Exposing Limitations of Language Model Agents in Sequential-Task Compositions on the Web

Anonymous authors
Paper under double-blind review

Abstract

Language model agents (LMA) recently emerged as a promising paradigm on muti-step decision making tasks, often outperforming humans and other reinforcement learning agents. Despite the promise, their performance on real-world applications that often involve combinations of tasks is still underexplored. In this work, we introduce a new benchmark, called *CompWoB* – 50 new compositional web automation tasks reflecting more realistic assumptions. We show that while existing prompted LMAs (`gpt-3.5-turbo` or `gpt-4`) achieve 94.0% average success rate on base tasks, their performance degrades to 24.9% success rate on compositional tasks. On the other hand, transferred LMAs (finetuned only on base tasks) show less generalization gap, dropping from 85.4% to 54.8%. By balancing data distribution across tasks, we train a new model, HTML-T5++, that surpasses human-level performance (95.2%) on MiniWoB, and achieves the best zero-shot performance on CompWoB (61.5%). While these highlight the promise of small-scale finetuned and transferred models for task compositionality, their performance further degrades under different instruction compositions changing combinational order. In contrast to the recent remarkable success of LMA, our benchmark and detailed analysis emphasize the necessity of building LMAs that are robust and generalizable to task compositionality for real-world deployment.

1 Introduction

Based on the exceptional capability of large language models (LLMs) (OpenAI, 2023; Anil et al., 2023; Touvron et al., 2023) in commonsense understanding (Brown et al., 2020; Chowdhery et al., 2022), multi-step reasoning (Wei et al., 2022; Kojima et al., 2022), program synthesis (Chen et al., 2021) and self-improvement (Shinn et al., 2023; Madaan et al., 2023; To et al., 2023), language model agents (LMAs) have recently emerged to tackle various decision making problems, such as robotics (Huang et al., 2022a; Ahn et al., 2022), information retrieval (Nakano et al., 2021; Yao et al., 2022b), and external tool use (Wu et al., 2023; Shen et al., 2023). Especially, *web automation* (Shi et al., 2017) has attracted attention a lot as an promising application of LMAs, because LMAs with prompting (Kim et al., 2023; Sun et al., 2023; Zheng et al., 2023) outperform humans and other learning-based agents, such as imitation (Furuta et al., 2023) or reinforcement learning (Humphreys et al., 2022).

Despite their human-level proficiency in MiniWoB (Shi et al., 2017), a standard web automation benchmark, it is still unclear whether LMAs could deal with challenges in the real world; such as complex observation (Gur et al., 2023), domain generalization (Deng et al., 2023), and ambiguity of instructions (Zhou et al., 2023b). These challenges are exacerbated due to the open-ended nature of real-world tasks, making it infeasible for the agent system to prepare exemplars and prompts for any unseen task in advance. Moreover, it is reasonable and practically important to assume the **zero-shot evaluation of the off-the-shelf agents at deployment**. This is an orthogonal assumption to the recent progress in LMAs learnable from execution/self-feedback (Shinn et al., 2023; Ma et al., 2023) during the deployment. Considering the agents are deployed as a service, (1) it is infeasible to cover all the possible prompts/demonstrations for user requests in advance and (2) iterative trial-and-error to adjust system prompts hurts user experiences.

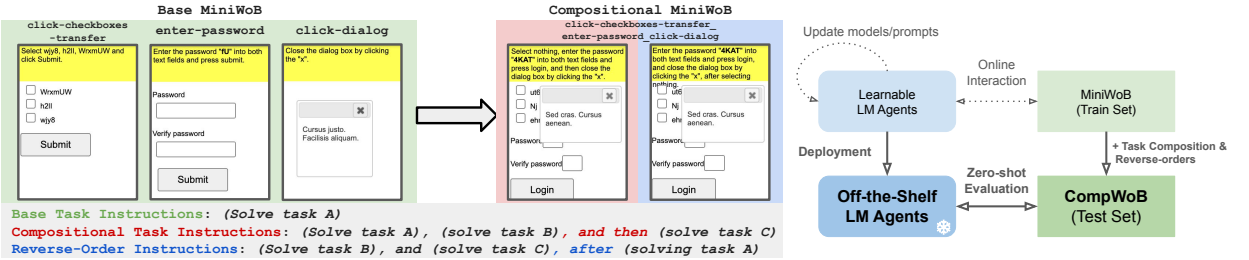

Figure 1: (**Left**) We design *CompWoB*, as a novel decision making benchmark for LMAs, by leveraging the high composability of simulated web environments. We first select base tasks from the original MiniWoB (Shi et al., 2017) based on the brute-force task complexity averaged among existing LMAs. While considering the feasibility and reality, we systematically combine them into a sequential single task (e.g. `click-checkboxes-transfer` + `enter-password` + `click-dialog`→ `click-checkboxes-transfer_enter-password_click-dialog`). The instructions of base tasks are stitched with "**and then**". LMAs are asked to satisfy the given instructions sequentially (e.g. satisfying the success criteria of task $A \rightarrow B \rightarrow C$). We also implement **reverse-order instruction** settings, where the instructions are provided upside down (e.g. solve task B, and solve task C, after solving task A). These complex yet controllable strategies make the analysis of LMA's behaviors easy, while maintaining complex and ambiguous aspects of real-world web environments. (**Right**) In this paper, we assume the zero-shot evaluation of the off-the-shelf agents at deployment, which is practically important because (1) it is infeasible to cover all the possible prompts/demonstrations for user requests in advance and (2) iterative trial-and-error to adjust system prompts hurts user experiences. CompWoB works as a hold-out test environment accompanying with MiniWoB as a train environment.

In this work, we extensively study the transferability of LMAs to more realistic sequential task compositions. We first design a new controlled test bed, called CompWoB, with 50 compositional tasks by combining a set of base tasks based on their difficulty (Figure 1, right). CompWoB works as a *hold-out* test environment accompanying with MiniWoB as a train environment (Figure 1, left). Each compositional task is implemented from 2 to 8 base tasks in a single-page or multi-page environment with instructions linked together using simple connectors such as "and then". Only providing the knowledge about base tasks, we investigate the generalization performance of existing SoTA prompted LMAs (Kim et al., 2023; Sun et al., 2023; Zheng et al., 2023) with planning, self-improvement, program synthesis, and structured prompts that are supported by `gpt-3.5-turbo` and `gpt-4`. Our findings indicate that their performance drops significantly, from 94.0% success on base tasks to 24.9% success on compositional tasks. In contrast, small-scale LMAs finetuned only on base tasks and zero-shot-transferred to compositional settings (i.e. transferred LMAs), deal with unknown task compositionality better, achieving 54.8% success rate on average. By rebalancing the data distribution, we train a new model, HTML-T5++, that achieves human-level performance on MiniWoB and performs the best among all the LMAs on compositional tasks. In contrast, small-scale LMAs finetuned only on base tasks and zero-shot-transferred to compositional settings (i.e. transferred LMAs), deal with unknown task compositionality better, achieving 54.8% success rate on average. By rebalancing the data distribution, we also train a new model, HTML-T5++, that achieves human-level performance on MiniWoB and performs the best among all the LMAs on compositional tasks. We further point out that LMAs struggle to handle complex instruction compositions permuting the order of sub-instructions, where prompted agents are more robust to the difference in the order of compositions compared to transferred agents (6.9% vs 23.8% drop in performance). Finally, we illustrate that instruction length and observation complexity are useful indicators of compositional task performance.

In contrast to the recent notable success of LMAs, our benchmark and detailed analysis highlight building robust and generalizable LMAs to be safely deployed in the real world. In summary, our key contributions are:

- We empirically show that (1) **prompted LMAs even with `gpt-4` suffer from generalizing to compositional web automation tasks much more than transferred LMAs**, and (2) **LMAs are highly sensitive to the order of instructions**.
- We develop *CompWoB* [1] , simulated web environments for LMAs to measure the generalization to the realistic task compositionality and complex instructions.

[1] We provide the code in Supplementary Material, and will release it online in de-anonymized version.

- We propose a new data mixture strategy for finetuning LMAs. HTML-T5++, trained on this strategy, achieves human-level performance on MiniWoB (95.2%) and the best zero-shot transfer to CompWoB (61.5%).
- Our experiments reveal that while more capable models, such as `gpt-4`, lead to better performance, they still struggle with task compositionality or complex instructions.

2 Related Works

Web Automation Although prior works have worked on imitation learning and reinforcement learning (Liu et al., 2018; Gur et al., 2019; Jia et al., 2019; Humphreys et al., 2022), web automation has become a popular domain as an application of LMAs (Gur et al., 2022; Kim et al., 2023). In earlier work, finetuned LMAs, based on at most 3-billion parameters, amortize the training costs with the strong prior knowledge on web environments (Gur et al., 2022; Furuta et al., 2023; Shaw et al., 2023), but they often result in sub-optimal performances due to the insufficient data coverage. Recently, by leveraging capable private LLMs (Brown et al., 2020; Ouyang et al., 2022) with self-refinement (Kim et al., 2023), program synthesis (Sun et al., 2023), structured instruction-state translation (Zheng et al., 2023), or hierarchical prompts (Sridhar et al., 2023; Ma et al., 2023), prompted LMAs with few-shot exemplars have outperformed finetuned LMAs and shown superior performance to humans and RL-finetuned agents. In contrast, our work discusses the transferability and robustness of those LMAs in zero-shot web automation with a set of compositional tasks, and resolves the sub-optimality of finetuned LMAs via data-rebalancing.

In addition to MiniWoB (Shi et al., 2017), a representative web simulator, several works have conducted real-world evaluation (Gur et al., 2023) and proposed novel benchmarks reflecting real-world assumptions, such as a simulated e-commerce site (Yao et al., 2022a), sand-boxed real-world websites (Zhou et al., 2023b; Drouin et al., 2024), an adaptive sim-to-real bridge with unsupervised auto-curricula (Gur et al., 2021), and large-scale web interaction dataset curated by human annotators (Deng et al., 2023). However, real-world web automation may make the analysis challenging because it often faces miscellaneous obstacles, such as complex HTML observations, domain gaps between websites, and ambiguous instructions. In this work, we design CompWoB under realistic assumptions while controlling task difficulty and ambiguity of instructions, and investigate what may prevent the generalization capability of LMAs in compositional tasks.

Language Model Agents Beyond the common NLP tasks, LLMs could act as autonomous agents (Wang et al., 2023b; Qin et al., 2023a) to solve the given instruction-following tasks, by considering the context in the prompt as states (Ahn et al., 2022; Yao et al., 2022b; Hsieh et al., 2023) and sequentially planning and manipulating external "tools" or "actuators", such as calculators (Parisi et al., 2022), retrievers (Schick et al., 2023; Hao et al., 2023), APIs (Qin et al., 2023b; Tang et al., 2023a), programs (Gao et al., 2023; Wang et al., 2023a; Liang et al., 2023; Song et al., 2023; Cai et al., 2023), robotic commands (Huang et al., 2022a;b; Tang et al., 2023b), computer game (Nottingham et al., 2023; Wang et al., 2023c), or other foundation models (Lu et al., 2023; Hsieh et al., 2023; Wu et al., 2023; Shen et al., 2023; Yang et al., 2023). Those prior works have worked on proposing novel benchmarks (Li et al., 2023; Xu et al., 2023; Patil et al., 2023) and comparing backbone LLMs (e.g. open-sourced v.s. private) (Ruan et al., 2023; Liu et al., 2023b;a). Despite their success, it is still unclear how such LMAs designed for specific tasks can generalize out-of-domain problems, which should be an important perspective since we may not prepare prompts and exemplars for all the possible problems in the real world. In this work, we measure the transferability and robustness to unseen compositionality and instructions in a realistic web automation scenario.

Large Language Models for Compositional Tasks Several works have investigated compositional natural language problems with LLMs, such as semantic parsing (Furrer et al., 2021; Shaw et al., 2021; Zhou et al., 2023a), logic grid puzzles (Dziri et al., 2023), mathematical reasoning (Chen et al., 2023), programming (Zelikman et al., 2023), and planning (Brahman et al., 2023), which shows that dynamical selection of exemplars for decomposed sub-problems (Drozdov et al., 2023) or model scaling (Qiu et al., 2022) could help generalization. While those are focused on static tasks, our paper deals with task compositionaliy in interactive decision making, especially, in web automation where the task may have more explicitly decomposable structures (Gur et al., 2021) than natural language tasks.

3 Preliminaries

Web automation could be described as a deterministic sequential decision making problem, which consists of a state space $\mathcal{S}$, action space $\mathcal{A}$, deterministic transition function $T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, a set of instructions $\mathcal{G}$, a set of contextual information (i.e. prompts for LLM) $\mathcal{C}$, and episodic reward function (i.e. success criteria) $r : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \rightarrow \{0, 1\}$. At each time step t, the language model agent π infers the action conditioned on the prompt, instruction, current state, and previous actions $\pi : \mathcal{S} \times \mathcal{A}^{\times t} \times \mathcal{C} \times \mathcal{G} \rightarrow \mathcal{A}$, and moves to the next state: $s_{t+1} = T(s_t, a_t)$. When the agent reaches the terminal state (e.g. `Login` button is clicked) or the max time step is exceeded, the episode is marked as a success if the instruction g is satisfied (i.e. $r(s_t, g, a_t) = 1$). The state $s_t \in \mathcal{S}$ is a raw HTML, and we assume the programmatic action space: `function(selector, text)`. `function` is either *click*, *move* or *type*, `selector` is an integer index or XPath that can uniquely specify the element, and `text` is a text input for *type* function.

Task Compositionality Web automation tasks can be decomposed into a set of primitive base tasks. For instance, (1) clicking several checkboxes, (2) fulfilling the password form, and (3) closing the dialog window. Such a combination could be open-ended and have some dependencies. In this work, we assume that the task $\psi \in \Psi$ is characterized by a corresponding subtree of HTML ($\mathcal{S}_\psi \subset \mathcal{S}$) and instructions ($\mathcal{G}_\psi \subset \mathcal{G}$), and can be dependently combined each other as long as the compositional task is executable.

