LUMOS: LEARNING AGENTS WITH UNIFIED DATA, MODULAR DESIGN, AND OPEN-SOURCE LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce LUMOS, a novel framework for training **l**anguage agents that employs a **u**nified data format and a **m**odular architecture based on open-source large language models (LLMs). LUMOS consists of three distinct modules: planning, grounding, and execution. The **planning module** breaks down a task into a series of high-level, tool-agnostic subgoals, which are then decomposed into a set of low-level actions via the **grounding module**. These actions are subsequently executed by the **execution module**, utilizing a range of off-the-shelf tools and APIs. In order to train these modules effectively, high-quality annotations of subgoals and actions were collected and are made available for fine-tuning open-source LLMs for various tasks such as complex question answering, web tasks, and math problems. Leveraging this unified data and modular design, LUMOS not only achieves comparable or superior performance to current, state-of-the-art agents, but also exhibits several key advantages: (1) LUMOS surpasses GPT-4/3.5-based agents in complex question answering and web tasks, while equalling the performance of significantly larger LLM agents on math tasks; (2) LUMOS outperforms open-source agents created through conventional training methods and those using chain-of-thoughts training; and (3) LUMOS is capable of effectively generalizing to unseen interactive tasks, outperforming larger LLM-based agents and even exceeding performance of specialized agents.

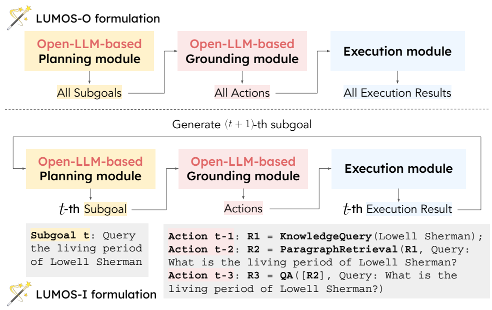
1 INTRODUCTION

Language agents, which employ the insights gained by language models (LMs) to address complex reasoning problems, carry out executable actions and interact with external tools or environments. They have evolved into crucial elements of AI systems targeted at solving complex interactive tasks. These tasks can range from question-answering (Yang et al., 2018; Geva et al., 2021), to web tasks (Deng et al., 2023; Zhou et al., 2023), and mathematical reasoning (Cobbe et al., 2021). The inherent challenge of these tasks springs from their typical demand for long-horizon planning and interactive reasoning abilities.

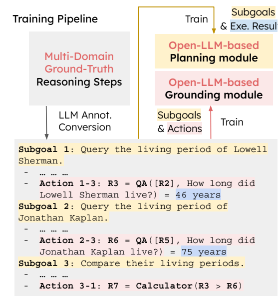
Prior agent frameworks (Yao et al., 2022b; Shinn et al., 2023; Lin et al., 2023; Patil et al., 2023; Lu et al., 2023; Liu et al., 2023b) have primarily relied on closed-source large language model (LLM) APIs such as OpenAI's GPT-4 and ChatGPT (OpenAI, 2023; 2022). Though powerful, these LLM APIs can be prohibitively expensive, particularly for tasks with long contexts such as web tasks, where encoding HTMLs is a necessity. Compounding these issues, these model APIs are seldom deterministic, making agent reproduction challenging. Furthermore, the lack of transparency in these closed-source LLMs results in limited understanding of their architectures and internal behaviors. We argue that such reliance on closed-source LLM-based agents is not conducive to the growth of the research community, suggesting a shift towards the use of open-source LLMs. However, recent studies (Xu et al., 2023a; Liu et al., 2023a; Zeng et al., 2023) indicate that agents built with open-source LLMs significantly lag behind those using GPT-4 in terms of performance.

In this paper, we propose LUMOS[1], a general language agent framework with a **u**nified data format and a **m**odular design, built on **o**pen-**s**ource LLMs. Contrary to conventional fine-tuning (Chen et al., 2023; Zeng et al., 2023), our focus resides on designing a *modular* framework for developing

[1] *"Lumos"* is a spell from the *Harry Potter* series that produces light from the caster's wand.



(a) Overview of our proposed LUMOS formulations – LUMOS-Onetime (LUMOS-O) and LUMOS-Iterative (LUMOS-I) (see details in §2).



(b) The process of acquiring annotations for training planning and grounding modules (see details in §3).

Figure 1: Overall architecture, formulations and training pipeline of LUMOS. The figure example is about answering a complex QA question "*Who lives longer, Lowell Sherman or Jonathan Kaplan?*"

language agents. LUMOS consists of three modules: a planning module, a grounding module, and an execution module. The planning module dissects a complex task into a sequence of high-level subgoals. The grounding module subsequently translates these generated subgoals into a series of low-level actions, which the execution module, a collection of off-the-shelf tools for problem-solving like calculators and search engines, executes. We propose two interaction formulations for implementing language agents, LUMOS-Onetime (LUMOS-O) and LUMOS-Iterative (LUMOS-I). As illustrated in Figure 1a, LUMOS-O is an efficient formulation that generates *all* the subgoals and executable actions using a *single* inference call in a one-pass fashion. On the other hand, LUMOS-I is an iterative formulation that generates one subgoal at a time based on its previous intermediate execution results and environment updates, thereby creating a more adaptive agent.

In addition, LUMOS utilizes a unified data format that encompasses multiple task types, thereby enabling the developed agent framework to conveniently support a range of interactive tasks. These include, but are not limited to, question answering, mathematics, coding, web browsing, and embodied simulation. This format is advantageous as it facilitates cross-task generalization. To obtain high-quality annotations for training LUMOS, we convert the rationale behind decision-making steps of existing benchmarks into a unified format (Fig. 1b). This conversion is achieved with the aid of GPT-4, ensuring that these steps share a universally-applicable formulation that is in line with the modular design we described earlier. Our proposed unified data representation aids not only in the efficient acquisition of high-quality training data for agents, but also in facilitating cross-task generalization for these agents. For both the planning and grounding modules, we have collected approximately 40K training annotations. These comprise one of the largest open-source resources for agent fine-tuning available. The training annotations could serve as an invaluable resource for universally enhancing any open-source LLMs with agent capabilities.

Our extensive evaluation affirms that LUMOS provides state-of-the-art or parallel performance across various complex interactive tasks, including QA, web, and mathematics tasks. We consolidate our contributions and results as follows:

A General Agent Framework with High-Quality Data. We introduce a modular agent learning framework that trains open-source LLMs with unified data, aimed at representing complex interactive tasks. LUMOS continually outclasses other learning methods such as the chain-of-thoughts fine-tuning and integrated agent training. Our unified task representation along with our gathered data (comprising 40K high-quality subgoal/action annotations for three prevalent task types) can be instrumental for future research in developing open-source agents for complex interactive tasks.

State-of-the-Art Performance. LUMOS surpasses GPT-based agents in complex QA and web tasks. Particularly, LUMOS exhibits a 3.5-8.1% LLM accuracy increase over GPT-3.5-turbo-based agents on HotpotQA (Yang et al., 2018) with only 7B-scale base models (a previously unseen dataset for LUMOS) and a substantial improvement on StrategyQA (Geva et al., 2021). It also shows a 5.0% enhancement in success rate over GPT-4 on Mind2Web. Interestingly, it even outperforms contemporaneous agents that have been fine-tuned with in-domain HotpotQA and Mind2Web annotations, with examples including FiReAct (Zeng et al., 2023) and AgentLM (Zeng et al., 2023). In mathematics tasks such as GSM8K (Cobbe et al., 2021) and SVAMP (Patel et al., 2021), LUMOS effectively competes against agents constructed on substantially larger LLMs.

