Hephaestus: Improving Fundamental Agent Capabilities of Large Language Models Through Continual Pre-Training

Yuchen Zhuang[1*] **Jingfeng Yang**[2] **Haoming Jiang**[2] **Xin Liu**[2] **Kewei Cheng**[2]
Sanket Lokegaonkar[2] **Yifan Gao**[2] **Qing Ping**[2] **Tianyi Liu**[2] **Binxuan Huang**[2]
Zheng Li[2] **Zhengyang Wang**[2] **Pei Chen**[2] **Ruijie Wang**[2] **Rongzhi Zhang**[1]
Nasser Zalmout[2] **Priyanka Nigam**[2] **Bing Yin**[2] **Chao Zhang**[1]
[1] Georgia Institute of Technology [2] Amazon

Abstract

Due to the scarcity of agent-oriented pre-training data, LLM-based autonomous agents typically rely on complex prompting or extensive fine-tuning, which often fails to introduce new capabilities while preserving strong generalizability. We introduce `Hephaestus-Forge`, the first large-scale pre-training corpus designed to enhance the fundamental capabilities of LLM agents in API function calling, intrinsic reasoning and planning, and adapting to environmental feedback. `Hephaestus-Forge` comprises 103B agent-specific data encompassing 76,537 APIs, including both tool documentation to introduce knowledge of API functions and function calling trajectories to strengthen intrinsic reasoning. To explore effective training protocols, we investigate scaling laws to identify the optimal recipe in data mixing ratios. By continual pre-training on `Hephaestus-Forge`, `Hephaestus` outperforms small- to medium-scale open-source LLMs and rivals commercial LLMs on three agent benchmarks, demonstrating the effectiveness of our pre-training corpus in enhancing fundamental agentic capabilities and generalization of LLMs to new tasks or environments.

1 Introduction

Large language models (LLMs) are rapidly evolving beyond traditional natural language processing tasks (Ouyang et al., 2022; Brown et al., 2020; Achiam et al., 2023), demonstrating increasing intelligence and autonomy by exhibiting capabilities in perception, reasoning, planning, and action within complex real-world environments (Yao et al., 2023; Lu et al., 2024; Sun et al., 2024a). Through well-crafted prompting or extensive post-training, LLM-based autonomous agents augmented with

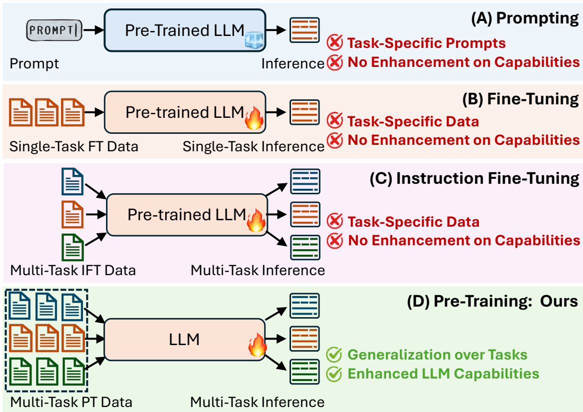


Figure 1: Training paradigms of LLM agents. *Prompting* alone fails to introduce new knowledge and capabilities, while heavy *fine-tuning* can hinder generalization and degrade performance in non-agent use cases, potentially suppressing the original base model capabilities.

external tools (*e.g.*, APIs) have demonstrated exceptional instruction-following capabilities in a wide range of tasks (Schick et al., 2024; Qin et al., 2024; Srinivasan et al., 2023; Zeng et al., 2023).

Despite their remarkable task-specific performance, existing LLM agents often face the following challenges: (1) **Overemphasis on instruction fine-tuning while ignoring the pre-training stage.** LLMs typically undergo a two-stage training process: pre-training to learn general knowledge and instruction fine-tuning to align to specific tasks and user preferences. The *Superficial Alignment Hypothesis* (Zhou et al., 2024; Gudibande et al., 2024; Lin et al., 2024b) posits that LLMs acquire most of their knowledge during pre-training, which is more important than instruction fine-tuning in terms of obtaining generalizable fundamental capabilities. However, the majority of existing agent frameworks (Figure 1) focus on instruction fine-tuning to align with specific patterns or formats, rather than fundamentally enhancing model knowledge or capabilities (*e.g.*, API function calling). (2) **Scarcity of agent-oriented pre-training data.**

*Work done during Yuchen's internship at Amazon. Correspondence to: Yuchen Zhuang (`yczhuang@gatech.edu`), Jingfeng Yang (`jingfengyangpku@gmail.com`), Chao Zhang (`chaozhang@gatech.edu`).

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 6041–6068
April 29 - May 4, 2025

Agent instructions and trajectories significantly differ from general instructions and responses (Zhang et al., 2024b). Thus, function-calling knowledge is difficult to derive directly from web archives, the primary pre-training data source. This notable lack of agent-specific pre-training corpora constrains LLMs from effectively acquiring new agentic knowledge and capabilities (Table 1). (3) **Limited generalization across multiple tasks.** LLM agents often struggle to generalize to new scenarios (*e.g.*, from single to multiple tools) that differ from their original fine-tuning data distributions (Qin et al., 2024).

To address these challenges, we introduce Hephaestus-Forge, a large-scale pre-training corpus specifically designed to enhance the fundamental capabilities of LLM agents in API function calling, intrinsic reasoning and planning, and adaptation to environmental feedback. Specifically, we focus on two primary objectives: (a) improving *comprehension of individual function calls*, and (b) strengthening *intrinsic reasoning capabilities* for solving problems requiring multiple function calls. To enhance (a) comprehension of API functions and alignment with their formats, we collect a large-scale dataset of tool documentation tailored for LLM pre-training on API function calls. Given the expanding range of tasks with growing complexity, we incorporate a vast number of function calling trajectories to improve (b) intrinsic reasoning abilities in sequencing API function calls. We then integrate this meticulously curated tool documentation and function-calling data with code (to bolster reasoning capabilities) and text data (to maintain robust text generation capabilities), creating a *multi-source*, *large-scale*, and *high-quality* training corpus, Hephaestus-Forge.

Building upon Hephaestus-Forge, we introduce a continual pre-trained open-source LLM, Hephaestus, an LLM with strong agentic and autonomous capabilities across domains, bringing open-source models closer to the capabilities of commercial LLMs. Our empirical evaluations demonstrate that Hephaestus-8B outperforms open-source LLMs at small to medium scales (*e.g.*, 9.6% over LLaMA-3-8B and 17.6% over Mixtral-8x22B) and performs comparably to API-based large commercial LLMs (*e.g.*, 18.9% over Claude-3-Haiku and 4.1% over GPT-3.5-turbo) across three agent benchmarks. Our large-scale ablation studies further demonstrate the effectiveness of retrieved agent data in scaling up and diversifying the coverage of scenarios in pre-training. Our contributions can be summarized as follows:

- We curate Hephaestus-Forge, a large-scale pre-training corpus designed to enhance understanding of API function calls and guide actionable trajectories for LLM agents. Remarkably, through exhaustive scaling law experiments, we discover a pioneering pre-training recipe with an empirically optimal data mix ratio.
- We propose Hephaestus, a foundation model that exhibits enhanced fundamental agentic capabilities, including API function calling, intrinsic reasoning and planning, and adaptation to environmental feedback, achieved through continual pre-training on Hephaestus-Forge.
- We extensively compare Hephaestus with strong baselines across three agent benchmarks, verifying its enhanced fundamental agentic capabilities and superior generalization derived from Hephaestus-Forge.

2 Related Work

Prompting-based LLM Agents. Due to the lack of agent-specific pre-training corpus, existing LLM agents rely on either prompt engineering (Hsieh et al., 2023; Lu et al., 2024; Yao et al., 2023; Wang et al., 2023) or instruction fine-tuning (Chen et al., 2023; Zeng et al., 2023) to understand human instructions, decompose high-level tasks, generate grounded plans, and execute multi-step actions. However, prompting-based methods mainly depend on the capabilities of backbone LLMs (usually commercial LLMs), failing to introduce new knowledge and struggling to generalize to unseen tasks (Sun et al., 2024a; Zhuang et al., 2024a).

Instruction Finetuning-based LLM Agents. Considering the extensive diversity of APIs and the complexity of multi-tool instructions, tool learning inherently presents greater challenges than natural language tasks, such as text generation (Qin et al., 2024). Post-training techniques focus more on instruction following and aligning output with specific formats (Patil et al., 2023; Hao et al., 2024; Qin et al., 2024; Schick et al., 2024), rather than fundamentally improving model knowledge or capabilities. Moreover, heavy fine-tuning can hinder generalization or even degrade performance in non-agent use cases, potentially suppressing the original base model capabilities (Ghosh et al., 2024).

Pretraining-based LLM Agents. While pre-training serves as an essential alternative, prior

Methods	Datasets	Training Paradigm	# PT Data (Tokens)	# IFT Data (Samples)	# APIs	Code	Nat. Lang.	Action Traj.	API Doc.	Func. Call	Multi. Step	Plan Refine	Multi. Turn
Instruction Finetuning-based LLM Agents for Intrinsic Reasoning													
FireAct (Chen et al., 2023)	FireAct	IFT	-	2.1K	10	✗	✓	✓	✗	✓	✗	✓	✗
ToolAlpaca (Tang et al., 2023)	ToolAlpaca	IFT	-	4.0K	400	✗	✓	✓	✗	✓	✗	✓	✗
ToolLLaMA (Qin et al., 2024)	ToolBench	IFT	-	12.7K	16,464	✗	✓	✓	✗	✓	✓	✓	✓
AgentEvol (Xi et al., 2024)	AgentTraj-L	IFT	-	14.5K	24	✗	✓	✓	✗	✓	✗	✗	✓
Lumos (Yin et al., 2024)	Lumos	IFT	-	20.0K	16	✗	✓	✓	✗	✓	✓	✗	✓
Agent-FLAN (Chen et al., 2024b)	Agent-FLAN	IFT	-	24.7K	20	✗	✓	✓	✗	✓	✓	✗	✓
AgentTuning (Zeng et al., 2023)	AgentInstruct	IFT	-	35.0K	-	✗	✓	✓	✗	✓	✗	✗	✓
Instruction Finetuning-based LLM Agents for Function Calling													
NexusRaven (Srinivasan et al., 2023)	NexusRaven	IFT	-	-	116	✓	✓	✓	✗	✓	✗	✗	✗
Gorilla (Patil et al., 2023)	Gorilla	IFT	-	16.0K	1,645	✓	✗	✗	✓	✓	✗	✗	✗
OpenFunctions-v2 (Patil et al., 2023)	OpenFunctions-v2	IFT	-	65.0K	-	✓	✓	✗	✓	✓	✗	✗	✗
API Pack (Guo et al., 2024b)	API Pack	IFT	-	1.1M	11,213	✓	✗	✓	✗	✓	✗	✗	✗
LAM (Zhang et al., 2024a)	AgentOhana	IFT	-	42.6K	-	✓	✓	✓	✗	✓	✗	✓	✓
xLAM (Liu et al., 2024e)	APIGen	IFT	-	60.0K	3,673	✓	✓	✓	✗	✓	✗	✓	✓
Pretraining-based LLM Agents													
`Hephaestus`	`Hephaestus-Forge`	PT	103B	95.0K	76,537	✓	✓	✓	✓	✓	✓	✓	✓

Table 1: Summary of existing instruction finetuning-based LLM agents for intrinsic reasoning and function calling, along with their training resources and sample sizes. "PT" and "IFT" denote "Pre-Training" and "Instruction Fine-Tuning", respectively.

works (Nijkamp et al., 2023; Roziere et al., 2023; Xu et al., 2024; Patil et al., 2023) have primarily focused on improving task-specific capabilities (*e.g.*, code generation) instead of general-domain LLM agents, due to single-source, uni-type, small-scale, and poor-quality pre-training data. Existing tool documentation data for agent training either lacks diverse real-world APIs (Patil et al., 2023; Tang et al., 2023) or is constrained to single-tool or single-round tool execution. Furthermore, trajectory data mostly imitate expert behavior or follow function-calling rules with inferior planning and reasoning, failing to fully elicit LLMs' capabilities and handle complex instructions (Qin et al., 2024). Given a wide range of candidate API functions, each comprising various function names and parameters available at every planning step, identifying globally optimal solutions and generalizing across tasks remains highly challenging.

