Agent LUMOS: Unified and Modular Training for Open-Source Language Agents

Anonymous ACL submission

Abstract

Closed-source agents suffer from several issues such as a lack of affordability, transparency, and reproducibility, particularly on complex interactive tasks. This motivates the development of open-source alternatives. We introduce LUMOS, one of the first frameworks for training open-source LLM-based agents. LUMOS features a learnable, unified and modular architecture with a planning module that learns high-level subgoal generation, and a grounding module trained to translate these into the actions using various tools in the execution module. The design allows for modular upgrades and wider applicability to diverse interactive tasks. To foster generalizable agent learning, we collect large-scale, unified, and high-quality training annotations derived from diverse ground-truth reasoning rationales across various complex interactive tasks. On 9 datasets, LUMOS exhibits several key advantages: (1) LUMOS excels multiple larger open-source agents on the held-out datasets (unused for training) for each task type. LUMOS even surpasses GPT agents on QA and web tasks; (2) LUMOS outperforms open-source agents produced by chain-of-thoughts and unmodularized integrated training; and (3) LUMOS effectively generalizes to unseen tasks, outperforming 33B-scale agents and domain-specific agents. Code and data will be released.

1 Introduction

Language agents execute actions and interact with external tools or environments, in service of a goal. They have evolved into crucial elements of AI systems targeted at solving complex interactive tasks. These tasks often require agents to perform long-horizon planning and interactive reasoning, and can range from QA (Yang et al., 2018; Geva et al., 2021), to web tasks (Deng et al., 2023; Zhou et al., 2023), math (Cobbe et al., 2021), and multimodal reasoning (Schwenk et al., 2022; Lu et al., 2022).

Prior agent frameworks (Yao et al., 2022b; Shinn et al., 2023; Lin et al., 2023; Lu et al., 2023; Liu et al., 2023c) have primarily relied on closed-source large language model (LLM) APIs such as GPT-4 and ChatGPT (OpenAI, 2023a, 2022). Though powerful, they can be prohibitively expensive, particularly for tasks with long contexts such as web tasks (which include encoding long HTML code). Furthermore, the lack of transparency in closed-source LLMs hinders scientific understanding of their architectures and effectiveness, and provides limited reproducibility, and controllability over their behavior. We argue that over reliance on closed-source LLM-based agents is not conducive to the growth of research on language agents.

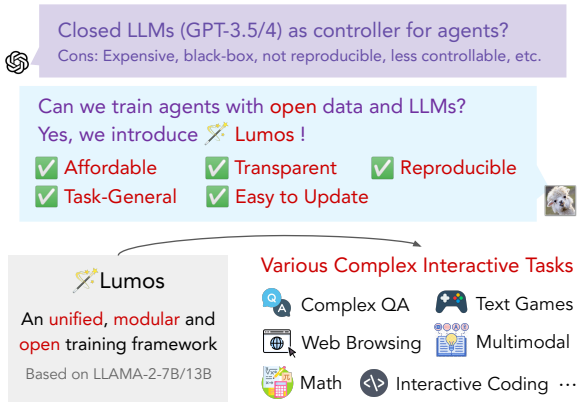


Figure 1: LUMOS is an unified, modular and open-source agent training framework that enables effective cross-task generalization while being easy to be updated. It also has advantages against closed-source agents from affordability, transparency and reproducibility aspects.

In this paper, we propose LUMOS, a generalizable **L**anguage agent framework via **U**nified, **M**odular, and **O**pen **S**ource training. LUMOS employs a unified and modular architecture broadly applicable to complex interactive tasks: a planning module, a grounding module, and an execution module. The planning module learns to decompose diverse complex tasks into a sequence of high-level subgoals. The grounding module is trained to communicate with the planning module

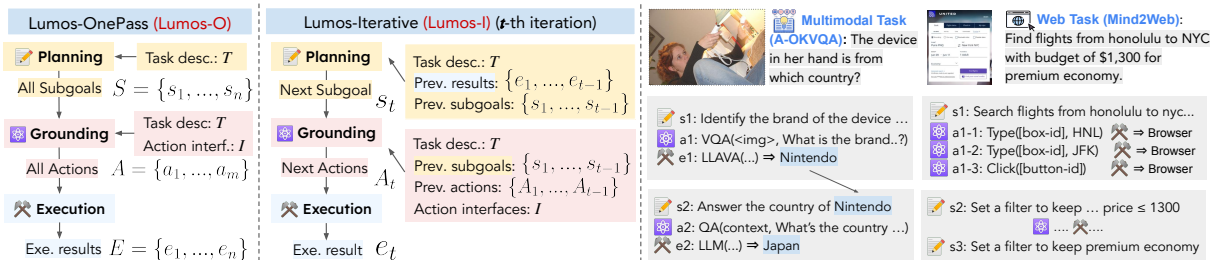


Figure 2: Overall framework of LUMOS. LUMOS are trained with 56K high-quality training annotations. We propose two agent training and inference formulations, LUMOS-O (§2.2) and LUMOS-I (§2.3). LUMOS-O is an efficient formulation that enables one-pass inference; LUMOS-I is an adaptive formulation that help agents flexibly plan based on the execution feedback. We showcase two LUMOS-I running examples in A-OKVQA and Mind2Web.

and translate its generated subgoals into the actions that can be executed through a collection of tools in the execution module. LUMOS design allows for easy module upgrades to enhance new task planning, novel action grounding and supplementing new tools, without impacting the others. To tackle the tasks through the agent modules, we propose two interaction formulations for implementing the language agents, LUMOS-OnePass (LUMOS-O) and LUMOS-Iterative (LUMOS-I). Outlined in Fig. 2, LUMOS-O is an efficient formulation that generates *all* the subgoals and actions through a *single* inference call, accomplishing the task in a one-pass manner. LUMOS-I is an iterative formulation that generates one subgoal at a time based on its previous execution results and environment updates, thereby enabling an adaptive agent.

In addition, LUMOS utilizes a unified data format that encompasses multiple task types, thereby enabling the proposed agent framework to conveniently support a range of interactive tasks. These include, but are not limited to: question answering, mathematics, coding, web browsing, multimodal reasoning, and text games. To obtain high-quality annotations for training LUMOS, we leverage the ground-truth rationales in existing benchmarks across various task types, and convert them into a unified format (§3). This conversion is achieved with the aid of strong LLMs, ensuring that the converted annotations adhere to a universally applicable format consistent with our modular design. Our proposed annotation conversion method results in approximately 56K multi-task multi-domain agent training annotations, one of the largest open-source resources for agent fine-tuning. The training annotations could serve as a resource for universally enhancing any open-source LLMs with agent capabilities.