Cross-Task Generalization. LUMOS generalizes well to unseen tasks and actions. When evaluated on an unseen task, namely WebShop (Yao et al., 2022a), a text game for online shopping, LUMOS surpasses the performance of larger LLM-based agents (for instance, WizardLM-30B (Xu et al., 2023b)) by approximately 20 reward points and also delivers a notable 5-10 reward point improvement over domain-specific agents. This suggests that LUMOS can easily generalize across tasks, hinting at potential benefits for a wide spectrum of language agent applications.

2 LUMOS: A MODULAR AGENT FRAMEWORK

2.1 LUMOS AGENT ARCHITECTURE

To solve a complex interactive task, it is necessary to decompose the task into a series of subgoals. These subgoals should then be converted into sequences of executable actions before being carried out. This process corresponds to the planning, grounding, and execution modules in our framework.

Planning Module (PM). This module is designed to dissect a complex task into a series of high-level `subgoals`, expressed in natural language. For example, a question such as "Who lived longer, Lowell Sherman or Jonathan Kaplan?" necessitates three subgoals, as illustrated in Figure 1b: (1) Determine the lifespan of 'Lowell Sherman'; (2) Ascertain the lifespan of 'Jonathan Kaplan'; (3) Compare their lifespans and provide an answer. The devised subgoals assist in breaking down a complex task into low-level actions in an interpretable and tool-agnostic manner.

Grounding Module (GM). The purpose of this module is to transform the high-level subgoals produced by the PM into low-level executable `actions`. For instance, the GM translates the subgoal, "`Query the living period of Lowell Sherman`," into one or more actions. These actions could be `KnowledgeQuery(Lowell Sherman)`, or `QA([R2], Query: ''What is the living period of Lowell Sherman?'')`. Here, `R2` refers to the previously retrieved knowledge that may be helpful in answering the query. The grounding module can be easily customized and updated to learn new actions without updating the planning module, thus significantly improving the flexibility of agent development.

Execution Module (EM). The Execution Module (EM) is a program that parses actions into a series of operations. It deploys a variety of off-the-shelf tools, including APIs, neural models, and virtual simulators. The module functions by interacting with these tools and the external environment. For instance, the execution module could call the Wikipedia API or Google Search APIs to accomplish the `KnowledgeQuery` action, thereby enabling the system to query a document.

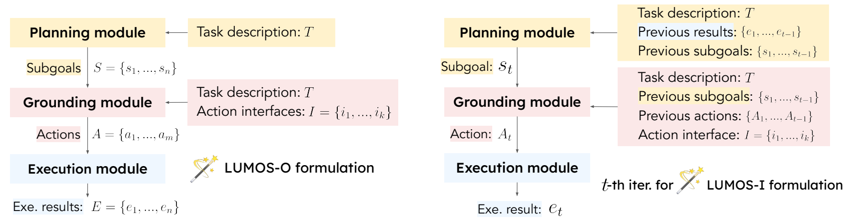
The main characteristic of the LUMOS framework is the interaction among the planning, grounding, and execution modules. We propose two formulations promoting enhanced communication between these three modules: LUMOS-Onetime (LUMOS-O) and LUMOS-Iterative (LUMOS-I).

2.2 LUMOS-ONETIME (LUMOS-O)

The LUMOS-Onetime (LUMOS-O) formulation is an efficient method that generates all subgoals and grounded actions simultaneously. As depicted in Figure 2a, this formulation uses the planning module to generate all three subgoals in a single inference call, without requiring interaction with the other two modules. We then pass all the generated subgoals at the same time to the grounding module, which translates them into a sequence of low-level actions without any interaction with the execution modules. Given both the task and the three subgoals, the grounding module can produce low-level actions, such as `KnowledgeQuery(Lowell Sherman)` and `ParagraphRetrieve(..., Query: How long did Lowell Sherman live?)`.

In addition to the task description and subgoals, we also provide `action interfaces` to the grounding module as inputs. These action interfaces define the functionalities of actions and their valid arguments, guiding the grounding module to produce executable actions. For example, we provide the grounding module with the action definition, `SolveEquation(equation): Solve a previously set equation and return the solution`, which is implemented by math APIs.

More formally, the overall planning and grounding process of LUMOS-O is illustrated in Figure 2a. In the planning phase, the task description T is input into the planning module, given the parameters of the module θ_{plan}. This generates an output series of subgoals, expressed as $S = \theta_{plan}(T) =$



(a) Inputs & outputs of each module in LUMOS-O formulation.

(b) Inputs & outputs of each module for the t-th iteration in LUMOS-I formulation.

Figure 2: Overview of the inputs & outputs contents for the modules in LUMOS.

$\{s_1, ..., s_n\}$. In the grounding phase, grounded actions are obtained via $A = \theta_{ground}(T, I, S)$, with reliance on the task description, action interface $I = \{i_1, ..., i_k\}$, and the subgoals generated by the planning module. Here, θ_{ground} represents the parameters of the grounding module.

2.3 LUMOS-ITERATIVE (LUMOS-I)

LUMOS-Iterative (LUMOS-I) is a formulation that generates one subgoal and its corresponding executable actions in each iteration. When generating the current t-th subgoal, the planning module requires the previous planned subgoals and the execution results of their grounded actions. The execution results assist the planning module to be aware of the dynamic environmental change and thus decide actions according to the most up-to-date environments. LUMOS-I has the advantage of being flexible in adapting to the environmental change.

Take the complex QA task as an example when answering the question "`Which state was the U.S. president in 2014 born in?`": in the first iteration, the planning module will produce "`Subgoal 1: Query who the U.S. president in 2014 was`". This subgoal is passed to the grounding module to generate the corresponding query actions, and obtain the executed results "`Obama`". The planning module then takes "`Obama`" along with the prior planning context as input to generate the next subgoal "`Subgoal 2: Query which state Obama was born`". Planning upon the latest execution results would mitigate the risk of introducing a non-existent object in the environment or a random entity during the reasoning process.

We demonstrate a single iteration of planning and grounding process of LUMOS-I in Figure 2b. In order to plan t-th subgoal, we input the 1) task description T, 2) prior subgoals $\{s_1, ..., s_{t-1}\}$, and 3) their executed results $\{e_1, ..., e_{t-1}\}$ into the planning module. We concatenate them in the format of $T, s_1, e_1, ..., s_{t-1}, e_{t-1}$ where the most recent subgoals and their results are placed in the end, as they have higher influence for planning t-th subgoal. The generated output would be the t-th subgoal, $s_t = \theta_{plan}(T, s_1, e_1, ..., s_{t-1}, e_{t-1})$. After the t-th subgoal is obtained, it will be directly incorporated into grounding module together with the prior grounding history and action interface I to generate the next set of executable actions, $A_t = \theta_{ground}(T, I, s_1, A_1, ..., s_{t-1}, A_{t-1}, s_t)$. Note that A_t is an executable action list, as the high-level subgoal might be decomposed into multiple low-level actions. We finally put the low-level actions A_t into execution module and the final execution result e_t can be sent back for planning $(t+1)$-th subgoal.

3 LEARNING TO PLAN & GROUND WITH OPEN-SOURCE LLMS

To guide planning and grounding modules to generate subgoals and valid low-level actions under our specified formulations, we fine-tune the two modules to produce the expected outputs.

Training the modules requires high-quality task descriptions, subgoals, and low-level actions. To equip smaller LLMs with instruction-following ability, prior works leverage methods such as Self-Instruct (Wang et al., 2023b) to synthesize instructions, inputs and outputs based on few-shot examples. However, these methods are not suitable for generating high-quality annotations for complex interactive tasks. For example, GPT-4 can only achieve around 20% step success rate in a web agent benchmark Mind2Web (Deng et al., 2023; Liu et al., 2023a). Relying on such methods to generate complex interactive task annotations may introduce noise and degrade the annotation quality.