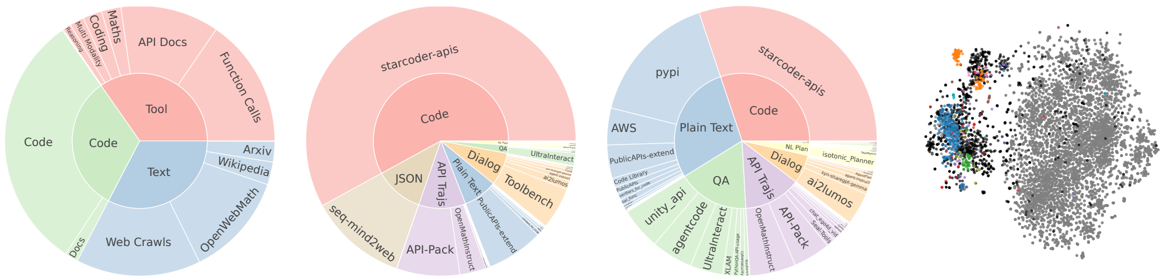
3 Preliminaries

Problem Formulation. We conceptualize leveraging LLMs as autonomous agents for problem-solving as a planning process. Initially, we augment the LLM agent with access to a pool of candidate API functions, denoted as $\mathcal{A} = \{\text{API}_0, \text{API}_1, \cdots, \text{API}_m\}$, along with a natural language task description $g \in \mathcal{G}$ from the task space $\mathcal{G}$. The objective of the LLM agent is to translate the task description g into an ordered sequence of T_g API function calls $p_g = \{a_0, \cdots, a_{T_g}\}$. Specifically, considering the task description g as the initial state s_0, we then sample the plan p_g by prompting the LLM agent with the API definitions $\mathcal{I}$ and demonstration samples $\mathcal{D}$ as follows: $p_g \sim \rho(a_0, a_1, \cdots, a_{T_g}|s_0; \mathcal{I}, \mathcal{D}) : \mathcal{G} \times \mathcal{I} \times \mathcal{D} \to \Delta(\mathcal{A}^{T_g})$, where $\Delta(\cdot)$ denotes a probability simplex function. The final output is derived after executing the entire plan $y \sim \pi(y|s_0, a_1, a_2, \cdots, a_{T_g})$, where $\pi(\cdot)$ denotes a plan executor.

During this procedure, we focus on three fundamental capabilities of LLM agents:

Accurate Function Calling. It involves accurately understanding the API definitions and demonstration samples to generate correct API function calls with corresponding parameters in a given scenario. Specifically, the model should accurately understand the API definitions $\mathcal{I}$ and demonstration samples $\mathcal{D}$, as well as generate an accurate API function call in the given scenario $p(a_t|s_0, a_1, \cdots, a_{t-1}, \mathcal{I}, \mathcal{D})$, where a_t is the ground-truth API function call with corresponding parameters at t-th step.

Intrinsic Reasoning and Planning. It refers to the intrinsic reasoning and planning ability to devise a sequence of multiple tool functions as a solution when addressing complex (multi-step) real-world problems. In such cases, LLMs are often required to generate a sequence of API function calls, $p(a_1, a_2, \cdots, a_{T_g}|s_0; \mathcal{I}, \mathcal{D})$, where $\{a_1, a_2, \cdots, a_{T_g}\}$ constitutes the ground-truth solution plan of length T_g. This process relies on



(a) Hephaestus-Forge (b) Tool Data (c) Retrieved Data (d) t-SNE: Retrieved Data

Figure 2: Data composition of (a) the entire Hephaestus-Forge, (b) seed data collection (§ 4.1), and (c) retrieved agent data from the open web (§ 4.2). A t-SNE visualization (d) depicts seed data (**colorful** points, with each color representing different data sources), retrieved data (**black**), and general text (**gray**) within the semantic space, where retrieved data is closer to the selected seed data than to the general text. Detailed data sources are in appendix A.1.

intrinsic reasoning embedded within the model parameters; enhanced reasoning capabilities lead to a solution plan with a higher chance of success.

Adaptation with Environment Feedback. It focuses on adapting the current plan or action based on environmental feedback when the environments support interaction with the LLM agent. When such feedback is available, it is crucial for the agent to adjust its actions accordingly: $p(a_t|s_0, a_1, o_1, a_2, \cdots, o_{t-1}; \mathcal{I}, \mathcal{D})$, where o_k represents the feedback from the environment after the k-th action. Incorporating environmental feedback allows the agent to take reflections to refine its plan and improve task performance iteratively.

4 `Hephaestus-Forge`

To scale and diversify the pre-training corpus for LLM agents, we introduce a three-stage construction process for Hephaestus-Forge (see Figure 2): (1) **Seed Data Collection** (§ 4.1), where we gather initial high-quality samples; (2) **Web Data Retrieval** (§ 4.2), which expands the seed data by retrieving relevant data from the web; and (3) **Data Quality Control** (§ 4.3), where we ensure the integrity and relevance of the collected data.

4.1 Seed Data Collection

For seed data collection, we first traverse available public resources to gather high-quality API documentation and action trajectories, including: (1) **Public APIs.** High-quality API documentation is collected from over $1,400$ public APIs and official websites, including detailed function definitions and parameter descriptions. (2) **Public Repositories.** To improve intrinsic reasoning, we integrate action trajectories from over 60 public repositories across diverse domains, such as programming code and web interactions. (3) **Code-to-Text Synthesis.** Given the limited coverage of curated data, we use LLMs to synthesize additional API documentation from *StarCoder-API*, generating examples based on code snippets. (4) **Simulated Agent Data.** We gather simulated action sequences with observational data to facilitate adaptation to environmental feedback. Importantly, we offer step-by-step details of the seed data collection process in appendix D.1 for reproducibility.

4.2 Web Data Retrieval

Given the limited availability of agent-oriented data, we use the high-quality data described in § 4.1 as seed data for further expansion. To enhance agentic capabilities, we retrieve a diverse set of examples from web crawls, focusing on content relevant to API documentation and action trajectories. Our retrieval process involves the following steps: (1) **Web Data Corpus Creation.** Similar to CommonCrawl (Raffel et al., 2020) and FineWeb (Penedo et al., 2024), we first compile a large-scale web data corpus. (2) **Semantic Matching.** We utilize COCO-DR (Yu et al., 2022) to encode semantic representations of documents in the seed data and the large-scale web corpus. We then retrieve the top-K similar documents by calculating the cosine similarity between the corresponding embeddings. It allows us to identify and retrieve documents from the web corpus that are semantically similar to our seed data, effectively enriching our dataset with relevant and diverse information. (3) **Quality Control.** To ensure the quality of the retrieved corpus, we perform data pruning to remove semantically redundant content and maintain the diversity of knowledge, preventing overrepresentation of certain topics and ensuring generalization and robustness across domains.

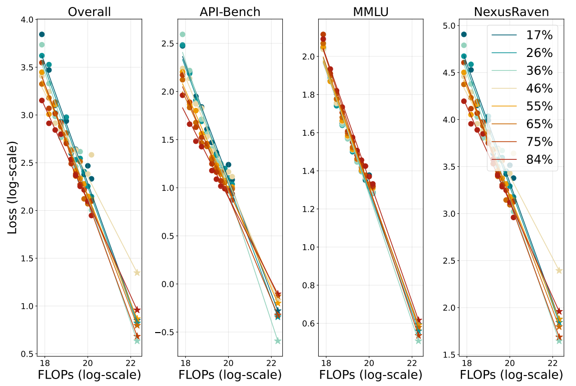


Figure 3: Scaling law of the relationship between agent data mixing ratio (%) and benchmark loss.

4.3 Data Quality

After retrieving semantically relevant data from the web corpus, we obtain a collection of noisy agent data. To ensure the integrity and relevance of our dataset, it is essential to consistently monitor data quality and filter out content that resembles general text rather than agent-specific data. First, we employ `Claude-3-Sonnet` (Anthropic, 2024) as the data annotator to annotate a total of $71,473$ samples from the retrieved data, identifying $37,714$ as agent-relevant and $33,767$ as general text paragraphs. Using the annotated samples, we train a `fastText` (Joulin, 2016) model to effectively recall additional agent-relevant web data. This filtering process then reduces the data volume from approximately 200B to 80B tokens, ensuring that the preserved data maintains high relevance and quality. See details in appendix D.2.

5 Scaling Laws for Data Composition

When designing LLMs, the scaling law (Kaplan et al., 2020; Hoffmann et al., 2022) is an important predictive tool that can estimate the performance (*e.g.*, benchmark loss) of a large-sized target model using a scaling curve fitted over much smaller models (referred to as sampling models). We develop scaling laws to determine the optimal data proportion among agent data, text data, and code data. With the total budget of the data volume fixed, our scaling law experiments show that the effect of agent data ratio x on the loss $\mathcal{L}$ of a pre-trained model follows power laws:

$$\mathcal{L} = c + kx^{\alpha},$$

where c, k, and α are parameters to be fitted. By fitting these parameters using a collection of small models, training data, or computational resources, scaling laws can extrapolate to precisely predict the test loss of larger cases over orders of magnitude.

Scaling Law Experiments. Concretely, we construct our scaling laws by pre-training models ranging in 45M to 0.65B parameters. To simulate the continual pre-training setting, we amplify the target data volume used for training each small model to $50\times$ model parameters. Consequently, the total compute budgets for the scaling law experiments span from 7×10^{17} to 2×10^{20} FLOPs. Regarding data proportions, we begin with the seed agent data and progressively incorporate the retrieved web corpus to increase the agent data ratio. Concurrently, as the agent data ratio increases, we proportionally decrease the volumes of general text and code data to maintain the fixed total data volume. Following Dubey et al. (2024), we leverage the benchmark loss of Nexus (Srinivasan et al., 2023), API-Bank (Li et al., 2023b), API-Bench (Patil et al., 2023) to monitor the agent capabilities, and MMLU (Hendrycks et al., 2020) to monitor the general capabilities of LLMs.

Optimal Data Mixing Ratio. Figure 3 illustrates that the optimal mixture of agent data within the entire pre-training corpus is approximately 36%, indicating that the proportion of agent data, text data, and code data should be roughly $1:1:1$. This balanced distribution promotes both specialized agent capabilities and general language understanding, ensuring that the model remains versatile and robust across diverse tasks and domains.

Remark. The established scaling laws provide critical insights into the data composition for pre-training LLM agents. By identifying the optimal ratio of agent data, we ensure that the model effectively balances specialized agentic capabilities with general language proficiency.

6 `Hephaestus`

In this section, we propose `Hephaestus`, a foundation model with enhanced fundamental capabilities of LLM agents. `Hephaestus` undergoes a two-stage continual pre-training process, followed by instruction fine-tuning (see Figure 4): (1) **Stage I**, continual pre-training stage on the entire `Hephaestus-Forge` corpus to inject general agent knowledge (§ 6.1); (2) **Stage II**, continual pre-training stage on the high-quality seed set of `Hephaestus-Forge` to further enhance specific capabilities (§ 6.1); and (3) **Stage III**, instruction

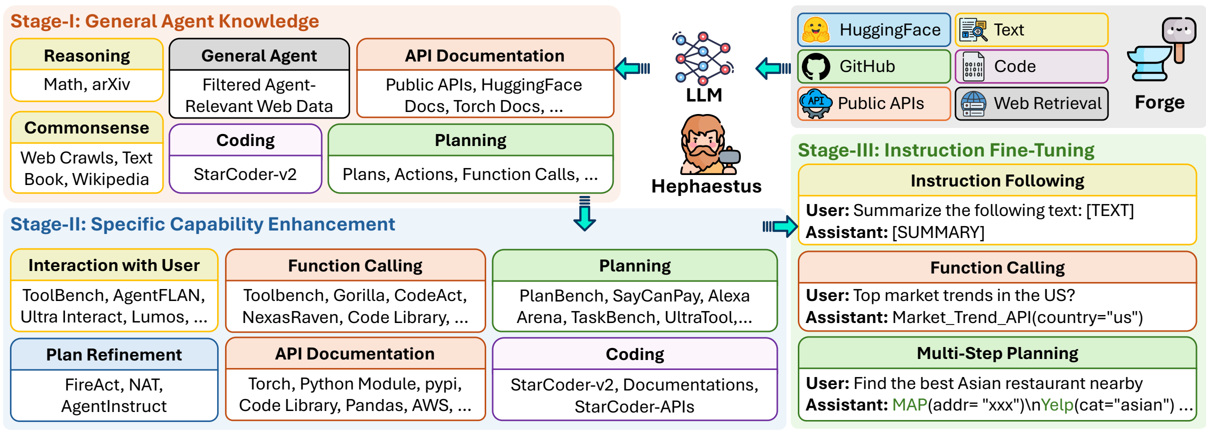


Figure 4: Overview of the pre-training (Stages I & II) and instruction fine-tuning (III) framework in Hephaestus.

fine-tuning to follow general instructions and downstream task requirements (§ 6.2).

6.1 Stage I & II: Continual Pre-Training

Following Caccia et al. (2022); Lange et al. (2023), we revisit the concept of *stability gap*, which describes the phenomenon where the performance on old tasks initially drops and then recovers when learning a new task. Specifically, in the continual pre-training of LLMs, if the data distribution shifts too significantly between the initial pre-training and the continual pre-training stages, the model's capabilities can deteriorate markedly until it assimilates knowledge from the new data distribution (Guo et al., 2024a). To this end, we propose a two-stage continual pre-training framework:

Stage I: Injecting General Agent Knowledge. Stage I infuses general agent knowledge, accompanied by commonsense knowledge and code snippets. We pre-train Hephaestus on the entire Hephaestus-Forge, whose data distribution is carefully balanced between general corpus and agent-specific data, facilitating a smooth and gradual integration of agent knowledge.

Stage II: Enhancing Agent-Specific Capabilities. Stage II leverages high-quality agent data to further enhance the specific capabilities of an agent LLM, including user interaction, function calling, planning, plan refinement, and coding capabilities. We continually pre-train the model obtained from Stage I on the high-quality seed data in § 4.1 to further align the behavior with agent-specific requirements, ensuring that the specialized functionalities are robustly learned and integrated.

