Solving Real-World Tasks with AI Agents

Shuyan Zhou

CMU-LTI-24-014

July 25, 2024

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15123

*Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.*

Keywords: artificial intelligence, autonomous agents, digital automation, web navigation, code generation

To my family, for their unwavering support and unconditional love.

Abstract

For years, my dream has been to create autonomous AI agents to handle tedious procedural tasks (*e.g.*, arranging conference travel), freeing me to focus on creative endeavors. Modern AI models, especially large language models (LLMs) like ChatGPT, have brought us closer to this goal. But has my dream already come true? This thesis spans AI agent research from 2020 to 2024, acknowledging LLMs as a crucial yet early step in the broader AI agent applications. While LLMs show promise in well-defined tasks (*e.g.*, drafting emails), they struggle with procedural tasks requiring agents to *comprehend* and *apply how-to knowledge* during *dynamic interactions.* Current LLMs are inconsistent in complex procedural tasks. This thesis aims to *create AI agents to perform procedural tasks with accuracy, robustness, and trust in an ever-evolving environment* and is centered around three key pillars.

First, we **study the evaluation of AI agents to systematically understand agent behavior**. There was a lack of benchmarks that mimic real-world complexity, emulate diverse and complex human tasks, and support dynamic interactions to perform systematic evaluations. This led to evaluations that are only partially representative of real-world scenarios. We create a comprehensive benchmark on performing interactive web-based tasks (*e.g.*, book a hotel room near Pittsburgh airport online) that meets these criteria and developed more robust evaluation metrics. Our works reveal the deficiencies of LLM-powered agents in realistic interactive tasks and offer an accessible environment to advance the field.

Second, we **augment the expressiveness of AI agents with a more versatile "language" for agents**. Beyond knowledge, humans demonstrate versatility in procedural tasks: we break tasks into smaller sub-tasks, leverage past experiences, use tools, etc. Representing this versatility is challenging with unstructured text. We design a new formalization that equates task-solving to writing Python programs. The inherent expressiveness and structured nature of programs enable AI agents to more accurately and explicitly represent complex processes (*e.g.*, planning sub-tasks → composing nested functions, recalling memory → reusing functions). This new formalization enhances LLMs in reasoning about and performing procedural tasks, significantly improving task execution accuracy.

Finally, we **develop resources and design innovative methodologies to enable agents to adapt to unfamiliar tasks**. It is particularly challenging for LLMs to handle information that is not included or included sparsely in their training corpora. Hence, LLMs can benefit from access to external knowledge. We investigate how to make human-authored external knowledge (*e.g.*, manuals) comprehensible to AI agents by enriching such knowledge with detailed breakdowns of sub-tasks. We also propose new mechanisms of knowledge-augmented execution via retrieval, which allows the agents to perform challenging tasks by referring to external knowledge and via data synthesis. Both approaches circumvent the reliance on exact demonstrations.

Acknowledgments

In 2018, I felt somewhat daunted at the thought of pursuing a Ph.D. People had told me about many frightening aspects — stress, endless deadlines, a lack of work-life balance, etc. Looking back, I suppose I did experience some of these challenges, but perhaps only to a minimal extent. My Ph.D. journey was filled with joy and happiness.

The biggest shoutout goes to my advisor Graham Neubig. I first knew Graham during my senior year while watching his 11-747 Neural Networks for NLP course on YouTube. I have immense respect for him — he always provides comprehensive answers to any questions students ask, and I appreciated the way he delivered the course material. When I first met Graham in person at a multilingual reading group over the summer, I was excited, like a fan. I could not have imagined that I would one day work with him. I am grateful to Graham for giving me tremendous freedom and support in exploring the topics I am passionate about. One piece of work in this thesis was rejected five times; it was Graham's encouragement and support that kept me from giving up. Even after six years, I still feel there is so much to learn from Graham, from being an excellent researcher to being an amazing advisor. He is my lifelong role model.

I am honored to have had Daniel Fried, Tom Mitchell, and Yoav Artzi on my thesis committee. Their research has significantly influenced my work at various stages. Yoav's tutorial on "Semantic Parsing with CCG" sparked my fascination with semantic parsing, which eventually led to my interest in code generation and AI agents. Exploring Tom's early work inspired my study of computer agents. Our discussions have been particularly enriching, often leaving me pondering his thought-provoking questions for weeks. I am fortunate to have collaborated with Daniel on several projects; his insights have been invaluable to my progress. Their constructive feedback has been instrumental in shaping this thesis.

I have been incredibly lucky to work with Shruti Rijhwani, Pengcheng Yin, and Uri Alon during my junior years, They were inspiring role models, showing me how to craft compelling research problems, design and execute solid experiments, and effectively communicate the findings. Their enthusiasm and expertise made a lasting impact on my approach to research. I would like to thank Chin-Yew Lin, Jing Liu, Jin-Ge Yao, and Feng Nie for guiding me as I began my research journey at Microsoft Research Asia in 2017. Their support was invaluable in helping me establish a strong foundation in research. I also want to express my gratitude to Kaushik Chakrabarti and Alex Polozov for being my mentors during my internship at Microsoft, and X. Their mentorship sharpened my research skills.