Our evaluation demonstrates that LUMOS provides improved or comparable performance with GPT-based or larger open-source agents across various complex interactive tasks that are commonly used for agent evaluation, including QA, web, math, and multimodal tasks. We summarize our contributions and results as follows:

General Agent Framework with High-Quality Data. We introduce an open-source agent learning framework that trains LLMs with unified data, aimed at unifying complex interactive tasks and enhancing generalization on unseen tasks with new environments and actions. We hope our framework and annotations can facilitate future research in developing open-source language agents.

Competitive Performance. LUMOS outperforms a great number of open-source agents on the LUMOS held-out datasets unused in LUMOS training data across the four training task types. LUMOS even surpasses GPT-based agents in web and QA tasks. Specifically, LUMOS shows a 5.0% enhancement over GPT-4 on Mind2Web, and 4.1% and 3.5% LLM accuracy[1] improvement on HotpotQA over the ReAct and ReWOO agents fully based on GPT-3.5-turbo, respectively.

Cross-Task Generalization. We evaluate LUMOS on two unseen tasks, WebShop (Yao et al., 2022a), a text game for online shopping, and InterCode$_{SQL}$ (Yang et al., 2023), an interactive code generation task. LUMOS even surpasses 30B-scale agents, especially by nearly 20 reward points on WebShop. LUMOS also delivers a consistent reward improvement over domain-specific agents. This suggests that LUMOS can generalize across tasks, hinting at potential benefits for a wide spectrum of language agent applications.

2 LUMOS: A Modular Open-Source LLM-Based Agent Framework

We introduce the overall design and two formulations for developing agents within this framework.

[1] A metric defined in Xu et al. (2023) to identify the semantic equivalence between predictions and gold answers.

2.1 LUMOS Agent Architecture

For various complex interactive tasks, a common solution would include: (1) decomposing the task into a series of subgoals, (2) converting subgoals into concrete actions, (3) executing those actions. 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 multimodal question such as "`The device in her hand is from which country?`" necessitates two subgoals: (1) Identify the brand of the device in her hand; (2) Answer the country of the device brand. The devised subgoals assist in breaking down a complex task into low-level actions in an interpretable and tool-agnostic manner. The planning module is designed for easy debugging and learning new task planning, without affecting other modules.

Grounding Module (GM). This module transforms 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, such as `KnowledgeQuery(Lowell Sherman)` and `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 to learn new actions without impacting the planning module.

Execution Module (EM). The Execution Module (EM) is a program that implements the actions generated by the grounding module and gets `execution results`. It deploys a variety of off-the-shelf tools, including APIs, neural models, and virtual simulators. For instance, the execution module could call the Wikipedia or Google Search APIs to accomplish the `KnowledgeQuery` action.

The main characteristic of the LUMOS framework is the interaction among the three modules. We propose two formulations promoting the communication: LUMOS-OnePass (LUMOS-O) and LUMOS-Iterative (LUMOS-I).

2.2 LUMOS-OnePass (LUMOS-O)

The LUMOS-OnePass (LUMOS-O) formulation is an efficient method that generates all subgoals and grounded actions at once (efficiency study in App. E). As depicted in Fig. 2, this formulation uses the planning module to generate all n subgoals in a single inference call. We then pass all the generated subgoals to the grounding module, which translates them into a sequence of m low-level actions. Note that in addition to the task description and subgoals, we also provide action interfaces I to the grounding module as inputs. These action interfaces (e.g., "`VQA(Image_Context, Query): Given the image context, answer the query.`") define the functionalities of actions and their valid arguments, guiding the grounding module to produce executable actions. Lastly, for example, the grounding module can produce all the corresponding actions, from `VQA([IMG], Question: What's the brand of the device in her hand?)` to the final one `QA(..., Question: What's the country of ...?)`.

Formally, the overall planning and grounding process of LUMOS-O is illustrated in Fig. 2. In the planning phase, the task description T is input into the planning module. This generates an output series of subgoals, expressed as $S = \pi_{plan}(T) = \{s_1, ..., s_n\}$, where π_{plan} is the parameters of trained planning module. Grounded actions are obtained via $A = \pi_{ground}(T, I, S)$, with reliance on the task description, action interfaces $I = \{i_1, ..., i_k\}$, and the generated subgoals S. π_{ground} represents the parameters of the grounding module. We take the last execution result e_n as the final inference result for the given task.

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 t-th subgoal, the planning module requires the previous planned subgoals and the execution results of their grounded actions as input. The execution results assist the planning module to be aware of the environmental change and decide next subgoal according to the up-to-date environments.

Take the VQA question "`The device in her hand is from which country?`" in Fig. 2 as an example. In the first iteration, the planning module will produce "`Subgoal 1: Identify the brand of the device in her hand`". This subgoal is passed to the grounding module to generate the query actions, and obtain the executed results `Nintendo`. The planning module then takes `Nintendo` along with the prior planning context as input to generate the next subgoal "`Subgoal 2: Answer`

the country of Nintendo". 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 (a case study in App. E).

We demonstrate a single iteration of planning and grounding process of LUMOS-I in Fig. 2. To plan the 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 order 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 output would be the t-th subgoal, $s_t = \pi_{plan}(T, s_1, e_1, ..., s_{t-1}, e_{t-1})$. After then, the t-th subgoal will be directly incorporated into grounding module together with the prior grounding history and action interface I to generate the corresponding actions, $A_t = \pi_{ground}(T, I, s_1, A_1, ..., s_{t-1}, A_{t-1}, s_t)$. 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. 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 action interfaces, we fine-tune the two modules to produce the expected outputs.

Training the modules requires the supervisions consisting of high-quality tasks, 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 training tasks and inputs, and *directly* generate ground-truth task outputs based on its created tasks. However, these methods are not suitable for generating high-quality annotations for training agents. For example, given a web browsing task in Mind2Web, GPT-4 only achieves around 20% step success rate (Liu et al., 2023b) when completing the task. Relying on such methods to generate complex interactive task annotations may degrade the annotation quality.

Instead of creating annotations with LLMs from scratch, we exploit LLMs as a "style transfer" tool to convert ground-truth reasoning steps in existing benchmarks into the expected format in LUMOS formulations. There are a considerable number of the datasets annotated with either human-written solutions or structured action sequences[2]. For example, PRM800K (Lightman et al., 2023) is a math dataset containing the natural language solutions interleaved with formulas; StrategyQA (Geva et al., 2021) are a QA dataset with decomposed questions, supporting facts, and relevant Wikipedia paragraph indices; Mind2Web includes ground-truth action sequences, etc. They provide LLMs with fundamental information that sufficiently contributes to the annotation conversion.

Next, we introduce 1) how we prompt LLMs to obtain the subgoal and action supervisions for training modules; 2) how to organize the subgoals and actions into the conversational forms aligning with LUMOS formulations; 3) how we train the modules with the final annotations.

3.1 Annotation Conversion Prompts

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

Action Interfaces. Action interfaces define the available actions that LLMs could ground to. Table 8 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 references, LLMs are able to summarize high-level subgoals and synthesize corresponding actions according to the given action interfaces.