Pre-Training Objectives. For both stages, we employ language modeling as the primary pre-training task. The objective is to auto-regressively predict the next token, defined as follows:

$$\mathcal{L}_{\text{PT}} = -\mathbb{E}_{\mathbf{x}\in\mathcal{D}_{\text{PT}}} \sum_{i=1}^{n} p(x_i|\mathbf{x}_{<i}),$$

where $\mathcal{D}_{PT}$ denotes the pre-training data, and x_i represents the i-th token in the training sample $\mathbf{x}$.

6.2 Stage III: Instruction Fine-Tuning

To further improve its instruction-following capabilities to align with complex agent environments, Hephaestus undergoes instruction fine-tuning on a blend of high-quality instruction-completion datasets, including *ShareGPT* (Chiang et al., 2023), *ToolACE* (Liu et al., 2024c), and *AgentFlan* (Chen et al., 2024b). The Stage III employs a negative log-likelihood loss function, defined as:

$$\mathcal{L}_{\text{IFT}} = -\mathbb{E}_{(\mathbf{x},\mathbf{y})\in\mathcal{D}_{\text{IFT}}} \sum_{i=1}^{n} p(y_i|\mathbf{y}_{<i},\mathbf{x}),$$

where $\mathbf{x}$ represents the given instruction, and $\mathbf{y}$ is the expected solution to fill. Here, $(\mathbf{x},\mathbf{y}) \in \mathcal{D}_{\text{IFT}}$ indicates that the data pairs are sampled from the instruction-tuning dataset.

7 Experiments

7.1 Experiment Setup

Tasks and Datasets. We mainly evaluate our Hephaestus on the following benchmarks: (1) *AgentBench* (Liu et al., 2024d) for intrinsic reasoning and adaptation to environment feedback; (2) *Berkeley Function Calling Leaderboard (BFCL)-v3* and (3) *BFCL-v2* (Patil et al., 2023) for accurate function calling. To test generalizability instead of memorization, we intentionally exclude all evaluation benchmarks from pre-training corpora. Task and dataset details are available in appendix A.3.

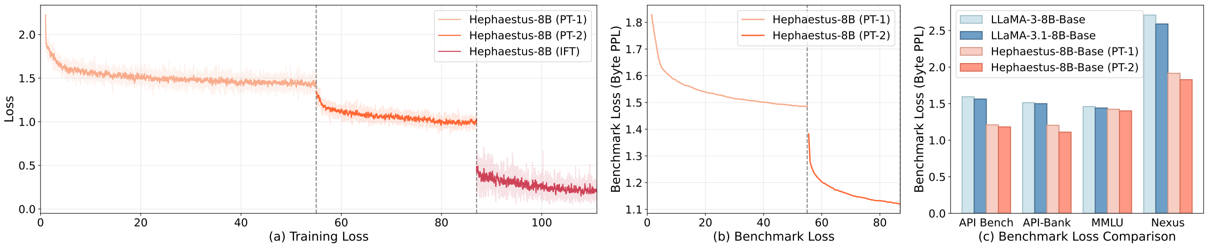


Figure 5: Training and benchmark loss. (a) Training loss of Hephaestus during continual pre-training and instruction fine-tuning. (b) Benchmark loss at periodic training checkpoints and (c) a comparison across base models.

Baselines. We mainly compare to the following baselines: (1) *Base LLMs* and (2) *Open-Source Instruction Fine-Tuned LLMs* with varying model sizes. We also show the performance of (3) *API-based Commercial LLMs* as reference. We exclude prompting and instruction fine-tuned agent frameworks from our main experiments to focus on evaluating the fundamental agentic capabilities of LLMs. Details of baseline models are in appendix B.

Evaluation. Following Liu et al. (2024d); Patil et al. (2023), for AgentBench, we report *success rate* for the OS, DB, HH, and WB environments, *F1 score* for the KG environment, and *reward score* for the WS environment; for BFCL-v2 and -v3, we use *accuracy* as the primary metric for all scenarios to assess correct function calls. Implementation details can be found in appendix E.

7.2 Main Experiments: Hephaestus-8B-Base

Following Shao et al. (2024); Dubey et al. (2024), we evaluate our two-stage pre-trained Hephaestus-8B-Base on three agent-specific benchmarks (API-Bank, API-Bench, NexusRaven) and one general benchmark (MMLU). We observe that incorporating more agent data during pre-training consistently reduces benchmark loss on agent tasks in Figure 5 (b). Additionally, Figure 5 (c) demonstrates that Hephaestus-8B-Base achieves significantly lower benchmark loss compared to the LLaMA-3-8B series of base models. Furthermore, Table 2 reports the benchmark scores, where Hephaestus-8B-Base leads in performance across all benchmarks among the open-source base models. Our findings indicate that both pre-training stages (I & II) enhance Hephaestus's fundamental capabilities across a wide range of agent tasks without compromising general capabilities.

7.3 Main Experiments: Hephaestus-8B-IFT

Table 2 presents the main experimental results of instruction fine-tuned Hephaestus and baselines. Hephaestus consistently outperforms small to medium size open-source LLMs. Moreover, Hephaestus-8B-IFT remains competitive compared to baseline models with significantly more parameters or commercial LLMs.

Enhanced Capabilities Through Pre-training. We conduct a direct comparison between Hephaestus and LLaMA-3-8B-Base (Dubey et al., 2024), both instruction-tuned using the same instruction fine-tuning data. Hephaestus-8B-IFT outperforms LLaMA-3-8B-IFT across all three benchmarks, indicating that the observed improvements can be attributed to the pre-training stage. Moreover, incorporating more domain-specific knowledge during the pre-training stage leads to better performance, without requiring additional instruction fine-tuning data.

Excelling in Complex Multi-turn Tasks. BFCL-v3, the latest benchmark, emphasizes multi-turn tool function-calling tasks requiring intrinsic reasoning capabilities and function-calling proficiency. Due to its recent introduction, the limited availability of task-specific data for instruction-tuning has led to suboptimal performance, particularly in multi-turn function-calling accuracy, as observed with models like Groq-8B-Tool-Use (Groq, 2024). In contrast, Hephaestus exhibits significantly better performance on BFCL-v3, suggesting that its improvements in core agentic capabilities and generalization stem from pre-training on our large-scale, diverse agent-oriented corpus.

7.4 Ablation Studies

Table 3 presents the ablation results of Hephaestus on AgentBench and BFCL-v2.

Effect of Pre-Training Stages. Removing the second pre-training stage results in a slight performance decline for both base and instruction-tuned models across all tasks. Although the Stage-I pre-training data, comprising a large volume of general and retrieved agent data from the web, brings the Hephaestus-Forge closer to the general data distribution, it still differs from the data used in down-

Datasets (→)	Model		AgentBench							BFCL-v3					BFCL-v2
Models (↓)	**Size**	**Type**	**OA**	**OS**	**DB**	**HH**	**KG**	**WB**	**WS**	**OA**	**NL-AST**	**Exec**	**L-AST**	**MT**	**OA**
Base LLMs															
LLaMA-3-8B (Dubey et al., 2024)	8B	OSS	0.56	2.8	12.0	0.0	8.9	11.0	1.4	17.73	4.3	2.5	39.1	0.0	17.77
LLaMA-3.1-8B (Dubey et al., 2024)	8B	OSS	1.05	15.3	5.3	8.0	12.7	**18.0**	41.9	19.50	16.3	10.7	37.5	0.0	21.08
Hephaestus-8B-Base	8B	OSS	**1.87**	**20.8**	**32.3**	**30.0**	**16.0**	16.0	**60.5**	**22.12**	**18.1**	**12.1**	**42.2**	**4.0**	**25.18**
Open-Source Instruction Fine-Tuned LLMs (Small)															
LLaMA-2-7B-Chat (Touvron et al., 2023)	7B	OSS	0.36	4.2	8.0	0.0	2.1	7.0	11.6	-	-	-	-	-	-
Vicuna-7B-v1.5 (Chiang et al., 2023)	7B	OSS	0.43	9.7	8.7	0.0	2.5	9.0	2.2	-	-	-	-	-	-
CodeLLaMA-7B-Instruct (Roziere et al., 2023)	7B	OSS	0.65	4.9	12.7	0.0	8.2	12.0	25.2	-	-	-	-	-	-
CodeLLaMA-13B-Instruct (Roziere et al., 2023)	13B	OSS	0.74	3.5	9.7	0.0	10.4	14.0	43.8	-	-	-	-	-	-
LLaMA-2-13B-Chat (Touvron et al., 2023)	13B	OSS	0.66	4.2	11.7	6.0	3.6	13.0	25.3	-	-	-	-	-	-
Vicuna-13B-v1.5 (Chiang et al., 2023)	13B	OSS	0.86	10.4	6.7	8.0	9.4	12.0	41.7	-	-	-	-	-	-
Groq-8B-Tool-Use (Groq, 2024)	8B	OSS	1.27	15.3	11.7	4.0	17.6	<u>23.0</u>	53.4	30.44	42.8	35.5	45.5	0.0	**89.06**
LLaMA-3-8B-Instruct (Dubey et al., 2024)	8B	OSS	1.51	18.1	12.3	24.0	15.9	19.0	56.1	35.79	60.6	66.2	48.4	0.5	59.57
LLaMA-3.1-8B-Instruct (Dubey et al., 2024)	8B	OSS	1.74	<u>21.5</u>	5.3	<u>34.0</u>	18.4	**25.0**	59.5	46.76	70.3	76.5	<u>62.2</u>	2.5	61.39
LLaMA-3-8B-IFT	8B	OSS	<u>2.07</u>	**22.2**	<u>29.7</u>	32.0	**25.3**	19.0	**66.1**	<u>48.52</u>	<u>72.5</u>	<u>81.8</u>	**66.8**	<u>2.6</u>	62.12
Hephaestus-8B-IFT	8B	OSS	**2.29**	20.8	**41.7**	**46.0**	<u>21.2</u>	17.0	<u>63.9</u>	**50.59**	**84.3**	**86.2**	60.1	**9.6**	<u>70.78</u>
For Reference: Open-Source Instruction Fine-Tuned LLMs (Medium to Large) and API-based Commercial LLMs															
LLaMA-2-70B-Chat (Touvron et al., 2023)	70B	OSS	0.66	9.7	13.0	2.0	8.0	19.0	5.6	-	-	-	-	-	-
CodeLLaMA-34B-Instruct (Roziere et al., 2023)	34B	OSS	1.13	2.8	14.0	4.0	23.5	20.0	52.1	-	-	-	-	-	-
Gemini-1.5-Flash (Reid et al., 2024)	-	API	1.81	20.1	46.0	22.0	14.2	17.0	39.1	53.01	77.1	71.2	71.2	13.1	70.75
text-davinci-003 (Ouyang et al., 2022)	-	API	1.90	20.1	16.3	20.0	34.9	26.0	61.7	-	-	-	-	-	-
DeepSeek-v2 (Liu et al., 2024a)	236B	OSS	1.97	20.8	21.7	38.0	21.7	22.0	57.4	-	-	-	-	-	-
Mixtral-8x22B (Jiang et al., 2024)	176B	OSS	2.00	24.3	25.7	14.0	31.1	28.0	62.8	43.00	56.1	59.7	65.3	8.9	63.26
gpt-3.5-turbo-0125 (OpenAI, 2022)	-	API	2.12	32.6	36.7	16.0	25.9	20.0	64.1	51.90	84.5	81.7	59.0	19.1	66.53
Claude-3-Haiku (Anthropic, 2024)	-	API	2.13	14.6	41.0	42.0	27.3	14.0	57.8	38.39	62.6	60.7	58.1	1.6	55.47
Command-R-Plus-FC (Cohere, 2024)	-	API	-	-	-	-	-	-	-	45.22	77.7	77.4	54.2	6.1	76.29
LLaMA-3-70B-Instruct (Dubey et al., 2024)	70B	OSS	2.73	28.6	50.3	44.0	39.5	22.0	53.6	49.55	87.2	87.4	63.4	1.1	84.95
gpt-4-0613 (Achiam et al., 2023)	-	API	4.52	42.4	32.0	78.0	58.8	29.0	61.1	-	-	-	-	-	89.26

Table 2: Main experiments on three agent benchmarks across various model scales. **Bold** and <u>underlined</u> texts represent the best and the second-best results, respectively. Notations are consistent throughout all tables. "OSS", "API", and "OA" denote "Open-Sourced LLMs", "API-based Commercial LLMs", and "Overall", respectively.

Datasets (→)	AgentBench							BFCL-v2
Models (↓)	**OA**	**OS**	**DB**	**HH**	**KG**	**WB**	**WS**	**OA**
Hephaestus-8B-Base	**1.87**	20.8	32.3	30.0	16.0	**16.0**	60.5	**25.18**
w/ Stage-1 PT Only	1.76	20.1	29.0	28.0	**17.5**	14.0	56.1	23.88
w/o Data Filtering	1.85	20.8	**36.3**	28.0	**17.5**	14.0	54.0	21.08
w/o Retrieval Data	1.84	**22.9**	16.7	**48.0**	5.4	**16.0**	**66.4**	19.35
Hephaestus-8B-IFT	**2.29**	20.8	**41.7**	**46.0**	**21.2**	17.0	63.9	**70.78**
w/ Stage-1 PT Only	2.00	20.8	**41.7**	34.0	18.4	10.0	63.9	64.23
w/o Data Filtering	2.10	**23.6**	28.3	44.0	17.2	**18.0**	**64.0**	59.34
w/o Retrieval Data	1.99	21.5	30.3	38.0	17.2	17.0	60.7	49.86

Table 3: Ablation studies on the effect of (1) different pre-training stages and (2) retrieved data.

stream applications and evaluations. The Stage-II pre-training is essential for effectively bridging the gap between the pre-training corpus and the instruction fine-tuning data, thereby enhancing overall model performance.

