30,000 APIs and 300 domains, structured using our designed Unified Format 2.0 to facilitate agent research.
- We demonstrate ActionStudio's effectiveness across public and realistic industry agent benchmarks, showing its utility and practical value for real-world agent applications.

2 Related Work

2.1 Agent Data

While proprietary models often restrict data accessibility, the research community has significantly advanced open-source initiatives by releasing extensive agent datasets (Liu et al., 2023; Zeng et al., 2023; Tang et al., 2023; Li et al., 2023; Xi et al., 2024; Liu et al., 2024a). However, due to the inherent complexity and heterogeneous nature of agent trajectories across different environments, datasets often vary widely in format, creating substantial hurdles for industrial-scale model development. Recent efforts such as Lumos (Yin et al., 2023), AgentOhana (Zhang et al., 2024a), and xLAM (Zhang et al., 2024b) have aimed to standardize datasets into unified formats to reduce errors and simplify training. Nonetheless, these initiatives have not fully open-sourced the data conversion and automation pipelines, limiting widespread adoption and scalability.

2.2 Large Action Models

Beyond proprietary model APIs, significant progress has been made toward developing open-source large action models, specifically tailored for complex agent-oriented tasks (Xu et al., 2023; Qin et al., 2024; Zhang et al., 2024b; Liu et al., 2024b). These models have achieved impressive performance on benchmarks, showing the growing capability of open-source initiatives. While established frameworks like Transformers (Wolf et al., 2020) and LLAMA-Factory (Zheng et al., 2024) facilitate general language model training, specialized frameworks designed explicitly for fine-tuning LAMs remain limited. Our work contributes directly to addressing this gap by providing an efficient, lightweight training pipeline within ActionStudio, significantly reducing the complexity and resource requirements associated with training high-performing LAMs.

3 Framework

To support the development of high-performing large action models (LAMs), we present **ActionStudio**, a comprehensive framework for agentic data constructing and model training under a single, modular system. ActionStudio is composed of two core pipelines: a *data pipeline* for standardizing and preparing diverse agentic data sources, and a *training pipeline* for fine-tuning large language models on agent tasks at various scales.

3.1 Data Pipeline

The data pipeline in ActionStudio is designed to process diverse agentic data sources into standardized, training-ready formats. It includes four major components: data collection, format unification, quality filtering, and format conversion. This modular structure ensures the framework is extensible, scalable and compatible with a wide range of agent environments and models.

3.1.1 Data Collection

To support agentic model training, we compile a diverse set of high-quality datasets from multiple agent environments and domains, including but not limited to function calling, tool-use, and robotic agent trajectories. The datasets vary in structure and components, which poses significant challenges for LAM training.

3.1.2 Format Unification 2.0

Previous work (Zhang et al., 2024a,b) proposed a modular schema (Unified Format 1.0) to standardize agentic interaction data, with fields such as `task_instruction`, `query`, `tools`, and a list of `steps` capturing tool calls, intermediate thoughts, user feedback, and observations. While partially

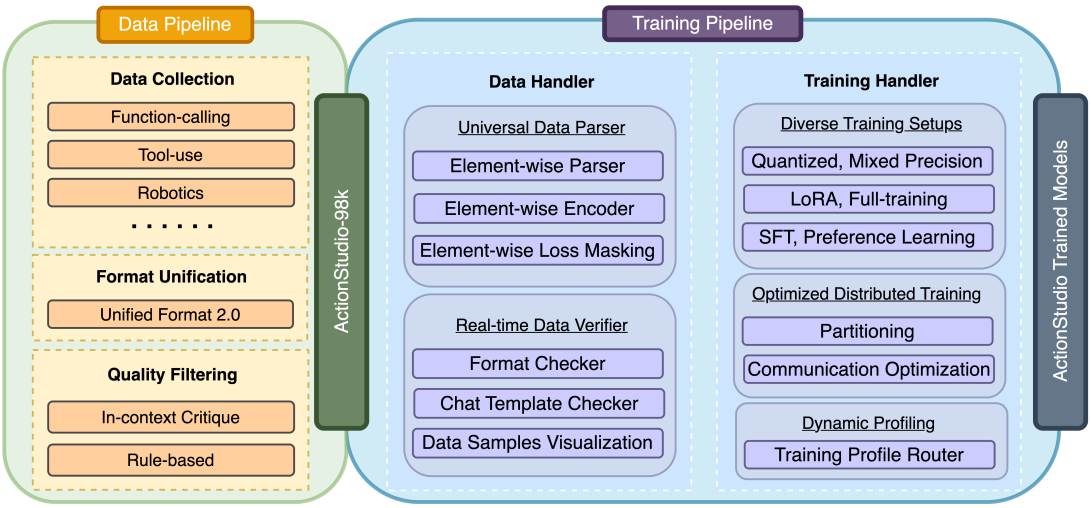

Figure 1: Framework of ActionStudio.

effective for general-purpose processing and augmentation, Unified Format 1.0 was not natively compatible with message-based interfaces expected by most open-source LLMs, resulting in non-trivial conversion and error fixing overhead for training and deployment. Examples of Unified Format 1.0 and its corresponding training prompt are shown in Figures 5 and 6 in the Appendix.