Instead of creating annotations with API-based LLMs directly, we exploit LLMs as a "style transfer" tool to convert ground-truth intermediate reasoning steps in existing benchmarks into the expected format in LUMOS formulations. We notice that there are several complex interactive tasks annotated with either human-written solutions or structured action sequences. For example, PRM800K (Lightman et al., 2023) is a math dataset containing the solution steps written in natural language, interleaved with formulas; Musique (Trivedi et al., 2022) and StrategyQA (Geva et al., 2021) are complex QA datasets annotated with decomposed questions, supporting facts, and relevant Wikipedia paragraph indices; Mind2Web includes ground-truth action sequences. They provide LLMs with fundamental information that sufficiently contributes to the annotation conversion.

3.1 CONVERSION PROMPTS

To help LLMs better follow the annotation conversion instructions, we add 4-/5-shot examples in conversion prompts (see Appendix F for prompt details). We discuss the important properties of these in-context examples. The notations of the converted annotations have hat over letters.

Action Space. Action space defines the available actions that LLMs could ground to. Table 6 shows a comprehensive list of action definitions and their implementations.

Ground-Truth Intermediate Reasoning Steps. We provide LLMs with ground-truth intermediate reasoning steps in existing benchmarks. With these as a reference, LLMs are able to summarize high-level subgoals and synthesize corresponding actions according to the given action space.

Subgoals and Corresponding Actions. When converting ground-truth reasoning steps into our expected annotations, it is necessary to provide LLMs with examples about how to distill the high-level subgoals from the reasoning steps and map them into corresponding actions. In the in-context examples, we manually decompose a complex task into several high-level subgoals according to the context of ground-truth reasoning steps. Under each high-level subgoal, we write down multiple corresponding actions that help to accomplish the subgoal (see examples in Appendix F). Given the aligned exemplar subgoals and actions in the prompt, LLMs would emulate to generate subgoals and their paired actions when converting annotations for new tasks.

As the executed results of prior subgoals might be useful in future action implementation, we interlink the grounded actions in the in-context examples to allow context-dependent execution. One typical example of the interlinked actions is `R1 = KnowledgeQuery(Zombies); R2 = ParagraphRetrieve(R1, Query: What color skin are zombies typically depicted with?)`. The agent could first find the relevant paragraphs in the zombie knowledge page. Written in interlinked style, the second paragraph retrieval action is able to receive the knowledge about zombies (`R1`) as the context, and performs query-based retrieval.

Intermediate Executed Results of Subgoals. The intermediate executed results $\hat{E}$ play an important role in increasing LUMOS's adaptability to environmental changes. Some datasets (e.g., GSM8K, Mind2Web) offer execution results in their reasoning steps, including the computation results of formulas and the HTML code after operating on a website. We leverage them as $\hat{E}$. For the datasets without any execution results, their reasoning steps actually contain the relevant documents that include the clues for solving subgoals. We take an annotated example in StrategyQA dataset. Although the direct answer of the decomposed question "`What color skin are zombies typically depicted with?`" is not provided, the annotation contains a related fact "`Zombies are often depicted as green in pallor.`" that mentions the answer "`green`". Thus, for each in-context example, we concatenate the relevant documents (e.g., "`Zombies are often depicted as green in pallor.`") as well as our manually captured execution results (e.g., "`green`") in the conversion prompts. When converting new samples into our expected annotations, LLMs would automatically extract answers from the given documents as the executed results.

After prompting LLMs with the conversion prompts, we can acquire the key elements in training annotations, including subgoals $\hat{S}$, their corresponding actions $\hat{A}$ and execution results $\hat{E}$.

3.2 TRANSFERRING ANNOTATIONS INTO CONVERSATIONS

Finally, to build the interaction between planning and grounding modules, we organize the annotations into conversational format:

Conversational Planning Module Annotation. As shown in Figure 3a, we first play a user role to provide the task $\hat{T}$ and an instruction that guides the module to generate subgoals. For LUMOS-O formulation, the planning module should reply all the annotated subgoals $\hat{S}$ at once. There would be no further conversation needed.

LUMOS-I requires multi-turn conversational style. The planning module appends the first ground-truth subgoal $\hat{s}_1$ with index "`Subgoal 1`" in the beginning. We then act as user again and put the executed results of $\hat{s}_1$ with prefix "`The executed result for Subgoal 1 is` ". Similarly,for the rest turns, we assume ourselves as user, tell the execution results $\hat{e}_{t-1}$ of the last subgoal $\hat{s}_{t-1}$ to planning module, and ask whether the planning should be stopped; The response would be whether the planning should stop; if no, the response should contain a new ground-truth subgoal $\hat{s}_t$.

Conversational Grounding Module Annotation. Shown in Figure 3b, we also first play a user role to provide the task $\hat{T}$, action space and interfaces $\hat{I}$. For LUMOS-O formulation, we feed all the subgoal annotations $\hat{S}$ in the first user prompt. All the action annotations $\hat{A}$ would be the response of the user instruction. For LUMOS-I formulation, we provide the current ground-truth subgoal $\hat{s}_t$, with prefix "`Subgoal to be grounded:` ". Its response would be $\hat{s}_t$'s corresponding actions $\hat{A}_t$.

3.3 TRAINING WITH CONVERTED ANNOTATIONS

As LUMOS training annotations are conversational, we formulate them in the format of $\{x_1, y_1, ..., x_i, y_i, ..., x_n, y_n\}$, where x_i is i-th user prompt and y_i indicates its ground-truth responses. Following Wang et al. (2023a), during training process, we feed each entire multi-turn annotation into a decoder-only model while merely calculating the decoding loss on the tokens of ground-truth responses $Y = \{y_1, ..., y_i, ..., y_n\}$. We apply binary masking on the user prompt tokens to prevent computing loss on them. The final loss function is $L = -\sum_j \log p_\theta(t_j \mid t_{<j}) \times \mathbf{1}(t_j \in Y)$ where t_j denotes j-th input token and $\mathbf{1}(\cdot)$ is a Boolean indicator function.

4 EXPERIMENTS

We present the details of our experimental setup, including annotation conversion, module training, and the tools used in our execution module. To demonstrate the effectiveness of LUMOS, we evaluate LUMOS by: 1) comparing LUMOS with larger open-source LLM agents and GPT-based agent frameworks, 2) contrasting LUMOS against other potential open-source agent baselines, 3) manifesting LUMOS's generalizability on an unseen task that involves new environments and actions, and finally 4) assessing the quality of LUMOS training annotations.

4.1 EXPERIMENTAL SETUP

Data Collection. Utilizing the conversion prompts discussed in Section 3.1, we employ GPT-4 (OpenAI, 2023) to perform annotation conversion on the high-quality ground-truth reasoning steps present in existing benchmarks. Appendix A provides a list of the data sources used for annotation conversion. These include a diverse range of complex interactive tasks, such as math, complex QA and web tasks. After filtering out the annotations that contain mismatched parentheses, invalid execution outputs or excessively lengthy outputs, we obtain 39,441 and 39,558 annotations for training the planning and grounding modules, respectively. Detailed statistics are provided in Appendix A.

Training. We adopt LLAMA-2-7B and LLAMA-2-13B as the base models for both the planning and grounding modules. In all of our experiments, we implement training over two epochs with a learning rate of 2×10^{-5}. Details regarding the training process can be found in Appendix B.

Action Space. To solve the complex interactive tasks, we integrate commonly used actions into the pre-defined action spaces. Each action can be perceived as a function that inputs intermediate results as arguments and outputs text. Further details of the execution module's supported executable actions are included in Appendix E.