Effect of Retrieved Data. Degrading the retrieved data to unfiltered, low-quality data or removing it entirely negatively impacts overall performance. For tasks with numerous hand-crafted instructions and simulated trajectories available on the open web (*e.g.*, HH and WS), the seed data of

Models (↓)/Datasets (→)	AgentBench	BFCL-v3	BFCL-v2
Hephaestus-8B-IFT	**2.29**	**51.59**	**70.78**
LLaMA-3-8B-IFT	2.07 (-9.6%)	48.52 (-6.0%)	62.12 (-12.2%)
Groq-8B-Tool-Use (Groq, 2024)	1.27 (-44.5%)	30.44 (-41.0%)	89.06 (+25.8%)
AgentLM-7B (Zeng et al., 2023)	2.36 (+3.1%)	16.67 (-67.7%)	19.18 (-72.9%)
ToolACE-8B (Liu et al., 2024c)	1.48 (-35.3%)	58.20 (+12.8%)	91.41 (+29.2%)

Table 4: Generalization across three agent benchmarks.

Hephaestus-Forge can lead to model overfitting on specific patterns. When the large volume of retrieval data is removed, the seed data predominates, leading to improved performance on these specific tasks but reduced performance on others.

7.5 Cross-Task Generalization

Table 4 compares Hephaestus with several instruction fine-tuned agent frameworks across three agent benchmarks for cross-task generalization. While models fine-tuned on task-specific data excel in corresponding tasks (Groq, 2024; Zeng et al., 2023; Liu et al., 2024c), they struggle to generalize across different agent benchmarks. In contrast, Hephaestus performs consistently well across all tasks, suggesting that the large and diverse pre-

Models (↓) / Datasets (→)	Benchmark Metrics (↑) GSM8K	HumanEval	HumanEval+	BBH	OA	Benchmark Loss (↓) IFEval	hellaswag	MMLU	BBH	OA
LLaMA-3-8B (Dubey et al., 2024)	0.420	0.372	0.317	0.613	0.431	0.648	0.759	0.526	0.361	0.573
Hephaestus-8B-Base	0.460	0.411	0.356	0.584	0.453	0.683	0.769	0.536	0.374	0.591
LLaMA-3-8B-IFT	0.695	0.343	0.337	0.596	0.493	1.046	0.908	0.725	0.503	0.795
Hephaestus-8B-IFT	0.686	0.373	0.373	0.567	0.500	0.657	0.784	0.559	0.369	0.592
ToolACE-8B (Liu et al., 2024c)	0.623	0.385	0.324	0.120	0.363	0.774	0.848	0.602	0.442	0.666
AgentLM-7B (Zeng et al., 2023)	0.549	0.122	0.110	0.071	0.213	0.783	0.915	0.657	0.450	0.701
LLaMA-3-8B-Instruct (Dubey et al., 2024)	0.797	0.646	0.573	0.660	0.669	0.619	0.769	0.533	0.361	0.570

Table 5: Comprehensive evaluation of general model capabilities across diverse benchmarks. Hephaestus maintains general capabilities while achieving competitive performance against baseline and specialized models.

training corpora, Hephaestus-Forge, effectively enhance function calling and agentic reasoning, leading to better generalization. Furthermore, the compared methods are based on continued instruction fine-tuning of LLaMA-3-8B-Instruct, which inherently possesses strong instruction-following and understanding capabilities due to its meticulously curated post-training data. Unlike models relying solely on instruction fine-tuning, the pre-training of Hephaestus effectively improves its fundamental capabilities, thereby offering a more robust foundation for diverse agentic applications.

7.6 Preservation of General Capabilities

To evaluate the preservation of general capabilities, we further conduct comprehensive experiments across seven additional benchmarks (Table 5) besides MMLU, spanning mathematics (Cobbe et al., 2021), software development (Chen et al., 2021; Liu et al., 2024b), logical reasoning (Suzgun et al., 2022), and broad language model abilities (Zhou et al., 2023; Zellers et al., 2019; Hendrycks et al., 2020). Our results demonstrate that Hephaestus maintains comparable performance to the base model across these diverse domains while significantly enhancing agent-specific capabilities.

8 Conclusion

In summary, Hephaestus-Forge and Hephaestus collectively advance open-source LLM-based autonomous agents by addressing critical gaps in pre-training corpora. Through exhaustive scaling law experiments, we identify an empirically optimal data mix ratio of approximately 1:1:1 for agent, code, and text data, maximizing the fundamental and generalization capabilities of LLM agents. Empirical evaluations underscore the efficacy and validity of Hephaestus-Forge in fostering enhanced fundamental agentic capabilities and superior generalization in LLM-based autonomous agents.

Limitations

Data Composition. While knowledge of the composition of pre-training or instruction fine-tuning data would further enhance the effectiveness of Hephaestus, most prominent open-source LLMs (*e.g.*, LLaMA-3-8B-Instruct) do not disclose detailed data information. Nevertheless, our continual pre-training experiments with LLaMA-3-8B demonstrate that significant improvements are achievable even without this knowledge.

Model Scalability. Computational constraints currently restrict our ability to extend these experiments to larger models. In future work, we aim to validate our findings and methodologies on more expansive LLM architectures, pending access to increased computational resources.

Ethical Statement

Data Contamination. A potential concern in our evaluations is test set contamination, which occurs when some task-specific examples overlap with data used during continual pre-training (Oren et al., 2024). To mitigate this issue, we follow Wang et al. (2024b) and conduct a string-matching analysis, which indicates no overlap between our training data and the datasets of the target tasks. Moreover, we intentionally exclude all evaluation benchmark data from both our pre-training and fine-tuning datasets to ensure a fair comparison.

Reproducibility. To promote transparency, reproducibility, and generalizability in our research, we include all details of the dataset construction (*e.g.*, data collection, processing, retrieving, filtering, scaling law, *etc.*) of Hephaestus-Forge in § 4 and the training procedures for Hephaestus in § 6. Experimental setups and results are presented in § 7. Additionally, we detail the pre-training, instruction fine-tuning, and testing tasks and datasets in appendices A.1 to A.3, respectively.

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*.

Prithviraj Ammanabrolu and Mark Riedl. 2021. Modeling worlds in text. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

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

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Lucas Caccia, Rahaf Aljundi, Nader Asadi, Tinne Tuytelaars, Joelle Pineau, and Eugene Belilovsky. 2022. New insights on reducing abrupt representation change in online continual learning. In *International Conference on Learning Representations*.

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*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. 2024a. T-eval: Evaluating the tool utilization capability of large language models step by step. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9510–9529, Bangkok, Thailand. Association for Computational Linguistics.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. 2024b. Agent-FLAN: Designing data and methods of effective agent tuning for large language models. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 9354–9366, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2(3):6.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Cohere. 2024. Introducing command r+: A scalable llm built for business. *Cohere AI Blog*.

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

Qiaozi Gao, Govind Thattai, Suhaila Shakiah, Xiaofeng Gao, Shreyas Pansare, Vasu Sharma, Gaurav Sukhatme, Hangjie Shi, Bofei Yang, Desheng Zhang, et al. 2024. Alexa arena: A user-centric interactive platform for embodied ai. *Advances in Neural Information Processing Systems*, 36.

Sreyan Ghosh, Chandra Kiran Reddy Evuru, Sonal Kumar, Ramaneswaran S, Deepali Aneja, Zeyu Jin, Ramani Duraiswami, and Dinesh Manocha. 2024. A closer look at the limitations of instruction tuning. In *Forty-first International Conference on Machine Learning*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2024. ToRA: A tool-integrated reasoning agent for mathematical problem solving. In *The Twelfth International Conference on Learning Representations*.

Groq. 2024. Introducing llama-3-groq-tool-use models. *Groq Blog*.

Arnav Gudibande, Eric Wallace, Charlie Victor Snell, Xinyang Geng, Hao Liu, Pieter Abbeel, Sergey Levine, and Dawn Song. 2024. The false promise of imitating proprietary language models. In *The Twelfth International Conference on Learning Representations*.

Yiduo Guo, Jie Fu, Huishuai Zhang, Dongyan Zhao, and Yikang Shen. 2024a. Efficient continual pre-training by mitigating the stability gap. *arXiv preprint arXiv:2406.14833*.

Zhen Guo, Adriana Meza Soria, Wei Sun, Yikang Shen, and Rameswar Panda. 2024b. Api pack: A massive multilingual dataset for api call generation. *arXiv preprint arXiv:2402.09615*.

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

Rishi Hazra, Pedro Zuidberg Dos Martires, and Luc De Raedt. 2024. Saycanpay: Heuristic planning with large language models using learnable domain knowledge. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20123–20133.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, pages 30016–30030.

Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. 2023. Tool documentation enables zero-shot tool-usage with large language models. *arXiv preprint arXiv:2308.00675*.

Shijue Huang, Wanjun Zhong, Jianqiao Lu, Qi Zhu, Jiahui Gao, Weiwen Liu, Yutai Hou, Xingshan Zeng, Yasheng Wang, Lifeng Shang, Xin Jiang, Ruifeng Xu, and Qun Liu. 2024. Planning, creation, usage: Benchmarking LLMs for comprehensive tool utilization in real-world complex scenarios. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 4363–4400, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.

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*.

Armand Joulin. 2016. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Diederik P Kingma. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Matthias De Lange, Gido M van de Ven, and Tinne Tuytelaars. 2023. Continual evaluation for lifelong learning: Identifying the stability gap. In *The Eleventh International Conference on Learning Representations*.

Changhao Li, Yuchen Zhuang, Rushi Qiang, Haotian Sun, Hanjun Dai, Chao Zhang, and Bo Dai. 2024. Matryoshka: Learning to drive black-box llms with llms. *arXiv preprint arXiv:2410.20749*.

Haoran Li, Dadi Guo, Wei Fan, Mingshi Xu, Jie Huang, Fanpu Meng, and Yangqiu Song. 2023a. Multi-step jailbreaking privacy attacks on chatgpt. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 4138–4153.

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

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. 2024a. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. *Advances in Neural Information Processing Systems*, 36.

Bill Yuchen Lin, Abhilasha Ravichander, Ximing Lu, Nouha Dziri, Melanie Sclar, Khyathi Chandu, Chandra Bhagavatula, and Yejin Choi. 2024b. The unlocking spell on base LLMs: Rethinking alignment via in-context learning. In *The Twelfth International Conference on Learning Representations*.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. 2024a. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2024b. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, et al. 2024c. 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, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2024d. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, et al. 2024e. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *arXiv preprint arXiv:2406.18518*.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. 2024. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*.

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

Weijie Lv, Xuan Xia, and Sheng-Jun Huang. 2024. Codeact: Code adaptive compute-efficient tuning framework for code llms. *arXiv preprint arXiv:2408.02193*.

Zixian Ma, Weikai Huang, Jieyu Zhang, Tanmay Gupta, and Ranjay Krishna. 2024. m&m's: A benchmark to evaluate tool-use for multi-step multi-modal tasks. In *Synthetic Data for Computer Vision Workshop @ CVPR 2024*.

Mistral. 2024. Large enough. *Mistral AI Blog*.

Yao Mu, Qinglong Zhang, Mengkang Hu, Wenhai Wang, Mingyu Ding, Jun Jin, Bin Wang, Jifeng Dai, Yu Qiao, and Ping Luo. 2024. Embodiedgpt: Vision-language pre-training via embodied chain of thought. *Advances in Neural Information Processing Systems*, 36.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. Codegen: An open large language model for code with multi-turn program synthesis. In *The Eleventh International Conference on Learning Representations*.

OpenAI. 2022. Introducing chatgpt. *OpenAI Blog*.

Yonatan Oren, Nicole Meister, Niladri S. Chatterji, Faisal Ladhak, and Tatsunori Hashimoto. 2024. Proving test set contamination in black-box language models. In *The Twelfth International Conference on Learning Representations*.

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

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. 2024. Openwebmath: An open dataset of high-quality mathematical web text. In *The Twelfth International Conference on Learning Representations*.

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*.

Guilherme Penedo, Hynek Kydlicek, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. 2024. The fineweb datasets: Decanting the web for the finest text data at scale. *Preprint*, arXiv:2406.17557.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

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*.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, YK Li, Yu Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. 2023. Taskbench: Benchmarking large language models for task automation. *arXiv preprint arXiv:2311.18760*.

Wenqi Shi, Ran Xu, Yuchen Zhuang, Yue Yu, Haotian Sun, Hang Wu, Carl Yang, and May D Wang. 2024a. Medadapter: Efficient test-time adaptation of large language models towards medical reasoning. *arXiv preprint arXiv:2405.03000*.

Wenqi Shi, Ran Xu, Yuchen Zhuang, Yue Yu, Jieyu Zhang, Hang Wu, Yuanda Zhu, Joyce Ho, Carl Yang, and May Dongmei Wang. 2024b. Ehragent: Code empowers large language models for few-shot complex tabular reasoning on electronic health records. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 22315–22339.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. {ALFW}orld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*.

Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, et al. 2023. Restgpt: Connecting large language models with real-world restful apis. *arXiv preprint arXiv:2306.06624*.

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*.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2024a. Adaplanner: Adaptive planning from feedback with language models. *Advances in Neural Information Processing Systems*, 36.

