xLAM: A Family of Large Action Models to Empower AI Agent Systems

Jianguo Zhang[*], **Tian Lan**[*], **Ming Zhu**[*], **Zuxin Liu**[*], **Thai Hoang**[*], **Shirley Kokane**[††], **Weiran Yao**[†], **Juntao Tan**[†], **Zhiwei Liu**, **Yihao Feng**, **Juan Carlos Niebles**, **Shelby Heinecke**, **Huan Wang**, **Silvio Savarese**, **Caiming Xiong**
Salesforce AI

Abstract

Autonomous agents powered by large language models (LLMs) have attracted significant research interest. However, the open-source community faces many challenges in developing specialized models for agent tasks, driven by the scarcity of high-quality agent datasets and the absence of standard protocols in this area. We introduce **xLAM**, a series of large action models designed for AI agent tasks. The **xLAM** series includes five models with both dense and mixture-of-expert architectures, ranging from 1B to 8x22B parameters, trained using a scalable, flexible pipeline that unifies, augments, and synthesizes diverse datasets to enhance AI agents' generalizability and performance across varied environments. Our experimental results demonstrate that **xLAM** consistently delivers exceptional performance across multiple agent ability benchmarks, notably securing the 1st position on the Berkeley Function-Calling Leaderboard, outperforming GPT-4, Claude-3, and many other models in terms of tool use. By releasing the **xLAM** series, we aim to advance the performance of open-source LLMs for autonomous AI agents, potentially accelerating progress and democratizing access to high-performance models for agent tasks.

1 Introduction

The field of autonomous agents has witnessed significant advancements in recent years, with large language models (LLMs) playing a crucial role in enhancing agent capabilities across diverse tasks. Researchers have made substantial progress in developing sophisticated frameworks (Hong et al., 2023; Team, 2023; Wu et al., 2023; Xie et al., 2023) and specialized environments (Deng et al., 2023; Yao et al., 2022; Zhou et al., 2023) to enhance agent capabilities, such as tool use (Qin et al., 2024) and web browsing (Zhou et al., 2023). Concurrently, comprehensive benchmarks like AgentBench (Liu et al., 2023a), ToolBench (Qin et al., 2024), and AgentBoard (Ma et al., 2024) have been established to rigorously assess agent performance in reasoning, planning, and multi-turn interactions.

While proprietary LLMs developed by industry leaders have demonstrated competitive performance in various agent tasks (Anthropic, 2024; OpenAI, 2023; Reid et al., 2024; Touvron et al., 2023), the open-source community faces limited choices for specialized models in this domain. This scarcity stems from several challenges in adapting open-source LLMs to agent tasks, primarily due to the lack of comprehensive, high-quality datasets and the heterogeneity of existing data formats. These factors complicate the unification of diverse datasets and obstruct the learning of transferable knowledge across different agent tasks.

Recently, the agent research community has intensified efforts in open-source agent data processing and model training (Qin et al., 2024; Chen et al., 2023; Xu et al., 2023; Patil et al., 2023; Zeng et al., 2023; Yin et al., 2023; Zhang et al., 2024). However, these works still face challenges in managing complex environments and generalizing to new scenarios, primarily due to limitations in the collected agent data. A major obstacle is the homogeneity of content and format in existing datasets, resulting in models that lack diversity across various tasks and struggle to adapt to new or slightly different data structures in practical applications. While previous efforts have attempted to design pipelines for unifying data, they typically cover only a few scenarios or lack flexibility in their unified formats. For instance, Lumos (Yin et al., 2023) primarily addresses question answering, web agents, and mathematical tasks involving planning and grounding; while AgentOhana (Zhang et al., 2024), despite encompassing a more diverse range of environments, lacks an extendable unified format to accommodate

[*] Co-first Authors.
[†] Essential Contributors.

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), pages 11583–11597

new environments.

Moreover, open-source datasets often suffer from quality issues, such as incorrect agent outputs, hallucinated actions, and repeated interaction turns within trajectories (Zhang et al., 2024; Chen et al., 2024). The lack of detailed analysis and understanding of agent data further complicates these challenges, hindering the development of robust and versatile open-source agent models. Addressing them is crucial for advancing the field of open-source agent models and bridging the performance gap with proprietary LLMs in agent tasks.

In this work, we introduce and open-source **xLAM**, a series of powerful models with varying sizes. This diverse set is tailored for a variety of applications, with smaller models (1B and 7B) optimized for on-device deployment, while larger models (8x7B and 8x22B) are designed to tackle more challenging tasks. Alongside the model release, we offer several insights and lessons learned from our experience in agent model training:

- **Data Processing:** We highlight the importance of data unification and augmentation in enhancing dataset diversity and mitigating overfitting. Our developed data preprocess and augmentation pipeline significantly improves the generalizability of agent models across diverse environments.
- **Data Synthesis:** We showcase the impact of scalable, high-quality data synthesis on agent model performance. Our synthetic dataset enabled **xLAM** models to achieve 4 of the top 20 positions on the Berkeley Function Calling Leaderboard, including securing the top-1 spot, with smaller models achieving performance comparable to much larger counterparts, showing great potential in this direction.

We evaluate the **xLAM** series on public agent benchmarks, demonstrating exceptional performance across various agent tasks. By open-sourcing these models, we aim to advance open-source agent models and provide valuable insights into data processing and synthesis techniques, addressing key challenges in developing competitive alternatives to proprietary models.

2 Related Work

2.1 LLM Agents

Recent advancements in LLMs have significantly enhanced their utility in various agent tasks. Several innovative prompt techniques have been developed to improve performance, including Chain of Thought (Wei et al., 2022), ReACT (Yao et al., 2023), and Reflection (Shinn et al., 2023). Additionally, considerable efforts have been made to fine-tune open-sourced agent models for better capabilities (Qin et al., 2024; Chen et al., 2023; Patil et al., 2023; Zeng et al., 2023; Zhang et al., 2024). These include enhancements in data collection and processing to facilitate effective agent learning (Zeng et al., 2023; Li et al., 2023; Tang et al., 2023; Yin et al., 2023; Zhang et al., 2024; Chen et al., 2024), covering a range from simple question answering to more complex scenarios like web interactions, tool operations, reasoning, and planning. However, many of these agent frameworks still depend on proprietary models as their core engine to achieve optimal performance, revealing a substantial gap in the availability of high-quality open-source models for these tasks.

2.2 Agent Benchmarks

A variety of benchmarks have been established to assess the abilities of LLM agents across diverse scenarios (Yao et al., 2022; Qin et al., 2024; Liu et al., 2023a; Ma et al., 2024; Huang et al., 2023; Liu et al., 2023b; Wang et al., 2023; Liu et al., 2024a; Du et al., 2024; Wang et al., 2024; Yan et al., 2024). Notably, AgentBench (Liu et al., 2023a), Mint-Bench (Wang et al., 2023), and AgentBoard (Ma et al., 2024) encompass environments ranging from code generation and games to web interactions and reasoning tasks. ToolBench (Qin et al., 2024) specifically evaluates multi-turn reasoning and tool-usage abilities, while the Berkeley Function-Calling Leaderboard (Yan et al., 2024) broadly assesses models' capabilities in function calling across various contexts.