4.2 OVERALL PERFORMANCE

We evaluate our agents across an array of complex interactive tasks, such as complex QA, web, and math tasks. We conduct this evaluation in accordance with the setting conventions established by AgentBench (Liu et al., 2023a) and ReWOO (Xu et al., 2023a) when evaluating LUMOS (see Appendix D). The overall performance on each task type is displayed in Table 1. It's worth noting that

Agents	Web Task Mind2Web
GPT/API-based Agents	
GPT-3.5-turbo†	15.7
Claude†	21.0
GPT-4†	22.6
Open-source Agents	
Baichuan-13B-chat†	2.3
WizardLM-30B†	3.1
Koala-13B†	6.0
AgentLM-70B♡	13.5
LUMOS-I_{Web}	27.6
LUMOS-I_{Web}-13B	**31.3**

(a) Performance on web tasks. The metric is step-wise success rate (%).

Agents	Math Tasks	
	GSM8K	SVAMP
Open-source Agents		
AgentLM-13B♡	32.4	-
Code-Llama (PoT)-13B¶	36.1	60.0
Platypus-30B¶	37.8	51.7
ReWOO-open‡	≈38	-
Orca-Platypus-13B¶	38.4	56.9
Alpaca-7B‡	≈39	-
Galactica-30B¶	41.7	41.6
LUMOS-O_{Math}	50.5	65.5
LUMOS-I_{Math}	47.1	63.6
LUMOS-O_{Math}-13B	**55.4**	**69.3**

(b) Performance on math tasks in accuracy (%).

Agents	Agent Model	QA Tool	Complex QA Tasks	
			StrategyQA	HotpotQA (LLM Acc. / EM)
GPT/API-based Agents				
GPT-3.5-CoT‡	GPT-3.5-turbo	GPT-3.5-turbo	56.0	37.8 / 22.4
ReAcT‡	GPT-3.5-turbo	GPT-3.5-turbo	64.6	40.8 / 32.4
ART*	GPT-3	GPT-3	66.4	- / -
ReWOO‡	GPT-3.5-turbo	GPT-3.5-turbo	66.6	42.4 / 30.4
Open-source Agents				
ReWOO-open‡	LLAMA-7B	GPT-3.5-turbo	≈56	≈37 / -
AgentLM‡♡	LLAMA-2-7B	LLAMA-2-7B	-	- / 22.3
FiReAct‡♠	LLAMA-2-7B	LLAMA-2-7B	-	- / 26.2
FiReAct‡♠	CodeLLAMA-34B	CodeLLAMA-34B	-	- / 27.8
LUMOS-O_{QA}	LLAMA-2-7B	GPT-3.5-turbo	60.6	39.2 / 24.9
LUMOS-I_{QA}	LLAMA-2-7B	GPT-3.5-turbo	65.7	45.9 / 29.4
LUMOS-I_{QA}	LLAMA-2-13B	GPT-3.5-turbo	65.3	50.2 / 31.4
LUMOS-I_{QA}	LLAMA-2-7B	GPT-4	**72.4**	**56.8 / 36.0**

(c) Performance on complex QA tasks. The evaluation metric for StrategyQA and HotpotQA is accuracy (%), and LLM accuracy / Exact Match (%), respectively.

Table 1: Overall performance on diverse complex interactive tasks. Baseline results are reported in †Liu et al. (2023a), ‡Xu et al. (2023a), *Paranjape et al. (2023), ¶Yue et al. (2023), ♡Zeng et al. (2023) and ♠Chen et al. (2023), respectively.

in Table 1, task-specific agents like LUMOS-I_X are trained using task-specific data corresponding to task type X (e.g., Web, Math, QA). The details for performance evaluation is shown in Appendix D.

LUMOS vs. GPT-based Agents. Although LUMOS is built on LLAMA-2-7B and LLAMA-2-13B, it does not inherently underperform the strong GPT-4/3.5-based agents. Specifically, LUMOS-I delivers superior performance by 5.0% over GPT-4 on the Mind2Web dataset, and a 3.5% increase in LLM accuracy over the GPT-based ReWOO agent on the HotpotQA dataset when employing GPT-3.5-turbo as the implementation of the QA tool to ensure fair comparisons. The emphasis for these experiments was to draw a fair comparison with existing performance benchmarks.

LUMOS vs. Larger Open-Source Agents. We have conducted comparisons of LUMOS with other existing agents powered by open-source LLMs. According to our observations, LUMOS consistently outperforms various open-source LLM agents across all five datasets. Despite the fact that the base models of the compared language agents are approximately $2-4\times$ larger than LUMOS in terms of model size, LUMOS significantly excels in performance. Specifically, 7B LUMOS-I achieves 24.5% and 14.1% step success rate improvements over WizardLM-30B and AgentLM-70B, respectively, on the Mind2Web dataset. Additionally, LUMOS-I provides a 1.6% higher EM than CodeLLAMA-34B (Roziere et al., 2023). It is relevant to point out that despite ReWOO and FiReAct being fine-tuned with in-domain HotpotQA annotations, LUMOS-I, without any fine-tuning using HotpotQA annotations, still presents an impressive improvement. In particular, LUMOS-I surpasses the 7B-scale FiReAct by 3.2% EM. A similar trend is also observed on Math tasks.

4.3 Ablation Analysis on Modular Training of Agents

In this subsection, we evaluate the effectiveness of the modular design in LUMOS. We train models using the same base model and data, but with different training methods - **Chain-of-Thoughts (CoT) Training**: For a given task T, the agent learns to produce both the chain-of-thoughts solution

Agents	Web Task
	Mind2Web
Integrated Training	25.3
LUMOS-I$_{\text{Web}}$	**27.6**

(a) Web tasks.

Agents	Math Tasks	
	GSM8K	SVAMP
CoT Training	40.4	52.2
Integrated Training	45.5	61.7
LUMOS-O$_{\text{Math}}$	**50.5**	**65.5**
LUMOS-I$_{\text{Math}}$	47.1	63.6

(b) Math tasks.

Agents	Complex QA Tasks	
	StrategyQA	HotpotQA
CoT Training	58.3	22.1
Integrated Training	62.3	39.6
LUMOS-O$_{\text{QA}}$	60.6	39.2
LUMOS-I$_{\text{QA}}$	**65.7**	**45.9**

(c) Complex QA tasks.

Table 3: Comparison among different formulations of training language agents. The metric for HotpotQA is LLM accuracy (%). All the experiments are based on LLAMA-2-7B.

and the final answer directly; **Integrated Agent Training**: For a given task T, the agent learns to generate all the *subgoals* and *actions* using the same model. The execution modules remains the same. This training paradigm is adopted in ReWOO-open, FiReAct and AgentLM.

From Table 3, both LUMOS-I and LUMOS-O significantly outperform CoT Training[2]. Moreover, LUMOS formulations also perform better than the integrated formulation that solely relies on a single model to perform planning and grounding. It highlights the importance of disentangling the skills for subgoal planning and action grounding during the agent training.

Agents	Unseen Task
	WebShop
Open-source Agents	
Baichuan-13B-chat†	5.7
Koala-13B†	6.0
WizardLM-30B†	10.6
Vicuna-v1.1-13B†	12.6
ChatGLM2†	19.4
Vicuna-v1.3-33B†	23.9
Vicuna-v1.5-13B†	41.7
OpenChat-v3.2-13B†	46.9
Claude-instant†	49.7
Domain-specific Agents	
LUMOS-I$_{\text{Web}}$	34.7
LUMOS-I$_{\text{Math}}$	30.1
LUMOS-I$_{\text{QA}}$	33.5
LUMOS-I$_{\text{All}}$	39.8
LUMOS-I$_{\text{All}}$-13B	**50.3**

Table 2: Performance on the unseen task, WebShop. The metric is the average reward defined in Yao et al. (2022a). † indicates the results reported in the two versions of AgentBench (Liu et al., 2023a).