To address these limitations, we introduce **Unified Format 2.0**, a redesigned schema that natively aligns with modern chat-based LLM APIs and HuggingFace-style chat templates. Unlike prior work, Unified Format 2.0 is designed to support both *training, inference and evaluation workflows*-including Alpaca-style (Taori et al., 2023) (`input`, `output`) pairs, ShareGPT-style (Zhang et al., 2023) multi-turn dialogues, and general chat-based interaction formats. Its structure minimizes data transformation overhead, enabling direct use in common fine-tuning pipelines and runtime LLM interfaces.

Unified Format 2.0 introduces new abstractions that modularize agent trajectories into semantically grounded and model-compatible components, such as task instructions, available tools, and user-agent exchanges (including tool calls and execution results). This simplifies downstream formatting, promotes consistency across various data sources, and enables plug-and-play integration into both training and deployment systems. An example in Unified Format 2.0 and its converted chat format are shown in Figures 3 and 4 in the Appendix.

3.1.3 Data Quality Filtering

Prior work (Chen et al., 2023; Zhang et al., 2024b) has explored leveraging LLMs-like GPT-4-class models-for automatically evaluating and filtering trajectory quality, thereby reducing the reliance on manual annotation. These approaches typically score trajectories along dimensions such as correctness, hallucination, tool-use appropriateness, and overall response quality. However, we observe that off-the-shelf LLM evaluators tend to produce overly confident or median-biased scores and often fail to detect subtle or context-dependent hallucinations, and the issues are also noted in prior studies.

To address these limitations, we design a novel agent trajectory quality filtering method based on **In-Context Critique Filtering**. We augment the LLM evaluator with a small set of curated exemplars illustrating common failure cases and preferred critique behaviors. This simple yet effective in-context guidance leads to more fine-grained, human-aligned evaluations, particularly for ambiguous or borderline trajectories. In addition, we fine-tune open-source models using agent critique data to reduce reliance on commercial LLMs and further improve performance, making quality filtering more cost-effective and accessible. The critique pipeline is complemented by *rule-based* filtering pipeline that catch systematic errors (e.g., missing function calls, wrong function names or arguments, hallucinated agent actions).

Together, these components provide a scalable, high-precision pipeline for selecting training trajectories. Human verification in Section 4.6 demonstrates the effectiveness of the approach.

3.1.4 ActionStudio-98k

To facilitate open-source agent research, we release *actionstudio-98k*, a curated collection of 97,755 high-quality agent trajectories spanning diverse environments and task domains. The dataset includes 69,271 single-turn and 28,484 multi-turn trajectories, with multi-turn examples averaging 9 steps per trajectory. Filtered, critiqued, and corrected through the ActionStudio data pipeline, the trajectories are sourced from public agent datasets such as (Liu et al., 2023; Zhang et al., 2023, 2024b; Liu et al., 2024a; Xi et al., 2024; Guo et al., 2024b) and cover over 30,000 APIs across more than 300 domains. The dataset includes programmatic tool-use sequences, embodied agent interactions, and both single- and multi-turn tasks, all represented in our unified format 2.0.

3.2 Training Pipeline

Our framework is designed to deliver unparalleled versatility and efficiency in agentic training, ensuring it addresses diverse needs while optimizing scalability and performance. The following outlines how these are achieved.

3.2.1 Data Handler

Universal Data Parser. Training agentic models often involves working with highly diverse data structures, ranging from single-step responses to multi-step reasoning processes, multi-turn conversations, and various role configurations among users and agents (or groups of agents). To manage this complexity and maximize flexibility, our framework uses element-wise parsing and encoding. Each part of the conversation history is parsed as independently as possible, while still following the chat template. This approach simplifies the process of applying fine-grained loss masking and supports different training objectives. As a result, researchers have full control over these processes, making it easy to fine-tune agentic models for specific tasks. Ultimately, this design accelerates experimentation and speeds up model development.

Real-Time Data Verifier. Given the wide range of configurations and processing capabilities possible with our framework, it is critical to have robust mechanisms to validate data integrity throughout the training pipeline. The Real-Time Data Verifier keeps training pipeline under-control by dynamically running three checks: the Format Checker instantly flags data instances with missing fields or wrong structures, the Chat Template Checker ensures every conversation fits the provided Chat template, and the Data Samples Visualization presents "before-and-after" views of each data entry at every stage - before and after preprocessing, applying conversational templates, and encoding. By offering visibility into the transformations applied at each step, the verifier enables users to validate that their data complies with the expected format. This minimizes the risk of unexpected behaviors during training, ensuring a smoother development process and more reliable model performance.

3.2.2 Training Handler.

Comprehensive Support for Diverse Training Setups We provide an extensive range of functionalities to accommodate virtually any training pipeline for agentic models. From Supervised Instruction Fine-tuning to Human Preference Alignment, from lightweight approaches like quantized training and LoRA (Low-Rank Adaptation) (Hu et al., 2021) to full-scale training in mixed precision, our framework has it covered. This flexibility allows researchers and practitioners to seamlessly adapt to varying training requirements and computational resource constraints.