Subgoals and Corresponding Actions. When converting ground-truth reasoning steps into our expected annotations, we 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 high-level subgoals according to the context of ground-truth reasoning steps. Under each high-level subgoal, we write down multiple corresponding ac-

[2]More available resources for future extension and the discussion about the scalability of our annotation conversion methods are described in App. C.

tions that help to accomplish the subgoal (shown in App. I). Given the exemplar subgoals and actions in the prompt, LLMs would emulate to generate subgoals and their paired actions when performing the conversion 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 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 zombie knowledge page (`R1`). Written in interlinked style, the `ParagraphRetrieve` 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) offer execution results in their reasoning steps, i.e., the computation results of formulas. For the datasets without any execution results, their reasoning steps actually contain the relevant clues for the execution results. We take an example in StrategyQA. Though the answer of the annotated decomposed question "`What color skin are zombies typically depicted with?`" is not directly 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 as well as our manually captured execution results in the conversion prompts. When converting new samples into LUMOS annotations, LLMs would automatically extract the executed results from the given documents.

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 Organizing Conversational Annotations

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

Conversational Planning Module Annotation. As shown in App. A's Fig. 3a, we first play a user role to provide the task $\hat{T}$ in the user prompt. For LUMOS-O, all the subgoals $\hat{S}$ are the planning module's final outputs. LUMOS-I requires multi-turn conversational style. From Fig. 3a, the planning module appends the first ground-truth subgoal $\hat{s}_1$ with index "`Subgoal 1`" as the first response supervision. We then put Subgoal 1's executed result $\hat{e}_1$ with prefix "`The executed result for Subgoal 1 is`" as the second user input. For the remaining turns, we act as the user, provide the execution results $\hat{e}_{t-1}$ of the last subgoal $\hat{s}_{t-1}$ to the planning module, and ask if the planning should be stopped. The response supervisions cover whether the planning should be terminated; if no, the response should contain a new gold subgoal $\hat{s}_t$.

Conversational Grounding Module Annotation. As shown in App. A's Fig. 3b, we also first provide the task $\hat{T}$ and action interfaces $\hat{I}$ to the grounding module in the first user turn. For LUMOS-O, we feed all the converted subgoal annotations $\hat{S}$ in the user prompt. All the action annotations $\hat{A}$ would be the user prompt response. For LUMOS-I, we input the current gold 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 annotations are conversational, we formulate them as $\{x_1, y_1, ..., x_i, y_i, ..., x_n, y_n\}$, where x_i is i-th user prompt and y_i indicates its ground-truth responses. Following Wang et al. (2023a), during training, 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 $L = -\sum_j \log p_\pi(t_j \mid t_{<j}) \times \mathbf{1}(t_j \in Y)$ where t_j denotes j-th input token and $\mathbf{1}(\cdot)$ is a Boolean indicator function.

4 Experiments

We begin with the details of our experimental setup, including annotation conversion, module training, and the tools used in the execution module. We then evaluate LUMOS by: 1) comparing LUMOS with existing open-source LLM agents and GPT-based agents, 2) contrasting LUMOS against other agent training methods, 3) manifesting LUMOS generalizability on two unseen tasks involving new environments and actions, and 4) assessing LUMOS annotation quality.

4.1 Experimental Setup

Data Collection. Utilizing the conversion prompts discussed in §3.1, we employ GPT-4 (Achiam et al., 2023) versions on 8/13/2023 and 9/13/2023, and GPT-4V (OpenAI, 2023b) version on 1/24/2023 to perform annotation conversion on the ground-truth reasoning steps in existing benchmarks. App. B provides the data sources used for annotation conversion. These include the datasets of math, QA, web and multimodal tasks. To help LUMOS be aware of the visual inputs in multimodal tasks, we append a detailed image caption generated by LLAMA-1.5-7B (Liu et al., 2023a) to the task description in train and test sets. After filtering out the ones that contain mismatched parentheses, invalid execution outputs or excessively lengthy outputs, we obtain 55,479 and 55,596 annotations for training the planning and grounding modules, respectively.

Training and Action Interfaces. We adopt LLAMA-2-7B and LLAMA-2-13B (Touvron et al., 2023a) as the base models for both the planning and grounding modules. Details regarding the training process can be found in App. D. For solving interactive tasks, we integrate commonly used actions for each task into the pre-defined action interfaces. Details of supported executable actions are included in App. G.

4.2 Training Task Performance

We evaluate LUMOS across an array of complex interactive tasks - QA, web, math and multimodal tasks. The evaluation mainly follows the settings established by AgentBench (Liu et al., 2023b) and ReWOO (Xu et al., 2023) (see App. H). For each task type, excluding web tasks, we include a held-out dataset to assess the model's generalizability across the domains within the same task category. The performance is displayed in Tab. 1. Note that in Tab. 1, task-specific agents LUMOS-I_X are trained using task-specific data belonging to task type X (e.g., Web, Math, QA, MM). LUMOS-I_{All} represents the agent after unified training with the combination of four task type annotations. More evaluation details are shown in App. H.

LUMOS vs. Open-Source Agents. Overall, we find that LUMOS consistently outperforms various open-source agents across the seven datasets. Though the base models of some compared agents are 2-10× larger than LUMOS, 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 on Mind2Web.

The effectiveness of LUMOS is particularly manifested on the held-out datasets which are unused in LUMOS training data. Our observations reveal that LUMOS outperforms many baseline open-source agents across all held-out datasets. Notably, even though Orca-Platypus-13B (Lee et al., 2023) has been trained on a math corpus that includes GSM8K, its performance still 8.6% lower than LUMOS-O on SVAMP (Patel et al., 2021). Moreover, despite ReWOO and FiReAct being fine-tuned with in-domain HotpotQA annotations, LUMOS-I, without any fine-tuning on HotpotQA, still presents an impressive improvement. A similar trend can be observed on ScienceQA. We compare AutoAct-7B (Qiao et al., 2024), an open-source agent specifically trained on ScienceQA. LUMOS achieves 67.3% on the entire test set of ScienceQA, while AutoAct-7B's performance is only 53.3%.

LUMOS vs. GPT-based Agents. Although LUMOS is built on LLAMA-2-7B/13B, LUMOS-I delivers superior performance by 5.0-8.7% over GPT-4 on Mind2Web. We also notice a 3.5-7.8% increase in LLM accuracy over the GPT-based ReWOO on the HotpotQA dataset when employing GPT-3.5-turbo as the implementation of the QA tool to ensure fair comparisons.

4.3 Agent Formulation Comparison

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 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 Tab. 3, both LUMOS-I and LUMOS-O outperform CoT Training[3]. They also exceed the integrated formulation based on a single module to operate planning and grounding. It highlights the importance of disentangling subgoal planning and action grounding skills during the agent training.

[3]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.