4.4 LUMOS ON UNSEEN TASKS

Given that LUMOS deploys a unified format to represent complex interactive tasks, we envision it to possess superior cross-task generalization ability. In other words, when faced with a task that was not present in the training data, LUMOS can adapt to it more effectively with few-shot examples.

In order to examine the generalization ability of LUMOS, we use WebShop (Yao et al., 2022a) as the unseen task. It should be noted that WebShop does not qualify as a Web task by the definition we used earlier, which incorporates HTML pages and simulates operations such as clicking and typing. WebShop more closely resembles a text game[3], with its shopping environment and action space considerably differing from those covered in the training annotations of LUMOS. To adjust LUMOS to this new task, we supplement the input of the planning and grounding modules with two-shot examples, enabling them to learn how to generate subgoals and ground to new sets of available actions (for more details, see Appendix G).

As demonstrated in Table 2, LUMOS-I achieves a 5-10% higher average reward than domain-specific agents on WebShop. Moreover, it significantly outperforms larger language agents such as WizardLM-30B and Vicuna-v1.1-13B (Chiang et al., 2023). For 13B-scale LUMOS-I, it even surpasses Vicuna-v1.3-33B and API-based Claude-instant[4]. This suggests that unified training enables agents to improve their cross-task generalization capacity, thereby enhancing their ability to plan for unseen tasks and action types.

Training Data	Complex QA	
	StrategyQA	HotpotQA
Downstream Perf. of Training Different Data		
ReWOO-Planner Data	≈57	≈37
LUMOS-I$_{\text{QA}}$ Data	**58.3**	**38.1**
Perf. Using High-Level and Low-Level Subgoal Annots.		
LUMOS-I$_{\text{QA}}$ w/ Low-Level Subgoals	63.3	44.3
LUMOS-I$_{\text{QA}}$ Data	**65.7**	**45.9**

Table 4: Comparison between the 7B-sized agents trained with different annotations.

[2]Note that we do not implement CoT training on web tasks, as updates to the environment (e.g., changes to HTML) are necessary intermediaries for planning subsequent actions.

[3]WebShop utilizes four actions in its training annotations: `Search`, `FeatureRetrieve`, `Pick`, and `Click`. The argument of `Click` is a shopping item, differing from the argument for `Click` in Mind2Web which includes an HTML element description.

[4]https://www.anthropic.com/product.

4.5 FURTHER ANALYSIS ON TRAINING ANNOTATIONS

In the analysis of annotations, we aim to address two questions pertinent to quality and format decisions. **Q1: How good are our converted training annotations? Q2: Would adopting low-level subgoals be more effective than using high-level subgoals?**

Assessment of Annotation Quality. We assess the quality of our annotations by training models with these annotations and evaluating the agents' performance. We conduct a comparison with ReWOO-Planner annotations, another source of training annotations for language agents. These annotations, which are constructed based on HotpotQA and TriviaQA (Joshi et al., 2017) using Self-Instruct method. To foster a fair comparison, we train the base model of ReWOO-Planner, LLAMA-7B, using LUMOS annotations. We sample 2,000 training data to keep the same size with ReWOO-Planner's annotations. Given that ReWOO-Planner data exclusively relies on QA benchmarks, we primarily focus on complex QA tasks for our comparison. Shown in Table 4, our training annotations yield a 2.3% improvement in EM and 1.1% in LLM accuracy when compared to the ReWOO-Planner on StrategyQA and HotpotQA. Note that even though the ReWOO-Planner data is based on HotpotQA, it still does not outperform LUMOS on HotpotQA. It further proves the effectiveness of our proposed annotation conversion method.

Low-Level Subgoal vs. High-Level Subgoal. As described in §2, we ask LLMs to generate high-level subgoals corresponding to one or many low-level actions. An alternative annotation could be one where each subgoal corresponds solely to one low-level action, i.e., the subgoal is "low-level". We direct LLMs to create these annotations with low-level subgoals by modifying the annotation conversion prompt to fit the format where a subgoal is strictly linked to one action. Table 4 reveals a decrease after replacing high-level subgoals with low-level ones across both the QA datasets. This result hence reaffirms the appropriateness of our initial subgoal design.

5 RELATED WORK

Language Agents. Empowered by reinforcement learning, language agents have been previously deployed in text game environments, such as TextWorld (Côté et al., 2019) and LIGHT (Urbanek et al., 2019). With the advancement of LLMs, language agents have shown potential in solving diverse complex interactive tasks. ReAct (Yao et al., 2022b) introduced a prompting method that shaped LLMs as language agents and grounded them in an external environment. Subsequently, several methods (Shen et al., 2023; Lu et al., 2023; Xu et al., 2023a; Lin et al., 2023; Liu et al., 2023b) aimed at improving agent performance and increasing their applicability in diverse scenarios. However, these agents mainly rely on closed LLMs, lacking of the consideration of affordability, reproducibility and transparency issues when being applied on complex interactive tasks. We focus on exploring 1) the training paradigms that enable open LLMs to tackle complex tasks, and 2) new processes to generate high-quality training annotations adhering to these paradigms.

Improving Small Models for Building General Agents. Knowledge distillation (Hinton et al., 2015) aims at transferring knowledge from larger models to smaller models. Several recent works have utilized LLMs to generate explicit training data for fine-tuning smaller models (Bosselut et al., 2019; West et al., 2022; Wang et al., 2023b; Brahman et al., 2023; Li et al., 2023). However, we observed that directly generating annotations for training planning and grounding modules may introduce a large number of errors, given that LLMs (e.g., GPT-4) behave badly on some complex interactive tasks (Liu et al., 2023a). Instead, we use LLMs to transform the diverse gold reasoning steps into LUMOS format to acquire large-scale high-quality data to train agents.

6 CONCLUSION

We introduce LUMOS, a language agent learning framework. We propose two training formulations, LUMOS-I and LUMOS-O, which promote collaboration among agent modules to solve complex interactive tasks. To obtain annotations for training modules, we use LLMs to transform reasoning steps in existing benchmarks into a unified format applicable within the LUMOS framework. Built solely on LLAMA-2-7B/13B, LUMOS performs better than GPT-series agents on QA and web tasks, and competes with $2-4\times$ larger agents on math tasks. We also show that LUMOS outperforms potential agent training formulations and exhibits superior generalization on unseen interactive tasks.

REFERENCES

Antoine Bosselut, Hannah Rashkin, Maarten Sap, Chaitanya Malaviya, Asli Celikyilmaz, and Yejin Choi. COMET: Commonsense transformers for automatic knowledge graph construction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4762–4779, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1470. URL https://aclanthology.org/P19-1470.

Faeze Brahman, Chandra Bhagavatula, Valentina Pyatkin, Jena D Hwang, Xiang Lorraine Li, Hirona J Arai, Soumya Sanyal, Keisuke Sakaguchi, Xiang Ren, and Yejin Choi. Plasma: Making small language models better procedural knowledge models for (counterfactual) planning. *arXiv preprint arXiv:2305.19472*, 2023.

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

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. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023. URL https://lmsys.org/blog/2023-03-30-vicuna/.

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

Marc-Alexandre Côté, Akos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, et al. Textworld: A learning environment for text-based games. In *Computer Games: 7th Workshop, CGW 2018, Held in Conjunction with the 27th International Conference on Artificial Intelligence, IJCAI 2018, Stockholm, Sweden, July 13, 2018, Revised Selected Papers 7*, pp. 41–75. Springer, 2019.