3 Data Processing Pipeline

In this section, we discuss the data pipeline for training xLAM, including data unification, augmentation, quality verification, general instruction data synthesis, and preference data generation.

3.1 Data Unification

Existing agent datasets are collected from diverse environments and designed in various formats, introducing noise and complicating data augmentation and verification. Models like NexusRaven (Srinivasan et al., 2023), Gorilla-Openfunctions (Cheng-Jie Ji et al., 2024), and

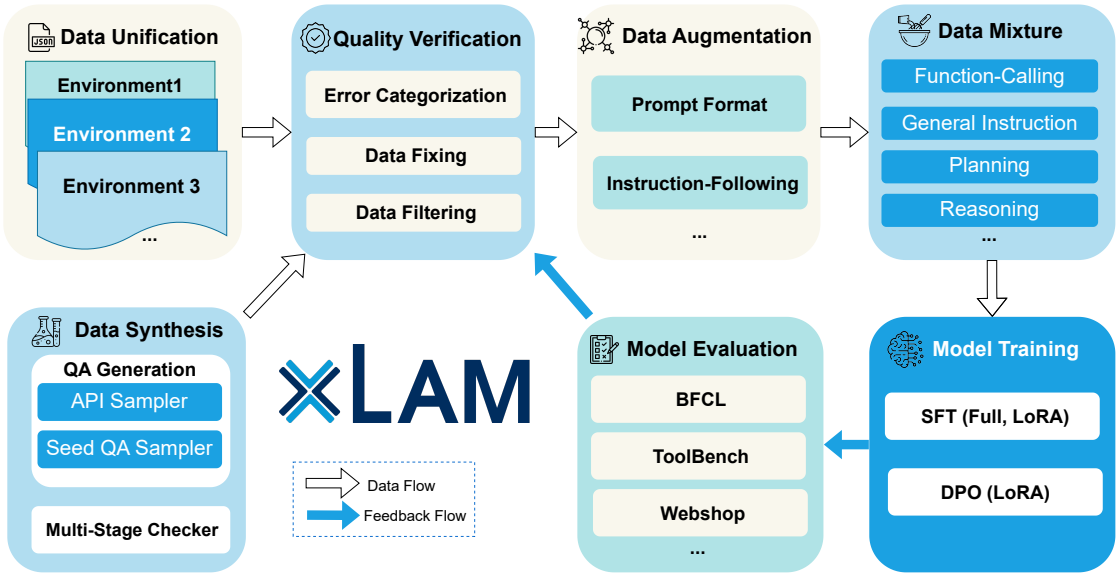

Figure 1: Overview of the data processing, training and evaluation of xLAM. We take the diagnostic feedback from the model evaluation results to iteratively improve the data quality.

AgentOhana (Zhang et al., 2024) have demonstrated superior performance in function-calling, suggesting that a well-defined, universal format could significantly enhance model performance. By standardizing the format of existing data, we can reduce noise and facilitate easier data augmentation and quality verification, leading to a more efficient and robust framework for model training and evaluation. Furthermore, a standardized format ensures consistency, simplifies model training, and enhances the model's ability to generalize across various benchmarks.

Function-calling formats form the basis for how models understand and execute tasks, motivating us to design our unified data format in a function-calling style. As illustrated in Figure 3, the unified format consists of several modules: task instruction, available tools, format instruction, few-shot examples, query, and steps. Specifically, the available tools define the agent's action space, and the format instruction specifies the output format the agent should follow when generating a response. In each step, the agent's output, the environment's feedback/execution results, and the user's follow-up input are organized into a dictionary. It's quite common for there to be purely conversational interactions between users and agents that don't trigger any APIs or receive corresponding observations. In these instances, the related entry values would simply remain empty.

This unified format is compatible with various environments and tasks, making our data processing pipeline adaptable to different datasets and scalable to large amounts of data. Moreover, the modularized design allows for fine-grained data augmentation and quality verification, which are essential in improving agent data quality. For example, by unifying all the available tools and tool calls, we can easily inspect for hallucination and function-call errors, and apply various augmentation techniques.

3.2 Data Augmentation

Our data augmentation strategy focuses on improving the diversity of the data. It involves applying various transformations to the existing dataset, thereby generating new, synthetic data samples. The data unification step significantly simplifies the application of various augmentation techniques. A standardized data format ensures consistency and ease of implementation, allowing for more efficient augmentation processes. Specifically, the augmentation techniques we adopted can be categorized as prompt format augmentation and instruction-following augmentation.

Prompt Format Augmentation: Prompt format augmentation focuses on creating various prompt formats based on the structured, unified data format. The format augmentation can be further divided into two categories: 1) *Order Shuffling*. In the unified format, the available tools are provided in a list, and each tool contains the name, description, and parameters. To avoid model overfitting to the specific order of the tools, we randomly shuf-

fle the tool list. Furthermore, we also shuffle the order of the name, description, parameters, and within the parameters to present the information in different ways. We do the same thing within the tool_calls in each step. Additionally, we also shuffle the order of different sections of the input, including task instruction, tools, format instruction, few-shot examples etc. 2) *Concatenation Tokens*. Each training data point is a pair of input and output sequences. To convert the structured unified format to the training prompt, we use special tokens to concatenate different sections into one sequence. We create several different special token styles, including "[START/END OF QUERY]", "<query></query>", and plain text.

Instruction-Following Augmentation: It focuses on adding diversity to the instructions in order to improve the model's instruction-following capability. It involves rephrasing existing instructions and adding new instructions, without introducing inaccuracy and inconsistency. Therefore, verification of the new instructions is a crucial step for this type of augmentation. We employ two methods for instruction-following augmentation: 1) *Task Instruction Rephrasing.* We rephrase the task instructions using powerful LLMs to accommodate various input styles from users. To ensure the rephrased instructions still align with the original version, we verify them by prompting the LLMs with the rephrased instructions and check if the LLMs can still follow them and generate correct function calls. 2) *Format Instruction-Following.* In our unified format, the output format is a JSON string with `thought` and `tool_calls`. To avoid the model overfitting on JSON format and to enable the model to follow various output formats upon different format instructions, we prepare 15 different output formats along with their corresponding format instructions and format converters. The formats include JSON, XML, YAML, plain text, etc.

3.3 Data Quality Verification

To further understand the data quality and to thoroughly investigate the sources of errors in the evaluation, we conduct a detailed analysis of the unified dataset. We identify a list of errors in the data using both rule-based and LLM-as-a-judge approaches.