Highly Optimized Distributed Training at Scale. Our framework is purpose-built for highly efficient training of large-scale agentic models. To this end, heavy effort has been invested in optimizing performance for distributed settings. We have studied communication patterns in common Transformer architectures (Wolf et al., 2020), focusing on reducing inefficiencies in layer-to-layer interactions and communications between experts under Mixture of Experts settings. Additionally, we have optimized GPU-to-GPU communication within a single node as well as cross-node communication, reducing bottlenecks and enabling seamless scalability to industrial-scale training clusters. Users can also benefit from widely adopted parallelization and sharding strategies, such as those offered by DeepSpeed (Rasley et al., 2020), ensuring compatibility with industry-standard training practices. These enhancements ensure that our framework consistently delivers top-tier performance and efficiency, even in demanding distributed environments.

Dynamic Profiling for Model-Specific Optimizations. Recognizing the variety of architectures and model sizes in the agentic training space, we have implemented an initial dynamic profiling sys-

tem to enhance efficiency and improve ease-of-use. When provided with a model checkpoint, our framework dynamically routes it to an optimized configuration tailored for that architecture and model size under the current resource situation. This automation eliminates the need for labor-intensive tuning, allowing researchers to achieve higher efficiency with less manual effort.

4 Experiments

4.1 Model Training

Utilizing the ActionStudio framework, we conducted supervised fine-tuning on selected open-source models including Mistral series (Jiang et al., 2023, 2024) and Llama 3 series (Gratt. et al., 2024). To showcase ActionStudio's effectiveness across various sizes, we fine-tune from smaller-scale models such as Mistral variants to larger-scale models like Llama-3.1-70b-inst and Mixtral-8x22b-inst.

We set the sequence length between 8k and 16k, the batch size between 32 and 96 and the learning rate between 2e-6 and 5e-5, employing a cosine learning rate scheduler with 5% warm-up steps. Smaller models were fine-tuned on single NVIDIA H200 pod, while larger models were fine-tuned on both single and more H200 pods for comparison. Each H200 pod is equipped with 8 NVIDIA H200 GPUs, each having 141GB of memory.

4.2 Benchmarks

To demonstrate the effectiveness of ActionStudio, we selected NexusRaven and CRM Agent Bench.

NexusRaven (Srinivasan et al., 2023) provides a diverse benchmark for function calling, comprising 318 test examples across 65 distinct APIs. The dataset is curated through a structured pipeline that mines function definitions, docstrings, and execution contexts from open-source corpora. LLMs are then prompted to generate natural language queries, Chain-of-Thought (CoT) traces, and hard-negative function candidates to enhance evaluation difficulty. NexusRaven specifically evaluates model performance using precision, recall, and F1-score metrics for both function retrieval and argument inference tasks, offering a comprehensive assessment framework for function calling capabilities.

The CRM Agent Benchmark [2] is a proprietary evaluation developed by Salesforce. It assesses proficiency of AI models across critical and real CRM agent scenarios, focusing on accuracy in agent topic identification, accurate generation of function calls, and creation of contextually appropriate free-text responses. This benchmark emphasizes realistic business use cases by incorporating several hundred real CRM data points with expert assessments, providing valuable insights into the practical utility and reliability of language models for commercial deployment. Importantly, for all experiments, we fine-tune models exclusively on public datasets, ensuring that no customer data from any companies is utilized during training.

We set the model temperature to 0 during evaluations to ensure deterministic and replicable results.

4.3 NexusRaven

Table 1 shows the comparative performance of various models evaluated on NexusRaven. ActionStudio-trained models consistently outperform baseline and prominent commercial models. In particular, our fine-tuned ActionStudio-Mixtral-8x22b-inst-exp model achieves the highest overall F1-score (0.969), reflecting strong precision and recall, significantly surpassing commercial alternatives such as GPT-4 and GPT-4o. It also surpasses recently released large-scale models such as GPT-4.1 and Llama-4, both of which highlight strong agentic capabilities. Similarly, other fine-tuned models also exhibit robust performance. These results show the effectiveness of ActionStudio's pipeline in enhancing function-calling capabilities.

4.4 CRM Agent Bench

Table 2 illustrates our performance on the realistic industry Agent Benchmark. Our ActionStudio-Llama-3.3-70b-inst-exp model achieves the highest overall performance with an average accuracy of 0.87, surpassing the base Llama-3.3-70b-inst model (0.84). This reflects balanced capabilities across all three dimensions. Similarly, the ActionStudio-Llama-3.1-70b-inst-exp shows robust performance.