4 Language Model Agents for Web Automation

To be self-contained, we here review the representative LMAs, which are selected based on their superior performance and novelty in using LLMs for web automation; RCI (Section 4.1), AdaPlanner (Section 4.2), Synapse (Section 4.3). These are characterized with different base LLMs, prompting and pipeline. To clarify their methodological difference, we provide the pseudo code in Appendix B. Moreover, we resolve the sub-optimal performance of transferred LMAs (Section 4.4) with data-rebalanced finetuning (Section 4.5).

4.1 RCI

The agent with Recursive Criticism and Improvement (RCI) prompting (Kim et al., 2023) first generates an open-loop plan to follow a given instruction using few-shot demonstrations. Next, it uses a prompt-based critic to identify the errors in the plan and improves it by reflecting on self-criticized outputs, which is referred as an explicit RCI (ERCI) loop. After ERCI, the agent follows the self-improved plan step-by-step. Before executing the action at each step, the agent grounds the action to the current state (i.e. HTML, open-loop plan, and previous actions) and refines its formatting to be parsable, which increases the feasibility and reduces hallucinations. These final steps are referred as an implicit RCI (IRCI) loop without the self-criticism. All of those play complementary roles to achieve human-level proficiency. While 1 ERCI and 3 IRCI loops are recommended, we observe that the optimal number of self-improvement iterations may differ across the tasks (see Appendix C.1).

4.2 AdaPlanner

In contrast to other LMAs, AdaPlanner (Sun et al., 2023) leverages the capability of program synthesis in LLMs to mitigate the hallucination in a plan. Conditioning on the instruction, the description of permissible actions in the web environments, and few-shot demonstrations, the agent first generates an open-loop plan in a Python function, where each snippet corresponds to the action. Once the agent receives environmental feedback at each step, such as assertion errors in the code, other functional errors, or "ask" action to LLMs, it adaptively re-generates the plan for the remaining steps in a closed-loop manner. LLMs more capable of code have performed better, such as `text-davinci-003` than `gpt-3.5-turbo`.

4.3 Synapse

Synapse (Zheng et al., 2023) argues that LMAs perform better if well-designed structured prompts are provided, even without self-improvement or program synthesis. The structured prompting is formed by two

pre-processing strategies: state filtering and task reformulation. State filtering gradually transforms raw HTML into simple formatted text, such as a Pythonic list or dictionary, in a multi-step manner, which may improve the state understanding of LMAs. Task reformulation translates given instructions or raw HTML into decomposed queries: for instance, translating *"select 12/03/2016 as the date and hit submit"* into *"select the datepicker at step 1, click 'Prev' 7 times at step 2-8 (May is 7 months before December), click the date '12' at step 9, and finally submit at step 10"* (translated instruction), or mapping proper noun into corresponding XPath (translated HTML). While detailed structured prompts have led to strong performances, those should be specialized for each primitive task in MiniWoB by leveraging 7 different types of reformulation. See Appendix C.3 for further details.

4.4 Finetuned and Transferred Language Model Agents

In addition to the prompted LMAs, LMAs finetuned on base tasks have also been developed (Gur et al., 2022; Furuta et al., 2023; Shaw et al., 2023), which are built on pre-trained language models, such as T5 (Raffel et al., 2020), Flan-T5 (Chung et al., 2022), HTML-T5 (Gur et al., 2023), or Pix2Struct (Lee et al., 2023), with web automation demonstrations. Those LMAs take HTML (or screenshots) and previous actions as inputs and predict the text-format next actions in a closed-loop manner. Since pre-trained language models have a sufficient inductive bias for web environments and instruction-following, finetuned LMAs can data-efficiently achieve competitive performance to the RL-finetuned agents trained from scratch with domain-specific architectures (Humphreys et al., 2022; Liu et al., 2018). Compared to the prompted LMAs relying on private LLM API, it is possible to build on-premise agents based on tractable-size models (at most 3 billion parameters), which may reduce inference time and costs. However, prior works have pointed out that finetuned LMAs struggle to the sub-optimal performance (Zheng et al., 2023) and they require demonstrations on the order of hundreds of thousands while prompted LMAs just need hundreds of episodes (Kim et al., 2023). In this paper, we extensively evaluate such finetuned LMAs in **zero-shot transfer** settings; LMAs are finetuned only with base task demonstrations and should deal with unseen compositional tasks. We call those *transferred* LMAs in the later sections.

Models	Dataset Size	Success Rate
HTML-T5	347K episodes	85.6%
HTML-T5++ (ours)	424K (+77K)	94.1%
	351K (+77K - 73K)	94.6%
	322K (+77K - 102K)	94.8%
	282K (+77K - 142K)	**95.2%**
	241K (+77K - 183K)	95.0%
RCI (Kim et al., 2023)		90.6%
AdaPlanner (Sun et al., 2023)		92.9%
Human		93.5%
CC-Net (Humphreys et al., 2022)		93.5%
RCI (`gpt-4`) (Kim et al., 2023)		94.0%
Synapse (Zheng et al., 2023)		**98.5%**

Table 1: Average success rate of finetuned LMAs in 56 tasks on MiniWoB. Adding 77K episodes and reducing redundant thousands of episodes, HTML-T5++ achieves competitive performance to prompted LMAs, RL-finetuned agents, and humans, while improving the success rate from 85.6% to 95.2%.

4.5 Data-Rebalancing Improves Finetuned Language Model Agents

Furuta et al. (2023) utilized agent-driven data collection instead of humans to improve the performance of finetuned LMAs further. For each task, 10k demonstrations are collected and filtered based on task success, which resulted in challenging tasks having much less than 10k demonstrations due to the sub-optimal performance of LMA on these tasks. We identify that by fixing the data-imbalance problem, the performance of finetuned LMAs can be significantly improved, achieving super-human performance on MiniWoB. We first run Synapse (Zheng et al., 2023) on MiniWoB and collect 77K additional demonstrations across 16 tasks on top of 347K demonstrations (Furuta et al., 2023) to compensate for the lack of data in specific tasks. We then estimate the "brute-force" task difficulty averaging success rates for representative web automation agents. Based on those proximal measures, we classify 65 base tasks into three categories, such as `easy` (0.8 - 1.0), `medium` (0.6 - 0.8), and hard (0.0 - 0.6) (see Appendix D). We then balance the number of episodes based on the task difficulty, where we gradually reduce the ratio of easier tasks to focus more on challenging tasks. For instance, we remove $X\%$ episodes from top-k tasks in `easy` group (see Appendix E for the details).

We finetune HTML-T5-XL (Gur et al., 2023), a pre-trained language model with local and global attention in the encoder and a mixture of long-span denoising, on these rebalanced datasets. Table 1 shows that all the data-rebalance strategies improve the success rate, and reducing 50% episodes from `easy` tasks (finally 282K episodes in total) is the most effective rebalancing strategy. This suggests that finetuned LMAs can be as

Benchmark	# of Episode Steps					# of Instruction Tokens				
	Average	**50th**	**90th**	**95th**	**Max**	**Average**	**50th**	**90th**	**95th**	**Max**
MiniWoB (Shi et al., 2017)	3.6	4.3	5.5	5.8	7.0	8.5	14.8	30.5	12.9	23.2
Mind2Web (Deng et al., 2023)	7.7	6.0	14.0	17.0	37.0	19.2	16.0	33.0	42.0	84.0
CompWoB (**Ours**)	7.2	6.8	12.8	14.2	17.8	37.3	34.2	55.2	57.9	100.0

Table 2: The statistics (Mean, Max, 50/90/95th percentiles) of episode steps and instruction tokens from MiniWoB (Shi et al., 2017), Mind2Web (Deng et al., 2023), and CompWoB. CompWoB successfully increases task complexity from MiniWoB, and has sufficient difficulties the same as Mind2Web, a representative benchmark from real websites.

capable as prompted LMAs in decision making tasks. We include HTML-T5++ as a baseline in the later sections.

5 Designing CompWoB Controlling Task Complexity and Compositionality

Even though MiniWoB includes a spectrum of simulated environments, they have still focused on narrow and single-task instances. We need more advanced environments to measure the generalization to the various functional challenges in real-world web automation, such as complex observation (Deng et al., 2023), instruction (Zhou et al., 2023b), and task compositionality (Gur et al., 2021). We design *CompWoB* (i.e. compositional MiniWoB), as a general test bed for LMAs, by leveraging the high composability of simulated web environments (Figure 1). In CompWoB, we systematically combine several base tasks (from 2 to 8) in the original MiniWoB, which LMAs can already solve, into a single task (e.g. `click-checkboxes-transfer` + `enter-password` + `click-dialog` → `click-checkboxes-transfer_enter-password_click-dialog`). This allows us to control the complexity of HTML and instructions, ensuring novel tasks are solvable to some extent. Moreover, CompWoB can work as a *hold-out* test environment accompanying with MiniWoB as a train environment. While tasks are visually simplified, CompWoB reflects a broader spectrum of functionalities in real-world websites without sacrificing controllability. For instance, the task in Figure 1 sketches the structure of a login form with agreement checkboxes and a pop-up window, like Internet banking. We also designed tasks with page transitions, such as from the login form to the email browser.

5.1 Details of Task Design

As mentioned in Section 4.5, we first calculate the average success rates among representative web automation agents as brute-force task complexity, and classify 65 primitive tasks in MiniWoB into 3 categories (`easy`, `medium`, `hard`) based on those scores. The details of classification are described in Appendix D. We randomly select base tasks from `easy` group, filter those combinations by their feasibility and reality, and make 50 compositional tasks. We divide those into five categories: **two-way** tasks (20), **three-way** tasks (10), **n-way** tasks (5), **transition** tasks (5), and **easy-medium two-way** tasks (10). In n-way tasks, we combine from 4 to 8 tasks sequentially, and in transition tasks, we implement explicit page transition, for instance, transiting from the login form to the email browser. We also sample several tasks from `medium` group to construct easy-medium two-way tasks. You can find the full list of tasks in Appendix H. For the implementation, we stitch the instructions with "*and then*" and put each HTML on the same depth. LMAs should satisfy the given instructions sequentially, such as from task A, B, to C. Moreover, to test whether LMAs can deal with complex and ambiguous instructions, we propose **reverse-order instruction** settings, where the instruction is provided upside down while its task order is semantically the same (e.g. solve task B and C, after solving A). These simple yet controllable strategies make the analysis of LMA's behaviors tractable while reflecting compositional aspects of real-world tasks.

5.2 Number of Tasks

MiniWoB has around 104 base tasks in total; all the two-way and three-way combinations of these tasks would give 185,460 compositional tasks. It is infeasible to manually curate all the two-way/three-way combinations. Unfortunately, it is also nontrivial to automate this process due to the locality of the environment implementations in MiniWoB. Each environment is mostly self-contained, making it nontrivial

to modularize the whole benchmark/codebase so that every combination can be automatically generated. Alternatively, we outline a set of design principles – solvability, feasibility, and reality, which we follow to manually curate 50 compositional tasks and 50 reverse-order instruction tasks that could cover a wide variety of difficulty, and compositionality. We believe these guiding principles would be applicable to other compositional generalization problems such as robotic navigation. Based on these design principles, our compositional benchmark can easily be extended in the future to study even more challenging and compositional web automation problems. We would also like to highlight that, because previous works were tested around 50 - 60 MiniWoB tasks (Kim et al., 2023; Sun et al., 2023; Zheng et al., 2023; Gur et al., 2022; Furuta et al., 2023), 50 compositional tasks is a decent number of tasks to evaluate the agents. Our addition of a hold-out compositional test set doubles the number of tasks for the community to study.

5.3 Inherent Compositionality and Sub-Task Dependency in Web Automation

While many of the real-world tasks have inherent compositionality to some degree, these tasks are not explicitly designed for compositionality, making it difficult to systematically investigate the generalization gap. This can be analyzed with our CompWoB by separating the difficulty of the base tasks themselves and the difficulty of the task compositions.

Real-world websites have dependencies between sub-tasks. For instance, an email client requires a successful login to proceed to writing an email or reading social media posts of a specific user requires navigating to the personal page of that user. In CompWoB, we have implemented these kinds of dependencies in the reward function using logical AND operations. For instance, while we allowed agents to continue to "write an email" sub-task without successful login, the episode is flagged as a failure (i.e. zero reward) even if the "write an email" sub-task is successful. We inform agents of these dependencies using connectors in instructions such as "solve B after A" or "solve A then B". Our analysis in Section 6.4 also suggests that, in addition to task compositionality, long instructions and deep HTML sub-tree can lead to challenging tasks.

5.4 Comparison to Other Web Automation Benchmarks

We design CompWoB on top of MiniWoB, which enables us to test the human-level LMAs (Kim et al., 2023; Sun et al., 2023; Zheng et al., 2023; Gur et al., 2023) as strong baselines. Because the success of in-domain tasks is ensured, we could focus on the analysis of generalization to unknown task compositions and their functionality in a controllable way. In contrast, other benchmarks copying real websites fail to obtain decent agents (for instance, the episode success rate of Mind2Web (Deng et al., 2023) is around 5-10% at most; around 15% success at most in WebArena (Zhou et al., 2023b)) and it seems to be harder to identify the attribution of errors due to the complex nature of real websites. For the task complexity compared to other benchmarks, we provide the statistics (mean, max, percentiles) of episode steps and instruction tokens from MiniWoB (Shi et al., 2017), Mind2Web, and CompWoB in Table 2. In terms of episode length and instruction length, CompWoB successfully increases task complexity from MiniWoB, and has sufficient difficulties the same as Mind2Web, a representative benchmark from real websites.

6 Results

Evaluation Methodology We evaluate both transferred and prompted LMAs with base MiniWoB demonstrations on the unseen compositional tasks in a "**zero-shot**" manner; i.e. **we do not provide any demonstrations on the compositional tasks for the training corpus and exemplars to measure the generalization**. We test 50 compositional tasks and run 100 episodes per task. We adopt `gpt-3.5-turbo` as a backbone LLM, unless otherwise mentioned. We assume the optimal exemplar retriever throughout experiments and always provide the pre-defined prompts to LMAs. We borrow hyper-parameters and prompt templates from respective papers with minimal change to respect our zero-shot transfer setting. As a sanity check for the quality of environments and baseline agents, we also test LMAs with oracle demonstrations in Appendix I.