Undefined Function Call: In function-calling, a list of available functions is provided, and the model should generate a function_call using one of the given functions. However, we found that in many cases, the predicted function_call is not from the given list. We match the predicted function with the given functions by comparing the function names and the list of parameter names. When the function_call name does not match any given functions, we refer to it as *Undefined Functions Invoked*. When the function name matches but the argument list contains undefined arguments, we refer to it as *Undefined Arguments Passed*. We also take into consideration optional parameters.

Incorrect Argument Type: Other than the error types mentioned above, we also observe that sometimes the model generates the correct argument's value, but in the wrong types. For example, when a parameter expects a `[val1, val2, val3]`, the generated arguments is `"[val1, val2, val3]"`, which is a string version of the list. When executing the function call, errors will occur due to incorrect data type. We identify trajectories containing the incorrect argument type error by comparing the parameter type in the available tools and the actual argument type. We also found that most argument type errors can be fixed by converting the arguments to the correct parameter types.

Argument Hallucination: Upon examining the unified dataset from public sources, we discovered that tool calls frequently include argument values not present in the user query or prior steps. This issue arises because much of this data is generated by LLMs, which are prone to hallucination, a common problem in LLM-generated content. We identified two types of hallucination: 1) the generated tool names or argument names do not appear in the provided tool and argument list; and 2) the argument values do not align with the user query or observations from previous steps. The first type of hallucination is straightforward to address by searching the generated tool call and argument names and matching them with the provided tool list, as they are all structured in JSON, making this process efficient. However, detecting the second type, where argument values are misaligned, is more challenging, as simple string matching is ineffective for complex queries and tasks. To tackle this, we use LLMs as judges to perform step-wise argument hallucination detection, detecting if there is a mismatch between the arguments and the intended query or prior observations.

Low-Quality Reasoning and Planning: We observe many data trajectories where the reasoning and planning steps are of low quality, which is a common issue in the outputs of many LLMs. To address this, we first filter out low-

quality data using rule-based methods informed by heuristics, then prompt models like Mixtral-8x22b-Instruct-v0.1 (Jiang et al., 2024) and DeepSeek-V2 (DeepSeek-AI, 2024) to evaluate both the overall trajectory and individual thought steps on the selected data. A portion of these rating results is then sampled and verified by humans. We also attempted to iterate on this process using specifically fine-tuned models.

3.4 Data Synthesis

Based on our findings in Sec. 3.3, we observe that most of these publicly available datasets have several limitations. First, these datasets are often static, synthesized by weak models, limited in scope, and, more importantly, not verified by execution. Second, these datasets mainly focus on a single type of function-calling category, i.e., outputting a single function call based on the provided tools. However, real-world scenarios might consist of many other types of use cases, such as the parallel function-calling scenario (Yan et al., 2024), where the user query contains multiple requests and the model should respond with concurrent function calls in parallel within a single response.

To address these two issues, we utilize a systematic data synthesis framework called APIGen (Liu et al., 2024b), to generate 50k verifiable data points based on a collection of 3,673 executable APIs. The key idea is a multi-stage verification process to ensure the accuracy and quality of the generated data. This process includes format verification, semantic verification and quality control as we developed in Sec. 3.3, along with execution verification. Together, these steps collectively help to identify and filter out low-quality data points, such as those with hallucination issues or inaccurate argument parameter values.

3.5 Data Mixture

For supervised fine-tuning (SFT), our dataset combines training samples from three main sources: cleaned and augmented agent datasets, a synthetic function-calling dataset, and general instruction-tuning datasets. These sources are used to train the general xLAM models.

Specifically, to enhance the general instruction capability of xLAM, we integrate diverse instruction-tuning datasets from DialogStudio (Zhang et al., 2023) and Data Provenance (Longpre et al., 2023, 2024). This instruction data comprises 20% to 30% of our training set. To enhance the function-calling capability of xLAM-7b-fc-r and xLAM-1b-fc-r, we employ a targeted training approach, with 50% of their training data drawn from our high-quality synthetic function-calling dataset. The remaining 50% of the data is sampled from other tasks within our training set.

For Direct Preference Optimization (DPO) (Rafailov et al., 2023), we prompt less powerful models to generate and rate responses for selected data from each source, then sample a subset for human verification. After adjustments to models and prompts, we classify the selected responses as rejected samples.

4 Model Training

4.1 Modeling

We use a supervised fine-tuning (SFT) approach, further aligning model checkpoints with the DPO method, and leverage the robustness of our flexible data pipeline. Our training code is based on the HuggingFace Transformers and Accelerate libraries(Wolf et al., 2020; Gugger et al., 2022), as well as PyTorch FSDP(Zhao et al., 2023a).

The fine-tuning of general xLAM models is conducted on Nvidia H100 GPUs. For SFT, we use a full fine-tuning framework that employs the fully sharded data parallel algorithm (Zhao et al., 2023b). In the case of xLAM-8x22b-r, we integrate LoRA (Hu et al., 2021; Dettmers et al., 2023) to better preserve the model's original capacities and prevent catastrophic forgetting (Liu et al., 2023c). LoRA is also used for DPO alignment across all xLAM models. We utilize a total batch size of 128 and a learning rate that ranges from 5×10^{-6}, to 5×10^{-5}. Additionally, we use a cosine learning rate scheduler with 100 warm-up steps to optimize performance. Training times range from 6 to 40 hours.

4.2 xLAM Model Series

We introduce a series of agent models tailored for different use cases. Our flagship model series, xLAM, is built upon the Mixtral Instruct (Jiang et al., 2024) models and aims to achieve balanced performance across a diverse range of agent tasks, from complex multi-turn interactions to function-calling applications.

In addition to general xLAM models, we develop two specialized models for function-calling use cases, xLAM-7b-fc-r and xLAM-1b-fc-r, based on DeepSeek-Coder-7B-instruct-v1.5 and DeepSeek-Coder-1.3B-instruct, respectively (Guo et al.,

2024). The smaller model sizes offer increased accessibility, allowing users to easily host them on a single GPU to address various function-calling tasks, ranging from simple user queries to parallel concurrent requests.

By offering a suite of models with varying sizes and specializations, the xLAM series caters to a wide range of user needs and computational resources, making powerful agent capabilities more accessible and adaptable to real-world applications.

5 Experiments

5.1 Benchmarks

After considering the stability of environments and research budget limitations, we evaluate the performance of models across four rigorous benchmarks: Webshop (Yao et al., 2022), ToolQuery (Ma et al., 2024), ToolBench (Qin et al., 2024), and the Berkeley Function-Calling Benchmark (BFCL) (Yan et al., 2024). Each benchmark is designed to assess different aspects of model capabilities under a variety of settings and constraints.