Agents	Web Task
	Mind2Web
GPT/API-based Agents	
GPT-3.5-turbo	15.7
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$_{\text{Web}}$	27.6★
LUMOS-I$_{\text{Web}}$-13B	**31.3**★

(a) **Web** performance in 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$_{\text{Math}}$	50.5★	65.5
LUMOS-I$_{\text{Math}}$	47.1★	63.6
LUMOS-O$_{\text{Math}}$-13B	**55.4**★	**69.3**

(b) **Math** performance in accuracy.

Agents	Multimodal Tasks	
	A-OKVQA	ScienceQA (IMG)
GPT/API-based Agents		
GPT-3 + GT Caption	45.4	-
Open-source Agents		
ClipCap (VL)	44.0★	-
KRISP (VL)	51.9★	-
GPV-2 (VL)	60.3★	-
MiniGPT-4-13B (VL)	67.2	42.8
LLAVA-1.5-7B (VL)	-	57.6
LUMOS-O$_{\text{MM}}$	70.1★	56.9
LUMOS-I$_{\text{MM}}$	71.3★	**58.4**
LUMOS-I$_{\text{MM}}$-13B	**72.4**★	58.2

(c) **Multimodal** performance in accuracy.

Agents	Agent Model	QA Tool	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
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$_{\text{QA}}$	LLAMA-2-7B	GPT-3.5-turbo	60.6★	39.2 / 24.9
LUMOS-I$_{\text{QA}}$	LLAMA-2-7B	LLAMA-2-7B	58.3★	37.3 / 23.5
LUMOS-I$_{\text{QA}}$	LLAMA-2-7B	GPT-3.5-turbo	65.7★	45.9 / 29.4
LUMOS-I$_{\text{QA}}$	LLAMA-2-7B	GPT-4	72.4★	56.8 / 36.0
LUMOS-I$_{\text{QA}}$	LLAMA-2-13B	GPT-3.5-turbo	65.3★	50.2 / 31.4
LUMOS-I$_{\text{QA}}$	LLAMA-2-13B	GPT-4	**76.7**★	**57.4 / 36.3**

(d) **QA** performance. The evaluation metric for StrategyQA and HotpotQA is accuracy, and LLM accuracy / Exact Match (EM), respectively.

Agents	Unseen Tasks	
	WebShop	InterCode$_{\text{SQL}}$
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	6.7
Vicuna-v1.5-13B	41.7	4.8
OpenChat-v3.2-13B	46.9	-
Claude-instant	49.7	-
LUMOS-I$_{\text{Web}}$-13B	46.2	4.2
LUMOS-I$_{\text{Math}}$-13B	45.7	5.8
LUMOS-I$_{\text{QA}}$-13B	47.3	3.5
LUMOS-I$_{\text{MM}}$-13B	43.8	4.0
LUMOS-I$_{\text{All}}$-13B	**50.3**	**7.3**

(e) Unseen tasks, **WebShop** and **InterCode**$_{\text{SQL}}$. The metric is average reward and success rate, respectively.

Table 1: Overall performance of language agents on diverse complex interactive tasks. The tasks highlighted in red and blue are the held-in/held-out datasets for the trained task types. ★ indicates that the results are obtained after fine-tuning on the task's training set. We adopt multiple-choice setting for A-OKVQA. IMG denotes the subset with image inputs in ScienceQA test set. GPT-3 in Tab. 1c indicates `text-davinci-002`.

Training Data	QA	
	StrategyQA	HotpotQA
Downstream Perf. of Training Different Data w/ LLAMA		
ReWOO-open Data	≈57	≈37
LUMOS-I$_{\text{QA}}$ Data	**58.3**	**38.1**
Perf. Using High- & Low-Level Subgoal Annots. w/ LLAMA-2		
LUMOS-I$_{\text{QA}}$ w/ Low-Level Subgoals	63.3	44.3
LUMOS-I$_{\text{QA}}$ Data	**65.7**	**45.9**

Table 2: Comparison between the 7B-sized agents trained with different annotations.

4.4 LUMOS Generalizability Evaluation

Since LUMOS employs a unified format to represent complex interactive tasks, we envision that after trained with the combined annotations across the four training task types, i.e., unified training, when faced with an unseen task type, LUMOS may adapt to it more effectively.

To examine the generalization ability of LUMOS, we evaluate it on the unseen tasks, WebShop (Yao et al., 2022a) and InterCode$_{\text{SQL}}$. WebShop resembles a text game[4], with its shopping environment and action space considerably differing from those covered in the training annotations of LUMOS. InterCode$_{\text{SQL}}$ (Yang et al., 2023) is an interactive code generation task that requires the agent to generate SQL code based on the external databases and involves unseen SQL commands. To make LUMOS adapt to these unseen tasks, we supplement 2-/3-shot examples in the module inputs, enabling them to learn how to generate subgoals and ground to new sets of available actions (see App. J).

As outlined in Tab. 1e, LUMOS-I achieves a 3-7 higher average reward than domain-specific agents on WebShop. It also significantly outperforms larger agents such as WizardLM-30B and Vicuna-v1.3-33B (Chiang et al., 2023). We observe the similar trend on InterCode$_{\text{SQL}}$. It suggests that unified training enables agents to enhance the generalization on unseen tasks and novel actions. We provide more unified training results in App. F to show unified training also boosts the performance on most training task types.

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

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

(a) Web task.

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	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) QA tasks.

Agents	Multimodal Tasks	
	A-OKVQA	ScienceQA
CoT Training	68.3	57.3
Integrated Training	70.9	57.6
LUMOS-O$_{\text{MM}}$	70.1	56.9
LUMOS-I$_{\text{MM}}$	**71.3**	**58.4**

(d) Multimodal 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.

4.5 Further Analysis on Training 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 them and evaluating the agents' performance. We compare them with ReWOO-open data, constructed based on HotpotQA and TriviaQA (Joshi et al., 2017) using the Self-Instruct method. For fair comparison, we train the base model of ReWOO-open, LLAMA-7B, using LUMOS annotations. We also adopt the integrated training formulation and sample 2,000 training data to keep the same training settings as ReWOO-open. Given that ReWOO-open data exclusively relies on QA benchmarks, we primarily focus on QA task evaluation. Shown in Tab. 2, LUMOS data yields an improvement when compared to ReWOO-open data on StrategyQA and HotpotQA. Note that even if the ReWOO-open data are based on HotpotQA, it still underperforms LUMOS on HotpotQA.

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 low-level subgoals by modifying the annotation conversion prompt to fit the format where a subgoal is strictly linked to one action. Tab. 2 reveals a drop after replacing high-level subgoals with low-level ones on both QA datasets. This result hence reaffirms the appropriateness of our initial subgoal design.

5 Related Work

LLM Agents. 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 external environments. Subsequently, several methods (Shen et al., 2023; Lu et al., 2023; Xu et al., 2023; Lin et al., 2023; Liu et al., 2023c) aimed at improving agent performance and increasing their applicability in diverse scenarios. These agents mainly rely on closed-source LLMs, lacking of the consideration of affordability, reproducibility and transparency issues on complex interactive tasks.