Furthermore, our fine-tuned ActionStudio-Mixtral-8x22b-inst-exp significantly outperforms its base, Mixtral-8x22b-inst, by 11%. Models such as ActionStudio-Mistral-7b-inst-exp also exhibit marked improvements of 13% compared to their instruct baselines, confirming that the pipeline benefits both large and small checkpoints. Additionally, our models are also ahead of strong agentic models such as o1-preview (0.85) and AgentOhana-8x22b-inst (0.80). These findings show ActionStudio's

[2] https://www.salesforceairesearch.com/crm-benchmark

Model	P_{api}	R_{api}	$F1_{api}$
ActionStudio-Llama-3.3-70b-inst-exp	0.950	0.953	<u>0.951</u>
ActionStudio-Llama-3.1-70b-inst-exp	0.940	0.943	<u>0.942</u>
ActionStudio-Mixtral-8x22b-inst-exp	0.969	0.969	**0.969**
ActionStudio-Mistral-latest-12b-inst-exp	0.953	0.956	<u>0.954</u>
ActionStudio-Mistral-7b-inst-exp	0.884	0.884	0.884
Llama-3.3-70b-inst (Gratt. et al., 2024)	0.917	0.934	0.925
Mistral-latest-12b-inst (Jiang et al., 2024)	0.906	0.940	0.923
Llama-3.1-70b-inst (Gratt. et al., 2024)	0.907	0.915	0.911
GPT-4o-latest (Hurst et al., 2024)	0.943	0.840	0.889
GPT-4.1-2025-04-14 (OpenAI, 2025)	0.841	0.846	0.843
DeepSeek-r1-671b (Guo et al., 2024a)	0.837	0.840	0.838
Mistral-7b-inst (Jiang et al., 2023)	0.814	0.827	0.821
Llama-4-Maverick-400b-inst (Meta, 2025)	0.796	0.796	0.796
Llama-4-Scout-109b-inst (Meta, 2025)	0.787	0.789	0.788
Mixtral-8x22b-inst (Jiang et al., 2024)	0.758	0.786	0.772
GPT-4 (Achiam et al., 2023)	0.894	0.635	0.743

Table 1: Performance comparison on **NexusRaven**. The best-performing result is indicated in **bold**, while the second and third-best results are marked with <u>underline</u>.

	Topic Acc	Function Call Acc	Free Text Acc	**Average Acc**
ActionStudio-Llama-3.3-70b-inst-exp	0.98	0.79	0.83	**0.87**
ActionStudio-Llama-3.1-70b-inst-exp	0.96	0.77	0.86	<u>0.86</u>
ActionStudio-Mixtral-8x22b-inst-exp	0.98	0.75	0.82	<u>0.85</u>
ActionStudio-Mistral-latest-12b-inst-exp	0.98	0.64	0.78	0.80
ActionStudio-Mistral-7b-inst-exp	0.95	0.49	0.74	0.73
DeepSeek-r1-671b (Guo et al., 2025)	0.82	0.83	0.94	<u>0.86</u>
o1-preview (Jaech et al., 2024)	0.98	0.75	0.81	<u>0.85</u>
Llama-3.3-70b-inst (Gratt. et al., 2024)	0.99	0.72	0.80	0.84
GPT-4-turbo (Achiam et al., 2023)	0.99	0.60	0.92	0.83
Llama-3.1-70b-inst (Gratt. et al., 2024)	1.0	0.62	0.82	0.81
AgentOhana-8x22b-inst (Zhang et al., 2024a)	0.90	0.66	0.84	0.80
Mixtral-8x22b-inst (Jiang et al., 2024)	0.98	0.65	0.60	0.74
GPT-4o-mini (Hurst et al., 2024)	0.94	0.42	0.81	0.72
Mistral-latest-12b-inst (Jiang et al., 2024)	0.96	0.18	0.70	0.61
Mistral-7b-inst (Jiang et al., 2023)	0.99	0.19	0.63	0.60

Table 2: Accuracy on the **CRM Agent Benchmark**. The best-performing result is indicated in **bold**, while the second and third-best results are marked with <u>underline</u>.

capability to train versatile and reliable models for practical and realistic agent scenarios.

4.5 Training Efficiency

Table 3 benchmarks raw training throughput (tokens / s) for three model sizes, Llama-3.1-8b, Mixtral-8x7b, and Mixtral-8x22b, under the four most commonly used fine-tuning regimes: (i) NF4-quantized LoRA (Q+LoRA), (ii) BF16 LoRA, (iii) full BF16 fine-tuning (FT) on a single pod (FT-1), and (iv) full FT on multiple pods (FT-2/FT-4). For the LoRA setting, we update the q_proj, k_proj, v_proj, and o_proj layers, with lora_r set to 32 and lora_a to 64. To contextualize these results, we replicated all configurations on the same cluster and GPU infrastructure for two competing systems that support agentic model trainings, AGENTOHANA (Zhang et al., 2024a) and LUMOS (Yin et al., 2023). The results from the comparison highlight the efficiency and capability of our framework.