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

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361, 2021. doi: 10.1162/tacl_a_00370. URL https://aclanthology.org/2021.tacl-1.21.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.550. URL https://aclanthology.org/2020.emnlp-main.550.

Liunian Harold Li, Jack Hessel, Youngjae Yu, Xiang Ren, Kai-Wei Chang, and Yejin Choi. Symbolic chain-of-thought distillation: Small models can also "think" step-by-step. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2665–2679, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.150. URL https://aclanthology.org/2023.acl-long.150.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Prithviraj Ammanabrolu, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. *ArXiv preprint*, abs/2305.17390, 2023. URL https://arxiv.org/abs/2305.17390.

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

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

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. *arXiv preprint arXiv:2304.09842*, 2023.

OpenAI. ChatGPT. 2022. URL https://openai.com/blog/chatgpt.

OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023.

Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. Art: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*, 2023.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*, 2021.

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

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

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*, 2023.

Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*, 2023.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. doi: 10.1162/tacl_a_00475. URL https://aclanthology.org/2022.tacl-1.31.

Jack Urbanek, Angela Fan, Siddharth Karamcheti, Saachi Jain, Samuel Humeau, Emily Dinan, Tim Rocktäschel, Douwe Kiela, Arthur Szlam, and Jason Weston. Learning to speak and act in a fantasy text adventure game. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 673–683, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1062. URL https://aclanthology.org/D19-1062.

Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Raghavi Chandu, David Wadden, Kelsey MacMillan, Noah A Smith, Iz Beltagy, et al. How far can camels go? exploring the state of instruction tuning on open resources. *arXiv preprint arXiv:2306.04751*, 2023a.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13484–13508, Toronto, Canada, July 2023b. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.754. URL https://aclanthology.org/2023.acl-long.754.

Peter West, Chandra Bhagavatula, Jack Hessel, Jena Hwang, Liwei Jiang, Ronan Le Bras, Ximing Lu, Sean Welleck, and Yejin Choi. Symbolic knowledge distillation: from general language models to commonsense models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4602–4625, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.341. URL https://aclanthology.org/2022.naacl-main.341.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. Rewoo: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*, 2023a.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*, 2023b.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2022b.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. Mammoth: Building math generalist models through hybrid instruction tuning. *arXiv preprint arXiv:2309.05653*, 2023.

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

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

APPENDIX

A STATISTICS OF CONVERTED TRAINING ANNOTATIONS

As discussed in §4.1, the data sources for constructing training annotations cover a broad range of complex interactive tasks. Table 5 shows the benchmarks leveraged for annotation conversion, along with the task type information.

To train agents like LUMOS-I$_{\text{Math}}$ mentioned in Table 1b, we need to leverage the annotations converted from 19778 data specific to maths domain. For training a unified agent such as LUMOS-I, we would use the annotations transformed from all the listed data as training set.

B DETAILS OF TRAINING MODULES

We describe additional details about our training experiments. We set the maximum sequence length to 1024. We also apply linear warmup for 3% of the total training steps to adjust the learning rate. All the training experiments are implemented with 2 NVIDIA 80GB A100 GPUs.

Task Types	Datasets	# Conversion	# Total
Math	PRM800K	10000	19778
	GSM8K	7473	
	ASDiv	2305	
Complex QA	Musique	17632	19409
	StrategyQA	1777	
Web	Mind2Web	1009	1009

Table 5: Statistics of data sources for conversion.

C ILLUSTRATION OF ANNOTATION ORGANIZATION

D DETAILS OF PERFORMANCE EVALUATION

Metrics. Here we mainly discuss the special metrics adopted to evaluate the agent performance. For HotpotQA, instead of merely using strict exact matching, we follow Xu et al. (2023a) to also use GPT-4 as an evaluator to judge whether the predicted answer shares the same semantics with the gold answer. We call this metric as LLM accuracy, frequently mentioned in §4. For Mind2Web, we adopt the same metric step success rate used for AgentBench evaluation. A step is deemed successful solely when both the chosen HTML tag and predicted action type exactly match the gold action. For WebShop, we leverage the reward utilized in both AgentBench and original WebShop paper, which quantify the similarity between gold and predicted products with regard to product titles and selected attributes.

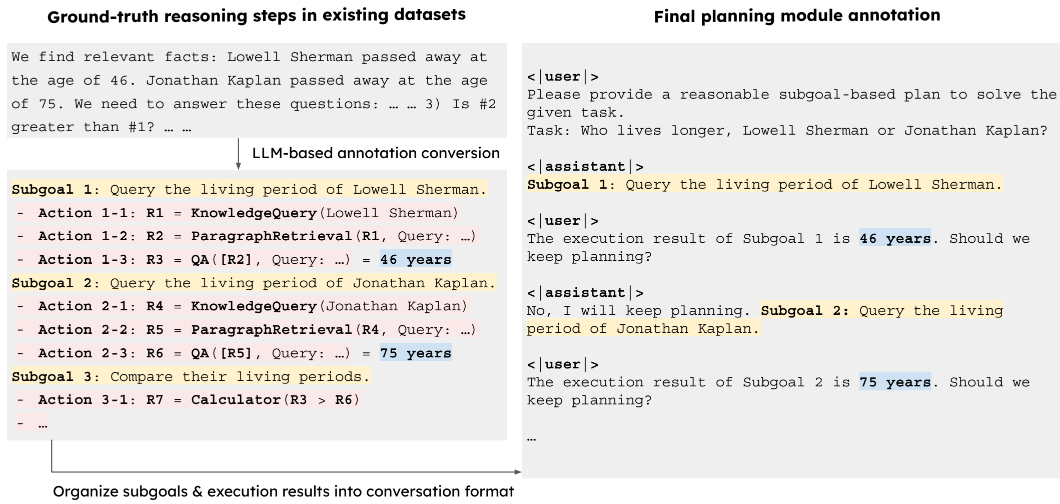
Evaluation Data. Following Xu et al. (2023a), we only evaluate 300 and 1000 randomly selected examples from StrategyQA and HotpotQA evaluation set, respectively. The results reported in Table 1c are the average performance on three different sets of sampled data. Regarding Mind2Web, we only evaluate on the "cross-domain" test set that AgentBench utilizes for evaluation. For WebShop, we evaluate the first 500 instances from the entire test set as AgentBench used to do.

E EXECUTION TOOLS ASSOCIATED WITH ACTION SPACES

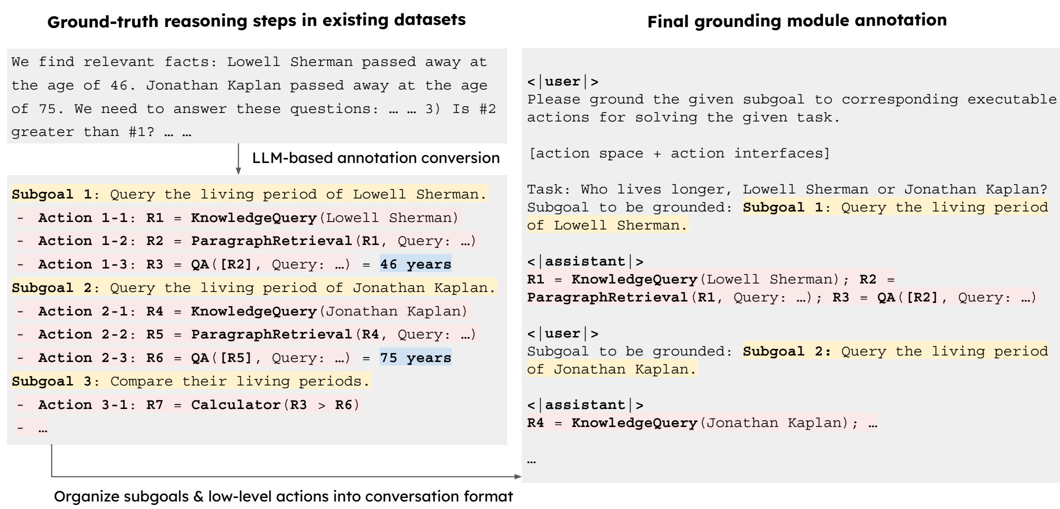
For each available action defined in the action spaces, there are at least one associated backend execution tools that help to implement the actual grounded actions.