Improving Small Models for Building Agents. Recent works have utilized larger models to generate training data for fine-tuning smaller models (Bosselut et al., 2019; West et al., 2022; Wang et al., 2023b; Hsieh et al., 2023; Brahman et al., 2023) to enable them to follow instructions and perform chain-of-thoughts reasoning. We also observe contemporaneous efforts ReWOO-open (Xu et al., 2023), FireAct (Chen et al., 2023), AgentLM (Zeng et al., 2023), and AutoAct (Qiao et al., 2024), focusing on training agents on smaller LLMs. Unlike FireAct and AutoAct, our work delves into a more in-depth analysis, aiming to discover a unified task representation that enables agents to generalize across unseen interactive tasks effectively. In contrast to ReWOO-open and AgentLM, we extend to examining proper training formulations and studying multiple strategies for creating large-scale, high-quality datasets for agent training. We demonstrate LUMOS superior performance in §4.

6 Conclusion

We introduce LUMOS, an open-source, generalizable language agent training framework. We propose two formulations, LUMOS-I and LUMOS-O, which promote collaboration among agent modules to solve complex tasks. For module training data, we use LLMs to transform reasoning steps in existing benchmarks into a unified format applicable within LUMOS framework. LUMOS outperforms a variety of open-source agents across the 9 datasets. It performs even better than GPT agents on QA and web tasks. LUMOS also exceeds potential agent training formulations and exhibits superior generalization on two unseen interactive tasks.

Limitations

Covered Training Task Types. Currently, LUMOS is trained using annotations for four specific types of complex interactive tasks, which may still limit its generalization capabilities for novel tasks. To address this, we aim to enrich the training data for LUMOS by incorporating a wider variety of task types. As outlined in §3 and App. C, a substantial array of benchmarks already exists, providing ground-truth reasoning steps that could serve as a foundation for expanding LUMOS's annotations. By broadening the scope of annotations, we not only enhance the language agents but also offer a valuable resource for practitioners looking to develop their own models.

Backtracking and Replanning Ability. In situations where language agents encounter invalid execution outcomes or navigate erroneous solution pathways, it is crucial for them to possess the capacity for self-diagnosis and replanning their reasoning processing. The current LUMOS lacks these sophisticated self-corrective features. Future versions should be designed with advanced mechanisms that enable the agents to recognize and rectify their planning errors.

Open-Source Tool Replacement. For part of our QA experiments, we employ GPT models to address decomposed sub-questions. It is designed for fair comparison with the agents that also use GPT models as QA tools, as elaborated in §4.2. Our future strategy involves transitioning to fully open-source QA frameworks that leverage models such as LLAMA-2-70B, aiming to establish a completely open-source framework.

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*.

Antoine Bosselut, Hannah Rashkin, Maarten Sap, Chaitanya Malaviya, Asli Celikyilmaz, and Yejin Choi. 2019. COMET: Commonsense transformers for automatic knowledge graph construction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4762–4779, Florence, Italy. Association for Computational Linguistics.

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

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*.

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. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

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*.

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

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond iid: three levels of generalization for question answering on knowledge bases. In *Proceedings of the Web Conference 2021*, pages 3477–3488.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8003–8017, Toronto, Canada. Association for Computational Linguistics.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Ariel N. Lee, Cole J. Hunter, Nataniel Ruiz, Bleys Goodson, Wing Lian, Guan Wang, Eugene Pentland, Austin Cook, Chanvichet Vong, and "Teknium". 2023. Openorcaplatypus: Llama2-13b model instruct-tuned on filtered openorcav1 gpt-4 dataset and merged with divergent stem and logic dataset model. https://huggingface.co/Open-Orca/OpenOrca-Platypus2-13B.

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

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

Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D Ernst. 2018. Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system. *arXiv preprint arXiv:1802.08979*.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2023a. Improved baselines with visual instruction tuning. *arXiv preprint arXiv:2310.03744*.

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

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

Pan Lu, Swaroop Mishra, Tony Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. 2022. Learn to explain: Multimodal reasoning via thought chains for science question answering. In *The 36th Conference on Neural Information Processing Systems (NeurIPS)*.

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

Xing Han Lu, Zdenvek Kasner, and Siva Reddy. 2024. Weblinx: Real-world website navigation with multi-turn dialogue.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A diverse corpus for evaluating and developing English math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 975–984, Online. Association for Computational Linguistics.

Swaroop Mishra, Matthew Finlayson, Pan Lu, Leonard Tang, Sean Welleck, Chitta Baral, Tanmay Rajpurohit, Oyvind Tafjord, Ashish Sabharwal, Peter Clark, and Ashwin Kalyan. 2022. LILA: A Unified Benchmark for Mathematical Reasoning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5807–5832, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

OpenAI. 2022. ChatGPT.

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

OpenAI. 2023b. Gpt-4v(ision) system card.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Eleanor Jiang, Chengfei Lv, and Huajun Chen. 2024. Autoact: Automatic agent learning from scratch via self-planning. *arXiv preprint arXiv:2401.05268*.

Dustin Schwenk, Apoorv Khandelwal, Christopher Clark, Kenneth Marino, and Roozbeh Mottaghi. 2022. A-okvqa: A benchmark for visual question answering using world knowledge. In *European Conference on Computer Vision*, pages 146–162. Springer.

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

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

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. *ArXiv preprint*, abs/2302.13971.

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

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

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

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023b. 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)*, pages 13484–13508, Toronto, Canada. Association for Computational Linguistics.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. 2017. Crowdsourcing Multiple Choice Science Questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 94–106, Copenhagen, Denmark. Association for Computational Linguistics.

Peter West, Chandra Bhagavatula, Jack Hessel, Jena Hwang, Liwei Jiang, Ronan Le Bras, Ximing Lu, Sean Welleck, and Yejin Choi. 2022. 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*, pages 4602–4625, Seattle, United States. Association for Computational Linguistics.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*.

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

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. 2023. Intercode: Standardizing and benchmarking interactive coding with execution feedback.

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, Brussels, Belgium. Association for Computational Linguistics.

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

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

Rowan Zellers, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. From recognition to cognition: Visual commonsense reasoning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6720–6731.

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*.

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

Appendix

A Illustration of Annotation Organization

As discussed in §3.2, we organize the ground-truth subgoals and actions converted by GPT-4 into the conversational format that boosts the interaction between modules. We show the final conversation format in Fig. 3.

B 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. Tab. 4 shows the benchmarks leveraged for annotation conversion, along with the task type information.

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

We calculate the average turn numbers in each task's converted training annotations. The average numbers are 4.75, 3.75, 8.25 and 3.39 for Math, QA, Web and Multimodal tasks, respectively.