RCI We test 4 prompting strategies: (1) zero-shot (without any exemplars), few-shot with (2) first-task exemplars, (3) second-task exemplars, and (4) combination exemplars (i.e. both first and second tasks). For

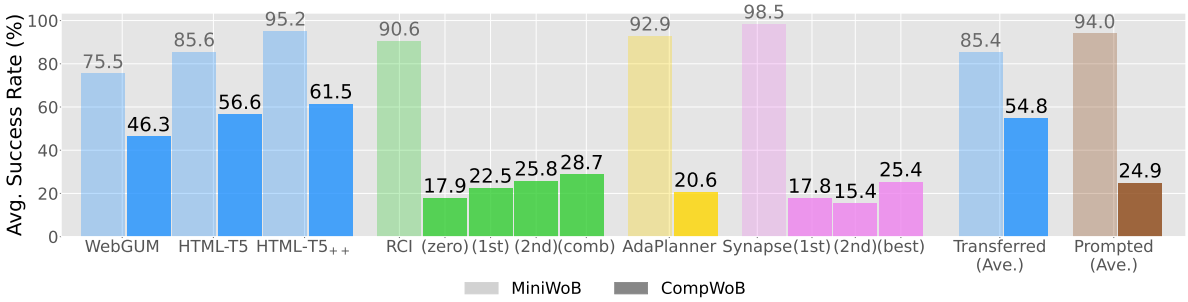

Figure 2: Average success rate of LMAs in 50 CompWoB tasks. The light color represents the performance in the original MiniWoB, and the dark color for CompWoB. We use `gpt-3.5-turbo` as the backbone LLM for prompted LMAs (RCI (Kim et al., 2023), AdaPlanner (Sun et al., 2023), Synapse (Zheng et al., 2023)), and transferred LMAs with 3-billion parameters. Transferred LMA, especially HTML-T5++, achieves the best generalization in compositional tasks, suppressing the performance degradation (from 95.2% to 61.5%). On the contrary, prompted LMAs drop their performance significantly; even the best RCI that uses combined task prompts in the composition just achieves 28.7% success (from 90.6% in base tasks). This indicates, in contrast to the base task performances, prompted LMAs are more vulnerable to, and transferred LMAs can deal with unknown task compositionality better than expected.

consistency and limited context length, we always consider the first two tasks even if the number of primitive tasks is more than two, and fix the number of self-improvement iterations to 1 explicit RCI and 3 implicit RCI as recommended in the original paper. The exemplars we use are provided by Kim et al. (2023).

AdaPlanner Following the original code, we use the same exemplars provided by Sun et al. (2023) for tasks where those base tasks are included, such as `enter-text` and `click-widget` (see Appendix C.2). Otherwise, the agents are prompted in a zero-shot manner.

Synapse We test 3 prompting strategies: few-shot with (1) first-task, (2) second-task exemplars, and (3) best exemplars (i.e. maximum score between (1) and (2)). Because prompts and modules are quite different among tasks, we do not merge the prompts and use proper hyper-parameters corresponding to the given exemplars designed by Zheng et al. (2023).

6.1 Language Model Agents Struggle to Handle Task Compositionality

Figure 2 shows that, in CompWoB, all the LMAs face performance degradation. Among those, transferred LMAs achieve better success rate (54.8%) than prompted LMAs (24.9%) on average. In particular, HTML-T5++ achieves the best generalization, suppressing the performance drop from 95.2% to 61.5%. This might be because finetuned LLMs are better at natural language compositional tasks prompted LLMs in general. See Appendix K for further details. In contrast, prompted LMAs degrade their performance drastically; even the best RCI with few-shot combination exemplars (comb) just degrades the success rate to 28.7% from 90.6% in base MiniWoB. These results indicate that LMAs suffer from generalization to task compositionality, and transferred LMAs can relatively deal with that better than prompted LMAs, which is an opposite trend to base MiniWoB performance. Among prompted LMAs, RCI performs better than AdaPlanner and Synapse, which suggests that multiple iterations of self-criticism and improvement might be more robust to out-of-domain decision making from the exemplars than program synthesis with feedback or structured prompting with state translations.

RCI (Kim et al., 2023)		**AdaPlanner** (Sun et al., 2023)		**Synapse** (Zheng et al., 2023)	
Click button ONE, then click button TWO, and then select whX, 1Nk, fUK3 and click Submit		*Enter the password "UBKR" into both text fields, and then select KwpUv and click Submit*		*Select yE, and then enter "Juan" into the text field and press Submit*	
✔	✗	✔	✗	✔	✗
1. click //button[@id="subbtn1"]	1. type //button[@id="subbtn2"]	1. click //*[@id="password"]		1. click //*[text()="yE"]/input	
2. click //button[@id="subbtn2"]	2. click //*[text()="whX"]/input	2. type UBKR	1. type UBKR	2. click //input[@id="tt"]	1. click //input[@id="tt"]
3. click //*[text()="whX"]/input	3. click //*[text()="1Nk"]/input	3. click //*[@id="verify"]			2. type yE
4. click //*[text()="1Nk"]/input	4. click //*[text()="fUK3"]/input	4. type UBKR	2. type UBKR	3. type Juan	3. type Juan
5. click //*[text()="fUK3"]/input	5. click //*[text()="gSm"]/input	5. click //input[@id="ch0"]	3. click //input[@id="ch0"]	4. click //*[@id="subbtn"]	4. click //*[@id="subbtn"]
6. click //*[@id="subbtn"]	6. click //*[@id="subbtn"]	6. click //*[@id="subbtn"]	4. click //*[@id="subbtn"]		

Table 3: Failure examples in CompWoB. The left columns have correct plans and the right have failure plans. LMAs often ignore necessary intermediate steps or predict incorrect action types and XPath.

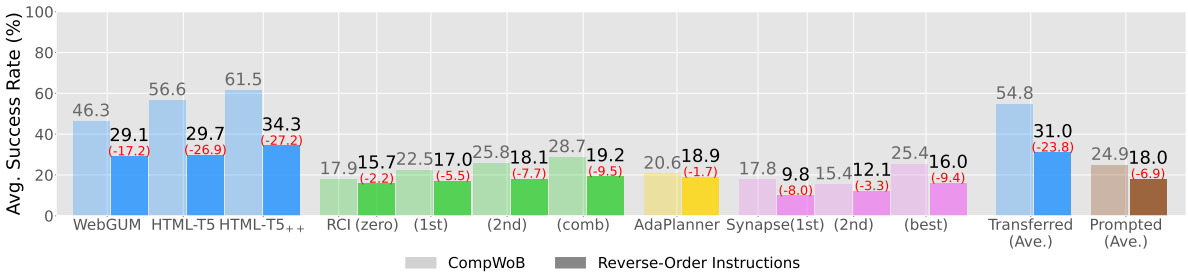

Figure 3: Average success rate of language model agents in **reverse-order instruction** settings. We use `gpt-3.5-turbo` as the backbone LLM for prompted LMAs (RCI, AdaPlanner, Synapse), and transferred LMAs with 3-billion parameters. Notably, most LMAs significantly degrade the success rate when reverse-order instructions are provided. This trend is more remarkable in transferred LMA (54.8% → 31.0% on average) than prompted LMA (24.9% → 18.0% on average), which suggests that any kind of LMAs are susceptible to the order of compositional instructions. The capability as general language models might be important to parse semantically complex instructions into the correct plan.

In the failure episodes (Table 3), LMAs often miss necessary steps, common to all the prompted LMAs. Since the instructions get long in compositional settings, LMAs may skip important intermediate steps to satisfy the instructions. In addition, they predict incorrect action types and XPath: for instance, hallucination in XPath (RCI) and mixing up *click* and *type* action (Synapse). Appendix H also provides average success rates per category.

6.2 Reverse-Order Instructions Degrade Language Model Agents

As shown in Figure 3, all the LMAs significantly degrade the success rate when reverse-order instructions are provided. This trend is more remarkable in transferred LMAs dropping from 54.8% to 31.0% than prompted LMAs from 24.9% to 18.0% on average, which suggests that any kind of LMAs is susceptible to the order of compositional instructions and that transferred LMAs may not generalize well to diverse instructions beyond the dataset distribution. As opposed to Section 6.1, the performance differences among prompted LMAs are marginal, which, however, implies that existing prompting methods, even with self-improvement, may not handle complex task instructions enough. The stronger capability as general-purpose language models or other prompting methods might be important to parse semantically complex instructions into the executable sequential plan.

Compared to Section 6.1, RCI and Synapse cannot parse reverse-order instructions into plans correctly (Table 4). This can be because base-task exemplars in the prompt just have "left-to-right" instructions from base MiniWoB tasks; strongly conditioning LMAs to process instructions in a linear order while compositionality could imply a non-linear processing of instructions. The mismatch between the prompt and tasks could cause the performance drops. LMAs still fail to select correct action types (AdaPlanner) or XPath (Synapse), and they also predict unnecessary actions (RCI).

6.3 Do Advanced LLMs Solve Compositional Tasks?

Figure 4 presents the results when we adopt other advanced LLMs, than `gpt-3.5-turbo`, as a backbone of each LMA. The more capable models, such as `gpt-4` in a generalist aspect and `text-davinci-003` in

RCI (Kim et al., 2023)		**AdaPlanner** (Sun et al., 2023)		**Synapse** (Zheng et al., 2023)	
Select rJ and click Submit, after clicking on the "yes" button		*Select OkRi7, and click Submit, after clicking on the "previous" button*		Select 2ld1 and click Submit, after entering the password "Zy4XI" *into both text fields*	
✔	✗	✔	✗	✔	✗
1. click //button[text()="yes"] 2. click //*[text()="rj"]/input 3. click //*[@id="subbtn"]	1. click //*[text()="rj"]/input 2. click //button[text()="yes"] 3. type rj 4. click //*[@id="subbtn"]	1. click //*[text()="previous"] 2. click //*[text()="OkRi7"]/input 3. click //*[@id="subbtn"]	1. click //*[text()="previous"] 2. type OkRi7 3. click //*[@id="subbtn"]	1. click //*[@type="password"] 2. type Zy4XI 3. click //*[text()="verify"] 4. type Zy4XI 5. click //*[text()="2ld1"]/input 6. click //*[@id="subbtn"]	1. click //*[text()="2ld1"]/input 2. click //*[@type="password"][1] 3. type Zy4XI 4. click //*[@type="password"][2] 5. type Zy4XI 6. click //*[@id="subbtn"]

Table 4: Failure examples in CompWoB with **reverse-order** instructions. LMAs often fail to parse the instruction into the correct-order plan, and hallucinate unnecessary actions (e.g. *type*).

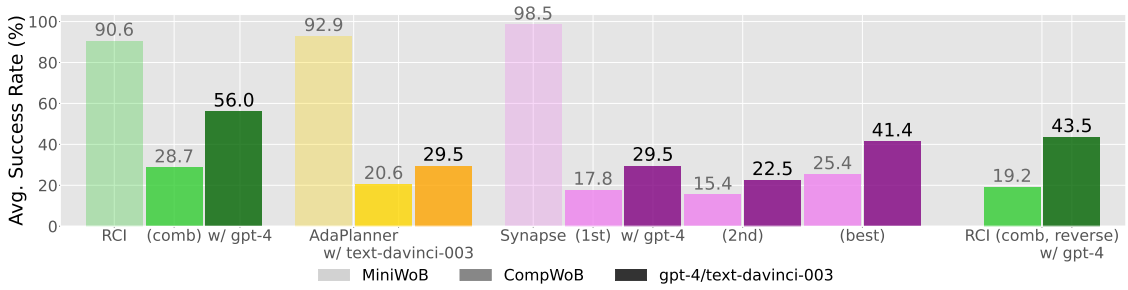

Figure 4: Average success rate of large language models with advanced LLMs (`gpt-4` for RCI and Synapse, and `text-davinci-003` for AdaPlanner). The lighter color represents the performance in MiniWoB, the medium color does in CompWoB with `gpt-3.5-turbo`, and the darker color does with `gpt-4` or `text-davinci-003`. The more capable models (`gpt-4` as a generalist and `text-davinci-003` as a coder) can improve the success rate of prompted LMAs but still struggle to generalize to compositional tasks (e.g. 56.0% by RCI) or to deal with reverse-order instructions (e.g. 43.5% by RCI). This may indicate that we need much better LLMs to realize deployable agents in the complex real world.

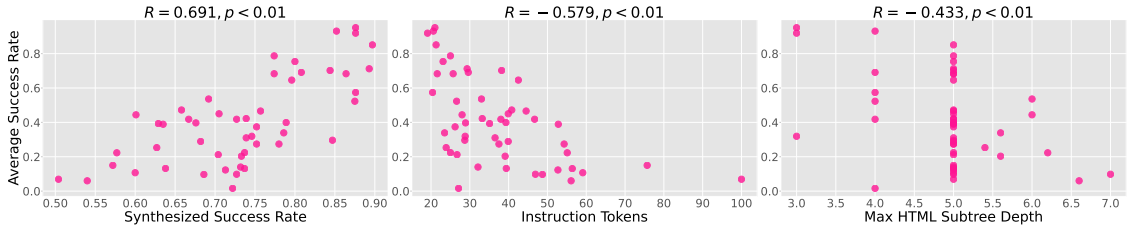

Figure 5: 2D-scatter plots between success rate averaged among LMAs (y-axis) and each statistic of compositional task (x-axis), such as success rate synthesized with a product of base task success rate, the number of instruction tokens, and max depth of HTML subtrees. Synthesized success rate positively correlates with an average success rate ($R = 0.691$, statistically significant in paired t-test with $p < 0.01$), indicating that base task difficulty may determine compositional task difficulty. In addition, the number of instruction tokens ($R = -0.579$; $p < 0.01$) and the max depth of HTML subtrees ($R = -0.433$; $p < 0.01$) show negative correlations, which suggests the high complexity of observation and long instructions make the compositional tasks hard to resolve.

a code generation, can improve the success rate of all the prompted LMAs. However, even `gpt-4` is still far from the generalization in compositional tasks (from 28.7% to 56.0% by RCI) or from dealing with reverse-order instructions (from 19.2% to 43.5% by RCI). This indicates that we need much better LLMs to realize deployable systems in complex real-world decision-making tasks. We provide failure examples in Appendix G.