Webshop is an interactive web environment designed to mimic online shopping experiences, testing an agent's ability to navigate and assist in e-commerce tasks. Webshop comprising approximately 250 test cases.

ToolQuery evaluates an agent's skills in using tools to retrieve and process information across domains. ToolQuery features 60 test cases across three distinct settings: Weather, Movie, and Academia. We use the testing configurations of AgentBoard (Ma et al., 2024) with Success Rate and Progress Rate for both Webshop and ToolQuery.

We also evaluate on **ToolQuery-Unified**, which is essentially ToolQuery but requires an agent to ingest the task instruction and tools following the augmented prompt format described in §3.2 and likewise solve the task following the unified format. The purpose of testing agents in this setting is to assess their reasoning and tool-use abilities when evaluated on structured formats (Tam et al., 2024).

ToolBench is developed for real-time evaluation of multi-turn reasoning and interactive capabilities via RapidAPI, includes around 1,000 test cases. It asks GPT-4 to evaluate Pass Rate of agent responses in both in-domain and out-of-domain settings, including unseen instructions with familiar tools, unseen tools within previously known categories, and entirely new categories of unseen tools. Due to space limit, we put testing results into Appendix A.2.

Berkeley Function-Calling Benchmark offers an extensive evaluation of an agent's ability to reason and execute function calls across various programming languages and application domains. With 2,000 test cases, it tests complex scenarios such as parallel and multiple function calls in languages including Java, JavaScript, and Python. The evaluation metrics include: Abstract Syntax Tree (AST) accuracy for non-executable test queries, executable accuracy by running the APIs to get the results, and relevance detection score which measures the agent's ability to distinguish non-relevant queries and provided tools. Since BFCL updates frequently, we used their stable V2 version, which has a cutoff date of September 19, 2024.

5.2 Experimental Results

5.2.1 Webshop and ToolQuery

Webshop. Table 1 presents detailed comparisons of state-of-the-art language and agent models in the Webshop and ToolQuery environments, illustrating the robust and strong performance of the xLAM models. In the Webshop environment, xLAM-7b-r not only achieves the highest Success Rate at 0.414, surpassing other general LLMs like GPT-4-0125-preview, GPT-4o-2024-0523, and Claude2, but also outperforms specialized agent models such as AgentOhana-8x7b and Lemur-70b. This shows xLAM's superior ability to navigate and execute tasks effectively in the web interactions.

ToolQuery. In the more complex and unseen ToolQuery environment, xLAM-8x7b-r and xLAM-8x22b-r also demonstrate high performance as shown in Table 1, ranking second with a Success Rate of 0.683. This shows a significant improvement over the baseline performance of Mixtral-8x7b-inst and Mixtral-8x22b-inst, which are 0.167 and 0.400, respectively. Notably, all three xLAM models surpass the Mixtral-8x22B-Instruct model. Despite Mixtral-8x22B-Instruct having a large number of parameters and specialized tuning for advanced functionalities such as function calling, reasoning, and complex tool usage, it falls short of the xLAM models' performance. Furthermore, same as other general LLMs, it lacks transparency regarding the data collection, unification processes, and other critical details, contrasting with the open source purposes provided for xLAM. These results show the efficacy of our proposed data unification and synthetic data pipeline.

ToolQuery-Unified. When the system prompt

LLM	Webshop		ToolQuery	
	Success Rate	Progress Rate	Success Rate	Progress Rate
xLAM-7b-r	**0.414**	0.767	0.550	0.674
xLAM-8x7b-r	0.410	0.763	0.683	0.745
xLAM-8x22b-r	0.390	0.763	0.683	0.758
GPT-4-0125-preview	0.375	0.760	**0.750**	0.803
GPT-4o-2024-0523	0.323	0.694	0.633	0.801
AgentOhana-8x7b (Zhang et al., 2024)	0.331	0.737	0.533	0.766
Claude2	0.378	0.746	0.483	0.735
Mixtral-8x22b-inst (Jiang et al., 2024)	0.383	0.739	0.400	0.740
DeepSeek-67b-chat (Bi et al., 2024)	0.319	0.727	0.400	0.714
GPT-3.5-Turbo-0125	0.323	0.749	0.367	0.545
Llama3-70b-inst (AI@Meta, 2024)	0.299	0.746	0.367	0.526
Lemur-70b-chat-v1 (Xu et al., 2023)	0.116	0.718	0.283	0.720
Mixtral-8x7b-inst (Jiang et al., 2024)	0.222	0.766	0.167	0.654
CodeLlama-34b-inst (Roziere et al., 2023)	0.235	0.717	0.133	0.600
Llama2-70b-chat (Touvron et al., 2023)	0.131	0.536	0.000	0.483

Table 1: Testing results on Webshop and ToolQuery. **Bold** and Underline results denote the best result and the second best result for Success Rate, respectively.

	Success Rate	Academia	Movie	Weather
xLAM-7b-r	0.466 (0.550)	0.45	0.25	0.35
xLAM-8x7b-r	0.533 (0.683)	0.45	0.40	0.45
xLAM-8x22b-r	**0.733** (0.683)	0.75	0.40	0.60
GPT-4-0125-preview	0.566 (0.750)	0.65	0.35	0.25
GPT-4o-2024-05-13	0.366 (0.633)	0.45	0.20	0.25

Table 2: Testing results on ToolQuery-Unified. **Bold** and Underline results denote the best result and the second best result for Success Rate, respectively. Values in brackets indicate corresponding performance on ToolQuery.

from ToolQuery is presented to the model in the unified format shown in Fig. 4, and the model is required to follow the provided format instructions to generate a structured output, we observe that xLAM's performances are more consistent compared to GPT models, as shown in Table 2. While GPT-4o's performance significantly degrades by 42% compared to ToolQuery, our best xLAM 8x22b model maintains comparable performance. This can be attributed to xLAM being trained on trajectories that adhere to the unified format, enabling it to perform consistently during inference. Concurrent research (Tam et al., 2024) observed a similar decline in performance on reasoning tasks when LLMs are constrained to produce output in specific formats. Deeper analysis indicated that the degradation is more than just due to incorrectly formatted output in a specific format, but rather due to a drop in the reasoning ability of the model itself.

5.2.2 Berkeley Function-Calling Benchmark

Table 3 presents the experimental results on the BFCL v2 benchmark (cutoff date 09/19/2024), which shows the exceptional performance of our xLAM model series in function-calling tasks. Notably, xLAM models secure four out of the top twenty positions, demonstrating the effectiveness of our data pipeline and training methodology across various model sizes.

Our flagship model, xLAM-8x22b-r, achieves the highest overall accuracy of 87.31%, surpassing all other models in the benchmark. This result validates the effectiveness of our data processing and model training pipeline in improving models' function-calling ability. Following closely, xLAM-8x7b-r ranks 6th, outperforming most prominent models including GPT-4o-mini and Claude-3.