C Available Resources for LUMOS Training Data Extension

As discussed in §3, we seek the existing datasets with ground-truth intermediate reasoning steps to synthesize LUMOS training annotations from four complex interactive task categories, math, QA, web and multimodal tasks.

The methodology for converting annotations as described is not limited to the four task types previously mentioned. Any training datasets that include gold reasoning rationales are suitable for the construction of annotations using the LUMOS method. We present an exhaustive list of datasets that span a variety of task types in Tab. 5. For example, the AlfWorld dataset requires actions that have not been encountered in existing LUMOS annotations, such as `open`, `take`, `move`, and others; the TravelPlanner dataset requires actions like `CitySearch`, `FlightSearch`, which are equally unseen in the existing training set as well. This approach could significantly enhance the scalability of LUMOS annotations, thereby augmenting the method's capability to adapt to novel environments and acquire proficiency in executing new actions.

D Details of Training Modules

We describe additional details about our training experiments. In all of our experiments, we implement training over two epochs with a learning rate of 2×10^{-5} and a batch size 128. 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 or 4 NVIDIA 48GB A6000 GPUs.

E Efficiency Study on LUMOS and Efficiency-Performance Trade-Off

We compute the inference time for LUMOS-O and LUMOS-I across 100 instances on GSM8K and HotpotQA, respectively. The experiments are run with 2 NVIDIA A6000 48GB GPUs with inference batch size 16. As depicted in Tab. 6, we find that LUMOS-O is much more efficient than LUMOS-I on both datasets.

LUMOS-O completes its inference in a single round, whereas LUMOS-I necessitates multiple rounds of inference until it autonomously concludes its planning. The iterative planning and grounding in LUMOS-I contribute to a higher time cost for solving individual instances. However, this property leads to better LUMOS-I's capacity to generate appropriate subgoals based on the current external environment compared to LUMOS-O.

For example, when tackling a complex question "`What government position was held by the woman who portrayed Corliss Archer in the film Kiss and Tell?`", LUMOS-I can first identify the woman who portrayed Corliss Archer in Kiss and Tell, Shirley Temple, then ask the government position she held. However, though LUMOS-O can first ask who the woman is as well, without the interaction with external knowledge in Wikipedia, it will then generate a subgoal which inquires the government position held by Madonna, a random entity irrelevant with the original question. Hence, LUMOS-O is a more efficient solution, but not as effective as LUMOS-I due to the lack of the adaptability to external environments.

We also notice that in Tab. 1b, LUMOS-O achieves superior performance on math tasks. Consider a mathematical problem such as, "`James decides to run 3 sprints 3 times a week. He runs 60 meters each sprint. How many total meters does he run a week?`" Even if the agent calculated how many times James sprints

Task Types	Datasets	# Source Data	# Total	# Final Converted for Planning	# Final Converted for Grounding
Math	PRM800K (Lightman et al., 2023) GSM8K (Cobbe et al., 2021) ASDiv (Miao et al., 2020)	10000 7473 2305	19778	19386	19471
QA	Musique (Trivedi et al., 2022) StrategyQA (Geva et al., 2021)	17632 1777	19409	19048	19080
Web	Mind2Web (Deng et al., 2023)	1009	1009	1007	1007
Multimodal	A-OKVQA (Schwenk et al., 2022)	17056	17056	16038	16038

Table 4: Statistics of data sources for conversion and the number of final successfully converted annotations for each task type.

Datasets	Task Types	# Total
WebLINX (Lu et al., 2024)	Web Browsing	2337
TravelPlanner (Xie et al., 2024)	Travel Planning	225
SciQ (Welbl et al., 2017)	Question Answering	11679
GrailQA (Gu et al., 2021)	Knowledge Graph Reasoning	44337
NL2Bash (Lin et al., 2018)	Interactive Coding	8090
AlfWorld (Shridhar et al., 2020)	Embodied Task	3553
VCR (Zellers et al., 2019)	Multimodal Reasoning	212923
LILA (Mishra et al., 2022)	Math	93670

Table 5: Statistics of potential data sources for LUMOS annotation extension.

Agents	GSM8K	HotpotQA
LUMOS-O	102s	556s
LUMOS-I	851s	1007s

Table 6: The time cost of LUMOS-O and LUMOS-I when performing inference 100 instances from GSM8K and HotpotQA datasets.

a week, which is $3 \times 3 = 9$, the mere number 9 does not affect the next subgoal generation. This is because no matter the result is 9 or 10000, the next high-level subgoal to calculate the total meters remains the same for solving the question. Therefore, the high environment adaptability of LUMOS-I may not be a good supplement on math tasks.

F More Unified Training Results

Agents	Web	QA	Multimodal	Math
LUMOS-I_X-13B	31.3	65.3/50.2/31.4	72.4/58.2	**51.9/66.3**
LUMOS-I_{All}-13B	**31.9**	**66.7/51.0/31.6**	**72.8**/58.2	50.5/65.8

Table 7: Unified training performance on trained task types. We aggregate the performance on held-in/held-out datasets for each trained task type with the symbol '/'. As discussed in the footnote 2 of §4.4, LUMOS-O is not applicable to web tasks. Thus, we only conduct unified training for LUMOS-I as it can be universally applied to any task types.

We evaluate the performance of LUMOS-I-13B after the unified training that leverages combined annotations from four distinct training task types. We observe that the unified training enhances performance across a majority of tasks, including web, QA, and multimodal tasks. The decline in performance for mathematical tasks is marginal, which is only 0.7% and 1.4%. The unified task representation may enable LUMOS agents to uncover intrinsic links among these complex interactive tasks, thereby yielding additional advantages for certain training tasks.

G Execution Tools Associated with Action Interfaces

For each available action defined in the action interfaces, there is at least one corresponding backend execution tool to facilitate the implementation of the concrete actions.

Displayed in Tab. 8a, for QA tasks, we rely on Wikipedia and Google search APIs to locate relevant entity knowledge. Besides, we leverage a semantic matching model `dpr-reader-multiset-base`[5] used in Dense Passage Retrieval (DPR) (Karpukhin et al., 2020) to capture paragraphs according to the query. Following ReWOO (Xu et al., 2023), we also include GPT-series model as a simple QA tool to answer the query based on our retrieved knowledge or previously interaction history.