Quantised LoRA. First, we can look at the throughput for quantized-based trainings (Q+LoRA). For the Llama-3.1-8B model, ActionStudio achieves a throughput of 79k tokens/s under Q+LoRA, outperforming AgentOhana (27.9k; 2.8x slower) and Lumos (8.4k; 9.4x slower). On the medium-sized Mixtral-8x7b, ActionStudio reaches 46k tokens/s, outpacing AgentOhana (25k; 1.9x slower) and Lumos (5k; 9.0x slower). For the larger configuration, Mixtral-8x22b, ActionStudio

Training Setup	Framework	Q+LoRA	LoRA	FT 1 pod	FT 2 pods	FT 4 pods
Llama-3.1 8b (BS/GPU=6, Seq=8k)	**ActionStudio (Ours)**	**79,306**	**76,766**	**64,179**	**125,097**	**224,192**
	AgentOhana	27,868 (-65%)	53,718 (-30%)	38,550 (-40%)	Not Sup.	Not Sup.
	Lumos	8,399 (-89%)	59,578 (-22%)	52,852 (-18%)	Not Sup.	Not Sup.
Mixtral-8x7b (BS/GPU=8, Seq=4k)	**ActionStudio (Ours)**	**46,193**	**47,404**	**33,661**	**71,858**	**137,146**
	AgentOhana	24,966 (-46%)	OOM	OOM	Not Sup.	Not Sup.
	Lumos	5,115 (-89%)	33,608 (-29%)	8,151 (-76%)	Not Sup.	Not Sup.
Mixtral-8x22b (BS/GPU=8, Seq=4k)	**ActionStudio (Ours)**	**14,703**	**14,654**	OOM	OOM	**44,438**
	AgentOhana	8,375 (-43%)	OOM	OOM	Not Sup.	Not Sup.
	Lumos	1,660 (-89%)	5,072 (-65%)	OOM	Not Sup.	Not Sup.

Table 3: Training throughput (tokens/s) for each setup under our **ActionStudio**, the **AgentOhana** and **Lumos** frameworks. "OOM" indicates an out-of-memory error. "Not Sup." denotes that the feature configuration is unsupported by the current version of the framework.

sustains 14.7k tokens/s, 1.8x and 8.9x faster than AgentOhana and Lumos, respectively.

BF16 LoRA. Under BF16-LoRA setting, ActionStudio also demonstrates a clear advantage. For Llama-3.1-8B, we obtain 76.8k tokens/s compared to AgentOhana (53.7k; 1.4x slower) and Lumos (59.6k; 1.3x slower). For Mixtral-8x7b, our system reaches 47.4k tokens/s, while AgentOhana encounters out-of-memory (OOM) errors; Lumos manages 33.6k tokens/s (1.4x slower). For Mixtral-8x22b, ActionStudio delivers a throughput of 14.7k tokens/s, while AgentOhana fails to complete training under this regime due to OOM and Lumos gets 5.1k tokens/s, which is 2.9x slower than us.

Full-model tuning. Next, for full model training, where much more parameters needed to be updated, and a more efficient usage of resources needed to be done in order to enable this. As we can see from the table, ActionStudio is the only framework able to fully update all model parameters across all three model sizes, with full fine-tuning on a single pod is only 16-30% slower than LoRA. For smaller models like Llama-3.1-8b and Mixtral-8x7b, AgentOhana and Lumos can also support training, but at noticeable slower performance than ActionStudio.

Multi-pod scalability. Finally, both AgentOhana and Lumos do not support multi-pods trainings, while in ActionStudio, the throughput remains to be linearly scaled for both 2 and 4 pods: Llama-8B throughput rises from 64 k (1 pod) to 125 k (2 pods) and 224 k (4 pods); similar scaling appears for both Mixtral variants. This not only demonstrate our capability to support larger model trainings with ActionStudio, but also at a very efficient speed.

Long-context support. We stress-tested ActionStudio on an extreme setting, Mixtral-8x22b with `BS/GPU=1`, `Seq=32k`, and observed *no* throughput loss. In fact, throughput *increased* slightly compared with the standard `BS/GPU=8`, `Seq=4k`: 15.2 k vs. 14.7 k tokens/s for Q+LoRA (+4 %), 15.4 k vs. 14.7 k for BF16-LoRA (+5 %), and 46.9 k vs. 44.4 k for FT-4 (+5 %). These results confirm that ActionStudio's memory scheduler scales seamlessly to much long token contexts, enabling efficient long-horizon agent training without manual tuning.

In summarization, across the three model sizes and four tuning regimes, ActionStudio is consistently the fastest-up to 9x quicker than AGENTOHANA and LUMOS-and the *only* framework that (i) completes every LoRA configuration without out-of-memory (OOM) failures, (ii) supports full-model tuning on multiple pods with linear speed scaling, and (iii) in our tests under ActionStudio framework, it remains similar throughput when Mixtral-8x22b is trained with a 32k-token context.

4.6 Ablation Study

Human Verifications. Figure 2 demonstrates that ActionStudio reliably removes low-quality trajectories across all datasets. To quantify this effect, we commissioned an independent third-party annotation company to audit a uniformly random sample of 150 trajectories, stratified across datasets. Annotators followed the four rubric items defined in §3.1.3-*correctness*, *hallucination*, *tool-use appropriateness*, and *overall response quality*-and then indicated whether ActionStudio's keep/remove decision was correct. Agreement between ActionStudio and human judges reached **85%**, roughly 15% over previous LLM-based agent data critiquing baseline. This shows the effectivenes of ActionStudio on handling complex trajectories. Besides, through detailed analysis, the residual 15% disagreement highlights future scopes such as inte-

Model / Data	Topic Acc	Function Call Ac	Free Text Acc	Average Acc
Mixtral-8x22b-inst	**0.98**	0.65	0.60	0.74
FT on ActionStudio (processed)	**0.98**	**0.75**	**0.82**	**0.85**
FT on Raw data	**0.98**	0.44	0.75	0.72
FT on AgentOhana (processed)	0.90	0.66	0.84	0.80

Table 4: Accuracy on the **CRM Agent Benchmark**. *FT* denotes full-tuning; **ActionStudio** and **AgentOhana** are two different training frameworks (each with its own data processing pipeline). Best score per column is **bold**.