Displayed in Table 6a, for complex QA tasks, we rely on Wikipedia and Google search to help us locate relevant entity knowledge. Besides, we leverage a pre-trained semantic matching model `dpr-reader-multiset-base`[5] used in Dense Passage Retrieval (DPR) (Karpukhin et al., 2020) to capture relevant paragraphs according to the given query. Following ReWOO (Xu et al., 2023a),

[5] https://huggingface.co/facebook/dpr-reader-multiset-base.



(a) Final planning module annotation organized from the converted subgoals & execution results.



(b) Final grounding module annotation organized from the converted subgoals & actions.

Figure 3: Process of converting converted subgoals, actions, and executions into the final conversational training annotations for LUMOS-I formulation.

we also include GPT-series model as a simple QA tool to answer the query based on our retrieved knowledge.

Demonstrated in Table 6b, for web tasks, the actions are real mouse and keyboard operations including typing, clicking and selecting HTML tags. To locate the relevant HTML tags to be operated, following AgentBench evaluation, we use a pre-trained DeBERTa model[6] that ranks and retrieves the tags according to the current action we would perform.

As shown in Table 6c, for maths tasks, the main execution tool is WolframAlpha API [7] as it is capable of performing a large collection of mathematical functions such as calculating formulas and solving equations. For complex mathematical operations such as sorting, we would leverage OpenAI Codex (Chen et al., 2021) to generate a short code snippet for execution.

For WebShop unseen task, the actions include `Search`, `FeatureRetrieve`, `Pick`, and `Click`. The implementation of `Search` and `Click` relies on the embedded implementation already pro-

[6] https://huggingface.co/osunlp/MindAct_CandidateGeneration_deberta-v3-base.
[7] https://www.wolframalpha.com/.

Task Type	Action Types	Function Descriptions	Implementation
Complex QA	`KnowledgeQuery(Entity) -> Knowledge`	Query the entity knowledge	Wikipedia, Google Search
	`ParagraphRetrieval(Knowledge, Query) -> Paragraphs`	Retrieve relevant paragraphs according to the query	`dpr-reader-multiset-base`
	`QA(Context, Query) -> Answer`	Answer the query based on the given context	GPT-series/open LLMs
	`Calculator(Expression) -> Value`	Calculate given math expressions	WolframAlpha

(a) Actions used in complex QA tasks.

Task Type	Action Types	Function Descriptions	Implementation
Web	`Click(Env, Query) -> Tag`	Locate the tag to be clicked according to the query	HTML Simulator
	`Type(Env, Query, Text) -> Tag, Text`	Locate the relevant tag according to the query and output the typed text	
	`Select(Env, Query, Text) -> Tag, Text`	Locate the relevant tag according to the query and output the selected option	

(b) Actions used in web tasks.

Task Type	Action Types	Function Descriptions	Implementation
Math	`Calculator(Expression) -> Value`	Calculate given math expressions	WolframAlpha
	`SetEquation(Expression) -> Equation`	Set equations based on given expressions	
	`SolveEquation(Equation) -> Solutions`	Solve the set equations	
	`Define(Variable) -> Variable`	Define a variable	
	`SolveInequality(Inequality) -> Solutions`	Solve the given inequality	
	`Code(Function_Description) -> Code`	Generate codes for math functions	`gpt-3.5-turbo`
	`Count(List) -> Number`	Count the element number in a list	Python

(c) Actions used in math tasks.

Table 6: Action space and execution module implementations for complex interactive tasks.

vided in official WebShop virtual environment[8]. `FeatureRetrieve` and `Pick` are based on `dpr-reader-multiset-base`, which helps to select the most relevant items and their item features according to the query.

F In-Context Examples in Conversion Prompts

As discussed in §3.1, in-context examples are helpful to instruct LLMs to generate annotations in our expected format. For each of the training task types, we showcase one in-context example to help readers better understand how the prompting conversion method works and the format of our expected annotations. We highlight subgoals and their corresponding actions and execution results with yellow, red and blue, respectively.

[8] `https://github.com/princeton-nlp/WebShop`.

F.1 IN-CONTEXT EXAMPLE FOR OBTAINING MATH TASK ANNOTATIONS

```
Please convert natural language plans into a series of subgoals and their corresponding
actions that lead to the successful implementation with respect to the given instructions.
Please use 'R[number]' to represent the intermediate results for each subgoal, without
generating any exact values. Please also use functions to represent the corresponding
actions. For the actions, they must be one of 'Calculator', 'SetEquation', 'SolveEquation',
'SolveInequality', 'Count', 'Code', and 'Define'.

Example 1:

Task: Peter goes to the store to buy a soda. The soda costs $.25 an ounch. He brought $2
with him and leaves with $.50. How many ounces of soda did he buy?

Natural language plan:
He spend $1.5 on soda because 2 - .5 = 1.5 He bought 6 ounces of soda because 1.5 / .25 =
6

Subgoal-based plan:
Subgoal 1: Calculate how much the soda costs in total.
Action 1-1: R1 = Calculator(2 - 0.5) = 1.5

Subgoal 2: Calculate the ounces of soda the price per ounch.
Action 2-1: R2 = Calculator(R1 / 0.25) = 6
```

F.2 In-Context Example For Obtaining Complex QA Task Annotations

```
Please convert natural language plans into a series of subgoals and their corresponding
actions that lead to the successful implementation with respect to the given instructions.
Please use ‘R[number]’ to represent the intermediate results for each subgoal, without
generating any exact values. Please also use functions to represent the corresponding
actions. For the actions, they must be one of one of ‘KnowledgeQuery’, ‘ParagraphRetrieve’,
‘QA’, ‘Calculator’ and ‘Code’.

Example 1:

Task: Are more people today related to Genghis Khan than Julius Caesar?

Natural language plan:
We find relevant facts: Julius Caesar had three children. Genghis Khan had sixteen
children. Modern geneticists have determined that out of every 200 men today has DNA that
can be traced to Genghis Khan. We need to answer these questions: 1. How many kids did
Julius Caesar have? (Can be answered based on paragraph ‘Julius Caesar-75’) 2. How many
kids did Genghis Khan have? (Can be answered based on paragraph ‘Genghis Khan-17’) 3. Is
#2 greater than #1? Based on these evidences and decomposed questions, the answer is True.

Subgoal-based plan:
Subgoal 1: Obtain the number of the kids that Julius Caesar had.
Action 1-1: R1 = KnowledgeQuery(Julius Caesar) = WikipediaPage(Julius Caesar)
Action 1-2: R2 = ParagraphRetrieve(R1, Query: How many kids did Julius Caesar have?) =
Paragraph(Julius Caesar-75).
Action 1-3: R3 = QA([R2], Question: How many kids did Julius Caesar have?) = 3.

Subgoal 2: Obtain the number of the kids that Genghis Khan had.
Action 2-1: R4 = KnowledgeQuery(Genghis Khan) = WikipediaPage(Genghis Khan).
Action 2-2: R5 = ParagraphRetrieve(R4, Query: How many kids did Genghis Khan have?) =
Paragraph(Genghis Khan-17).
Action 2-3: R6 = QA([R5], Question: How many kids did Genghis Khan have?) = 16.

Subgoal 3: Determine if Genghis Khan had more kids.
Action 3-1: R7 = Calculator(R6 > R3) = True
```