The performance of our models demonstrates clear scaling with model size, a trend exemplified by xLAM-7b-r, which ranks 14th with an accuracy of 80.33%. This model outperforms several larger and more resource-intensive alternatives, including multiple GPT-4 and GPT-4o versions. Remarkably, our smallest model, xLAM-1b-fc-r, achieves a 32nd place ranking with an accuracy of 75.43%, surpassing much larger models like Claude-3-Opus (FC) and GPT-3.5-Turbo. This performance underscores the power of our data synthesis framework in producing high-quality, diverse datasets that enhance function-calling effectiveness even for smaller language models.

It is also worth noting that the BFCL v2 benchmark (Yan et al., 2024) includes a live dataset released after our model training date. These fresh data are collected from real-world user queries that were entirely unseen by our models. Nevertheless, our models exhibit strong generalization capabilities in handling these real-world use cases. The consistently strong performance across our model

Rank	Overall Accuracy	Model	Abstract Syntax Tree (AST) Evaluation				Evaluation by Executing APIs				Relevance Detection	
			Simple	Multiple	Parallel	Parallel Multiple	Simple	Multiple	Parallel	Parallel Multiple	Irrelevance	Relevance
1	**87.31**	**xLAM-8x22b-r (FC)**	**72.79**	**86.37**	**87.13**	**84.75**	**98.57**	**94.00**	**92.00**	**85.00**	**74.96**	**97.56**
2	85.79	GPT-4-0125-Preview (Prompt)	78.82	88.44	91.00	83.75	99.00	96.00	82.00	80.00	61.35	97.56
3	85.00	GPT-4-1106-Preview (Prompt)	78.75	89.12	94.12	83.25	99.00	96.00	82.00	72.50	64.98	90.24
4	84.74	GPT-4-0613 (Prompt)	78.76	85.46	91.75	82.67	98.29	96.00	86.00	70.00	75.57	82.93
5	83.89	GPT-4-turbo-20240409 (Prompt)	80.47	88.81	88.12	84.25	99.00	96.00	80.00	77.50	61.82	82.93
6	**83.38**	**xLAM-8x7b-r (FC)**	**73.12**	**86.09**	**71.00**	**82.50**	**92.57**	**96.00**	**90.00**	**77.50**	**72.35**	**92.68**
7	83.35	GPT-4o-mini-20240718 (Prompt)	75.88	81.64	85.12	79.42	98.29	94.00	82.00	77.50	79.20	80.49
8	83.13	GPT-4o-2024-05-13 (Prompt)	76.18	86.01	92.12	81.00	98.00	94.00	76.00	72.50	77.44	78.05
9	82.55	Functionary-Medium-v3.1 (FC)	74.34	87.59	81.62	80.67	98.29	94.00	90.00	75.00	73.23	70.73
10	81.78	GPT-4-1106-Preview (FC)	69.32	84.19	86.38	71.92	95.43	94.00	86.00	75.00	72.70	82.93
11	81.59	Llama3-70B-Instruct (Prompt)	72.87	85.91	84.00	77.83	94.14	94.00	84.00	80.00	50.47	92.68
12	80.88	Claude-3-Opus (Prompt)	76.65	87.47	78.38	75.17	98.57	94.00	82.00	75.00	56.15	85.37
13	80.87	GPT-4-0125-Preview (FC)	68.76	84.95	80.38	74.00	84.21	94.00	88.00	75.00	74.03	85.37
14	**80.33**	**xLAM-7b-r (FC)**	**69.85**	**84.00**	**63.00**	**79.17**	**75.71**	**94.00**	**92.00**	**80.00**	**72.88**	**92.68**
15	80.23	Nemotron-340b-inst (Prompt)	68.51	80.38	78.62	79.17	86.00	90.00	80.00	77.50	84.10	78.05
16	80.21	Functionary-Small-v3.1 (FC)	72.70	83.31	85.62	72.92	87.79	90.00	86.00	70.00	68.36	85.37
17	**80.18**	**xLAM-7b-fc-r (FC)**	**70.52**	**78.22**	**73.88**	**68.50**	**95.21**	**90.00**	**88.00**	**77.50**	**79.54**	**80.49**
18	79.66	mistral-large-2407 (FC Any)	81.01	87.42	90.50	83.50	98.29	92.00	86.00	77.50	0.34	100.00
19	79.55	GPT-4o-2024-05-13 (FC)	70.40	82.33	89.00	76.08	88.93	84.00	88.00	72.50	73.50	70.73
32	**75.43**	**xLAM-1b-fc-r (FC)**	**64.63**	**72.33**	**64.50**	**61.42**	**80.21**	**92.00**	**86.00**	**75.00**	**60.65**	**97.56**
33	75.41	GPT-3.5-Turbo (FC)	69.79	83.58	71.88	68.83	95.14	88.00	86.00	57.50	35.83	97.56
34	74.97	Mistral-Nemo-2407 (FC Auto)	64.57	79.99	80.25	74.00	91.36	86.00	86.00	62.50	59.14	65.85

Table 3: Performance comparison on BFCL-v2 leaderboard (cutoff date 09/19/2024). The rank is based on the overall accuracy, which is a weighted average of different evaluation categories. “FC" stands for function-calling mode in contrast to using a customized “prompt" to extract the function calls. The complete table can be found in the Appendix Table 5.

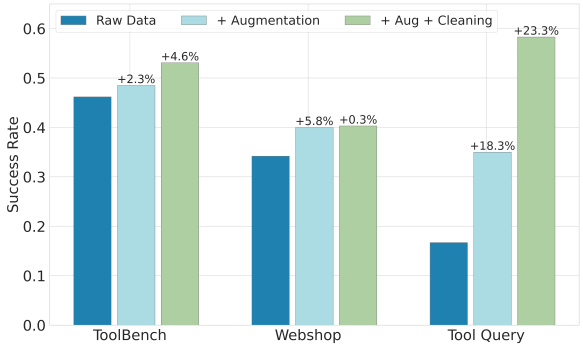

Figure 2: Ablation study for data augmentation and data quality verification (cleaning).

series, ranging from 8x22 billion to 1 billion parameters, demonstrates the scalability and versatility of our approach. This scalability is particularly noteworthy, as it enables strong results from compact models suitable for resource-constrained environments to large-scale models for more demanding applications. Furthermore, the ability of our smaller models to compete with much larger alternatives suggests significant potential for efficient deployment in various real-world scenarios.