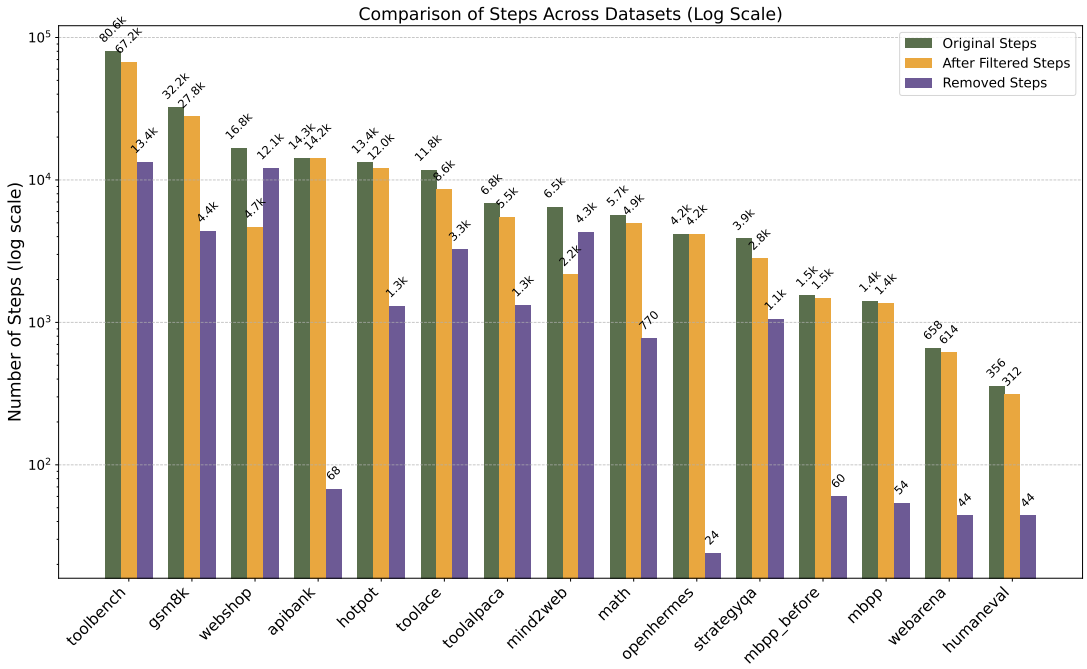

Figure 2: ActionStudio's critique–and–filter pipeline on each dataset. *Original* shows the total number of agent steps, *Removed* the steps discarded by filtering, and *Retained* the remaining steps.

grating multi-turn self-consistency checks, domain-specific hallucination detectors, or adaptive thresholds that evolve with new agent behaviors-to push reliability even closer to human parity.

Ablation on Data Pipelines. Table 4 examines how data quality influences full-tuning (FT) of the Mixtral-8x22b-inst backbone. We compare four settings that differ only in the data processing pipeline during FT: the untuned baseline, FT on *ActionStudio-processed* trajectories, FT on *raw* agent data trajectories, and FT on *AgentOhana-processed* trajectories.

ActionStudio preprocessing yields the largest gains. Applying ActionStudio's critique-and-filter pipeline lifts *function-call* accuracy from 0.65 to **0.75** (+10), *free-text* accuracy from 0.60 to **0.82** (+22), and the overall score from 0.74 to **0.85**.

Raw data degrades performance. Naively fine-tuning on unfiltered logs drives function-call accuracy down to 0.44 and reduces the aggregate metric to 0.72-*worse* than baseline-illustrating noisy agent trajectories can overwhelm the learning signal.

AgentOhana preprocessing is helpful but less effective. Cleaning the same corpus with AgentOhana's agent data pipeline partially recovers performance (0.66 / 0.84 / 0.80) yet still lags behind ActionStudio on every metric, implying that ActionStudio data pipeline could better target the error modes of complicated agent trajectories.

5 Conclusion

We introduced ActionStudio, a lightweight and flexible framework for training large action models. By integrating structured data preprocessing, advanced fine-tuning, and distributed training, ActionStudio simplifies agentic model development. Evaluations on NexusRaven and the CRM Agent Benchmark, which specifically reflects realistic industry agent scenarios, demonstrated its effectiveness and practical value for robust agentic model solutions.

6 Limitations

While ActionStudio provides a practical and extensible framework for developing robust large action models for complex agent scenarios, a few limitations remain. The current implementation primarily focuses on text-based and function-calling scenarios, with future support planned for multimodal and embodied environments. Additionally, although the framework includes standardized formats and a robust data processing pipeline, model effectiveness still depends on the quality and diversity of input datasets, which can vary across use cases. Despite these challenges, ActionStudio significantly lowers the barrier to LAM development and offers a scalable foundation for further innovation in industry and research.