Haotian Sun, Yuchen Zhuang, Wei Wei, Chao Zhang, and Bo Dai. 2024b. Bbox-adapter: Lightweight adapting for black-box large language models. In *ICML*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

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

Xiangru Tang, Chunyuan Deng, Hanminwang Hanminwang, Haoran Wang, Yilun Zhao, Wenqi Shi, Yi Fung, Wangchunshu Zhou, Jiannan Cao, Heng Ji, Arman Cohan, and Mark Gerstein. 2024. MIMIR: A customizable agent tuning platform for enhanced scientific applications. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 486–496.

Shubham Toshniwal, Ivan Moshkov, Sean Narenthiran, Daria Gitman, Fei Jia, and Igor Gitman. 2024. Openmathinstruct-1: A 1.8 million math instruction tuning dataset. *arXiv preprint arXiv:2402.10176*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2024. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *Advances in Neural Information Processing Systems*, 36.

Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. 2024a. LLMs in the imaginarium: Tool learning through simulated trial and error. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10583–10604, Bangkok, Thailand. Association for Computational Linguistics.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024b. Improving text embeddings with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11897–11916, Bangkok, Thailand. Association for Computational Linguistics.

Renxi Wang, Haonan Li, Xudong Han, Yixuan Zhang, and Timothy Baldwin. 2024c. Learning from failure: Integrating negative examples when fine-tuning large language models as agents. *arXiv preprint arXiv:2402.11651*.

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.

Mengsong Wu, Tong Zhu, Han Han, Chuanyuan Tan, Xiang Zhang, and Wenliang Chen. 2024. Sealtools: Self-instruct tool learning dataset for agent tuning and detailed benchmark. *arXiv preprint arXiv:2405.08355*.

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*.

Jiannan Xiang, Guangyi Liu, Yi Gu, Qiyue Gao, Yuting Ning, Yuheng Zha, Zeyu Feng, Tianhua Tao, Shibo Hao, Yemin Shi, et al. 2024. Pandora: Towards general world model with natural language actions and video states. *arXiv preprint arXiv:2406.09455*.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. 2023. On the tool manipulation capability of open-sourced large language models. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

Yiheng Xu, Hongjin SU, Chen Xing, Boyu Mi, Qian Liu, Weijia Shi, Binyuan Hui, Fan Zhou, Yitao Liu, Tianbao Xie, Zhoujun Cheng, Siheng Zhao, Lingpeng Kong, Bailin Wang, Caiming Xiong, and Tao Yu. 2024. Lemur: Harmonizing natural language and code for language agents. In *The Twelfth International Conference on Learning Representations*.

Yi Yang, Wen-tau Yih, and Christopher Meek. 2015. WikiQA: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2013–2018, Lisbon, Portugal. Association for Computational Linguistics.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

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)*.

Junjie Ye, Guanyu Li, Songyang Gao, Caishuang Huang, Yilong Wu, Sixian Li, Xiaoran Fan, Shihan Dou, Qi Zhang, Tao Gui, et al. 2024. Tooleyes: Fine-grained evaluation for tool learning capabilities of large language models in real-world scenarios. *arXiv preprint arXiv:2401.00741*.

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. 2024. Agent lumos: Unified and modular training for open-source language agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12380–12403.

Yue Yu, Chenyan Xiong, Si Sun, Chao Zhang, and Arnold Overwijk. 2022. Coco-dr: Combating distribution shift in zero-shot dense retrieval with contrastive and distributionally robust learning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1462–1479.

Lifan Yuan, Ganqu Cui, Hanbin Wang, Ning Ding, Xingyao Wang, Jia Deng, Boji Shan, Huimin Chen, Ruobing Xie, Yankai Lin, Zhenghao Liu, Bowen Zhou, Hao Peng, Zhiyuan Liu, and Maosong Sun. 2024a. Advancing LLM reasoning generalists with preference trees. In *AI for Math Workshop @ ICML 2024*.

Youliang Yuan, Wenxiang Jiao, Wenxuan Wang, Jen tse Huang, Pinjia He, Shuming Shi, and Zhaopeng Tu. 2024b. GPT-4 is too smart to be safe: Stealthy chat with LLMs via cipher. In *The Twelfth International Conference on Learning Representations*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

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*.

Kexun Zhang, Hongqiao Chen, Lei Li, and William Wang. 2023. Syntax error-free and generalizable tool use for llms via finite-state decoding. *arXiv preprint arXiv:2310.07075*.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. 2024. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*.

Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A. Rossi, Somdeb Sarkhel, and Chao Zhang. 2024a. Toolchain*: Efficient action space navigation in large language models with a* search. In *The Twelfth International Conference on Learning Representations*.

Yuchen Zhuang, Haotian Sun, Yue Yu, Rushi Qiang, Qifan Wang, Chao Zhang, and Bo Dai. 2024b. HYDRA: Model factorization framework for black-box LLM personalization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2024c. Toolqa: A dataset for llm question answering with external tools. *Advances in Neural Information Processing Systems*, 36.

Supplementary Materials for Hephaestus

A Task and Dataset Information

A.1 Pre-Training Corpus: `Hephaestus-Forge`

Agent Data Sources. To promote transparency, reproducibility, and potential generalization to novel domains in agent research, we publicly release the training recipe utilized for `Hephaestus-Forge` during the pre-training stage. To enhance the fundamental capabilities of `Hephaestus`, we compile a unique, comprehensive, and large-scale corpus of agent data sources, including API documentation, API function calling trajectories, code, and text data. Tables 9 and 10 provide a comprehensive overview of `Hephaestus-Forge` used in `Hephaestus`, detailing the data sources, their respective sizes, and public availability status. All data sources utilized in `Hephaestus-Forge` are licensed under `Apache-2.0`, `MIT`, or `LGPL-2.1`, permitting non-commercial use and aligning with the research objectives of this work. Examples of task formats in `Hephaestus-Forge` are available in Figure 6.

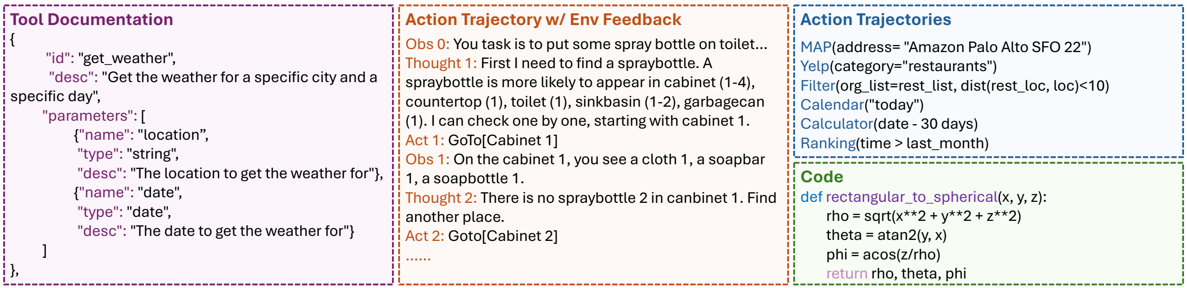


Figure 6: Examples of different task formats in `Hephaestus-Forge`, including tool documentation, action trajectory (w/ environmental feedback), and code data.

Text and Code Data. Since agent data typically includes detailed task descriptions, formatted function calls, and environmental feedback, significant gaps exist between agent data and standard text and code data. Given that current open-sourced LLMs have already been pre-trained on text and code data, and to preserve their generalization ability, it is necessary to mix agent data with text and code data during the continual pre-training stage. For the text data, we primarily select a corpus that covers commonsense reasoning, mathematical reasoning, scientific reasoning, and general text.

- **RedPajama_CommonCrawls**[1] (Raffel et al., 2020) is a large-scale web text dataset collected by the RedPajama project. It encompasses a diverse range of internet texts, including blogs, news articles, forum discussions, and social media posts. Incorporating this dataset helps to preserve general language understanding and generation capabilities, as it captures a wide variety of writing styles and topics, thus offering significant linguistic diversity.
- **Encyclopedic Content** is a comprehensive knowledge base sourced from Wikipedia[2] and WikiQA (Yang et al., 2015). This dataset includes extensively curated articles covering a wide range of human knowledge domains. Incorporating encyclopedic content during continual pre-training helps ensure factual accuracy and reliability in the model's learned information.
- **Textbooks** from OpenStax[3] provide peer-reviewed, openly licensed textbooks for higher education. These textbooks span topics such as mathematics, science, economics, and the humanities. Since textbooks are structured with well-organized chapters and summaries, continual pre-training on this corpus exposes the model to formal educational language and coherent knowledge representation.
- **Mathematical Content** from OpenWebMath (Paster et al., 2024) aggregates open-access mathematical texts, problem sets, and explanations. This dataset spans topics ranging from pure mathematics to applied fields, enabling the model to understand and generate mathematically rigorous content.
- **arXiv Papers**[4] include preprints hosted on arXiv in fields such as physics, mathematics, computer science, and more. This dataset features advanced terminology, methodologies, and academic discourse.

[1] `https://www.together.ai/blog/redpajama-data-v2`
[2] `https://www.wikipedia.org/`
[3] `https://openstax.org/`
[4] `arxiv.org`

Using this data for continual pre-training enhances the model's ability to grasp complex scientific concepts and fosters cross-disciplinary understanding.

• **StarCoder-v2** (Lozhkov et al., 2024) is a large-scale collection of source code curated to advance research in code generation and understanding. We select all documentation samples and randomly sample the remaining portion for inclusion in the `Hephaestus-Forge`. This dataset provides knowledge of complex programming patterns and semantics, which may benefit the tool-function-calling capabilities of LLMs.

A.2 Instruction Fine-Tuning Task and Dataset

The instruction fine-tuning stage empowers LLMs with instruction-following capabilities and aligns LLM agents with task-specific requirements and user preferences. To facilitate direct and fair comparison, we employ a diverse range of tasks for both the instruction fine-tuning baseline model, `LLaMA-3-8B-IFT`, and our model, `Hephaestus-8B-IFT`, including (1) a general conversation dataset, *ShareGPT* (Chiang et al., 2023); (b) a single-tool function-calling conversation dataset, *ToolACE* (Liu et al., 2024c); and (c) a multi-turn planning conversation dataset, *AgentFlan* (Chen et al., 2024b).

• **ShareGPT** (Chiang et al., 2023) is a general dataset comprising real-world conversations from 70K user data, designed to fine-tune models for enhanced instruction-following capabilities. It significantly improves LLMs' ability to handle complex, multi-turn dialogues. The dataset encompasses a wide range of topics and natural, human-generated prompts, enabling models to learn from authentic interactions. By leveraging real user data, ShareGPT allows models to better generalize across diverse tasks and navigate increasingly complex instructions, closely mimicking real-world conversational scenarios.

• **ToolACE** (Liu et al., 2024c) is a single-tool conversation dataset designed to enhance the function-calling capabilities of LLM agents. It comprises 26,507 APIs across 30 primary domains (*e.g.*, entertainment) and is categorized into 390 coarse-grained sub-domains (*e.g.*, music). In addition, ToolACE accommodates complex nested parameters, manages both parallel and dependent function calls, and encompasses a wide variety of tool-related data.

• **AgentFlan** (Chen et al., 2024b) is a multi-turn planning dataset that combines data in two formats: 10% in ReAct format and 90% in conversation format. It encompasses 24,703 instances derived from AgentInstruct and ToolBench. AgentFlan deliberately excludes format-following instructions and common reasoning tasks from its training corpus, aiming to elicit pure agent abilities from LLMs without overfitting to specific format protocols.

A.3 Evaluation Task and Dataset

We conduct the main experiments of `Hephaestus` on three widely used LLM agent benchmarks across a wide range of scenarios, including:

• **AgentBench** (Liu et al., 2024d) presents six distinct environments in a multi-turn, open-ended generation setting: Operating System (OS), Database (DB), Knowledge Graph (KG), House-Holding (HH), Web Shopping (WS), and Web Browsing (WB). We leverage AgentBench to evaluate intrinsic reasoning and adaptation to environmental feedback.

• **Berkeley Function Calling Leaderboard (BFCL)** (Patil et al., 2023) provides a rigorous framework for assessing the function-calling proficiencies of diverse LLM agents. This benchmark encompasses 2,000 question-function-answer triads, spanning multiple programming paradigms (Python, Java, JavaScript, REST API) and heterogeneous application domains. The BFCL's evaluation protocol incorporates varying degrees of complexity, ranging from single-function selection tasks to scenarios necessitating the concurrent execution of multiple-function calls. Notably, the latest iteration, BFCL-v3, represents a significant methodological advancement over its predecessor by introducing a novel category that evaluates multi-turn and multi-step function invocation, more closely simulating real-world tool usage scenarios. We leverage BFCL-v2 and -v3 to evaluate the function-calling capability of LLM agents.

Following Dubey et al. (2024), we leverage additional three agent benchmarks (Nexus (Srinivasan et al., 2023), API-Bank (Li et al., 2023b), and API-Bench (Patil et al., 2023)) and one general benchmark (MMLU) (Hendrycks et al., 2020) for benchmark loss in the scaling law experiments.