5.3 Ablation Study

We conducted an ablation study on the 7B models to measure the impact of various steps in our data pipeline. Three datasets were prepared for this analysis: raw data, augmented data, and augmented + cleaned data. The raw data represents the dataset before data unification, while the other two datasets are post-unification. Figure 2 presents the evaluation results of models trained on these three datasets. The metrics used for this evaluation are G1_instruction from ToolBench and success_rate from both Webshop and ToolQuery. The results indicate that augmented data consistently outperforms raw data across all metrics, with improvements of 2.3% on ToolBench, 5.8% on Webshop, and 18.3% on ToolQuery. Furthermore, the addition of data cleaning leads to a substantial performance increase on ToolQuery, with a further improvement of 23.4%. The results highlight the effectiveness of data augmentation and cleaning processes in the data pipeline.

6 Conclusion

This paper introduces xLAM series, a set of large action models for autonomous AI agents. Our models, ranging from 1B to 8x22B parameters, were trained with a scalable and flexible data pipeline that unifies, augments, and synthesizes diverse datasets. Evaluations show that xLAM models consistently perform exceptionally across various benchmarks. The insights we learned from training these models highlight the importance of rigorous data processing and the potential of data synthesis in developing capable AI agents. By releasing the xLAM series, we aim to democratize access to high-performance models for agent tasks, thereby accelerating progress in the field.

7 Limitations

Our data synthesis pipeline relies on large LLMs such as DeepSeek-V2 and Mixtral-8x22b-Inst. This dependence may result in extended data generation times when utilizing less powerful GPUs. Future work will focus on developing more efficient pipelines capable of operating effectively with both smaller and larger LLMs.

Additionally, due to the instability of some benchmarks and environments, along with the heavy reliance on API calls from commercial models such as OpenAI GPT-4, and constrained by limited research budgets, we face challenges in evaluating our models across even more benchmarks. Consequently, this may result in performance discrepancies on certain benchmarks. Moving forward, we aim to enhance our contributions to the research community by addressing instability and better maintaining and unifying existing benchmarks and environments.

References

AI@Meta. 2024. Llama 3 model card.

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

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. 2024. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*.

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*.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. 2024. Agent-flan: Designing data and methods of effective agent tuning for large language models. *arXiv preprint arXiv:2403.12881*.

Charlie Cheng-Jie Ji, Huanzhi Mao, Fanjia Yan, Shishir Patil, Tianjun Zhang, Ion Stoica, and Joseph Gonzalez. 2024. Gorilla openfunctions v2. In https://gorilla.cs.berkeley.edu//blogs/7_open_functions_v2.html.

DeepSeek-AI. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *Preprint*, arXiv:2405.04434.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *arXiv preprint arXiv:2306.06070*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *Preprint*, arXiv:2305.14314.

Yu Du, Fangyun Wei, and Hongyang Zhang. 2024. Anytool: Self-reflective, hierarchical agents for large-scale api calls. *arXiv preprint arXiv:2402.04253*.

Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, Marc Sun, and Benjamin Bossan. 2022. Accelerate: Training and inference at scale made simple, efficient and adaptable. https://github.com/huggingface/accelerate.

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

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2023. Metagpt: Meta programming for a multi-agent collaborative framework. *Preprint*, arXiv:2308.00352.

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*.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, et al. 2023. Metatool benchmark for large language models: Deciding whether to use tools and which to use. *arXiv preprint arXiv:2310.03128*.

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*.

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

Zhiwei Liu, Weiran Yao, Jianguo Zhang, Le Xue, Shelby Heinecke, Rithesh Murthy, Yihao Feng, Zeyuan Chen, Juan Carlos Niebles, Devansh Arpit, et al. 2023b. Bolaa: Benchmarking and orchestrating llm-augmented autonomous agents. *arXiv preprint arXiv:2308.05960*.

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*.

Zuxin Liu, Jesse Zhang, Kavosh Asadi, Yao Liu, Ding Zhao, Shoham Sabach, and Rasool Fakoor. 2023c. Tail: Task-specific adapters for imitation learning with large pretrained models. *arXiv preprint arXiv:2310.05905*.

Shayne Longpre, Robert Mahari, Anthony Chen, Naana Obeng-Marnu, Damien Sileo, William Brannon, Niklas Muennighoff, Nathan Khazam, Jad Kabbara, Kartik Perisetla, et al. 2023. The data provenance initiative: A large scale audit of dataset licensing & attribution in ai. *arXiv preprint arXiv:2310.16787*.

Shayne Longpre, Robert Mahari, Ariel Lee, Campbell Lund, Hamidah Oderinwale, William Brannon, Nayan Saxena, Naana Obeng-Marnu, Tobin South, Cole Hunter, et al. 2024. Consent in crisis: The rapid decline of the ai data commons. *arXiv preprint arXiv:2407.14933*.

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*.

OpenAI. 2023. Gpt-4 technical report. *ArXiv*.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*.

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*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*.

Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

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*.

Zhi Rui Tam, Cheng-Kuang Wu, Yi-Lin Tsai, Chieh-Yen Lin, Hung-yi Lee, and Yun-Nung Chen. 2024. Let me speak freely? a study on the impact of format restrictions on performance of large language models. *arXiv preprint arXiv:2408.02442*.

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*.

XAgent Team. 2023. Xagent: An autonomous agent for complex task solving.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Jize Wang, Zerun Ma, Yining Li, Songyang Zhang, Cailian Chen, Kai Chen, and Xinyi Le. 2024. Gta: A benchmark for general tool agents. *arXiv preprint arXiv:2407.08713*.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2023. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. *arXiv preprint arXiv:2309.10691*.

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.

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.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Tianbao Xie, Fan Zhou, Zhoujun Cheng, Peng Shi, Luoxuan Weng, Yitao Liu, Toh Jing Hua, Junning Zhao, Qian Liu, Che Liu, et al. 2023. Openagents: An open platform for language agents in the wild. *arXiv preprint arXiv:2310.10634*.

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*.

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, 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)*.

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. 2024. Agentohana: Design unified data and training pipeline for effective agent learning. *arXiv preprint arXiv:2402.15506*.

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*.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. 2023a. Pytorch fsdp: Experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. 2023b. Pytorch fsdp: Experiences on scaling fully sharded data parallel. *Preprint*, arXiv:2304.11277.

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 Benchmarks

After considering the stability of environments and research budget limitations, we evaluate the performance of models across four rigorous benchmarks: Webshop (Yao et al., 2022), ToolQuery (Ma et al., 2024), ToolBench (Qin et al., 2024), and the Berkeley Function-Calling Benchmark (Yan et al., 2024). Each benchmark is designed to assess different aspects of model capabilities under a variety of settings and constraints.

Webshop is an interactive web environment designed to mimic online shopping experiences, testing an agent's ability to navigate and assist in e-commerce tasks. Webshop comprising approximately 250 test cases.

ToolQuery evaluates an agent's skills in using tools to retrieve and process information across domains. ToolQuery features 60 test cases across three distinct settings: Weather, Movie, and Academia.