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*.

Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, et al. 2023. Alpagasus: Training a better alpaca with fewer data. *arXiv preprint arXiv:2307.08701*.

Aaron Gratt., Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

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

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y Wu, YK Li, et al. 2024a. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024b. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. *Preprint*, arXiv:2403.07714.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

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*.

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

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

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*.

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*.

Meta. 2025. https://ai.meta.com/blog/llama-4-multimodal-intelligence/.

OpenAI. 2025. https://openai.com/index/gpt-4-1/.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. *ICLR*.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3505–3506.

Venkat Krishna Srinivasan, Zhen Dong, Banghua Zhu, Brian Yu, Damon Mosk-Aoyama, Kurt Keutzer, Jiantao Jiao, and Jian Zhang. 2023. Nexusraven: a commercially-permissive language model for function calling. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Alpaca: A strong, replicable instruction-following model. *Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html*, 3(6):7.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, et al. 2024. Agentgym: Evolving large language model-based agents across diverse environments. *arXiv preprint arXiv:2406.04151*.

Yiheng Xu, Hongjin Su, Chen Xing, Boyu Mi, Qian Liu, Weijia Shi, Binyuan Hui, Fan Zhou, Yitao Liu, Tianbao Xie, et al. 2023. Lemur: Harmonizing natural language and code for language agents. *arXiv preprint arXiv:2310.06830*.

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. 2023. Lumos: Learning Agents with Unified Data, Modular Design, and Open-Source LLMs. *arXiv preprint arXiv:2311.05657*.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2023. Agenttuning: Enabling generalized agent abilities for llms. *arXiv preprint arXiv:2310.12823*.

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*.

Jianguo Zhang, Kun Qian, Zhiwei Liu, Shelby Heinecke, Rui Meng, Ye Liu, Zhou Yu, Silvio Savarese, and Caiming Xiong. 2023. Dialogstudio: Towards richest and most diverse unified dataset collection for conversational ai. *arXiv preprint arXiv:2307.10172*.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. *arXiv preprint arXiv:2403.13372*.

A Appendix

A.1 Unified Format 2.0

To support diverse agentic tasks in a model-friendly way, we introduce Unified Format 2.0, an upgraded version of the format used in prior work. While Unified Format 1.0 was designed to modularize agentic trajectories for general-purpose processing, it lacked alignment with the message-passing format expected by modern LLM APIs. Unified Format 2.0 is designed to be natively compatible with modern chat-based LLM APIs and HuggingFace-style chat templates, significantly reducing the effort required to convert raw data into model-ready training samples. An example of a trajectory in Unified Format 2.0 is shown in Figure 3.

Unified Format 1.0 (Zhang et al., 2024b) introduced a modular schema for representing agentic interaction data, including fields such as `task_instruction`, `query`, `tools`, and a list of `steps` that capture tool calls, intermediate thoughts, user follow-ups, and observations. While effective for general processing and augmentation, this format was not directly aligned with the message-based structure expected by most open-source LLMs, requiring non-trivial conversion logic to adapt the data for training. An example showcasing Unified Format 1.0 is presented in Figure 5.

In contrast, Unified Format 2.0 structure is based on the conversational schema commonly used in APIs like OpenAI and HuggingFace. It replaces the `steps` field with a `conversation` list, where each entry is a message with a specific role (e.g., `system`, `user`, `assistant`, or `tool`). Tool calls are now explicitly represented inside assistant messages using a `tool_calls` field, and tool responses are mapped to messages with the role `tool`, linking back via a `tool_call_id`. This structure is more compatible with LLM APIs and chat templates, which removes the need for custom scripts to flatten or restructure training samples. Figure 4 demonstrates that with Unified Format 2.0, the training format can be flexibly changed by applying the corresponding chat template from the tokenizer. In contrary, Figure 6 shows the fixed training format from Unified Format 1.0, which remains unchanged across different models, requiring substantial effort in data format conversion for both training phase and deployment phase.

A.2 Evaluation Details for NexusRaven

To evaluate function-calling performance on the NexusRaven benchmark, we follow a two-step process: (1) parsing tool calls from the model's output, and (2) computing precision, recall, and F1 scores by comparing the parsed predictions to ground-truth annotations.

Tool Call Parsing. NexusRaven includes a custom parser designed to extract structured tool call information from raw model outputs. Since different models may follow varying output formats (e.g., enclosing tool calls in special tags, including JSON blocks with markdown fencing, or appending irrelevant tokens), the parser applies a series of heuristics to sanitize the output and isolate the tool call content. It then parses the cleaned text into a structured format containing: 1) the function/tool name, 2) the arguments as a dictionary of key-value pairs, and 3) an optional tool call ID. The parser handles edge cases such as missing fields, extraneous formatting tags (e.g., `<think>`, `<tool_call>`), and malformed JSON.