6.4 What Determines Task Complexity on the Web?

Figure 5 visualizes the correlation between the success rate averaged across WebGUM, HTML-T5, RCI, AdaPlanner, and Synapse (y-axis) and each statistic of compositional tasks (x-axis), such as synthesized success rate – a product of base task success rates among compositional tasks – the number of instruction tokens, and max depth of HTML subtrees. Synthesized success rate positively correlates with an average success rate ($R = 0.691$), indicating that compositional task difficulty takes over base task difficulties. In addition, the number of instruction tokens ($R = -0.579$) and the max depth of HTML subtrees ($R = -0.433$) show negative correlations. All those are statistically significant in paired t-test with $p < 0.01$. In contrast, other task statistics, such as synthesized success rate with human performance, the number of HTML tokens, and elements in HTML, just show relatively weaker correlations (see Appendix F for the details). This analysis suggests that HTML with larger depth and long instructions make generalizing compositional tasks challenging. The complexity of HTML is determined by its depth rather than its length or the number of elements. This might come from the hierarchical nature of HTML; in deeper HTML subtrees, the elements near the root tend to be distant from each other after the traversal. Such sparsity may cause confusion during planning. We also report the individual characteristics of each agent in Appendix J.

7 Discussion

Generalizable Prompting Methods The results of Synapse and RCI in Figure 2 imply that those prompted LMAs have "over-fitting" trends to the base MiniWoB tasks. While the robustness across the prompts has been investigated in natural language tasks (Wei et al., 2022; Kojima et al., 2022), it is not well understood in decision making. Because we will not be able to prepare the optimal self-improvement iterations or decomposed prompts for all the possible instructions and task compositions, even if using optimal exemplar retrievers, we should care more about the generalization of prompting methods for the agent systems.

Agent-Specialized Large Language Models As shown in Figure 4, the more capable LLMs, such as `gpt-4`, can improve the performance of LMAs in CompWoB. However, it has not reached the base MiniWoB yet (e.g. from 90.6% to 56.0% in RCI, and from 98.5% to 41.4% in Synapse). Similarly, as described in Section 4.4, transferred LMAs can perform better if the training dataset has a good balance and coverage, but it is far from sufficient generalization to compositionality and instructions. The current pre-trained LLMs may still not be sufficient to generalize to complex decision making tasks, and then, in addition to prompting methods, the development of agent-specialized LLMs with enhanced reasoning and generalization through the instruction-tuning (Zeng et al., 2023) or RLHF/AIF (Furuta et al., 2024) would be expected.

Parsing Complex Instructions to Executable Plan Section 6.2 highlights that LMAs are fragile when we increase the complexity of instruction even by the most straightforward reverse-order instructions. This may not be preferable for the real-world application since the instructions might not be easy-to-parse and the users should carefully and concisely tell what they would like to do, which hinders the user's experience. It would be an interesting future direction to investigate better planning modules that could parse complex instructions to correct and executable plans.

8 Conclusion

The robustness and generalization of LMAs are important aspects for real-world deployment. We extensively examine how much existing LMAs, via transferring and prompting, can deal with a set of compositional web automation environments, CompWoB, that consists of easily-resolvable base primitive tasks. Our evaluation implies the contrary conclusion to the prior works (Table 5); the prompted LMAs are strong solver for primitive web automation tasks but significantly drop their performance in unknown task compositionality. The transferred LMAs often show sub-optimal performance in basic tasks but can deal with compositional problems much better. Our detailed analysis also highlights that LMAs also face catastrophic degradation when they receive complex, even in the simplest reversed-order instructions, and that the challenges in compositional tasks might come from instruction length and the depth of HTML subtree. We hope this inspires the community to build robust and generalizable LMAs to task compositionality toward real-world application.

	Base MiniWoB	CompWoB	Reverse-Order Instructions	Advanced Models
Prompted LMA	**94.0**% / **98.5**%	24.9% / 28.7%	18.0% / 19.2%	42.3% / 56.0%
Transferred LMA	85.4% / 95.2%	**54.8**% / **61.5**%	**31.0**% / **34.3**%	–

Table 5: Summary of average / max success rate in web automation.

Ethics Statements

This paper systematically evaluates the performance of existing language model agents for web automation in realistic compositional tasks, and unveils remaining challenges towards broader generalization and real-world deployments. This technique could realize capable AI assistant tools on digital devices (e.g. computers or smartphones), and improve productivity and accessibility for society.

While we anticipate the positive aspects of autonomous agents, for responsible development, we should also consider the potential harmful applications and unintended consequences. The misuse of web automation

would threaten cyber security, and the users may get scammed. To reduce these risks, it is essential for researchers, policymakers, and industry to discuss concrete guidelines and regulations.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arxiv:2204.01691*, 2022.

Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernandez Abrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vlad Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, Guy Gur-Ari, Steven Hand, Hadi Hashemi, Le Hou, Joshua Howland, Andrea Hu, Jeffrey Hui, Jeremy Hurwitz, Michael Isard, Abe Ittycheriah, Matthew Jagielski, Wenhao Jia, Kathleen Kenealy, Maxim Krikun, Sneha Kudugunta, Chang Lan, Katherine Lee, Benjamin Lee, Eric Li, Music Li, Wei Li, YaGuang Li, Jian Li, Hyeontaek Lim, Hanzhao Lin, Zhongtao Liu, Frederick Liu, Marcello Maggioni, Aroma Mahendru, Joshua Maynez, Vedant Misra, Maysam Moussalem, Zachary Nado, John Nham, Eric Ni, Andrew Nystrom, Alicia Parrish, Marie Pellat, Martin Polacek, Alex Polozov, Reiner Pope, Siyuan Qiao, Emily Reif, Bryan Richter, Parker Riley, Alex Castro Ros, Aurko Roy, Brennan Saeta, Rajkumar Samuel, Renee Shelby, Ambrose Slone, Daniel Smilkov, David R. So, Daniel Sohn, Simon Tokumine, Dasha Valter, Vijay Vasudevan, Kiran Vodrahalli, Xuezhi Wang, Pidong Wang, Zirui Wang, Tao Wang, John Wieting, Yuhuai Wu, Kelvin Xu, Yunhan Xu, Linting Xue, Pengcheng Yin, Jiahui Yu, Qiao Zhang, Steven Zheng, Ce Zheng, Weikang Zhou, Denny Zhou, Slav Petrov, and Yonghui Wu. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

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, 2023.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. *arXiv preprint arXiv:2305.17126*, 2023.

Jiaao Chen, Xiaoman Pan, Dian Yu, Kaiqiang Song, Xiaoyang Wang, Dong Yu, and Jianshu Chen. Skills-in-context prompting: Unlocking compositionality in large language models. *arXiv preprint arXiv:2308.00304*, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder,

Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models. *arXiv preprint arxiv:2210.11416*, 2022.

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.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. Workarena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arxiv:2403.07718*, 2024.

Andrew Drozdov, Nathanael Schärli, Ekin Akyürek, Nathan Scales, Xinying Song, Xinyun Chen, Olivier Bousquet, and Denny Zhou. Compositional semantic parsing with large language models. In *International Conference on Learning Representations*, 2023.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality. *arXiv preprint arxiv:2305.18654*, 2023.

Daniel Furrer, Marc van Zee, Nathan Scales, and Nathanael Schärli. Compositional generalization in semantic parsing: Pre-training vs. specialized architectures. *arXiv preprint arXiv:2007.08970*, 2021.

Hiroki Furuta, Tatsuya Matsushima, Tadashi Kozuno, Yutaka Matsuo, Sergey Levine, Ofir Nachum, and Shixiang Shane Gu. Policy information capacity: Information-theoretic measure for task complexity in deep reinforcement learning. In *International Conference on Machine Learning*, 2021.

Hiroki Furuta, Ofir Nachum, Kuang-Huei Lee, Yutaka Matsuo, Shixiang Shane Gu, and Izzeddin Gur. Multimodal web navigation with instruction-finetuned foundation models. *arXiv preprint arxiv:2305.11854*, 2023.

Hiroki Furuta, Kuang-Huei Lee, Shixiang Shane Gu, Yutaka Matsuo, Aleksandra Faust, Heiga Zen, and Izzeddin Gur. Geometric-averaged preference optimization for soft preference labels. *arXiv preprint arxiv:2409.06691*, 2024.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*, 2023.

Izzeddin Gur, Ulrich Rueckert, Aleksandra Faust, and Dilek Hakkani-Tur. Learning to navigate the web. In *International Conference on Learning Representations*, 2019.

Izzeddin Gur, Natasha Jaques, Yingjie Miao, Jongwook Choi, Manoj Tiwari, Honglak Lee, and Aleksandra Faust. Environment generation for zero-shot compositional reinforcement learning. In *Advances in neural information processing systems*, 2021.

Izzeddin Gur, Ofir Nachum, Yingjie Miao, Mustafa Safdari, Austin Huang, Aakanksha Chowdhery, Sharan Narang, Noah Fiedel, and Aleksandra Faust. Understanding html with large language models. *arXiv preprint arxiv:2210.03945*, 2022.

Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. A real-world webagent with planning, long context understanding, and program synthesis. *arXiv preprint arxiv:2307.12856*, 2023.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *arXiv preprint arxiv:2305.11554*, 2023.

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

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022a.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arxiv:2207.05608*, 2022b.

Peter C Humphreys, David Raposo, Toby Pohlen, Gregory Thornton, Rachita Chhaparia, Alistair Muldal, Josh Abramson, Petko Georgiev, Alex Goldin, Adam Santoro, and Timothy Lillicrap. A data-driven approach for learning to control computers. In *International Conference on Machine Learning*, 2022.

Sheng Jia, Jamie Ryan Kiros, and Jimmy Ba. DOM-q-NET: Grounded RL on structured language. In *International Conference on Learning Representations*, 2019.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks. *arXiv preprint arxiv:2303.17491*, 2023.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Advances In Neural Information Processing Systems*, 2022.

Kenton Lee, Mandar Joshi, Iulia Turc, Hexiang Hu, Fangyu Liu, Julian Eisenschlos, Urvashi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. Pix2struct: Screenshot parsing as pretraining for visual language understanding. *arXiv preprint arxiv:2210.03347*, 2023.

Minghao Li, Feifan Song, Bowen Yu, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*, 2023.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2023.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *International Conference on Learning Representations*, 2018.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 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, Ran Xu, Phil Mui, Huan Wang, Caiming Xiong, and Silvio Savarese. 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.

Kaixin Ma, Hongming Zhang, Hongwei Wang, Xiaoman Pan, and Dong Yu. Laser: Llm agent with state-space exploration for web navigation. *arXiv preprint arxiv:2309.08172*, 2023.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arxiv:2303.17651*, 2023.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

Kolby Nottingham, Prithviraj Ammanabrolu, Alane Suhr, Yejin Choi, Hannaneh Hajishirzi, Sameer Singh, and Roy Fox. Do embodied agents dream of pixelated sheep: Embodied decision making using language guided world modelling. In *International Conference on Machine Learning*, 2023.

OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. *arXiv preprint arxiv:2203.02155*, 2022.

Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models. *arXiv preprint arXiv:2205.12255*, 2022.

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.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, Bokai Xu, Zhen Zhang, Yining Ye, Bowen Li, Ziwei Tang, Jing Yi, Yuzhang Zhu, Zhenning Dai, Lan Yan, Xin Cong, Yaxi Lu, Weilin Zhao, Yuxiang Huang, Junxi Yan, Xu Han, Xian Sun, Dahai Li, Jason Phang, Cheng Yang, Tongshuang Wu, Heng Ji, Zhiyuan Liu, and Maosong Sun. Tool learning with foundation models. *arXiv preprint arXiv:2304.08354*, 2023a.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023b.

Linlu Qiu, Peter Shaw, Panupong Pasupat, Tianze Shi, Jonathan Herzig, Emily Pitler, Fei Sha, and Kristina Toutanova. Evaluating the impact of model scale for compositional generalization in semantic parsing. *arXiv preprint arXiv:2205.12253*, 2022.

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

Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi, Hangyu Mao, Xingyu Zeng, and Rui Zhao. Tptu: Task planning and tool usage of large language model-based ai agents. *arXiv preprint arXiv:2308.03427*, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? *arXiv preprint arXiv:2010.12725*, 2021.

Peter Shaw, Mandar Joshi, James Cohan, Jonathan Berant, Panupong Pasupat, Hexiang Hu, Urvashi Khandelwal, Kenton Lee, and Kristina Toutanova. From pixels to ui actions: Learning to follow instructions via graphical user interfaces. *arXiv preprint arXiv:2306.00245*, 2023.

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

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*, 2017.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arxiv:2303.11366*, 2023.

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

Abishek Sridhar, Robert Lo, Frank F. Xu, Hao Zhu, and Shuyan Zhou. Hierarchical prompting assists large language model on web navigation. *arXiv preprint arxiv:2305.14257*, 2023.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. *arXiv preprint arXiv:2305.16653*, 2023.

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

Yujin Tang, Wenhao Yu, Jie Tan, Heiga Zen, Aleksandra Faust, and Tatsuya Harada. Saytap: Language to quadrupedal locomotion. In *Conference on Robot Learning*, 2023b.

Hung Quoc To, Nghi D. Q. Bui, Jin Guo, and Tien N. Nguyen. Better language models of code through self-improvement. *arXiv preprint arxiv:2304.01228*, 2023.

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. Llama: Open and efficient foundation language models. *arXiv preprint arxiv:2302.13971*, 2023.

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

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*, 2023b.

Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. In *International Conference on Machine Learning*, 2023c.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*, 2023.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. On the tool manipulation capability of open-source large language models. *arXiv preprint arXiv:2305.16504*, 2023.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. Gpt4tools: Teaching large language model to use tools via self-instruction. *arXiv preprint arXiv:2305.18752*, 2023.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *arXiv preprint arxiv:2207.01206*, 2022a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022b.

Eric Zelikman, Qian Huang, Gabriel Poesia, Noah D. Goodman, and Nick Haber. Parsel: Algorithmic reasoning with language models by composing decompositions. *arXiv preprint arxiv:2212.10561*, 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.

Longtao Zheng, Rundong Wang, and Bo An. Synapse: Leveraging few-shot exemplars for human-level computer control. *arXiv preprint arXiv:2306.07863*, 2023.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. Least-to-most prompting enables complex reasoning in large language models. In *International Conference on Learning Representations*, 2023a.