The work in this thesis would not have been possible without my collaborators. I am deeply grateful to my co-leads: Frank Xu, who inspired me to become a full-stack researcher; Li Zhang, who demonstrated excellent project management skills; and Aman Madaan and Luyu Gao, who consistently brought fresh perspectives to research. I also extend my thanks to Yonatan Bisk, Chris Callison-Burch, Zhengbao Jiang, Zhiruo Wang, Pengfei Liu, Hao Zhu, Xuhui Zhou, JY Koh, and John Wieting for their insightful discussions across various projects. Additionally, I have had the privilege of mentoring several junior students, Grace Cuenca, Robert Lo, Abishek Sridhar, and Tianyue Ou. Seeing them build their projects from zero to one has been a key reason I am determined to pursue an academic career.

I am incredibly proud to be a member of NeuLab. Special thanks to Mengzhou Xia for the Insanity workouts we did together during the pandemic, which helped me lose 14 pounds; to Chunting Zhou for our countless discussions about research, life, and all sorts of random topics; to Patrick Fernandes and Vijay Viswanathan for hosting amazing house parties; to Zhiruo Wang and Lindia Tjuatja for being the best officemates; and to Amanda Bertsch, Emmy Liu, Simran Khanuja, Anjali Kantharuban, Lucio Dery, Junjie Hu, Junxian He, Hiroaki Hayashi, Cindy Wang, Kayo Yin, Alex Shypula, Zecong Hu, Junpei Zhou, Xinyu Wang for all the fun events. Also, thanks to Jia Yu and Zhili Feng for the wonderful meals we shared. To all my LTI and RI friends, my life would be completely different without you all.

I am deeply grateful to my family. My parents, Xiaoling Cai, and Daoqi Zhou, provide me with unconditional love and support. The guidance and encouragement from my sister, Hongwei Zhou, and brother-in-law, Feng Wu during my junior high school years opened my eyes to the incredible world beyond the small town where I grew up.

Finally, I would like to thank my partner, Xianyi Cheng. She has been my best friend, therapist, gym coach, and so much more since I was 19. From her, I have learnt to be more relaxed, to be present, to be kind. She makes me a better person.

Contents

Chapter 1

Introduction

Machines such as computers and robots can significantly improve the lives of human beings by performing tasks on behalf of humans. Natural language (NL) serves as a vital form of communication in human beings' lives and thus presents itself as a potential principle interaction protocol with machines. We study controlling AI agents through natural language. In this context, AI agents are expected to interact with an *external* environment and *autonomously* execute the specified tasks with minimal human intervention. [1]

1.1 Problem Scope

While many works consider AI agents more broadly as systems that perform tasks over multiple turns (e.g., multi-hop question answering), this thesis focuses on tasks that possess the following characteristics.

Let's denote an action space as $\mathcal{A}$, an action sequence as $\boldsymbol{a} = [a_1, a_2, ..., a_n]$ where $a_i \in \mathcal{A}$, an NL intent $\boldsymbol{u}$, the likelihood of an action sequence $\boldsymbol{a}$ given an intent $\boldsymbol{u}$ estimated from a reasonable parametric model as $p(\boldsymbol{a}|\boldsymbol{u})$. First, the task is associated with an external environment $\mathbb{E}$ with state space §. Any valid action in the action space can result in the state change in the environment defined by the environment dynamic function $f(s, a) \rightarrow s'$. Second, the action space is *enormous* and *semantically meaningful*. Tasks such as vision-language navigation (VLN) with a limited set of actions (*e.g.*, move forward/backward) or controlling the velocities and angles of the motors of a robot arm in a continuous space are not considered. Third, the length of an execution trace ($|\boldsymbol{a}|$) is substantial, which can roughly indicate the complexity of

[1] Throughout the thesis, we use "controlling AI agents" and "NL command and control" interchangeably.

the task. Therefore, tasks such as visual question answering and reference games, which usually consume a short action sequence, are not taken into account. Fourth, there is a diverse set of trajectories to achieve the desired objective in $\boldsymbol{u}$, thus exhibiting a high entropy of $\boldsymbol{a}$. For instance, one can purchase a flight ticket either online or by contacting the airline companies. Finally, for the task of interest, the extra information i that is helpful to the task completion is not specific to a particular instance; in other words, when $\log p(\boldsymbol{a}|\boldsymbol{u}, i) - \log p(\boldsymbol{a}|\boldsymbol{u}) > 0$, i can be seen as generic information pertinent to the task, instead of to the environment configuration. To illustrate, in VLN, having knowledge of the layout of a room would be helpful when navigating the room; however, this information is only germane to that exact room. On the contrary, when performing household tasks, the knowledge of "*preheating the oven*" is generic to many cooking tasks.

1.2 Challenges of Creating Autonomous AI Agents

One key challenge in creating AI agents capable of accomplishing arbitrary tasks is rooted in the *abstractness* and *ambiguity* of NL while the required underlying procedural knowledge is *concrete* and *broad*. A concise NL intent could practically express arbitrarily complex requests. For example, a short intent of "*prepare my conference travel*" may consist of complicated procedures such as "*book the flight tickets*", "*book the hotel*", "*registration*", and others. Each of these procedures, in turn, needs to be broken down into further finer-grained procedures. Therefore, there can be a huge *semantic gap* between the NL intent and the concrete executions.