F.3 IN-CONTEXT EXAMPLE FOR OBTAINING WEB TASK ANNOTATIONS

Since the data source for converting annotations, Mind2Web, already provides the ground-truth execution results after each action, as discussed in §3.1, we do not ask LLMs to capture each action's execution results. Therefore, there are no parts highlighted with blue in the in-context example.

```
Please convert natural language plans into a series of subgoals and their corresponding
actions that lead to the successful implementation with respect to the given instructions.
Please use ‘R[number]’ to represent the intermediate results for each subgoal, without
generating any exact values. Please also use functions to represent the corresponding
actions. For the actions, they must be one of they must be one of ‘TYPE’, ‘CLICK’, and
‘SELECT’.

Example 1:

Task: Find a Ricky Kej track to listen and share which has been added in the last year and
is between 2 to 10 minutes.

Natural language plan:
[searchbox] Search ⟶ TYPE: Ricky Kej; [link] Search for ‘‘Ricky Kej’’ ⟶ CLICK; [link]
Tracks ⟶ CLICK; [link] Added any time ⟶ CLICK; [link] Past year ⟶ SELECT; [link]
Any length ⟶ CLICK; [link] 2-10 min ⟶ CLICK; [link] To listen to ⟶ CLICK; [link]
To share ⟶ CLICK

Subgoal-based plan:
Subgoal 1: Type Ricky Kej to search his songs.
Action 1-1: R1 = TYPE(Env, QUERY: Type Ricky Kej to search his songs, TEXT: Ricky Kej)

Subgoal 2: Click on the option to search for Ricky Rej.
Action 2-1: R2 = CLICK(R1, QUERY: Click on the option to search for Ricky Rej)

Subgoal 3: Choose tracks as the search category.
Action 3-1: R3 = CLICK(R2, QUERY: Choose tracks as the search category)

Subgoal 4: Find the region to adjust the added time of our interested track.
Action 4-1: R4 = CLICK(R3, QUERY: Find the region to adjust the added time of our interested
track)

Subgoal 5: Choose the last year as the added date.
Action 5-1: R5 = SELECT(R4, QUERY: Choose the last year as the added date, TEXT: Past
year)

Subgoal 6: Find the region to adjust the track length of our interested track.
Action 6-1: R6 = CLICK(R5, QUERY: Find the region to adjust the track length of our
interested track)

Subgoal 7: Choose 2 to 10 minutes as the track length.
Action 7-1: R7 = CLICK(R6, QUERY: Choose 2 to 10 minutes as the track length)

Subgoal 8: Listen to our searched track.
Action 8-1: R8 = CLICK(R7, QUERY: Listen to our searched track)

Subgoal 9: Share our searched track.
Action 9-1: R9 = CLICK(R8, QUERY: Share our searched track)
```

G In-Context Example for Adapting Modules to Unseen Task — WebShop

As discussed in §4.4, the in-context examples are used for helping modules understand how to plan and ground to novel available actions. Mentioned in Appendix E, WebShop involves four unseen actions in training annotations: `Search`, `FeatureRetrieve`, `Pick`, and `Click`. The definition of `Click` is different from the one defined for web task. In WebShop task, the argument of `Click` is the item index. However, `Click`'s input arguments in Mind2Web include an HTML environment description and a specific query regarding the goal of click action. We showcase one in-context WebShop-related example used for the inference of planning and grounding modules, respectively.

G.1 In-Context WebShop-Related Example for Planning Module

```
<|user|>
Please provide a reasonable subgoal-based plan to solve the given task.
Task: Find me slim fit, loose fit women's tops, tees & blouses with long sleeve, short
sleeve with color: a01#black, and size: large, and price lower than 50.00 dollars

<|assistant|>
Subgoal 1: Search two relevant items and the product names according to the instruction.

<|user|>
The execution result for Subgoal 1 is b09s3bn15c - Mens Linen Shirt,Men's Striped Shirts
Casual Short Sleeve Button Down Shirts Regular Fit Hawaiian Shirts Beach Tees Tops **
b094q7b3ss - Women Cold Shoulder Tops, Summer Butterfly Print Shirts Fashion Casual Short
Sleeve Plus-Size Tunic Top Tee and Blouse.

<|assistant|>
Subgoal 2: Select the most relevant features of item b09s3bn15c.

<|user|>
The execution result for Subgoal 2 is short, casual, shoulder.

<|assistant|>
Subgoal 3: Select the most relevant features of item b094q7b3ss.

<|user|>
The execution result for Subgoal 3 is black, large, x-large.

<|assistant|>
Subgoal 4: Pick up the most related one from the two relevant items according to the
product names and their features.

<|user|>
The execution result for Subgoal 4 is b094q7b3ss.

<|assistant|>
Subgoal 5: Buy this item by selecting its relevant features.

<|user|>
The execution result for Subgoal 5 is Done.
```

G.2 In-Context WebShop-Related Example for Grounding Module

```
<|user|>
Please ground the given subgoal to corresponding executable actions for solving the given
task. The grounded actions must be the one in available action list.

The available action list is ‘Search’, ‘FeatureRetrieve’, ‘Pick’ and ‘Click’.
Search(Query): Search related items according to the Query; FeatureRetrieve(Feature_list,
Query): Select the most relevant features from Feature_list according to Query;
Pick(Item_names, Item_features, Query): Pick the most relevant item from Item_names
according to Query, Item_names and Item_features; Click(Item): Click the Item to check
more information.

Task: Find me slim fit, loose fit women’s tops, tees & blouses with long sleeve, short
sleeve with color: a01#black, and size: large, and price lower than 50.00 dollars
Subgoal to be grounded: Subgoal 1: Search two relevant items and the product names according
to the instruction.

<|assistant|>
R1 = Search(Find me slim fit, loose fit women’s tops, tees & blouses with long sleeve,
short sleeve with color: a01#black, and size: large, and price lower than 50.00 dollars)

<|user|>
Subgoal to be grounded: Subgoal 2: Select the most relevant features of item b09s3bn15c.

<|assistant|>
R2 = Click(b09s3bn15c); R3 = FeatureRetrieve(R2, Query: Find me slim fit, loose fit women’s
tops, tees & blouses with long sleeve, short sleeve with color: a01#black, and size: large,
and price lower than 50.00 dollars)

<|user|>
Subgoal to be grounded: Subgoal 3: Select the most relevant features of item b094q7b3ss.

<|assistant|>
R4 = Click(< prev); R5 = Click(b094q7b3ss); R6 = FeatureRetrieve(R5, Query: Find me slim
fit, loose fit women’s tops, tees & blouses with long sleeve, short sleeve with color:
a01#black, and size: large, and price lower than 50.00 dollars)

<|user|>
Subgoal to be grounded: Subgoal 4: Pick up the most related one from the two relevant
items according to the product names and their features.

<|assistant|>
R7 = Pick(Item_names: R1, Item_features: [R3, R6], Query: Find me slim fit, loose fit
women’s tops, tees & blouses with long sleeve, short sleeve with color: a01#black, and
size: large, and price lower than 50.00 dollars)

<|user|>
Subgoal to be grounded: Subgoal 5: Buy this item by selecting its relevant features.

<|assistant|>
R8 = Click(< prev); R9 = Click(R7); R10 = Click(black); R11 = Click(large); R12 =
Click(x-large); R13 = Click(buy now)
```