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

Appendix

A Details of LLM API

We used OpenAI API to call LLM inference in our experiments. Table 6 shows the API used for each method. We did most of our experiments from 2023/07 to 2023/09. We use the official implementations and prompts released by the authors [2][3][4]. We spent about \$3.6K for the experiments in total.

Methods	**API**	**Cost** (input/output; /1K tokens)	**Context Length**
RCI (Kim et al., 2023)	`gpt-3.5-turbo`	\$0.0015 / \$0.002	4K tokens
	`gpt-4`	\$0.03 / \$0.06	8K tokens
AdaPlanner (Sun et al., 2023)	`gpt-3.5-turbo`	\$0.0015 / \$0.002	4K tokens
	`text-davinci-003`	\$0.02 / \$0.02	4K tokens
Synapse (Zheng et al., 2023)	`gpt-3.5-turbo`	\$0.0015 / \$0.002	4K tokens
	`gpt-4`	\$0.03 / \$0.06	8K tokens

Table 6: List of LLM API used in this paper. We did those experiments from 2023/07 to 2023/09.

B Pseudo Code for Prompted Language Model Agents

We explicitly distinguish LLM itself and LMA (combination of LLM, prompting and pipeline) in this paper. We provide the pseudo code for prompted LMAs to clarify their methodological difference. RCI (Kim et al., 2023) is the first to use prompting in a self-refinement loop, outperforming imitation- or reinforcement-learned agents on MiniWoB benchmark that requires millions of demonstrations to work. AdaPlanner (Sun et al., 2023) and Synapse (Zheng et al., 2023) are the follow-up works outperforming RCI via code generation from environmental feedback or via well-designed decomposed prompts with retrieval.

Algorithm 1 Prompted Language Model Agents: RCI, AdaPlanner, Synapse

Input: prompt P, LMA π, task ψ, environment `Env`, large language model `LLM`, number of ERCI N_{ERCI}, number of IRCI N_{IRCI}

$s, g \leftarrow$ `Env.reset(`ψ`)`
$s, g \leftarrow$ `LLM`$(\cdot|P_{\text{syn}}, s, g)$ ▷ Task Reformulation (Synapse)
`history` $\leftarrow \{\}$
while `Env` is not terminated **do**
  $\{a_1, ..., a_T\} \leftarrow \pi(\cdot|P_\pi, s, g)$ ▷ Planning
  for i **in** `range(`N_{ERCI}`)` **do**
    `criticism` $\leftarrow$ `LLM`$(\cdot|P_{\text{rci}}, \{a_1, ..., a_T\})$ ▷ Criticism (RCI)
    $\{a_1, ..., a_T\} \leftarrow \pi(\cdot|P_\pi, s, g, \{a_1, ..., a_T\},$ `criticism`$)$ ▷ Improvement (RCI)
  end for
  for a **in** $\{a_1, ..., a_T\}$ **do**
    for j **in** `range(`N_{IRCI}`)` **do**
      $a \leftarrow \pi(\cdot|P_\pi, s, g, \{a, ..., a_T\},$ `history`$)$ ▷ Improvement (RCI)
    end for
    $s, r, \text{info} \leftarrow$ `Env.step(a)`
    $\{a, ..., a_T\} \leftarrow \pi(\cdot|P_\pi, s, g, \{a, ..., a_T\},$ `history`$, \text{info})$ ▷ Replanning (AdaPlanner)
    `history` $\leftarrow$ `history` $\cup \{a\}$
  end for
end while

[2] https://github.com/posgnu/rci-agent
[3] https://github.com/haotiansun14/AdaPlanner
[4] https://github.com/ltzheng/Synapse

C Details of Hyper-parameters

C.1 RCI

As we described in Section 4.1, RCI has two important hyper-parameters to control the number of self-improvement iterations. In Explicit RCI (ERCI) loop, LLMs criticize their own generated plans to identify the problem and then improve it, reflecting self-criticism. In Implicit RCI (IRCI) loop, LLMs ground the action to the current state (i.e. HTML) and refine its formatting to be parsable without self-criticism, which may reduce hallucinations or tiny errors. We here test how many self-improvement loops RCI requires (IRCI: 1-4, ERCI: 0-2). Table 7 shows that the optimal number of inference loops is different among tasks, while the recommendations are ERCI = 1 and IRCI = 3. These two hyper-parameters might need to be adjusted for each task.

	(ERCI, IRCI)											
Tasks	(0,1)	(0,2)	(0,3)	(0,4)	(1,1)	(1,2)	**(1,3)**	(1,4)	(2,1)	(2,2)	(2,3)	(2,4)
`click-button`	**1.00**	**1.00**	**1.00**	**1.00**	0.92	0.84	0.87	0.88	0.93	0.86	0.87	0.87
`click-checkboxes`	0.90	0.94	0.87	0.91	0.96	0.94	0.97	**1.00**	0.89	0.91	0.94	0.99
`click-dialog`	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
`click-link`	0.96	0.95	0.98	**0.99**	0.91	0.91	0.89	0.88	0.95	0.91	0.91	0.87
`click-option`	0.82	0.77	0.79	**0.87**	0.41	0.54	0.56	0.52	0.83	0.82	0.73	**0.87**
`click-scroll-list`	**0.86**	0.86	0.84	0.83	0.75	0.81	0.78	0.79	0.76	0.81	0.85	0.74

Table 7: The success rate of RCI with different hyper-parameters. The optimal parameters differ in each task, while the recommended one is (1,3).

C.2 AdaPlanner

We use the demonstrations of these 13 tasks where they are included in the task composition:

- `enter-text`
- `click-widget`
- `navigate-tree`
- `login-user-popup`
- `email-inbox-forward-nl-turk`
- `click-checkboxes-large`
- `click-tab-2-hard`
- `click-dialog-2`
- `search-engine`
- `click-checkboxes-soft`
- `use-autocomplete`
- `enter-date`
- `click-dialog-2`

C.3 Synapse

As we explained in Section 4.3, Synapse has several hyper-parameters to construct optimal structured prompts per task to specify whether LLMs translate the instruction or HTML.

Table 8 summarizes the type of reformulation into 7 categories and clarifies which transformed inputs are used for predicting open-loop plans. For instance, `Task` only requires translated instructions (and few-shot planning exemplars), although `Obs` takes raw instruction, HTML, and translated HTML as inputs. For the tasks that require temporal abstraction, it also employs state-conditional decomposition, which factorizes demonstrations into a set of exemplars conditioned on the environmental states, and can reduce error accumulation over the time step.

Table 9 provides the detailed values for state-filtering and task reformulation, which is quite different across the tasks. These well-designed structured prompts could be the source of the best performance in base MiniWoB. However, in compositional settings, it is challenging to modify them for any combinations. Instead, we assume the optimal retriever always picks up the exemplars for one of the base tasks, and we compute the maximum score among the results with given prompts.

	Reformulation Strategies						
Inputs	`Task`	`Obs`	`Obs_Task`	`Obs_Task_Filter`	`Raw_Task`	`None`	`None_Filter`
Instruction	Translated	Raw	Translated	Translated	Raw	Raw	Raw
HTML	✗	Raw+Translated	Raw+Translated	Translated	✗	Raw	Translated

Table 8: Summary of task reformulation for structured prompting used in Synapse (Zheng et al., 2023). Structured prompts are finely designed per task.

Tasks	**State Filtering**	**Exemplar Decomposition**	**Raw Task Only**	**Reformulation**
book-flight	True	True	False	None
choose-date	False	False	False	Task
choose-list	False	False	True	None
click-button	False	False	False	None
click-button-sequence	False	False	False	None
click-checkboxes	False	False	True	None
click-checkboxes-large	False	False	True	None
click-checkboxes-soft	False	False	False	Obs
click-checkboxes-transfer	False	False	True	None
click-collapsible	False	False	True	None
click-collapsible-2	False	False	False	Obs Task
click-color	False	False	True	None
click-dialog	False	False	True	None
click-dialog-2	False	False	False	None
click-link	False	False	True	None
click-menu	False	False	True	Task
click-option	False	False	True	None
click-pie	False	False	True	None
click-scroll-list	False	False	True	None
click-shades	False	False	True	Task
click-shape	True	False	False	Obs Task
click-tab	False	False	True	None
click-tab-2	True	False	False	Obs Task
click-tab-2-hard	True	False	False	Obs Task
click-test	False	False	False	None
click-test-2	False	False	False	None
click-widget	False	False	True	None
copy-paste	False	False	False	None
copy-paste-2	False	False	False	None
count-shape	True	False	False	Obs Task
email-inbox	False	False	False	Task
email-inbox-forward-nl	False	False	False	Task
email-inbox-forward-nl-turk	False	False	False	Task
email-inbox-nl-turk	False	False	False	Task
enter-date	False	False	True	None
enter-password	False	False	True	None
enter-text	False	False	True	None
enter-text-dynamic	False	False	True	None
enter-time	False	False	True	None
find-word	True	False	False	None
focus-text	False	False	True	None
focus-text-2	False	False	True	None
grid-coordinate	False	False	True	None
guess-number	False	False	False	None
identify-shape	False	False	False	None
login-user	False	False	True	None
login-user-popup	False	False	True	None
multi-layouts	False	False	False	None
multi-orderings	False	False	False	None
navigate-tree	False	False	False	None
read-table	False	False	False	None
search-engine	False	False	True	None
simple-algebra	False	False	False	None
simple-arithmetic	False	False	False	None
social-media	False	False	False	Obs Task
social-media-all	False	False	True	None
social-media-some	False	False	True	None
terminal	False	True	False	None
text-transform	False	False	False	None
tic-tac-toe	True	False	False	Obs Task
unicode-test	False	False	False	None
use-autocomplete	False	True	False	None
use-spinner	False	False	False	Task

Table 9: Hyperparameters for Synapse (Zheng et al., 2023). **Raw Task Only** is specified with **Task as Reformation** flag, and **Reformulation** is specified with **Reformat Input** flag in the original imprementation.

D Ranking Base MiniWoB Tasks

To ensure the solvability of CompWoB to some extent and to identify the data-redundant tasks for finetuned LMAs, we estimate the brute-force task difficulty (Furuta et al., 2021) (Table 10). We compute the average success rate for each task across representative previous web automation agents, such as CC-Net (SL, SL+RL) (Humphreys et al., 2022), WGE (Liu et al., 2018), WebN-T5 (Gur et al., 2022), WebGUM (Furuta et al., 2023), HTML-T5 (Gur et al., 2023), RCI (Kim et al., 2023), AdaPlanner (Sun et al., 2023), Pix2Act (BC, RL) (Shaw et al., 2023), and Synapse (Zheng et al., 2023). Based on those proxy difficulty measures, we classify 65 tasks into three categories (Kim et al., 2023): `easy` (from 0.8 to 1.0), `medium` (from 0.6 to 0.8), and `hard` (from 0.0 to 0.6).

Category	Task	Task Difficulty
`easy` (0.8 - 1.0)	click-button	0.923
	click-button-sequence	0.954
	click-checkboxes	0.936
	click-checkboxes-transfer	0.862
	click-collapsible	0.878
	click-dialog	0.923
	click-link	0.949
	click-option	0.839
	click-tab	0.898
	click-test	1.000
	click-test-2	0.996
	click-widget	0.945
	email-inbox-forward-nl	0.844
	email-inbox-forward-nl-turk	0.804
	enter-password	0.896
	enter-text	0.922
	enter-text-dynamic	0.933
	focus-text	0.999
	focus-text-2	0.990
	grid-coordinate	0.920
	identify-shape	0.918
	login-user	0.881
	login-user-popup	0.818
	multi-layouts	0.832
	navigate-tree	0.825
	unicode-test	0.900
`medium` (0.6 - 0.8)	choose-date-easy	0.740
	click-checkboxes-large	0.784
	click-checkboxes-soft	0.754
	click-collapsible-2	0.693
	click-color	0.742
	click-dialog-2	0.780
	click-menu	0.607
	click-pie	0.769
	click-shape	0.664
	click-tab-2	0.736
	click-tab-2-hard	0.651
	copy-paste	0.610
	email-inbox	0.778
	email-inbox-nl-turk	0.779
	enter-date	0.714
	multi-orderings	0.793
	read-table	0.660
	search-engine	0.723
	simple-algebra	0.799
	simple-arithmetic	0.782
	social-media	0.733
	text-transform	0.737
	use-autocomplete	0.782
`hard` (0.0 - 0.6)	book-flight	0.510
	choose-date	0.331
	choose-date-medium	0.497
	choose-list	0.520
	click-scroll-list	0.401
	click-shades	0.501
	copy-paste-2	0.547
	count-shape	0.536
	enter-time	0.521
	find-word	0.590
	guess-number	0.363
	social-media-all	0.432
	social-media-some	0.532
	terminal	0.592
	tic-tac-toe	0.598
	use-spinner	0.457

Table 10: Brute-force task complexity and difficulty classification of MiniWoB. We split 65 tasks into the three category based on the task complexity: `easy` (0.8 - 1.0), `medium` (0.6 - 0.8), and `hard` (0.0 - 0.6).

E Details of MiniWoB Dataset

To resolve the data-imbalance problem, we first run Synapse (Zheng et al., 2023) on MiniWoB and collect 77K additional demonstrations across 16 tasks on top of 347K demonstrations (Furuta et al., 2023) to compensate for the lack of data in specific tasks (**Strategy A**). We use PaLM 2-L (Anil et al., 2023) as a backbone LLM for Synapse. We then reduce the number of demonstrations for the tasks the agents can solve to focus on more challenging tasks. Based on brute-force task complexity (Appendix D), we consider the following four strategies:

- Removing 50% episodes from top-10 `easy` tasks (**Strategy B**; -73K)
- Removing 80% episodes from top-10 `easy` tasks (**Strategy C**; -102K)
- Removing 50% episodes from `easy` tasks (**Strategy D**; -142K)
- Removing 80% episodes from top-15 `easy` tasks and removing 50% episodes from other 11 `easy` tasks (**Strategy E**; -183K)

Through the empirical evaluations (Section 4.4), we find that **Strategy D** realizes a well-balanced dataset to improve the performance. Except for these dataset mixture, we follow Gur et al. (2023) for other training hyper-parameters of HTML-T5++.