Metric Computation. After extracting the predicted and ground-truth tool calls, we compute evaluation metrics at the API level level, which includes the Precision (P_api), Recall (R_api), and F1 Score (F1_api). P_api and R_api calculate the proportion of predicted tool calls whose function name matches one in the ground-truth, and the proportion of ground-truth tool calls that were successfully predicted, respectively. The F1_api is the harmonic mean of API precision and recall. This enables consistent and scalable comparison of function-calling capabilities across models, while maintaining tolerance to minor formatting differences.

```
{
    "unique_trajectory_id": "id",
    "task_instruction": "...",
    "tools": [
        {
            "type": "function",
            "function": {
                "name": "get_sleep_stats",
                "description": "Get the user's sleep statistics for a specified time period.",
                "parameters": {
                    "type": "object",
                    "properties": {
                        "user_id": {
                            "type": "string",
                            "description": "Unique identifier of the user whose sleep statistics will be
     retrieved.",
                        },
                    },
                    "required": [
                        "user_id",
                    ]
                }
            }
        },
    ],
    "conversation": [
        {
            "role": "user",
            "content": "I would like to get my sleep statistics from last night."
        },
        {
            "role": "assistant",
            "content": "",
            "tool_calls": [
                {
                    "type": "function",
                    "function": {
                        "name": "get_sleep_stats",
                        "arguments": {
                            "user_id": "1234",
                        }
                    },
                    "id": "808380806"
                }
            ]
        },
        {
            "role": "tool",
            "name": "get_sleep_stats",
            "content": {
                "data": {
                    "message": "..."
                }
            },
            "tool_call_id": "808380806"
        },
        {
            "role": "assistant",
            "content": "Your sleep statistics from last night has been retrived successfully."
        }
    ]
}
```

Figure 3: Unified format 2.0 for function calling data.

Prompt:

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>

Environment: ipython
Cutting Knowledge Date: December 2023
Today Date: 26 Jul 2024

<|eot_id|><|start_header_id|>user<|end_header_id|>

Given the following functions, please respond with a JSON for a function call
with its proper arguments that best answers the given prompt.

Respond in the format
{"name": function name,
"arguments": dictionary of argument name and its value}.
Do not use variables.

{
    "type": "function",

    "function": {
        "name": "get_sleep_stats",
        "description": "Get the user's sleep statistics
        for a specified time period.",
        "parameters": {
            "type": "object",
            "properties": {
                "user_id": {
                    "type": "string",
                    "description": "Unique identifier of the user whose sleep
                    statistics will be retrieved."
                }
            },
            "required": [
                "user_id"
            ]
        }
    }
}


I would like to get my sleep statistics from last night.<|eot_id|>
```

Output:

```
[{"name": "get_sleep_stats", "arguments": {"user_id": "1234"}}]
```

Figure 4: Example prompt and output for function-calling from unified format 2.0, by applying Llama-3.1-70B-Instruct chat template.

```
{
    "unique_trajectory_id": "id",
    "task_instruction": "...",
    "few_shot_examples": [],
    "query": "The task or the question that the user provides.",
    "tools": [
        {
            "name": "api_name1",
            "description": "description of this api",
            "parameters": {
                "param1": {
                    "type": "string",
                    "description": "",
                },
            }
        },
    ],
    "steps": [
        {
            "thought": "thinking and/or planning process",
            "tool_calls": [
                {
                    "name": "api_name1",
                    "arguments": {
                        "argument1": "xxx.",
                        "argument2": "xxx"
                    }
                }
            ],
            "step_id": 1,
            "next_observation": "observations or feedbacks from the environment/APIs after execution
     function."
            "user_input": "User follow up input at this turn if any."
        },
    ],
}
```

Figure 5: Unified format 1.0 for function calling data.

Prompt:

```
[BEGIN OF TASK INSTRUCTION]
Based on the previous context and API request history, generate an API
request or a response as an AI assistant.
[END OF TASK INSTRUCTION]

[BEGIN OF AVAILABLE TOOLS]
[
    {
        "name": "get_fire_info",
        "description": "Query the latest wildfire information",
        "parameters": {
            "location": {
                "type": "string",
                "description": "Location of the wildfire.",
                "required": true,
            },
            "radius": {
                "type": "number",

                "description": "The radius (in miles) around the location.",
            }
        },
    },...
]
[END OF AVAILABLE TOOLS]

[BEGIN OF FORMAT INSTRUCTION]
Your output should be in the JSON format, which specifies a list of
function calls. The example format is as follows. Please make sure the
parameter type is correct. If no function call is needed, please make
tool_calls an empty list "[]".
{"thought": "the thought process, or an empty string", "tool_calls":
[{"name": "api_name1", "arguments": {"argument1": "value1", "argument2":
"value2"}}]}
[END OF FORMAT INSTRUCTION]

[BEGIN OF QUERY]
Can you give me the latest information on the wildfires occurring in California?
[END OF QUERY]


[BEGIN OF HISTORY STEPS]
[
    {
        "thought": "Sure, what is the radius (in miles) around the location of
        the wildfire?",
        "tool_calls": [],
        "step_id": 1,
        "next_observation": "",
        "user_input": "User: Let me think... 50 miles."
    },
]
[END OF HISTORY STEPS]
```

Output:

```
{"thought": "", "tool_calls": [{"name": "get_fire_info",
"arguments": {"location": "California", "radius": 50}}]}
```

Figure 6: Example prompt and output for function-calling from unified format 1.0.