B Baseline Details

B.1 Base LLMs

• **LLaMA-3-8B-Base** (Dubey et al., 2024) is a small-scale flagship model in Meta's LLaMA-3 series, featuring 8 billion parameters. We compare Hephaestus with LLaMA-3-8B-Base, which also serves as the backbone of Hephaestus-8B-Base, to demonstrate the effectiveness of continual pre-training.
• **LLaMA-3.1-8B-Base** (Dubey et al., 2024) is an improved version of LLaMA-3-8B, offering more efficient parameter utilization and enhanced fine-tuning capabilities. The 3.1 series models are optimized for multilingual support and scalability, allowing for a longer context length of up to 128K tokens. We select LLaMA-3.1-8B-Base as the current state-of-the-art small-scale open-sourced base model for comparison.

B.2 Open-Source Instruction Fine-tuned LLMs

We compare Hephaestus-IFT with the following open-sourced instruction-tuned LLMs:
• **LLaMA-2-Chat** (Touvron et al., 2023) is a series of large language models developed by Meta, designed for conversational AI. The models support text-based interactions and come in varying parameter sizes, such as 7B, 13B, and 70B. For comparison, we select models of comparable scale, specifically LLaMA-2-7B-Chat and LLaMA-2-70B-Chat.
• **Vicuna-v1.5** (Chiang et al., 2023) is a collection of open-source LLMs fine-tuned from LLaMA models, optimized for high-quality conversational abilities. These models are fine-tuned using datasets derived from user-shared conversations and are available in sizes such as 7B and 13B parameters, both of which are included in our comparisons.
• **CodeLLaMA** (Roziere et al., 2023) is a specialized extension of the LLaMA family designed for code generation and understanding. Built upon LLaMA-2, CodeLLaMA introduces enhancements tailored to coding tasks. We evaluate multiple sizes, including CodeLLaMA-7B-Instruct, CodeLLaMA-13B-Instruct, and CodeLLaMA-34B-Instruct.
• **Groq-8B-Tool-Use** (Groq, 2024) is a specialized variant of LLaMA-3-8B, fine-tuned by Groq for advanced tool use and function-calling tasks. It leverages post-training techniques to achieve state-of-the-art performance in function-calling tasks, including BFCL.
• **LLaMA-3-Instruct** (Dubey et al., 2024) belongs to Meta's LLaMA-3 family, optimized for instruction-following tasks. These models excel at tasks requiring explicit instructions, making them suitable for applications such as chatbots, virtual assistants, and task-specific text generation. We compare LLaMA-3-8B-Instruct and LLaMA-3.1-8B-Instruct as small-scale state-of-the-art instruction-tuned models. Additionally, we use LLaMA-3-70B-Instruct as a reference model for comparison.
• **DeepSeek-v2** (Liu et al., 2024a) and **Mixtral-8x22B** (Jiang et al., 2024) are both cutting-edge language models utilizing Mixture-of-Experts (MoE) architectures to optimize efficiency and performance across various domains. We include both models as reference points in our comparisons.

B.3 API-based Commercial LLMs (for reference)

We also consider API-based commercial LLMs for reference only, including **Gemini-1.5-Flash** (Reid et al., 2024), **text-davinci-003** (Ouyang et al., 2022), **gpt-3.5-turbo-0125** (OpenAI, 2022), **gpt-4-0613** (Achiam et al., 2023), **Claude-3-Haiku** (Anthropic, 2024), and **Command-R-Plus-FC** (Cohere, 2024). We exclude prompting and instruction fine-tuned agent frameworks from our main experiments to focus on evaluating the fundamental agentic capabilities of LLMs.

C Additional Related Works

LLM-based intelligent agents and autonomous entities have demonstrated proficiency in tool utilization (Qin et al., 2024; Zhuang et al., 2024c), decision-making (Wang et al., 2023; Li et al., 2024), and action execution through interactions with diverse environments (Sun et al., 2024a; Shi et al., 2024b).

C.1 Black-box LLM Agents

Existing methods for enhancing commercial closed-source LLM-based agents primarily focus on designing task-specific prompts. These prompts often incorporate tool function documentation (Hsieh et al., 2023), few-shot demonstrations (Lu et al., 2024), environmental feedback (Yao et al., 2023; Sun et al., 2024a; Wang et al., 2023), and tree-like reasoning procedures (Yao et al., 2024; Zhuang et al., 2024a). While these approaches have yielded improved results and increased flexibility, they come with significant drawbacks. The use of closed-source LLMs incurs substantial financial costs and raises safety concerns (Li et al., 2023a; Zhuang et al., 2024b; Yuan et al., 2024b; Sun et al., 2024b; Shi et al., 2024a), limiting their wider deployment. Moreover, these prompting techniques do not fundamentally enhance the inherent agent abilities of the LLMs. Instead, they rely heavily on the function-calling capabilities of closed-source LLMs, which may lack stability across different updates or versions[5].

C.2 White-box LLM Agents

Open-source LLMs have recently emerged as promising alternatives, demonstrating effectiveness in various applications (Touvron et al., 2023; Jiang et al., 2024; Tang et al., 2024). While these models excel in natural language processing tasks, they still underperform when serving as the core of LLM agents (Zeng et al., 2023; Liu et al., 2024d). This limitation is primarily due to insufficient training samples and smaller model scales compared to their closed-source counterparts. Researchers have attempted to address these shortcomings through various approaches. Some have fine-tuned LLMs with specific API documentation and function call sequences (Qin et al., 2024; Gou et al., 2024). Others have leveraged domain-specific data to learn tool embeddings or modify the decoding process (Schick et al., 2024; Hao et al., 2024; Zhang et al., 2023). However, this focus on specialized capabilities often comes at the expense of the LLMs' general abilities and compromises their generalizability. A recent approach by Chen et al. (2024b) attempts to mitigate this issue by composing API function sequential data from diverse sources and reorganizing the training corpus. Yet, compared to the breadth of data included in the pre-training stage, the collected data from five to six different sources represents only a small fraction of real-world decision-making scenarios, limiting generalization to new tasks. Moreover, the superficial alignment hypothesis (Zhou et al., 2024) suggests that a model's fundamental knowledge and capabilities are acquired almost entirely during pre-training. Post-training techniques merely guide the model in selecting which subdistribution of formats to use when interacting with users. Consequently, core abilities cannot be significantly improved through prompting and post-training techniques alone.

C.3 Finetuning-based LLM Agents

Table 1 summarizes existing instruction fine-tuning-based LLM agents and their training samples. For example, Gorilla (Patil et al., 2023) fine-tuned a LLaMA-based model using API documentation and demonstrations from Huggingface, TorchHub, and TensorFlowHub. Toolformer (Schick et al., 2024) introduced special tokens around API function calls to teach the model when and how to leverage tools during fine-tuning. ToolkenGPT (Hao et al., 2024) incorporated tools as special tokens into the model's vocabulary, while ToolLLaMA (Qin et al., 2024) built datasets rich in various tools. However, these methods often rely on APIs and datasets from similar domains, potentially limiting their effectiveness to tasks within those domains. To address this limitation, recent instruction tuning methods (Achiam et al., 2023; Srinivasan et al., 2023; Zeng et al., 2023; Chen et al., 2024b) have expanded to include a diverse range of API function call data and tasks, aiming to equip models with broader generalization capabilities across different planning tasks. Nevertheless, the superficial alignment hypothesis (Zhou et al., 2024) suggests that a model's fundamental knowledge and capabilities are predominantly acquired during pre-training. According to this hypothesis, post-training techniques such as instruction tuning and alignment primarily teach the model which sub-distributions of formats to utilize when interacting with users, rather than fundamentally expanding its capabilities. Moreover, heavy fine-tuning prevents generalization and degrades performance in general use cases, potentially suppressing the original base model capabilities (Ghosh et al., 2024).

C.4 Pretraining-based LLM Agents

To overcome the limitations of prompting and tuning-based methods, recent initiatives have focused on pre-training or continual pre-training of language models to bolster their fundamental capabilities.

[5] https://openai.com/index/function-calling-and-other-api-updates/

Several notable examples have emerged in this domain: CodeGen (Nijkamp et al., 2023) and CodeLLaMA (Roziere et al., 2023) enhance the coding skills of LLMs. Building on the success of these code LLMs, LEMUR (Xu et al., 2024) further instruction tunes a code LLM with additional assistant and tool-related data. Pandora (Xiang et al., 2024) represents a pre-trained world model that incorporates visual encoders to process a wide array of multi-modal data, including videos and textual actions. The most closely related work to our proposed model is OpenFunctions-v2 (Patil et al., 2023). This model is pre-trained on a vast collection of data sources, including 19,353 Python packages, 16,586 Java repositories, 4,285 JavaScript repositories, 6,009 public APIs, and 19,090 command line tools. However, while OpenFunctions-v2 primarily focuses on making correct API function calls, it lacks emphasis on the intrinsic reasoning abilities required for managing multiple API function calls, as well as adapting to environmental feedback.

D Dataset Construction Details

To scale and diversify the pre-training corpus for LLM agents, we introduce a three-stage construction process (Figure 7) for `Hephaestus-Forge` in § 4. We then include additional data collection details as follows.

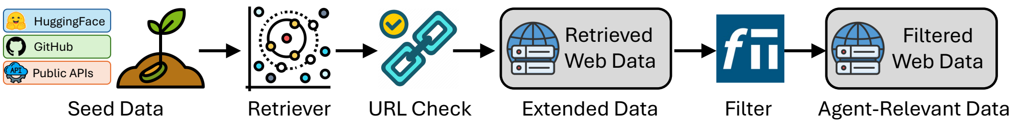


Figure 7: Overview of the data collection workflow in `Hephaestus-Forge`.

D.1 Seed Data Collection Details

We begin by assembling a set of high-quality initial data samples to establish a robust foundation. Specifically, we systematically explore publicly accessible resources to gather high-quality API documentation and associated action trajectories. This includes compiling a diverse dataset of agent behavior from public repositories, official API documentation sources, and data synthesized through LLMs. Given that the volume of tool-related data remains significantly smaller than that of plain text or code data, we employ data augmentation and generation techniques to expand the tool-related dataset.

D.1.1 Public APIs.

First, we collect data from over 1,400 public API documentations[6] and integrate additional data from official websites, including Huggingface[7], TorchHub[8], and Python Modules[9], among others. This compilation includes detailed API definitions and parameter descriptions, enabling the model to gain a better understanding of API functions. As the depth and location of the documentation vary across different API websites, we apply a three-level scraping strategy: (1) *Level 1:* the collected 1,400 URLs; (2) *Level 2:* 37,753 URLs appearing on the Level 1 web pages; (3) *Level 3:* 83,468 URLs appearing in the Level 2 web pages. We then apply URL checks to verify validity and filter for documentation-relevant data by searching for keywords (*e.g.*, "doc", "guide", "reference", *etc.*).

D.1.2 Public Repositories.

To strengthen the model's intrinsic reasoning and planning abilities, we integrate publicly available action trajectories from over 60 public repositories of related papers and datasets. These action trajectories span multiple domains, including programming code, natural language reasoning steps, embodied AI action sequences, grounded multi-modal data, web interactions, and function call sequences. This diverse range of trajectories, incorporated during the pre-training phase, enhances the model's reasoning capabilities and improves its generalization to various scenarios.

[6] https://github.com/public-apis/public-apis
[7] https://huggingface.co/docs
[8] https://pytorch.org/docs/stable/index.html
[9] https://docs.python.org/3/index.html

D.1.3 Code-to-Text Synthesis.

Given the limited quantity and API coverage of curated data from public APIs and repositories, we exploit the strong generative abilities of LLMs to synthesize additional API documentation and use cases. To produce high-quality synthetic agent data, we utilize *StarCoder-API*[10] as a knowledge base, which includes code snippets involving third-party APIs. Based on these code snippets and the API function calls within them, we generate corresponding API documentation and associated use cases. For efficiency, we utilize multiple LLMs from Amazon Bedrock[11] for data synthesis, including `Claude-3-Sonnet`, `Claude-3-Haiku` (Anthropic, 2024), `Mistral-Large` (Mistral, 2024), `LLaMA-3-70B-Instruct` (Dubey et al., 2024), and `Command-R-Plus` (Cohere, 2024).

D.1.4 Simulated Agent Data.

To improve the model's ability to adapt based on environmental feedback, we collect action sequences paired with observational data from various environments, represented as $\{o_0, a_1, o_1, a_2, o_2, \cdots, a_{T_g}, o_{T_g}\}$. This representation encodes the model's responses to environmental observations within its parameters. We execute official codes from agent frameworks (Yao et al., 2023; Sun et al., 2024a; Wang et al., 2024a; Shinn et al., 2024) in multi-step reasoning tasks (*e.g.*, HotpotQA (Yang et al., 2018)) and sequential decision-making tasks (*e.g.*, ALFWorld (Shridhar et al., 2021)) to collect action trajectories that involve interaction with and feedback from the environment.

D.2 Data Quality Control Details

We ensure the integrity and relevance of the collected data through continuous quality monitoring and validation procedures. After retrieving semantically relevant data from the web corpus, we obtain a collection of noisy agent-related data. To preserve the integrity and relevance of our dataset, it is critical to continuously monitor data quality and filter out content that resembles general text rather than agent-specific data. First, we employ `Claude-3-Sonnet` (Anthropic, 2024) as the data annotator to identify whether the sample belongs to agent data or a general web corpus. Specifically, we annotate a total of $71,473$ samples from the retrieved data, identifying $37,714$ as agent-relevant and $33,767$ as general text paragraphs. Using the annotated samples, we train a `fastText` (Joulin, 2016) model to effectively recall additional agent-relevant web data. We utilize the open-source `fastText` library[12] for training, configuring the vector dimension to 256, learning rate to 0.1, the maximum length of word n-gram to 3, the minimum number of word occurrences to 3, and the number of training epochs to 3. After training, the `fastText` model is used to recall agent-relevant data from the remaining retrieved samples. To filter out low-quality content, we rank the collected pages based on their predicted scores from the `fastText` model and retain only the top-ranking entries. This filtering process reduces the dataset from approximately 200 billion to 80 billion tokens, ensuring that the preserved data remains highly relevant and of sufficient quality for training LLM agents.