Task	**Original** (347K)	**Strat. A** (424K)	**Strat. B** (351K)	**Strat. C** (322K)	**Strat. D** (282K)	**Strat. E** (241K)
book-flight	2.88%	2.49%	2.84%	3.11%	3.54%	4.15%
choose-date	0.11%	1.25%	1.42%	1.55%	1.77%	2.07%
choose-date-easy	0.97%	0.84%	0.95%	1.04%	1.19%	1.39%
choose-date-medium	0.64%	0.55%	0.63%	0.69%	0.79%	0.92%
choose-list	0.54%	1.24%	1.42%	1.55%	1.77%	2.07%
click-button	2.82%	2.44%	1.39%	0.61%	1.73%	0.81%
click-button-sequence	2.88%	2.49%	1.42%	0.62%	1.77%	0.83%
click-checkboxes	2.81%	2.43%	1.36%	0.55%	1.69%	0.73%
click-checkboxes-large	0.57%	2.15%	2.46%	2.49%	3.06%	3.58%
click-checkboxes-soft	2.66%	2.30%	2.63%	2.87%	3.27%	3.83%
click-checkboxes-transfer	2.88%	2.49%	2.85%	3.11%	1.77%	2.07%
click-collapsible	1.71%	1.48%	1.69%	1.85%	1.06%	1.24%
click-collapsible-2	0.63%	0.55%	0.63%	0.68%	0.78%	0.91%
click-color	0.74%	1.23%	1.40%	1.53%	1.74%	2.04%
click-dialog	2.88%	2.49%	2.85%	3.11%	1.77%	0.83%
click-dialog-2	0.95%	1.36%	1.55%	1.70%	1.93%	2.26%
click-link	2.87%	2.48%	1.42%	0.62%	1.76%	0.83%
click-menu	0.93%	1.24%	1.42%	1.55%	1.77%	2.07%
click-option	2.88%	2.49%	2.84%	3.11%	1.77%	2.07%
click-pie	1.07%	2.32%	2.65%	2.90%	3.30%	3.86%
click-scroll-list	0.00%	1.01%	1.16%	1.26%	1.44%	1.69%
click-shades	0.00%	1.25%	1.42%	1.55%	1.77%	2.07%
click-shape	1.76%	1.80%	2.05%	2.24%	2.56%	2.99%
click-tab	2.88%	2.49%	2.84%	3.10%	1.77%	2.07%
click-tab-2	0.53%	0.46%	0.52%	0.57%	0.65%	0.76%
click-tab-2-hard	0.45%	1.67%	1.91%	2.09%	2.38%	2.78%
click-test	2.88%	2.49%	1.42%	0.62%	1.77%	0.83%
click-test-2	2.88%	2.49%	1.42%	0.62%	1.77%	0.83%
click-widget	2.87%	2.48%	1.41%	0.62%	1.76%	0.82%
count-shape	1.69%	1.10%	1.26%	1.37%	1.56%	1.83%
email-inbox	1.49%	1.29%	1.47%	1.60%	1.83%	2.14%
email-inbox-forward-nl	2.88%	2.49%	2.84%	3.11%	1.77%	2.07%
email-inbox-forward-nl-turk	1.41%	1.22%	1.39%	1.52%	0.86%	1.01%
email-inbox-nl-turk	1.25%	1.08%	1.24%	1.35%	1.54%	1.80%
enter-date	2.88%	2.49%	2.85%	3.11%	3.54%	4.15%
enter-password	2.88%	2.49%	2.84%	3.10%	1.77%	2.07%
enter-text	2.88%	2.49%	2.85%	3.11%	1.77%	0.83%
enter-text-dynamic	2.88%	2.49%	1.42%	0.62%	1.77%	0.83%
enter-time	0.00%	1.08%	1.23%	1.35%	1.54%	1.80%
focus-text	2.88%	2.49%	1.42%	0.62%	1.77%	0.83%
focus-text-2	2.88%	2.49%	1.42%	0.62%	1.77%	0.83%
grid-coordinate	2.41%	2.08%	2.38%	2.60%	1.41%	0.55%
guess-number	0.29%	1.25%	1.42%	1.55%	1.77%	2.07%
identify-shape	2.60%	2.24%	2.56%	2.80%	1.60%	0.75%
login-user	2.82%	2.44%	2.79%	3.05%	1.73%	2.03%
login-user-popup	2.82%	2.44%	2.78%	3.04%	1.73%	2.03%
multi-layouts	2.88%	2.49%	2.85%	3.11%	1.77%	2.07%
multi-orderings	2.88%	2.49%	2.85%	3.11%	3.54%	4.15%
navigate-tree	2.84%	2.46%	2.81%	3.07%	1.73%	2.02%
search-engine	2.56%	2.21%	2.52%	2.76%	3.14%	3.68%
social-media	0.76&	0.66%	0.75%	0.82%	0.93%	1.09%
social-media-all	0.03%	0.02%	0.03%	0.03%	0.03%	0.04%
social-media-some	0.09&	0.08%	0.09%	0.10%	0.11%	0.13%
tic-tac-toe	1.14%	1.43%	1.63%	1.78%	2.03%	2.37%
use-autocomplete	1.00%	0.86%	0.99%	1.08%	1.23%	1.44%
use-spinner	0.15%	1.19%	1.36%	1.48%	1.69%	1.98%

Table 11: Task ratio in rebalanced dataset for HTML-T5++.

F Task Complexity Analysis in Web Automation

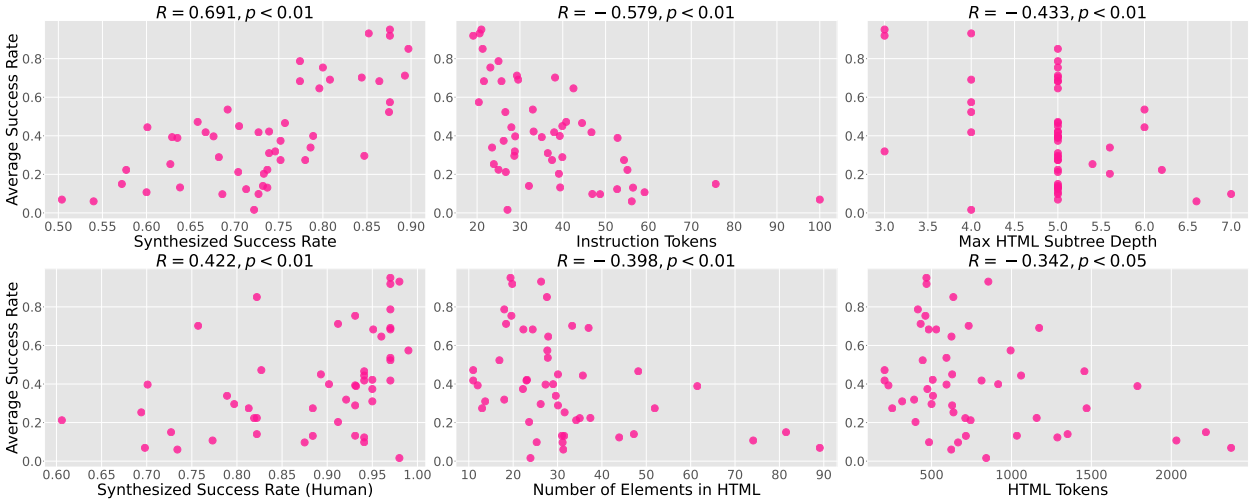

Figure 6: 2D-scatter plots between success rate averaged among LMAs (y-axis) and each statistic of compositional task (x-axis), such as success rate synthesized with a product of base task success rate, the number of instruction tokens, max depth of HTML subtrees, success rate synthesized with a product of base task human performances, the number of elements in HTML, and the number of HTML tokens. Synthesized success rate positively correlates with an average success rate ($R = 0.691$, statistically significant in paired t-test with $p < 0.01$), indicating that base task difficulty may determine compositional task difficulty. In addition, the number of instruction tokens ($R = -0.579$; $p < 0.01$) and the max depth of HTML subtrees ($R = -0.433$; $p < 0.01$) show negative correlations, which suggests the high complexity of observation and long instructions make the compositional tasks hard to resolve. In contrast, synthesized success rate from human performance, the number of HTML tokens, and elements in HTML just show relatively weaker correlations.

G Failure Examples with Advanced Language Models

We provide several failure episodes with advanced language models such as `gpt-4` and `text-davinci-003` in Table 12. The left columns have correct plans, and the right columns have failure plans. Qualitatively, LMAs based on advanced models manage to reduce the ratio of failure modes in `gpt-3.5-turbo` (Table 3), such as skipping necessary intermediate steps and hallucination of unnecessary actions. However, they tend to suffer from tiny errors more such as capitalization (RCI, AdaPlanner) or attributes in HTML (Synapse).

RCI (Kim et al., 2023)		**AdaPlanner** (Sun et al., 2023)		**Synapse** (Zheng et al., 2023)	
Click on the link "adipiscing", and then click on the "submit" button		Click on the link "Augue", and then click on a "button" widget		Select 0Qm9EUt, and then enter the username "cristin" and the password "M5" into the text fields and press login	
✔	✗	✔	✗	✔	✗
1. click //span[text()="adipiscing"] 2. click //button[text()="submit"]	1. click //span[text()="adipiscing"] 2. click //button[text()="Submit"]	1. click //span[text()="Augue"] 2. click //*[@data-type="button"]	1. click //span[text()="augue"] 2. click //*[@data-type="button"]	1. click //*[text()="0Qm9EUt"]/input 2. click //*[@id="username"] 3. type cristin 4. click //*[@id="password"] 5. type M5 6. click //*[@id="subbtn"]	1. click //*[@id="0Qm9EUt"] 2. click //*[@id="username"] 3. type cristin 4. click //*[@id="password"] 5. type M5 6. click //*[@id="subbtn"]

Table 12: Failure examples in CompWoB with advanced models (`gpt-4`, `text-davinci-003`).

H Per-Task Performance on MiniWoB and CompWoB

Task	**HTML-T5** (Gur et al., 2023)	**HTML-T5++** (Ours)
book-flight	0.99	0.99
choose-date	0.16	1.00
choose-date-easy	1.00	1.00
choose-date-medium	0.56	1.00
choose-list	0.22	1.00
click-button	1.00	1.00
click-button-sequence	1.00	1.00
click-checkboxes	1.00	1.00
click-checkboxes-large	0.90	0.97
click-checkboxes-soft	0.99	1.00
click-checkboxes-transfer	1.00	1.00
click-collapsible	1.00	1.00
click-collapsible-2	0.93	0.96
click-color	1.00	1.00
click-dialog	1.00	1.00
click-dialog-2	0.74	1.00
click-link	0.99	1.00
click-menu	0.37	0.96
click-option	1.00	1.00
click-pie	0.96	0.94
click-scroll-list	0.99	1.00
click-shades	0.00	1.00
click-shape	0.79	0.95
click-tab	1.00	1.00
click-tab-2	0.94	0.98
click-tab-2-hard	0.88	0.96
click-test	1.00	1.00
click-test-2	1.00	1.00
click-widget	1.00	1.00
count-shape	0.67	0.92
email-inbox	1.00	0.98
email-inbox-forward-nl	1.00	1.00
email-inbox-forward-nl-turk	1.00	1.00
email-inbox-nl-turk	0.99	1.00
enter-date	1.00	1.00
enter-password	1.00	1.00
enter-text	1.00	1.00
enter-text-dynamic	1.00	1.00
enter-time	1.00	1.00
focus-text	1.00	1.00
focus-text-2	1.00	1.00
grid-coordinate	1.00	1.00
guess-number	0.13	1.00
identify-shape	1.00	1.00
login-user	1.00	1.00
login-user-popup	1.00	1.00
multi-layouts	1.00	1.00
multi-orderings	1.00	1.00
navigate-tree	0.99	1.00
search-engine	0.93	0.96
social-media	0.99	1.00
social-media-all	0.31	0.31
social-media-some	0.89	0.85
tic-tac-toe	0.57	0.55
use-autocomplete	0.97	0.99
use-spinner	0.07	0.06
Average	**0.856**	**0.952**

Table 13: Per-task average success rate on 56 tasks from MiniWoB++. We refer to Gur et al. (2023) for the baseline performance.