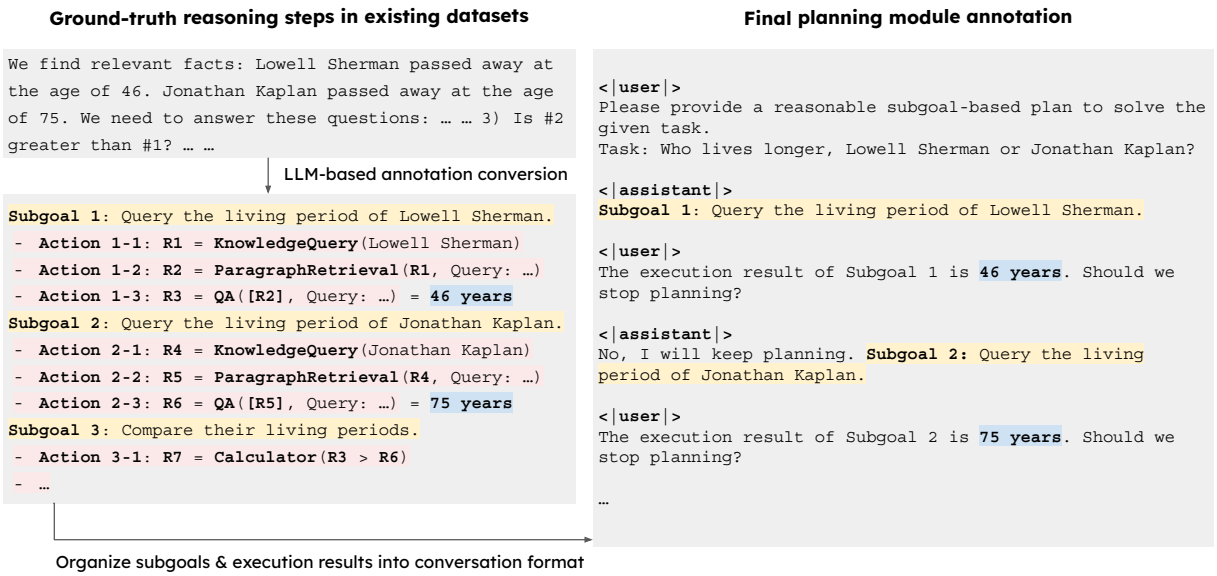
Demonstrated in Tab. 8b, for web tasks, the actions are real mouse and keyboard operations including typing, clicking and selecting HTML tags. To locate the relevant HTML tags to be operated, following AgentBench evaluation, we use a DeBERTa model[6] that ranks and retrieves the tags according to the current action.

As shown in Tab. 8c, for math tasks, the main execution tool is WolframAlpha API [7] as it can per-

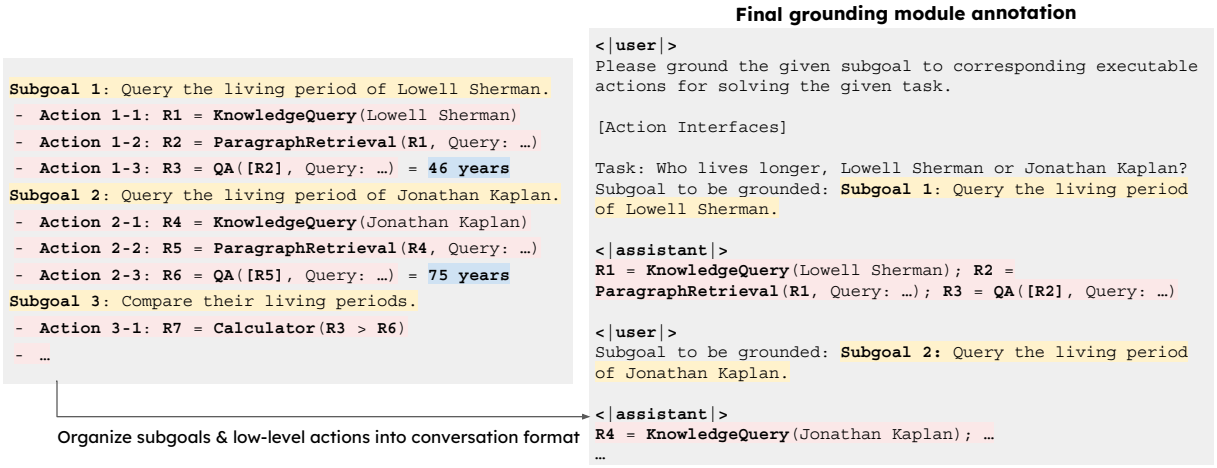
[5] https://huggingface.co/facebook/dpr-reader-multiset-base.

[6] https://huggingface.co/osunlp/MindAct_CandidateGeneration_deberta-v3-base.

[7] https://www.wolframalpha.com/.



(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.

form a large collection of mathematical functions such as calculating formulas and solving equations. For complex math operations such as sorting, we leverage OpenAI Codex (Chen et al., 2021) to generate a short code snippet for the execution.

For multimodal tasks, as illustrated in Tab. 8d, both Visual Question Answering (VQA) and Question Answering (QA) tools are considered. The employed VQA model is LLAVA-1.5-7B (Liu et al., 2023a), while the utilized QA model is LLAMA-2-13B-chat (Touvron et al., 2023b).

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

For the unseen task $\text{InterCode}_{\text{SQL}}$, the action interfaces include all the possible commands and functions provided in SQL programming language.

H 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. (2023) to also

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

Task Type	Action Types	Function Descriptions	Tools
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.

Task Type	Action Types	Function Descriptions	Implementation
Multimodal	`QA(Context, Query) -> Answer`	Answer the query based on the given context	`LLAMA-2-13B-chat`
	`VQA(Image_Context, Query) -> Answer`	Answer the query based on the given image context	`LLAVA-1.5-7B`

(d) Actions used in multimodal tasks.

Table 8: Action interfaces and execution module implementations for complex interactive tasks.

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 the product titles and selected attributes.

Evaluation Data. Following Xu et al. (2023), we only evaluate 300 and 1000 randomly selected examples from StrategyQA and HotpotQA evaluation set, respectively. The results reported in Tab. 1d 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. We leverage the entire evaluation set of the other testing benchmarks for assessment.

I 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 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, their actions and execution results with yellow, red and blue, respectively.

I.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
```

I.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
```

I.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)
```

I.4 In-Context Example For Obtaining Multimodal Task Annotations

Please convert natural language plans into a series of subgoals, their corresponding actions that lead to the successful implementation with respect to the given instructions. When generating the actions, please also attach the action's results contained in the natural language plans. Please use 'R[number]' to represent the execution results for each action. Please also use functions to represent the corresponding actions. For the actions, they must be one of the available actions, 'QA', 'VQA'.

Example 1:

Task: If the cameraman were driving what do they have to do from this position? There're some choices: A. turn left, B. drive straight, C. reverse course, D. turn right.

Natural language plan:
The would have to turn right because the lane has right turn arrows painted on it. The arrow on the street indicates that this lane can only go in one direction at the intersection. The sign on the road says to turn right. Overall, the final answer is 'turn right'.

Subgoal-based plan:
Subgoal 1: Describe the shape of the sign on the road lane the cameraman is in from the image.
Action 1-1: R1 = VQA([IMG], Question: What's the sign on the road lane?) = There's a right turn arrow on the road.

Subgoal 2: Answer which lane the cameraman is in.
Action 2-1: R2 = QA([R1], Question: Which lane is the cameraman in?) = The cameraman is in right turn lane.

Subgoal 3: Answer what the cameraman has to do from his position.
Action 3-1: R3 = QA([R1, R2], Question: What does the cameraman have to do from his position?) = turn right.

Subgoal 4: Select the choice from the answer choices.
Action 4-1: R4 = VQA([IMG], Question: If the cameraman were driving what do they have to do from this position? We can also see that There's a right turn arrow on the road, The cameraman is in right turn lane, turn right. There're some choices: A. turn left, B. drive straight, C. reverse course, D. turn right. You must choose one from 'A', 'B', 'C' or 'D'. Output only one letter!) = D

J 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 App. G, 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 of the two in-context WebShop-related examples used for the inference of planning and grounding modules, respectively.

J.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.
```