E Implementation Details

We use `LLaMA-3-8B` (Dubey et al., 2024) as the backbone for our main experiments. Our training process consists of two stages. In the two-stage pre-training, we set the batch size to 512 and train the model for $55,000$ steps in each stage, with a learning rate of $2e-4$ and weight decay of 0.01. For the instruction fine-tuning stage, we reduce the batch size to 16 and train the model for $24,000$ steps, using a learning rate of $1e-6$ while maintaining the same weight decay of 0.01. For parallel pre-training, we apply a tensor model parallel size of 8 and a pipeline model parallel size of 2. These values are adjusted to 4 and 2, respectively, for instruction fine-tuning. We use the Adam optimizer (Kingma, 2014) with $\beta_1 = 0.9$ and $\beta_2 = 0.98$ for all stages. During inference, we maintain a temperature of $T = 0$. Training `Hephaestus-8B-Base` requires 128 NVIDIA A100 (40G) GPUs for 11.1 days (7.7 days for Stage I pre-training and 3.4 days for Stage II pre-training). Training `Hephaestus-8B-IFT` uses 16 NVIDIA A100 (40G) GPUs for 11.6 hours.

[10] https://huggingface.co/datasets/luna-code/starcoderdata-apis
[11] https://aws.amazon.com/bedrock/
[12] https://fasttext.cc

F Additional Experimental Results and Analysis

F.1 Evaluation of the fastText Filter

To evaluate the precision of the fastText classifier in filtering general text from web retrieval data, we leverage `Claude-3-Sonnet` to annotate 20K samples. We then compare the predictions from the fastText filter against these annotated ground-truth labels. The evaluation results are presented in Table 6. The results indicate that the fastText filter achieves an accuracy of approximately 88%, suggesting that the filtering outcomes are reliable and trustworthy. Moreover, the higher recall score indicates that the filtered data encompasses most agent-relevant information from the retrieval.

Model (↓)	**Accuracy**	**F-1**	**Precision**	**Recall**
fastText	87.46	87.20	83.42	91.33

Table 6: Classification results of the fastText filter.

F.2 Evaluation of Base Models

As base models often struggle to follow instructions to solve problems, existing works evaluate these models using few-shot prompting (Wei et al., 2022; Shao et al., 2024) or by assessing the negative log-likelihood of the final answer (Dubey et al., 2024) (e.g., selecting the correct choice). However, these evaluation methods are not suitable for agent environments for the following reasons: (1) **Task Complexity.** Agent environment tasks are significantly more complex than multiple-choice questions, requiring the generation of long sequences of actions rather than selecting a single answer. (2) **Contextual Task Requirements.** Task requirements are often intricately embedded within the context, leaving insufficient space for few-shot exemplars. To this end, we evaluate `Hephaestus`-Base on three agent benchmarks (Nexus (Srinivasan et al., 2023), API-Bank (Li et al., 2023b), and API-Bench (Patil et al., 2023)) and one general benchmark (MMLU) (Hendrycks et al., 2020), reporting the benchmark loss in Figure 5.

F.3 Main Experimental Results on BFCL-v2

Datasets (→)	**AST**								**Exec**							**BFCL-v2**
Models (↓)	**OA**	**Simple**	**Python**	**Java**	**JS**	**MF**	**PF**	**PM**	**OA**	**Simple**	**Python**	**REST**	**MF**	**PF**	**PM**	**OA**
Base LLMs																
LLaMA-3-8B (Dubey et al., 2024)	0.94	1.3	1.0	2.0	1.5	0.5	0.5	0.5	0.40	2.0	1.0	1.0	0.0	0.0	0.0	17.77
LLaMA-3.1-8B (Dubey et al., 2024)	6.05	10.2	12.0	5.0	6.0	4.0	7.5	2.5	0.43	1.7	2.0	1.4	0.0	0.0	0.0	21.10
`Hephaestus-8B-Base`	**15.4**	**12.2**	**15.0**	**4.0**	**6.0**	**25.0**	**11.5**	**13.0**	**2.24**	**2.9**	**2.0**	**4.3**	**6.0**	**0.0**	**0.0**	**25.18**
Open-Source Instruction Fine-Tuned LLMs (Small)																
LLaMA-3-8B-Instruct (Dubey et al., 2024)	60.47	58.3	65.5	38.0	42.0	76.5	58.0	**49.0**	68.88	44.5	89.0	55.7	86.0	**78.0**	**55.0**	59.57
LLaMA-3.1-8B-Instruct (Dubey et al., 2024)	58.38	60.0	68.8	32.0	46.0	66.5	65.0	42.0	**72.60**	83.7	87.0	77.1	83.0	76.0	52.5	61.39
LLaMA-3-8B-IFT	47.43	66.7	75.5	37.0	**56.0**	45.5	54.0	23.5	63.41	**87.7**	93.0	**80.0**	68.0	58.0	40.0	62.12
`Hephaestus-8B-IFT`	**66.39**	**72.5**	**81.8**	**45.0**	54.0	**79.5**	**70.5**	43.0	69.82	85.3	**95.0**	71.4	**88.0**	66.0	40.0	**70.78**
For Reference: Open-Source Instruction Fine-Tuned LLMs (Medium to Large) and API-based Commercial LLMs																
Gemini-1.5-Flash (Reid et al., 2024)	77.44	67.3	92.8	55.0	54.0	94.0	71.5	77.0	73.23	57.9	93.0	22.9	86.0	74.0	75.0	70.75
Mixtral-8x22B (Jiang et al., 2024)	57.92	67.2	87.5	54.0	60.0	82.0	50.5	32.0	63.59	71.9	88.0	55.7	74.0	56.0	52.5	63.26
gpt-3.5-turbo-0125 (OpenAI, 2022)	66.31	63.8	75.3	50.0	66.0	78.0	68.0	55.5	65.88	44.5	89.0	0.0	86.0	78.0	55.0	66.53
Claude-3-Haiku (Anthropic, 2024)	62.52	77.6	95.8	63.0	74.0	93.0	47.5	32.0	60.73	89.4	96.0	82.9	94.0	32.0	27.5	55.47
Command-R-Plus-FC (Cohere, 2024)	77.65	69. 6	85.8	61.0	62.0	88.0	82.5	70.5	77.41	89.1	94.0	84.3	86.0	82.0	52.5	76.29
LLaMA-3-70B-Instruct (Dubey et al., 2024)	87.90	75.6	94.8	60.0	72.0	94.0	93.0	89.0	88.04	94.1	94.0	94.3	94.0	84.0	80.0	84.95
gpt-4-0613 (Achiam et al., 2023)	91.92	81.2	95.5	68.0	80.0	96.0	96.0	94.5	87.57	98.3	98.0	98.6	96.0	86.0	70.0	89.26

Table 7: Main experiment results on BFCL-v2.

Table 7 displays detailed experimental results on BFCL-v2, covering AST and Execution, two aspects in evaluation of function calling capabilities. Aside from the notations across the other tables, "JS" indicates "JavaScript"; "MF", "PF", and "PM" refer to "multiple functions", "parallel functions", "parallel multiple functions". The superior performance of Hephaestus-3-8B in AST evaluations indicates that the pre-training stage successfully introduced syntax knowledge of function calling into the model, which also contributes to improvements in the Execution aspect. However, the performance gain in Execution evaluations is less pronounced. This is because, lacking access to the instruction fine-tuning data used for LLaMA-3-8B, our Hephaestus-8B-IFT demonstrates limited instruction-following capabilities compared to LLaMA-3-8B-Instruct and LLaMA-3.1-8B-Instruct. Consequently, it is more challenging to follow instructions to generate executable functions.

F.4 Effect of Backbone LLMs

Datasets (→)	**AgentBench**						
Models (↓)	**OA**	**OS**	**DB**	**HH**	**KG**	**WB**	**WS**
Mistral-7B-v0.3-Base (Jiang et al., 2023)	0.40	7.6	0.7	0.0	8.9	11.0	1.4
Hephaestus-7B-Base (Mistral)	**1.46**	**18.3**	**21.0**	**24.0**	**12.7**	**14.0**	**46.2**
Mistral-7B-v0.3-Instruct (Jiang et al., 2023)	1.10	18.1	15.0	4.0	8.9	18.0	39.6
Mistral-7B-v0.3-IFT	1.32	17.4	**18.0**	8.0	15.9	20.0	45.1
Hephaestus-7B-IFT (Mistral)	**1.72**	**17.4**	11.7	**30.0**	**20.1**	**25.0**	**55.4**

Table 8: Experimental results of Hephaestus-7B (Mistral) with Mistral-7B-v0.3 as backbone LLM on AgentBench.

Table 8 reports the performance of Hephaestus and the baselines using Mistral-7B-v0.3 as backbone LLM on AgentBench. Notably, there exist consistent gains in terms of the average performance on both base model and instruction-tuned model (1.06 on base model and 0.4 on IFT model), justifying the advantage of pre-training on Hephaestus-Forge across different LLM types and architectures.

G Case Studies

G.1 Code-to-Text Synthesis Example

We present an example of synthesized API documentation as follows:

<Code_to_Text> Example

```
**API: `easyai.model.base_block.utility.utility_block.ConvBNActivationBlock`**

The `ConvBNActivationBlock` is a utility class that combines a convolutional layer, a batch
normalization layer, and an activation layer into a single block. This block is commonly used as a
building block in various neural network architectures, including the PNASNet architecture
demonstrated in the provided code.

**Parameters:**

- `in_channels` (int): The number of input channels for the convolutional layer.
- `out_channels` (int): The number of output channels for the convolutional layer.
- `kernel_size` (int or tuple): The size of the convolutional kernel.
- `stride` (int or tuple, optional): The stride of the convolutional operation. Default is 1.
- `padding` (int or tuple, optional): The padding added to the input tensor before applying
    the convolution. Default is 0.
- `bias` (bool, optional): Whether to include a bias term in the convolutional layer. Default is
    `False`.
- `bnName` (str or `easyai.base_name.block_name.NormalizationType`, optional): The normalization
    type to be used. Default is `NormalizationType.BatchNormalize2d`.
- `activationName` (str or `easyai.base_name.block_name.ActivationType`, optional): The activation
    function to be used. Default is `ActivationType.ReLU`.

**Attributes:**

- `conv`: The convolutional layer.
- `bn`: The batch normalization layer.
- `activation`: The activation layer.

**Methods:**
```

~~~~
- `forward(x)`: Defines the forward pass of the block. It takes an input tensor `x` and applies the
    convolutional, batch normalization, and activation operations sequentially.
- `get_name()`: Returns the name of the block, which is a combination of the class name and a unique
    index.

**Example Usage:**

```python
import torch
from easyai.model.base_block.utility.utility_block import ConvBNActivationBlock

# Create a ConvBNActivationBlock with input channels 3, output channels 64, kernel size 3x3, and
    stride 1
block = ConvBNActivationBlock(in_channels=3, out_channels=64, kernel_size=3, stride=1)

# Generate a random input tensor
x = torch.randn(1, 3, 32, 32)

# Pass the input tensor through the block
output = block(x)
```

In the provided code, the `ConvBNActivationBlock` is used as the first layer of the PNASNet
architecture, where it takes the input image data and applies a convolutional operation followed by
batch normalization and activation.
~~~~

G.2 Retrieved Data Examples

We present two examples of high-quality retrieved data as follows:

<Retrieval> Example-1

```
The Cardboard Kitchen : 6 Steps
By nicholasniski01 in Craft Cardboard
Introduction: The Cardboard Kitchen
In this instructable I will show you how to make a Cardboard Kitchen. The Cardboard Kitchen Is
    almost entirely made out of Cardboard. This Kitchen includes a Stove, Oven, Sink, Dishwasher,
    Fridge and Microwave.
lots of small boxes
and a medium size box
Step 1: How to Make a Fridge
    you need to disassemble the medium size box(Get rid of ALL the tape)...
Step 2: How to Make a Microwave
    First get a small box, make a rectangular hole in the box...
Step 3: How to Make a Sink
    First get a small box, cut off the top of the box...
Step 4: How to Make a Dishwasher
    Cut out a square of cardboard for the size of the dishwasher then you color the cardboard black,
    silver or any other color you would want for the dishwasher...
Step 5: How to Make a Stove
    To make the stove you make a black circle with for lines going out of the circle on an unused
    section of your big box or \"counter\"...
Step 6: How to Make an Oven
    Get a square the size you want your oven to be...
```

<Retrieval> Example-2

```
manual Prestigio MultiReader 5574
You can create your event and make a plan on your calendar. On the home screen or list menu,
    tap Calendar. View the calendar On the home screen or list menu, tap Calendar to check the
    calendar. Tap to change your calendar to Day, Week,Month or Agenda view. Create an event
1. Go to Calendar, select a date.
2. Tap to create a new event.
3. Edit reminder settings.
4. Tap Done to save the event.
```

G.3 Data Quality Filtering Failure Cases

We present a failure case of the fastText filter below:

<fastText_Filter> Failure Case

```
[TEXT]
We're making it even easier for you to stay connected to 99ROCK wherever you go! Besides tuning
    in on your radio, you can also stream your favorite station through your computer, smartphone,
    tablet, and your smart speaker.
If you are in or near the Fort Walton Beach-Destin broadcast area, tune your radio to: 99.5 FM
Stream 99ROCK at work or home from your computer on one of these web players: Triton Player iHeart
    Radio TuneIn
Listen to 99ROCK on-the-go thru one of these popular streaming apps or thru the 99ROCK mobile
    app: iOS App Google Play iHeart Radio TuneIn
First you need to enable the 99ROCK skill:
Say, ``Alexa, enable the ninety-nine rock Skill''
After you have enabled the Skill, listen to our station just by saying "Alexa, open ninety nine
    rock"
Just say, ``Hey Google, play ninety-nine rock''
[CATEGORY]
Agent
```

In this case, the fastText model incorrectly categorized the text as agent-relevant data. This misclassification likely occurred because fastText relies on gram frequency analysis, and the presence of multiple high-tech terms (e.g., iOS, App, Google Play) in the paragraph may have misled the model.