Task	WebGUM	HTML-T5	HTML-T5++	RCI (zero)	RCI (first)	RCI (second)	RCI (comb)	RCI (gpt-4)	Synapse (first)	Synapse (second)	Synapse (best)	Synapse (f;gpt-4)	Synapse (s;gpt-4)	Synapse (b;gpt-4)	AdaPlanner	AdaPlanner (davinci)
click-button_click-checkboxes	0.210	0.700	0.790	0.370	0.710	0.730	0.800	0.880	0.810	0.840	0.840	0.980	0.840	0.980	0.060	0.890
click-button_click-checkboxes-transfer	0.010	0.600	0.750	0.270	0.790	0.790	0.810	1.000	0.540	0.740	0.740	0.960	0.740	0.960	0.000	0.940
click-button_click-dialog	0.870	0.720	0.970	0.740	0.830	0.820	0.940	1.000	1.000	0.650	1.000	1.000	0.660	1.000	0.900	0.980
click-button_click-link	0.810	0.860	0.860	0.870	0.830	0.870	0.920	0.920	0.990	0.000	0.990	1.000	0.000	1.000	0.940	0.970
click-button_click-option	0.600	0.840	0.840	0.580	0.650	0.370	0.710	0.420	0.920	0.870	0.920	0.960	0.880	0.960	0.000	0.860
click-button-sequence_click-checkboxes	0.490	0.960	0.900	0.440	0.520	0.820	0.670	0.820	0.710	0.890	0.890	0.940	0.840	0.940	0.060	0.680
click-button-sequence_click-option	0.940	1.000	1.000	0.540	0.830	0.620	0.760	0.800	0.900	0.000	0.900	0.900	0.000	0.900	0.000	0.490
click-button-sequence_login-user-popup	0.950	0.390	0.310	0.000	0.000	0.000	0.000	0.100	0.000	0.000	0.000	0.300	0.000	0.300	0.420	0.000
click-link_click-button	0.900	0.980	0.970	0.790	0.830	0.970	0.980	0.980	0.010	1.000	1.000	0.000	1.000	1.000	0.950	0.800
click-link_click-dialog	0.930	0.970	1.000	0.200	0.200	0.460	0.600	0.640	0.000	0.000	0.000	0.000	0.000	0.000	0.540	0.490
click-link_click-widget	0.940	0.990	0.990	0.550	0.520	0.820	0.900	0.780	0.370	0.000	0.370	0.820	0.000	0.820	0.980	0.890
click-link_enter-text	0.130	0.900	1.000	0.090	0.160	0.200	0.060	0.840	0.000	0.000	0.000	0.000	0.000	0.000	0.860	0.920
click-option_enter-text	1.000	0.970	1.000	0.380	0.130	0.410	0.500	0.880	0.000	0.000	0.000	1.000	0.000	1.000	0.770	0.960
click-option_login-user	0.980	1.000	0.960	0.010	0.000	0.000	0.000	0.860	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
click-option_navigate-tree	0.020	0.480	0.370	0.550	0.470	0.500	0.720	0.820	0.000	0.240	0.240	0.560	0.960	0.960	0.620	0.590
click-widget_enter-password	0.370	0.740	0.810	0.000	0.000	0.000	0.000	0.740	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
click-widget_multi-layouts	0.430	0.400	0.220	0.000	0.000	0.000	0.000	0.240	0.000	0.760	0.760	0.000	1.000	1.000	0.000	0.000
enter-password_click-option	1.000	0.690	0.900	0.000	0.020	0.000	0.000	0.780	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
login-user_navigate-tree	0.000	0.040	0.140	0.020	0.010	0.000	0.000	0.700	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
multi-layouts_login-user	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.100	0.720	0.000	0.720	1.000	0.000	1.000	0.010	0.000
Average (two-way)	**0.579**	**0.712**	**0.739**	**0.320**	**0.375**	**0.419**	**0.469**	**0.715**	**0.349**	**0.300**	**0.469**	**0.521**	**0.346**	**0.641**	**0.356**	**0.523**
click-button_click-option_login-user	0.140	0.520	0.740	0.000	0.340	0.000	0.000	0.120	0.000	0.000	0.000	0.940	0.280	0.940	0.140	0.000
click-button-sequence_click-option_login-user	0.300	0.200	0.770	0.520	0.740	0.680	0.840	0.560	0.720	0.000	0.720	0.660	0.000	0.660	0.000	0.000
click-checkboxes_click-widget_click-button-sequence	0.200	0.580	0.780	0.400	0.660	0.660	0.820	1.000	0.820	0.000	0.820	0.820	0.180	0.820	0.580	0.720
click-checkboxes-transfer_click-button-sequence_enter-password	0.000	0.010	0.010	0.000	0.000	0.000	0.000	0.660	0.000	0.000	0.000	0.000	0.500	0.500	0.000	0.000
click-checkboxes-transfer_enter-password_click-dialog	0.000	0.010	0.000	0.000	0.000	0.000	0.000	0.280	0.000	0.420	0.420	0.000	0.400	0.400	0.000	0.000
click-dialog_click-button-sequence_enter-password	0.890	1.000	1.000	0.000	0.000	0.000	0.000	0.700	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
click-dialog_click-checkboxes-transfer_click-widget	0.140	0.690	0.560	0.140	0.540	0.340	0.400	0.680	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
click-link_click-button_click-dialog	0.730	0.940	0.980	0.240	0.270	0.480	0.610	0.600	0.000	0.600	0.600	0.000	0.620	0.620	0.560	0.580
click-widget_click-option_click-dialog	0.010	0.120	0.410	0.120	0.000	0.020	0.300	0.400	0.000	0.000	0.000	0.020	0.000	0.020	0.000	0.000
enter-password_click-checkboxes_login-user-popup	0.110	0.000	0.060	0.000	0.000	0.000	0.000	0.700	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Average (three-way)	**0.252**	**0.407**	**0.531**	**0.142**	**0.255**	**0.218**	**0.297**	**0.570**	**0.154**	**0.102**	**0.256**	**0.244**	**0.198**	**0.396**	**0.128**	**0.130**
click-button-sequence_click-widget_click-link_click-button_click-checkboxes_click-option_click-dialog	0.000	0.230	0.320	0.160	0.020	0.380	0.080	0.720	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
click-button-sequence_click-widget_click-link_click-button_click-checkboxes_click-option_click-dialog_login-user	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.620	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
click-link_click-button_click-checkboxes_click-dialog	0.280	0.630	0.810	0.150	0.230	0.320	0.370	0.540	0.000	0.310	0.310	0.000	0.600	0.600	0.050	0.600
click-link_click-button_click-checkboxes_click-option_click-dialog	0.180	0.430	0.780	0.200	0.180	0.370	0.360	0.560	0.000	0.060	0.060	0.000	0.620	0.620	0.020	0.490
click-widget_click-link_click-button_click-checkboxes_click-option_click-dialog	0.000	0.040	0.140	0.140	0.360	0.100	0.300	0.480	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Average (n-way)	**0.092**	**0.266**	**0.410**	**0.130**	**0.158**	**0.234**	**0.222**	**0.584**	**0.000**	**0.074**	**0.074**	**0.000**	**0.244**	**0.244**	**0.014**	**0.218**
click-checkboxes-transfer_multi-layouts_email-inbox-forward-nl-transition	0.240	0.460	0.650	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.220	0.440
click-option_login-user-transition	0.990	1.000	0.800	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
click-option_multi-layouts_click-widget_login-user-popup-transition	0.190	0.220	0.130	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
login-user_navigate-tree-transition	0.960	1.000	1.000	0.000	0.000	0.000	0.000	0.460	0.000	0.000	0.000	0.340	0.000	0.340	0.000	0.000
login-user-popup_email-inbox-forward-nl-turk-transition	1.000	0.960	0.910	0.000	0.000	0.200	0.000	0.120	0.000	0.000	0.000	0.000	0.000	0.000	0.700	0.560
Average (transition)	**0.676**	**0.728**	**0.698**	**0.000**	**0.000**	**0.040**	**0.000**	**0.116**	**0.000**	**0.000**	**0.000**	**0.068**	**0.000**	**0.068**	**0.184**	**0.200**
click-button_click-tab-2-hard	0.240	0.550	0.510	0.080	0.120	0.260	0.340	0.580	0.280	0.040	0.280	0.960	0.320	0.960	0.260	0.280
click-button-sequence_use-autocomplete	0.410	0.220	0.840	0.200	0.060	0.640	0.160	0.820	0.000	0.000	0.000	0.420	0.000	0.420	0.000	0.000
click-checkboxes-soft_enter-password	0.990	0.850	0.910	0.000	0.000	0.000	0.000	0.820	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
click-checkboxes-soft_multi-layouts	0.930	0.240	0.170	0.000	0.000	0.000	0.000	0.120	0.000	0.300	0.000	0.000	0.820	0.820	0.000	0.000
click-dialog_search-engine	0.930	0.970	0.940	0.100	0.020	0.000	0.000	0.920	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
click-dialog-2_click-widget	0.270	0.460	0.690	0.000	0.000	0.000	0.040	0.020	0.100	0.000	0.100	0.180	0.000	0.180	0.160	0.100
click-dialog-2_login-user-popup	0.330	0.440	0.360	0.000	0.000	0.000	0.000	0.060	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
click-widget_click-checkboxes-soft	0.120	0.290	0.460	0.080	0.100	0.060	0.200	0.600	0.000	0.000	0.000	0.000	0.000	0.000	0.100	0.140
enter-date_login-user	0.990	1.000	0.230	0.000	0.020	0.000	0.060	0.500	0.000	0.000	0.000	0.000	0.000	0.000	0.380	0.380
use-autocomplete_click-dialog	0.000	0.000	0.000	0.000	0.080	0.000	0.080	0.060	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Average (two-way, easy-medium)	**0.521**	**0.502**	**0.511**	**0.046**	**0.040**	**0.096**	**0.088**	**0.450**	**0.038**	**0.034**	**0.038**	**0.156**	**0.114**	**0.238**	**0.090**	**0.090**
Average (total)	**0.463**	**0.566**	**0.615**	**0.179**	**0.225**	**0.258**	**0.287**	**0.560**	**0.178**	**0.154**	**0.254**	**0.295**	**0.225**	**0.414**	**0.206**	**0.295**

Table 14: Per-task success rate on CompWoB.

Task	WebGUM	HTML-T5	HTML-T5++	RCI (zero)	RCI (first)	RCI (second)	RCI (comb)	RCI (gpt-4)	Synapse (first)	Synapse (second)	Synapse (best)	AdaPlanner
click-button_click-checkboxes	0.180	0.340	0.630	0.420	0.310	0.420	0.380	0.520	0.670	0.800	0.800	0.340
click-button_click-checkboxes-transfer	0.060	0.260	0.630	0.150	0.140	0.230	0.200	0.320	0.360	0.930	0.930	0.090
click-button_click-dialog	0.300	0.020	0.030	0.850	0.790	0.790	0.810	0.840	0.170	0.200	0.200	0.000
click-button_click-link	0.010	0.050	0.040	0.430	0.500	0.430	0.460	0.560	0.870	0.000	0.870	0.970
click-button_click-option	0.230	0.420	0.550	0.350	0.270	0.400	0.410	0.660	0.790	0.860	0.860	0.440
click-button-sequence_click-checkboxes	0.000	0.140	0.130	0.270	0.260	0.390	0.280	0.820	0.480	0.870	0.870	0.700
click-button-sequence_click-option	0.000	0.020	0.110	0.360	0.420	0.320	0.490	0.920	0.350	0.000	0.350	0.550
click-button-sequence_login-user-popup	0.000	0.720	0.020	0.000	0.000	0.000	0.000	0.160	0.000	0.000	0.000	0.280
click-link_click-button	0.910	0.750	0.580	0.840	0.760	0.800	0.830	0.920	0.000	0.670	0.670	0.950
click-link_click-dialog	0.450	0.050	0.230	0.330	0.430	0.470	0.880	0.660	0.000	0.000	0.000	0.200
click-link_click-widget	0.140	0.750	0.740	0.550	0.580	0.480	0.530	0.780	0.640	0.000	0.640	0.920
click-link_enter-text	0.000	0.950	0.910	0.240	0.280	0.300	0.310	0.480	0.000	0.000	0.000	0.970
click-option_enter-text	1.000	0.660	0.750	0.570	0.650	0.650	0.640	0.800	0.000	0.000	0.000	0.590
click-option_login-user	0.980	0.220	0.150	0.010	0.000	0.010	0.000	0.620	0.000	0.000	0.000	0.020
click-option_navigate-tree	0.000	0.060	0.050	0.450	0.390	0.500	0.490	0.440	0.050	0.590	0.590	0.880
click-widget_enter-password	0.280	0.470	0.530	0.000	0.000	0.000	0.000	0.300	0.000	0.000	0.000	0.000
click-widget_multi-layouts	0.260	0.310	0.220	0.000	0.000	0.000	0.000	0.440	0.000	0.700	0.700	0.020
enter-password_click-option	0.960	0.160	0.900	0.000	0.000	0.000	0.000	0.920	0.000	0.000	0.000	0.000
login-user_navigate-tree	0.000	0.030	0.080	0.010	0.000	0.000	0.000	0.280	0.000	0.000	0.000	0.000
multi-layouts_login-user	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.020	0.210	0.000	0.210	0.000
Average (two-way)	**0.288**	**0.319**	**0.364**	**0.292**	**0.289**	**0.310**	**0.336**	**0.573**	**0.230**	**0.281**	**0.385**	**0.396**
click-button_click-option_login-user	0.250	0.140	0.220	0.020	0.000	0.000	0.000	0.200	0.000	0.000	0.000	0.220
click-button-sequence_click-option_login-user	0.000	0.000	0.220	0.420	0.800	0.440	0.740	0.000	0.000	0.000	0.000	0.320
click-checkboxes_click-widget_click-button-sequence	0.000	0.270	0.100	0.060	0.140	0.140	0.080	0.820	0.000	0.000	0.000	0.000
click-checkboxes-transfer_click-button-sequence_enter-password	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.340	0.000	0.000	0.000	0.000
click-checkboxes-transfer_enter-password_click-dialog	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
click-dialog_click-button-sequence_enter-password	1.000	0.690	0.040	0.000	0.000	0.000	0.000	0.680	0.000	0.000	0.000	0.000
click-dialog_click-checkboxes-transfer_click-widget	0.080	0.520	0.460	0.140	0.340	0.000	0.360	0.580	0.000	0.000	0.000	0.000
click-link_click-button_click-dialog	0.040	0.050	0.130	0.180	0.320	0.320	0.310	0.360	0.000	0.000	0.000	0.000
click-widget_click-option_click-dialog	0.000	0.000	0.000	0.120	0.220	0.160	0.140	0.120	0.000	0.000	0.000	0.000
enter-password_click-checkboxes_login-user-popup	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.660	0.000	0.000	0.000	0.000
Average (three-way)	**0.137**	**0.167**	**0.117**	**0.094**	**0.182**	**0.106**	**0.163**	**0.376**	**0.000**	**0.000**	**0.000**	**0.054**
click-button-sequence_click-widget_click-link_click-button_click-checkboxes_click-option_click-dialog	0.000	0.070	0.090	0.140	0.100	0.340	0.080	0.600	0.000	0.000	0.000	0.000
click-button-sequence_click-widget_click-link_click-button_click-checkboxes_click-option_click-dialog_login-user	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.180	0.000	0.000	0.000	0.000
click-link_click-button_click-checkboxes_click-dialog	0.000	0.090	0.240	0.220	0.220	0.120	0.200	0.320	0.000	0.000	0.000	0.000
click-link_click-button_click-checkboxes_click-option_click-dialog	0.000	0.060	0.260	0.170	0.120	0.140	0.170	0.380	0.000	0.000	0.000	0.000
click-widget_click-link_click-button_click-checkboxes_click-option_click-dialog	0.000	0.000	0.030	0.060	0.060	0.040	0.180	0.080	0.000	0.000	0.000	0.000
Average (n-way)	**0.000**	**0.044**	**0.124**	**0.118**	**0.100**	**0.128**	**0.126**	**0.312**	**0.000**	**0.000**	**0.000**	**0.000**
click-checkboxes-transfer_multi-layouts_email-inbox-forward-nl-transition	0.170	0.360	0.780	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.140
click-option_login-user-transition	0.990	0.800	0.650	0.000	0.000	0.000	0.000	0.080	0.000	0.000	0.000	0.000
click-option_multi-layouts_click-widget_login-user-popup-transition	0.070	0.100	0.030	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
login-user_navigate-tree-transition	1.000	1.000	0.990	0.000	0.000	0.000	0.000	0.080	0.000	0.000	0.000	0.000
login-user-popup_email-inbox-forward-nl-turk-transition	0.650	0.770	0.820	0.000	0.000	0.220	0.100	0.300	0.000	0.000	0.000	0.580
Average (transition)	**0.576**	**0.606**	**0.654**	**0.000**	**0.000**	**0.044**	**0.020**	**0.092**	**0.000**	**0.000**	**0.000**	**0.144**
click-button_click-tab-2-hard	0.050	0.470	0.490	0.140	0.120	0.180	0.400	0.640	0.300	0.020	0.300	0.160
click-button-sequence_use-autocomplete	0.000	0.080	0.930	0.260	0.080	0.640	0.040	0.860	0.000	0.000	0.000	0.000
click-checkboxes-soft_enter-password	0.950	0.490	0.760	0.000	0.000	0.000	0.000	0.740	0.000	0.000	0.000	0.000
click-checkboxes-soft_multi-layouts	0.940	0.390	0.260	0.000	0.000	0.000	0.000	0.200	0.000	0.420	0.000	0.000
click-dialog_search-engine	0.950	0.820	0.550	0.000	0.000	0.000	0.020	0.980	0.000	0.000	0.000	0.000
click-dialog-2_click-widget	0.210	0.370	0.260	0.000	0.000	0.020	0.000	0.020	0.000	0.000	0.000	0.080
click-dialog-2_login-user-popup	0.380	0.360	0.330	0.000	0.000	0.000	0.000	0.020	0.000	0.000	0.000	0.000
click-widget_click-checkboxes-soft	0.090	0.240	0.290	0.080	0.080	0.120	0.080	0.520	0.000	0.000	0.000	0.000
enter-date_login-user	0.950	0.350	0.960	0.000	0.040	0.000	0.000	0.400	0.000	0.000	0.000	0.040
use-autocomplete_click-dialog	0.000	0.000	0.000	0.000	0.080	0.000	0.000	0.120	0.000	0.000	0.000	0.000
Average (two-way, easy-medium)	**0.452**	**0.357**	**0.483**	**0.048**	**0.040**	**0.096**	**0.054**	**0.450**	**0.030**	**0.044**	**0.030**	**0.028**
Average (total)	**0.291**	**0.297**	**0.343**	**0.157**	**0.170**	**0.181**	**0.192**	**0.435**	**0.098**	**0.121**	**0.160**	**0.189**