We use the testing configurations from AgentBoard (Ma et al., 2024) for both Webshop and ToolQuery. These configurations assess overall performance using the Success Rate and evaluate progressive performance across interactive turns with the Progress Rate, with Success Rate being the more critic metric.

We additionally evaluate on **ToolQuery-Unified**, which is essentially ToolQuery but requires an agent to ingest the task instruction and tools following the augmented prompt format described in §3.2 and likewise solve the task following the unified format. The purpose of testing agents in this setting is to assess their reasoning and tool-use abilities when evaluated on structured formats (Tam et al., 2024).

ToolBench is developed for real-time evaluation of multi-turn reasoning and interactive capabilities via RapidAPI, and includes around 1,000 test cases. It uses Pass Rate as the metric, where the trajectory and final response are sent to GPT-4-0125-preview to determine whether the agent's final response successfully addresses the given user query. The evaluations cover both in-domain and out-of-domain settings, including unseen instructions with familiar tools, unseen tools within previously known categories, and entirely new categories of unseen tools.

Berkeley Function-Calling Leaderboard Benchmark (Yan et al., 2024) provides a comprehensive evaluation framework for assessing an agent's capability to reason about and execute function calls across a variety of programming languages and application domains. The benchmark comprises over 2,200 test cases, challenging models with complex scenarios such as parallel and multiple function calls in languages like Java, JavaScript, and Python. The evaluation metrics include Abstract Syntax Tree (AST) accuracy for non-executable test queries, executable accuracy by running APIs to obtain results, and a relevance detection score that measures the agent's ability to distinguish non-relevant queries and provided tools.

Importantly, our evaluation utilizes the most stable BFCL v2 version, as of the cutoff date 09/19/2024. The v2 version introduces live function calls and real-world scenarios contributed by users, addressing issues such as data contamination, bias, and fairness by leveraging user-provided data. This updated dataset better reflects real-world distributions, characterized by a higher demand for selecting among multiple functions and a reduced demand for parallel function calls. For instance, our analysis indicates that in the v2 benchmark, the average number of available functions has doubled, while the average number of function calls has been halved compared to the non-live v1 data. It is important to note that all our models were trained prior to the release of the BFCL v2 live data.

A.2 ToolBench

Table 4 presents the results on ToolBench, where xLAM models demonstrate impressive performance. They surpass both TooLlama-V2 and GPT-3.5-Turbo-0125 across all test settings. Moreover, xLAM models outperform AgentOhana-8x7b in scenarios involving unseen instructions and unseen tools, while achieving performance comparable to GPT-4-0125-preview in the unseen tools setting. These results show xLAM models' robust capabilities in multi-turn reasoning and complex tool usage, effectively handling both in-domain and out-of-domain tasks.