H Prompt Templates

H.1 Prompt Template for Code-to-Text Synthesis

<Code_to_Text> Prompt

```
Please use your knowledge to write an API documentation for the given APIs and consider the given
code as the example usage.

API:
{api}

Code:
{code}

API Documentation:
```

H.2 Prompt Template for LLM Annotator in Data Quality Control

<LLM_Annotation> Prompt

```
Please categorize the given text belong to agent-relevant data or other general text. The
    definitions are as follows:
1. Agent: Tool documentation text that describes the usage of a tool, software, or API; and action
    trajectory text that describes a sequence of actions or steps to achieve a goal.
2. General: Other general text that does not belong to the above two categories.
Below are some examples:
[TEXT]
**API: `easyai.model.base_block.utility.utility_block.ConvBNActivationBlock`**
The `ConvBNActivationBlock` is a utility class that combines a convolutional layer, a batch
normalization layer, and an activation layer into a single block. This block is commonly used as a
building block in various neural network architectures, including the PNASNet architecture
demonstrated in the provided code.  **Parameters:**  - `in_channels` (int): The number of input
channels for the convolutional layer.
- `out_channels` (int): The number of output channels for the convolutional layer.
- `kernel_size` (int or tuple): The size of the convolutional kernel.
- `stride` (int or tuple, optional): The stride of the convolutional operation. Default is 1.
- `padding` (int or tuple, optional): The padding added to the input tensor before applying the
    convolution. Default is 0.
- `bias` (bool, optional): Whether to include a bias term in the convolutional layer. Default is
    `False`.
- `bnName` (str or `easyai.base_name.block_name.NormalizationType`, optional): The normalization
    type to be used. Default is `NormalizationType.BatchNormalize2d`.
- `activationName` (str or `easyai.base_name.block_name.ActivationType`, optional): The activation
    function to be used. Default is `ActivationType.ReLU`.
[CATEGORY]
Agent
```

```
[TEXT]
I want to deliver a Birthday Gift to my friend in London, UK. Then, I need to book a flight from
New York, USA to London, UK on August 1st, 2023 for myself. After arriving in London, I would like
to see Dr. Smith for my Migraine. Once my health is in check, I'd like to apply for a Software
Engineer job in London.
Step 1: Call deliver_package API with package: 'Birthday Gift' and destination: 'London, UK'
deliver_package(package=Birthday Gift, destination=London, UK)
Step 2: Call book_flight API with date: '2023-08-01', from: 'New York, USA' and to: 'London, UK'
book_flight(date=2023-08-01, from=New York, USA, to=London, UK)
Step 3: Call see_doctor_online API with disease: 'Migraine' and doctor: 'Dr. Smith'
see_doctor_online(disease=Migraine, doctor=Dr. Smith)
Step 4: Call apply_for_job API with job: 'Software Engineer' apply_for_job(job=Software Engineer)"
[CATEGORY]
Agent

[TEXT]
My New Crock Pot -- Creuzer Leave a comment on My New Crock Pot I went out and got myself a new
crock pot. I rather like this one. It has 3 settings, High, Low, and Warm. It is designed to
be hauled around even! There are latches on each side of the lid to clip the lid in place. It
even came with it's own spoon that clips into the lid! A really neat feature is that the lid has
little tabs so you can set the lid on one of the handles and it won't go all sliding all over
the place. I think this is a winner, I plan on using it a lot for the cooking club I am in.
[CATEGORY]
General

Please categorize the following text into agent-relevant data (Agent) or general text (General).
ONLY respond the category name (Agent/General) for each text. If you are unsure, please respond with
    'General'.
[TEXT]
{text}
[CATEGORY]
```

Data Source	Type	Format	Tokens (B)	URL Link
ToolBench (Qin et al., 2024)	Traj.	Dialog	0.530	https://github.com/OpenBMB/ToolBench
AgentInstruct (Zeng et al., 2023)	Traj.	ReAct	0.002	https://huggingface.co/datasets/THUDM/AgentInstruct
Alexa-Arena (Gao et al., 2024)	Traj.	NL Plan	0.035	https://github.com/amazon-science/alexa-arena/tree/main
chat_ego_4d (Mu et al., 2024)	Traj.	API Seq	0.025	https://github.com/EmbodiedGPT/EgoCOT_Dataset
FireAct (Chen et al., 2023)	Traj.	ReAct	0.002	https://fireact-agent.github.io/
NAT (Wang et al., 2024c)	Traj.	ReAct	0.003	https://github.com/Reason-Wang/NAT
ToolAlpaca (Tang et al., 2023)	Traj.	Plain Text	0.004	https://github.com/tangqiaoyu/ToolAlpaca/tree/main
Lumos (Yin et al., 2024)	Traj.	Dialog	0.109	https://huggingface.co/datasets/ai2lumos/lumos_complex_qa_ground_iterative?row=0
STE (Wang et al., 2024a)	Traj.	Plain Text	0.025	https://github.com/microsoft/simulated-trial-and-error
toolbench (Xu et al., 2023)	Traj.	API Seq	0.010	https://github.com/sambanova/toolbench
Gorilla (Patil et al., 2023)	Doc.	API Seq	0.009	https://gorilla.cs.berkeley.edu/
PublicAPIs	Doc.	Plain Text	0.008	https://github.com/public-apis/public-apis?tab=readme-ov-file
TaskBench (Shen et al., 2023)	Traj.	NL Plan	0.020	https://github.com/microsoft/JARVIS/tree/main/taskbench
RestBench (Song et al., 2023)	Traj.	API Seq	0.001	https://github.com/Yifan-Song793/RestGPT/tree/main/datasets
SayCanPay (Hazra et al., 2024)	Traj.	NL Plan	0.001	https://github.com/RishiHazra/saycanpay
AgentFlan (Chen et al., 2024b)	Traj.	Dialog	0.020	https://github.com/InternLM/Agent-FLAN
PlanBench (Valmeekam et al., 2024)	Traj.	NL Plan	0.001	https://github.com/karthikv792/LLMs-Planning
SwftSage (Lin et al., 2024a)	Traj.	NL Plan	0.022	https://github.com/yuchenlin/SwiftSage
T-Eval (Chen et al., 2024a)	Traj.	Dialog	0.040	https://github.com/open-compass/T-Eval
API-Bank (Li et al., 2023b)	Traj.	API Seq	0.001	https://github.com/AlibabaResearch/DAMO-ConvAI/tree/main/api-bank
JeirchoWorld (Ammanabrolu and Riedl, 2021)	Traj.	NL Plan	0.001	https://github.com/JerichoWorld/JerichoWorld
API-Pack (Guo et al., 2024b)	Traj.	API Seq	0.800	https://huggingface.co/datasets/zguo0525/API-Pack/tree/main
CodeAct (Lv et al., 2024)	Traj.	Dialog	0.009	https://huggingface.co/datasets/xingyaoww/code-act
UltraTool (Huang et al., 2024)	Traj.	NL Plan	0.002	https://github.com/JoeYing1019/UltraTool/tree/main
Tooleyes (Ye et al., 2024)	Doc.	JSON	0.001	https://github.com/Junjie-Ye/ToolEyes/tree/main
OpenMathInstruct (Toshniwal et al., 2024)	Traj.	API Seq	0.335	https://huggingface.co/datasets/nvidia/OpenMathInstruct-1
NexasRaven (Srinivasan et al., 2023)	Traj.	JSON	0.001	https://huggingface.co/Nexusflow
Seal-Tools (Wu et al., 2024)	Traj.	API Seq	0.002	https://github.com/fairyshine/Seal-Tools/tree/master
UltraInteract (Yuan et al., 2024a)	Traj.	QA	0.16	https://huggingface.co/datasets/openbmb/UltraInteract_sft?row=0
Python Module	Doc.	Plain Text	0.001	https://docs.python.org/3.12/
AgentTraj-L (Xi et al., 2024)	Traj.	Dialog	0.020	https://huggingface.co/datasets/AgentGym/AgentTraj-L
MNMs (Ma et al., 2024)	Traj.	API Seq	0.001	https://huggingface.co/datasets/zixianma/mnms
PythonQA-API-Usage	Doc.	QA	0.003	https://huggingface.co/datasets/RazinAleks/SO-Python_QA-API_USAGE_class
APIText	Traj.	API Seq	0.001	https://huggingface.co/datasets/havens2/apitext
StarCoder-APIs (Lozhkov et al., 2024)	Traj.	Code	6.147	https://huggingface.co/datasets/luna-code/starcoderdata-apis
APIs_v2	Traj.	API Seq	0.003	https://huggingface.co/datasets/vinilazzari/apis_v2
Ultimate	Traj.	QA	0.002	https://huggingface.co/datasets/Kris8an/ultimate_apicalls_and_topbot
xLAM (Zhang et al., 2024b)	Doc.	QA	0.022	https://huggingface.co/datasets/Salesforce/xlam-function-calling-60k

Table 9: Data sources of the seed data in Hephaestus-Forge.

Data Source	Type	Format	Tokens (B)	URL Link
API_doc	Doc.	Plain Text	0.001	https://huggingface.co/datasets/Prakhar1000/API_Documentation_dataset_alpaanco?row=0
ChatsBug	Traj.	NL Plan	0.009	https://huggingface.co/datasets/chats-bug/agent_action_plan?row=0
sample_scripts	Traj.	API Seq	0.002	https://huggingface.co/datasets/prantadi/tokenized_dataset_1024_SampleScripts_deduped_API-ref?row=1
Agent-Trajectories	Traj.	API Seq	0.001	https://huggingface.co/datasets/Agent-Eval-Refine/Agent-Trajectories/tree/main
Agent-Instruct	Traj.	Dialog	0.056	https://huggingface.co/datasets/sam-mosaic/agent-instruct
Agent007	Traj.	API Seq	0.001	https://huggingface.co/datasets/DepositorOP/agent007
AgentCode	Traj.	API Seq	0.010	https://huggingface.co/datasets/AlignmentLab-AI/agentcode
syn-web-agent	Traj.	JSON	0.001	https://huggingface.co/datasets/allyson-ai/synthetic-web-agent
syn-llama	Traj.	Dialog	0.004	https://huggingface.co/datasets/Cyleux/agent-machine-convo-llama-nicholas-2k-gpt4-verified
seq-Mind2Web	Traj.	JSON	1.243	https://huggingface.co/datasets/Izazk/Sequence-of-action-prediction-mind2web
syn-gemma	Traj.	Dialog	0.047	https://huggingface.co/datasets/NickyNicky/function-calling-sharegpt_chatml_gemma_agent
LLM Robot	Traj.	API Seq	0.001	https://huggingface.co/datasets/Aryaduta/llm_robot
Verifiers for Code	Traj.	Plain Text	0.05	https://huggingface.co/datasets/verifiers-for-code/CodeNet-Planner
isotonic planner	Traj.	NL Plan	0.005	https://huggingface.co/datasets/Isotonic/planner_dataset
Turing Solutions	Traj.	NL Plab	0.001	https://huggingface.co/datasets/TuringsSolutions/GlobalFunctionCallingTrainingSetLarge
G-PlanET	Traj.	NL Plan	0.003	https://huggingface.co/datasets/TuringsSolutions/GlobalFunctionCallingTrainingSetLarge
Pandas Doc	Doc.	Plain Text	0.004	https://pandas.pydata.org/
Sugarcrm	Doc.	Plain Text	0.001	https://huggingface.co/datasets/kaahila/sugarcrm_130_documentation
AWS	Doc.	Plain Text	0.033	https://huggingface.co/datasets/sauravjoshi23/aws-documentation-chunked
LangChain	Doc.	Plain Text	0.005	https://huggingface.co/datasets/jamescalam/langchain-docs-23-06-27
Code Library	Doc.	Plain Text	0.013	https://huggingface.co/datasets/code-rag-bench/library-documentation
PublicAPIs-extend	Doc.	Plain Text	0.718	https://github.com/public-apis/public-apis?tab=readme-ov-file
Torch	Doc.	Plain Text	0.005	https://pytorch.org/docs/stable/index.html

Table 10: Data sources of the seed data in Hephaestus-Forge (Cont'd).