Table 15: Per-task success rate on CompWoB with reverse-order instructions.

I Prompted Language Model Agents with Oracle Exemplars

In this paper, we assume, as a realistic and important constraint, that: (1) **prompted or finetuned LMAs are deployed as a service in the real world**; (2) **users are only allowed to interact with the agents via providing task instruction**, and (3) **not allowed to intervene the backend system or prompts**. Providing exemplars or preparing finetuning demonstrations for every compositional problem is infeasible given the huge space of web automation problems. Moreover, it significantly hinders the user experience that the agents would be good at some specific format of instructions and that their violation of those affects the performance even if semantically the same.

As a sanity check of the environments and baseline agents, we here demonstrate that prompted LMAs can change their behaviors adaptively by modifying their prompts and demonstrations. We evaluate the performance of RCI with `gpt-4` and oracle exemplars on 20 two-way tasks in CompWoB. We provide two demonstrations per task in the prompts. Figure 7 shows that RCI with `gpt-4` and oracle exemplars achieves 82.9% success rate, which is the best among the baselines, such as HTML-T5++ (73.9%), RCI with `gpt-3.5-turbo` and oracle exemplars (56.8%), RCI with combination exemplars (`gpt-3.5-turbo`: 46.9%, `gpt-4`: 71.5%). See Table 14 for other baselines. This ensures that the tasks are feasible, and that if a prompt includes how to perform on compositional tasks, the performance gets better. In contrast, while oracle demonstrations help improve performance, they could not fully resolve the issues from reverse-order instructions.

Through the experiments in Section 6, we have intended to shed light on their zero-shot generalization capabilities of dealing with unknown sequential-task compositions. For reliable and robust web automation agents, we might need to leverage both prompting and finetuning approaches at the different development stages. For instance, prompted LMAs might work well as adaptive automated data collectors to the novel domains and finetuned LMAs as deployed agents for service.

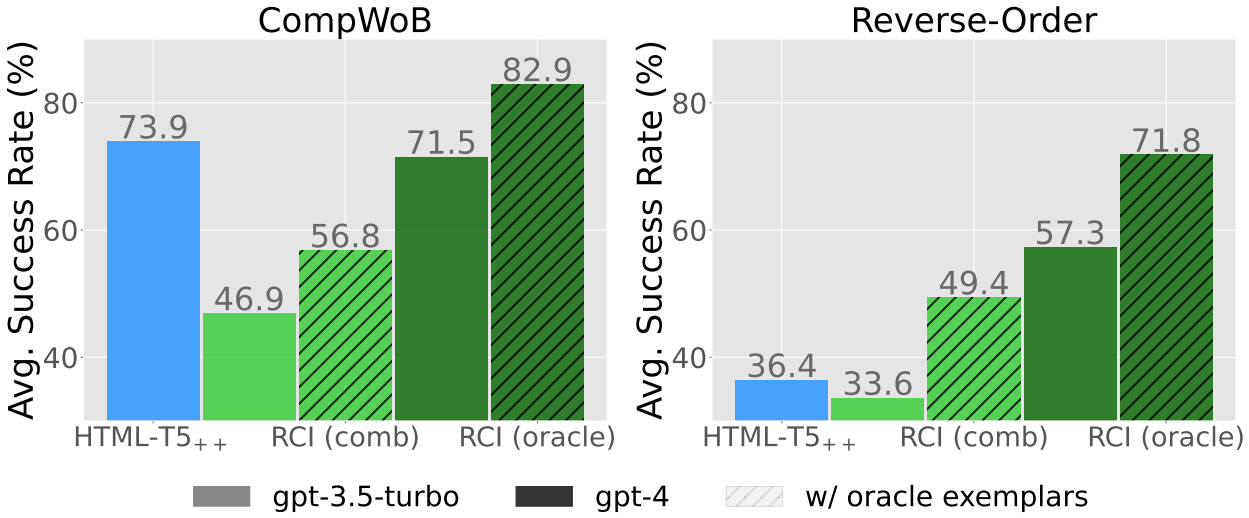

Figure 7: Average success rate of LMAs in 20 two-way tasks from CompWoB. RCI with `gpt-4` achieves the best performance when the oracle exemplars are provided (82.9%) in the prompt. While oracle demonstrations help improve performance, they could not fully resolve the issues from reverse-order instructions (for instance, 82.9% → 71.8% in RCI with `gpt-4`).

J Task Complexity Analysis with Language Model Agents

Following the recent work in deep reinforcement learning literature (Furuta et al., 2021), we measure average performance as a proxy of oracle task solvability. If many kinds of agents perform poorly, such tasks are regarded as challenging. This can shed light on"task" or "environment" themselves, rather than "language model agents" or "prompting methods", while such an analysis has been overlooked so far. Additionally, while the trend with average might be the same, distributional characteristics for each language model agent could be different. We extend our analysis by reporting the individual performances of each agent in Figure 8. Our results still indicate that while all the language model agents (HTML-T5++, RCI, AdaPlanner, Synapse) often show negative correlations between the success rate and instruction tokens or max subtree depth with statistical significance, the trends for other statistics may differ among the methods.

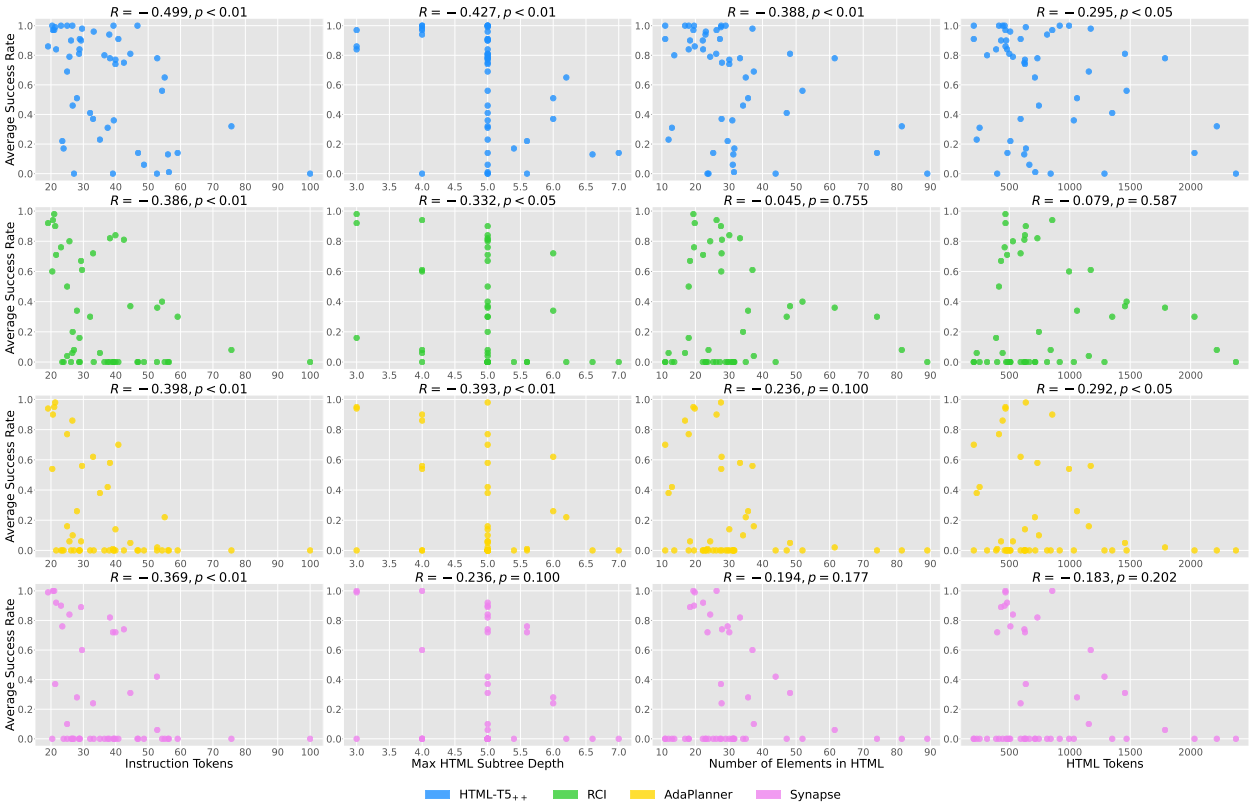

Figure 8: 2D-scatter plots between the success rate for each LMA (y-axis) and each statistic of compositional task (x-axis), such as the number of instruction tokens, max depth of HTML subtrees, the number of elements in HTML, and the number of HTML tokens. The results imply that while all the language model agents (HTML-T5++, RCI, AdaPlanner, Synapse) show negative correlations between the success rate and instruction tokens with statistical significance, the trends for other statistics may differ among the methods.

K Finetuning May Generally Outperform Prompting in Compositional Tasks

In this section, we examine the capability of finetuned LLMs and prompted LLMs for compositional generalization in natural language reasoning tasks, rather than agentic tasks. We employ Last Letter Concatenation (Wei et al., 2022; Zhou et al., 2023a) as a benchmark, where if LLMs take *"hans, structure"* as a question, LLMs are asked to answer it as *The last letter of "hans" is "s". The last letter of "structure" is "e". Considering "s", "e" leads to "se". So, "hans, structure" outputs "se"*. This reasoning problem exhibits a compositional nature as we increase the number of words provided as a question. For finetuned LLMs, we adopt 11B-parameter Flan-T5-XXL (Chung et al., 2022; Raffel et al., 2020) as a base model, and prepare 2000 examples per n-letter for a training dataset. For prompted LLMs, we employ Flan-PaLM-8B, 62B, and 540B (Chung et al., 2022; Chowdhery et al., 2022), and prepare randomized 4-shot CoT prompts from the same training dataset. We randomly use one of them during inference. Furthermore, we use up to 8 letters as a training dataset and few-shot exemplars and also use 9-12 letters as a hold-out test dataset to investigate the compositional generalization in natural language reasoning tasks.

Table 16 shows that finetuned LLM (Flan-T5-XXL) almost always outperforms prompted LLM (Flan-U-PaLM-540B) on average (76% v.s. 44% in an exact match) and also in both in-distribution (2-8 letters) and out-of-distribution (9-12 letters) settings. This implies that finetuned LLMs may generally outperform prompted LLMs in compositional tasks, which can also lead to the better performance of finetuned LMAs than prompted LMAs in sequential web automation tasks, as shown in the main text.

	# of Letters											
Models	2	3	4	5	6	7	8	9	10	11	12	Ave.
`Flan-PaLM-8B` (Prompted)	0.10	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01
`Flan-PaLM-62B` (Prompted)	0.48	0.16	0.05	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.07
`Flan-U-PaLM-540B` (Prompted)	**1.00**	**1.00**	0.75	0.76	0.57	0.41	0.16	0.08	0.06	0.00	0.01	0.44
`Flan-T5-XXL` (11B; Finetuned)	**1.00**	0.98	**0.99**	**0.98**	**0.98**	**0.96**	**0.96**	**0.83**	**0.47**	**0.20**	**0.05**	**0.76**

Table 16: The performance of finetuning (Flan-T5-XXL (Chung et al., 2022; Raffel et al., 2020), 11B parameters) and prompting (Flan-PaLM-8B/62B/540B (Chung et al., 2022; Chowdhery et al., 2022)) in Last Letter Concatenation (Wei et al., 2022; Zhou et al., 2023a). We measure the performance with an exact match. We use up to 8 letters as a training dataset and few-shot exemplars and also use 9-12 letters as a hold-out test dataset to investigate the compositional generalization in natural language reasoning tasks. The results show that finetuned LLM (Flan-T5-XXL) almost always outperforms prompted LLM (Flan-U-PaLM-540B) in both in-distribution (2-8 letters) and out-of-distribution (9-12 letters) settings.