J.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)
```

K In-Context Example for Adapting Modules to Unseen Task — InterCode$_{\text{SQL}}$

Similar to §J, as InterCode$_{\text{SQL}}$ is another unseen task, we showcase one of the three in-context InterCode$_{\text{SQL}}$-related examples used for the inference of planning and grounding modules, respectively.

K.1 In-Context InterCode$_{\text{SQL}}$-Related Example for Planning Module

```
<|user|>
Please provide a reasonable subgoal-based plan to solve the given task.
Task: What are the names of poker players, ordered ascending by the number of final tables they have
made?

<|assistant|>
Subgoal 1: Find all the related tables.

<|user|>
The execution result for Subgoal 1 is Results: Table players has Name (type: TEXT), People_ID (type:
INT); Table poker_player has Final_Table_Made (type: INT), People_ID (type: INT).; Subgoal 2: Write
the selection objects: the names of the people who are also poker players.

<|assistant|>
Subgoal 2: Write the selection objects: the names of the people who are also poker players.

<|user|>
The execution result for Subgoal 2 is T1.Name FROM people AS T1 JOIN poker_player AS T2 ON T1.People_ID
= T2.People_ID.

<|assistant|>
Subgoal 3: Write the selection condition: sort all the poker players by the number of final tables they
have made in ascending order.

<|user|>
The execution result for Subgoal 3 is ORDER BY T2.Final_Table_Made.

<|assistant|>
Subgoal 4: Finish the SQL query to the names of poker players, ordered ascending by the number of final
tables they have made.

<|user|>
The execution result for Subgoal 4 is SELECT T1.Name FROM people AS T1 JOIN poker_player AS T2 ON
T1.People_ID = T2.People_ID ORDER BY T2.Final_Table_Made.
```

K.2 In-Context InterCode$_{\text{SQL}}$-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.

Task: What are the names of poker players, ordered ascending by the number of final tables they have made?
Subgoal to be grounded: Subgoal 1: Find all the related tables.

<|assistant|>
R1 = SHOW TABLES

<|user|>
Subgoal to be grounded: Subgoal 2: Write the selection objects: the names of the people who are also poker players.

<|assistant|>
R2 = T1.Name FROM people AS T1 JOIN poker_player AS T2 ON T1.People_ID = T2.People_ID

<|user|>
Subgoal to be grounded: Subgoal 3: Write the selection condition: sort all the poker players by the number of final tables they have made in ascending order.

<|assistant|>
R3 = ORDER BY T2.Final_Table_Made

<|user|>
Subgoal to be grounded: Subgoal 4: Finish the SQL query to the names of poker players, ordered ascending by the number of final tables they have made.

<|assistant|>
R4 = SELECT T1.Name FROM people AS T1 JOIN poker_player AS T2 ON T1.People_ID = T2.People_ID ORDER BY T2.Final_Table_Made
```

L Multimodal Task Case Study

We provide two cases in the multimodal datasets, A-OKVQA and ScienceQA, to show the effectiveness of LUMOS. For the A-OKVQA case, LUMOS first identifies the device brand with the aid of VQA tool. After LUMOS planning module knows that the device is a Nintendo Wii controller, the module would generate the next subgoal "`Answer the country of origin for the Nintendo Wii controller`". Finally, LUMOS selects the closest option from the given answer choices. For the ScienceQA case, LUMOS identifies the highlighted area on the map, which includes Middle East area. Then the planning and grounding modules would leverage QA tool to answer which continent Middle East is located at, and then pick up the correct answer "`Asia`".

Planning module output
Grounding module output
Execution module output

Question: The company producing the device in her hand is from what country? There're some choices: A. … … C. Japan, D. … The image shows the following contents: The image features a young woman sitting on a couch, … … playing a video game, possibly on a Nintendo Wii console. The couch she is sitting on is … …

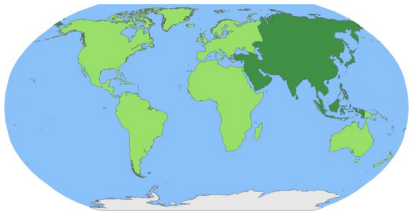
Lumos-I_{MM} Planning and Grounding Results:

Subgoal 1: Identify the brand of the device in her hand.
Action: R1 = VQA([IMG], Question: What brand is the device in her hand?) = The device in her hand is a Nintendo Wii controller.

Subgoal 2: Answer the country of origin for the Nintendo Wii controller.
Action: R2 = QA([R1], Question: What is the country of origin for the Nintendo Wii controller?) = The country of origin for the Nintendo Wii controller is Japan.

Subgoal 3: Select the choice from the four answer choices.
Action: R3 = VQA([IMG], Question: The company producing the device in her hand is from what country? The image features … We can also see that The device in her hand is a Nintendo Wii controller, The country of origin for the Nintendo Wii controller is Japan. There're some choices: A. … … C. Japan, D. … You must choose one from 'A', 'B', 'C' or 'D'. Output only one letter!) = C.

Figure 4: LUMOS-I_{MM} case study on A-OKVQA.



Planning module output
Grounding module output
Execution module output

Question: Which continent is highlighted? There're some choices: … … D. Asia. The image features a large, colorful map of the world, with various countries and continents depicted. The map is filled with green and blue colors, giving it a vibrant and lively appearance. … …

Lumos-I_{MM} Planning and Grounding Results:

Subgoal 1: Identify the highlighted area on the map.
Action: R1 = VQA([IMG], Question: Which area is highlighted on the map?) = The area highlighted on the map includes the Middle East, … .

Subgoal 2: Answer which continent the highlighted area is in.
Action: R2 = QA([R1], Question: Which continent is the Middle East in?) = The Middle East is located in the continent of Asia.

Subgoal 3: Select the choice from the answer choices.
Action: R3 = VQA([IMG], Question: Which continent is highlighted? The image features … We can also see that The area highlighted on the map includes the Middle East, …, The Middle East is located in the continent of Asia. There're some choices: … … D. Asia. You must choose one from 'A', 'B', 'C' or 'D'. Output only one letter!) = D.

Figure 5: LUMOS-I_{MM} case study on ScienceQA.