A.3 Berkeley Function-Calling Leaderboard Benchmark

Table 5 is an extended version of Table 3. It provide a comprehensive table with the results of various models.

```
{
    "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 3: Unified function calling data format.

	Unseen Insts & Same Set	Unseen Tools & Seen Cat	Unseen Tools & Unseen Cat
xLAM-7b-r	<u>0.5308</u>	<u>0.5300</u>	**0.5850**
xLAM-8x7b-r	<u>0.5308</u>	**0.5450**	<u>0.5700</u>
AgentOhana-8x7b (Zhang et al., 2024)	0.5077	0.5200	0.5650
GPT-4-0125-preview	**0.5462**	0.5050	0.5450
GPT-3.5-Turbo-0125	0.5000	0.4900	0.5150
TooLlama-V2 (Qin et al., 2024)	0.4385	0.4350	0.4300

Table 4: Pass Rate on ToolBench on three distinct scenarios. **Bold** and <u>Underline</u> results denote the best result and the second best result for each setting, respectively. The results for xLAM-8x22b-r are unavailable due to the ToolBench server being down during the evaluation phase in September 2024 and additional GPU budget limit for further testing.

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.",
                "required": false,
            }
        },
    },...
]
[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 4: Example prompt and output for function-calling using xLAM.

Rank	Overall Accuracy	Model	Abstract Syntax Tree (AST) Evaluation				Evaluation by Executing APIs				Relevance Detection	
			Simple	Multiple	Parallel	Parallel Multiple	Simple	Multiple	Parallel	Parallel Multiple	Irrelevance	Relevance
1	**87.31**	**xLAM-8x22b-r (FC)**	**72.79**	**86.37**	**87.13**	**84.75**	**98.57**	**94.00**	**92.00**	**85.00**	**74.96**	**97.56**
2	85.79	GPT-4-0125-Preview (Prompt)	78.82	88.44	91.00	83.75	99.00	96.00	82.00	80.00	61.35	97.56
3	85.00	GPT-4-1106-Preview (Prompt)	78.75	89.12	94.12	83.25	99.00	96.00	82.00	72.50	64.98	90.24
4	84.74	GPT-4-0613 (Prompt)	78.76	85.46	91.75	82.67	98.29	96.00	86.00	70.00	75.57	82.93
5	83.89	GPT-4-turbo-20240409 (Prompt)	80.47	88.81	88.12	84.25	99.00	96.00	80.00	77.50	61.82	82.93
6	**83.38**	**xLAM-8x7b-r (FC)**	**73.12**	**86.09**	**71.00**	**82.50**	**92.57**	**96.00**	**90.00**	**77.50**	**72.35**	**92.68**
7	83.35	GPT-4o-mini-20240718 (Prompt)	75.88	81.64	85.12	79.42	98.29	94.00	82.00	77.50	79.20	80.49
8	83.13	GPT-4o-2024-05-13 (Prompt)	76.18	86.01	92.12	81.00	98.00	94.00	76.00	72.50	77.44	78.05
9	82.55	Functionary-Medium-v3.1 (FC)	74.34	87.59	81.62	80.67	98.29	94.00	90.00	75.00	73.23	70.73
10	81.78	GPT-4-1106-Preview (FC)	69.32	84.19	86.38	71.92	95.43	94.00	86.00	75.00	72.70	82.93
11	81.59	Llama3-70B-Instruct (Prompt)	72.87	85.91	84.00	77.83	94.14	94.00	84.00	80.00	50.47	92.68
12	80.88	Claude-3-Opus (Prompt)	76.65	87.47	78.38	75.17	98.57	94.00	82.00	75.00	56.15	85.37
13	80.87	GPT-4-0125-Preview (FC)	68.76	84.95	80.38	74.00	84.21	94.00	88.00	75.00	74.03	85.37
14	**80.33**	**xLAM-7b-r (FC)**	**69.85**	**84.00**	**63.00**	**79.17**	**75.71**	**94.00**	**92.00**	**80.00**	**72.88**	**92.68**
15	80.23	Nemotron-340b-inst (Prompt)	68.51	80.38	78.62	79.17	86.00	90.00	80.00	77.50	84.10	78.05
16	80.21	Functionary-Small-v3.1 (FC)	72.70	83.31	85.62	72.92	87.79	90.00	86.00	70.00	68.36	85.37
17	**80.18**	**xLAM-7b-fc-r (FC)**	**70.52**	**78.22**	**73.88**	**68.50**	**95.21**	**90.00**	**88.00**	**77.50**	**79.54**	**80.49**
18	79.66	mistral-large-2407 (FC Any)	81.01	87.42	90.50	83.50	98.29	92.00	86.00	77.50	0.34	100.00
19	79.55	GPT-4o-2024-05-13 (FC)	70.40	82.33	89.00	76.08	88.93	84.00	88.00	72.50	73.50	70.73
20	79.25	GPT-4o-mini-2024-07-18 (FC)	67.83	80.16	85.38	77.17	83.21	92.00	82.00	70.00	71.83	82.93
21	79.14	Open-Mixtral-8x22b (Prompt)	73.47	76.14	79.12	73.67	91.86	96.00	84.00	75.00	71.42	70.73
22	79.10	Gorilla-OpenFunctions-v2 (FC)	70.81	79.47	75.75	66.67	95.86	96.00	78.00	70.00	73.13	85.37
23	79.09	GPT-4-turbo-2024-04-09 (FC)	64.21	82.72	82.50	75.75	88.71	88.00	86.00	72.50	79.79	70.73
24	78.96	Functionary-Small-v3.2 (FC)	69.50	81.50	80.12	73.50	90.64	88.00	86.00	67.50	72.32	80.49
25	78.87	GPT-4o-2024-08-06 (FC)	70.71	80.97	83.25	75.58	85.36	90.00	84.00	72.50	82.91	63.41
26	78.78	mistral-large-2407 (FC Auto)	68.28	86.44	90.25	83.50	76.86	92.00	86.00	77.50	48.93	78.05
27	77.92	Claude-3-Sonnet (Prompt)	71.80	85.26	82.75	73.92	96.14	90.00	84.00	77.50	30.01	87.80
28	77.45	FireFunction-v2 (FC)	74.11	81.49	73.62	67.58	94.43	88.00	82.00	72.50	52.94	87.80
29	76.63	Granite-20b (FC)	65.27	73.05	60.75	67.83	85.36	90.00	84.00	72.50	72.43	95.12
30	76.31	Mistral-Nemo-2407 (Prompt)	72.89	81.37	81.50	73.75	92.50	94.00	86.00	80.00	13.25	87.80
31	76.29	Claude-3.5-Sonnet (Prompt)	76.98	80.27	72.62	65.33	98.50	92.00	70.00	72.50	83.46	51.22
32	**75.43**	**xLAM-1b-fc-r (FC)**	**64.63**	**72.33**	**64.50**	**61.42**	**80.21**	**92.00**	**86.00**	**75.00**	**60.65**	**97.56**
33	75.41	GPT-3.5-Turbo (FC)	69.79	83.58	71.88	68.83	95.14	88.00	86.00	57.50	35.83	97.56
34	74.97	Mistral-Nemo-2407 (FC Auto)	64.57	79.99	80.25	74.00	91.36	86.00	86.00	62.50	59.14	65.85
35	74.78	Hermes-2-Pro-Llama3-70B (FC)	66.29	73.49	70.25	78.33	80.64	88.00	84.00	72.50	53.80	80.49
36	74.75	Gemini-1.5-Pro-0514 (FC)	56.15	78.89	82.38	65.50	75.71	88.00	84.00	75.00	83.31	58.54
37	74.57	Claude-2.1 (Prompt)	68.21	78.08	74.12	66.17	94.64	88.00	64.00	62.50	74.36	75.61
38	74.56	Gemini-1.5-Pro-0409 (FC)	55.08	79.43	83.12	64.75	76.00	88.00	80.00	72.50	83.27	63.41
39	74.12	GPT-4o-2024-08-06 (Prompt)	65.76	76.86	72.12	71.67	70.57	88.00	78.00	75.00	89.56	53.66
40	74.11	Command-R-Plus (Prompt)	68.14	78.13	77.50	62.17	91.29	86.00	78.00	55.00	69.31	75.61
41	73.12	Mistral-Nemo-2407 (FC Any)	67.98	82.46	77.38	76.08	92.07	86.00	86.00	62.50	0.72	100.00
42	72.19	Mistral-Medium-2312 (Prompt)	63.77	80.22	69.12	59.25	93.43	88.00	70.00	57.50	84.54	56.10
43	72.04	Command-R-Plus (FC) (Original)	64.25	72.45	66.25	62.33	89.14	86.00	82.00	52.50	52.75	92.68
44	70.75	Gemini-1.5-Flash-0514 (FC)	65.80	83.26	63.87	63.50	57.93	86.00	74.00	75.00	74.69	63.41
45	69.55	DBRX-Instruct (Prompt)	69.97	80.35	66.88	51.50	90.50	86.00	60.00	62.50	44.86	82.93
46	68.88	Claude-3.5-Sonnet (FC)	73.95	82.09	65.38	62.75	95.36	86.00	44.00	40.00	75.91	63.41
47	66.19	GPT-3.5-Turbo (Prompting)	59.01	67.74	65.25	48.58	44.50	86.00	78.00	55.00	69.97	87.80
48	66.18	Hermes-2-Pro-Llama3-8B (FC)	62.32	74.96	61.62	57.83	68.71	90.00	80.00	57.50	55.16	53.66
49	65.44	Hermes-2-Pro-Mistral-7B (FC)	60.98	71.49	60.38	50.42	60.50	90.00	84.00	62.50	38.55	75.61
50	64.83	Hermes-2-Theta-Llama3-8B (FC)	58.53	67.82	59.62	58.33	69.14	88.00	78.00	55.00	62.66	51.22
51	62.70	Llama3-8B-Instruct (Prompt)	58.53	70.26	53.50	53.25	84.50	88.00	68.00	50.00	22.88	78.05
52	61.89	Claude-3-Opus (FC)	69.41	79.95	39.38	27.92	84.64	86.00	52.00	30.00	76.40	73.17
53	60.82	Open-Mixtral-8x7b (Prompt)	61.49	70.70	47.12	36.83	71.86	74.00	56.00	52.50	71.84	65.85
54	60.34	Claude-3-Haiku (Prompt)	74.64	84.49	51.88	45.17	89.43	94.00	32.00	27.50	18.90	85.37
55	58.89	Open-Mixtral-8x22b (FC Any)	73.23	85.42	10.75	63.08	92.57	92.00	24.00	47.50	0.34	100.00

Table 5: Performance comparison on BFCL-v2 leaderboard (cutoff date 09/19/2024). The rank is based on the overall accuracy, which is a weighted average of different evaluation categories. “FC" stands for function-calling mode in contrast to using a customized “prompt" to extract the function calls. See (Yan et al., 2024) for details.