Existing Benchmarks Inadequately Reflect the Complexities of Real World Although we repeatedly highlight the complexities of everyday procedures and note that the structures of procedural knowledge are ubiquitous in daily human life, most existing agent benchmarks do not feature such complex procedures. Although there can be hierarchies embedded in vision-language navigation tasks [6], game playing through reading documentation [205] or through NL communication [75, 157] and mobile phone navigation [97], the hierarchies are shallow at best, or the occasional complex ones are limited in their breadth. This is potentially due to their emphasis on research questions regarding object grounding, spatial relations, interactions, and others, and therefore, they focus less on procedure hierarchies. On the other hand, benchmarks that generate examples programmatically [59, 153] often lack realistic and diverse conditional branching in their procedures, as opposed to real-world scenarios, where the same outcome may be achieved through a variety of means depending on the prevailing conditions. Finally, for those tasks that are more procedurally complex [46, 75, 132], embodied experiments are non-trivial due to the challenge of setting up the embodied environments of the same complexity and a lack of automatic evaluation metrics for task completeness. Others [52, 87] are in gaming environments where domain-specific knowledge, rather than general commonsense knowledge, is required to perform the tasks.

Current AI Agents' "Language" is Restricted While the effective predictions of agents are the atomic actions within the action space, there are intermediate predictions—such as the rationale behind issuing an action—that link these atomic actions together. Currently, these intermediate predictions are predominantly expressed in the unstructured text (e.g., "*let's think step-by-step, <... the reason>, so the next action I will take is <...the predicted atomic action>*"). The use of unstructured text has become especially prevalent with the rise of large language models (LLMs) due to their proficiency in text generation. However, free-form text has limited expressiveness for representing the versatility required for solving procedural tasks. First, generating unstructured text confines the *reasoning* and *solving* within the model, while LLMs are not necessarily the best option for the solving steps. For instance, the task of "*calculate the spending in online shopping last week*" requires adding up multiple orders, a task at which LLMs often struggle [95, 207]. Other solving steps involve accessing real-time data, which any model alone cannot accomplish by default. Second, while procedural tasks possess rich structure, expressing such structure in the text is unnatural, thus limiting the benefits of leveraging these structures. One key feature of procedures is their hierarchies, where a high-level task can be decomposed into multiple lower-level procedures. Due to this hierarchy, a procedure becomes *reusable* (or *composable*)—a single procedure can be combined with different procedures to achieve different higher-level goals. For example, "*learning to use a computer*" can be a sub-procedure of both "*book a flight ticket online*" and "*book a hotel room online*". Reusing existing procedures can enhance the efficiency and robustness of AI agents [8, 39, 44, 48, 59, 158, 188, 193]. However, reusing knowledge in free-form text is challenging unless it involves verbatim repetition. When task variations require revisions in some components, "reusing" the knowledge can become increasingly complex, eventually becoming as complex as generating the procedure from scratch. Many works propose alternative notations or domain-specific languages [11, 35, 144, 190], but these new notations may be less common in large pretraining corpora, thus losing the benefit from large-scale pretraining. Alternatively, the notation might be too domain-specific, sacrificing generalizability to broader tasks.

Existing Learning Mechanisms are Largely Annotation-driven The success of LLMs hinges on two key factors: large-scale pretraining and the extensive collection of human demonstrations for instruction finetuning and reinforcement learning from human feedback (RLHF). While AI agents can greatly benefit from similar pretraining, human demonstrations in dialogues typically have a different task distribution. They often assume a static context and fail to consider important aspects of agent tasks such as environmental dynamics and action impacts. Consequently, the benefits of leveraging these human demonstrations are limited. Moreover, collecting data for agentic tasks is often expensive and time-consuming. Annotators usually need to follow natural language instructions and act out the entire trajectory as if they were the agent [97, 132, 153, 157, *inter alia*]. This process is not only labor-intensive but also limited in scope. For instance, demonstrating a task like canceling a PayPal order requires an actual PayPal account with a legitimate subscription history. Without extensive data collection, models struggle to generalize to unseen intents or environments [153]. These limitations make it challenging to scale AI agents to real-world scenarios where long-tail requests and diverse environmental configurations are common.

1.3 Contributions Overview

This thesis aims at developing intelligent AI agents that are *generalizable towards a variety of tasks* and *data-efficient*. More specifically, *how* could an agent learn to perform a complex task *without* exact tedious demonstrations from humans? For example, how does an agent learn that the preparation of a conference travel will eventually consist of keyboard strikes and mouse clicks (*e.g.*, for flight booking), phone calls (*e.g.*, for hotel check-in) and others? The key philosophy of this thesis is that such knowledge exists in the open web in the form of natural language descriptions. Therefore, if an agent is capable of *knowledge acquisition and its application*, it could learn the executions of unseen tasks by referring to both past experiences as well as web knowledge. The development of an open-domain-knowledge-augmented agent is underpinned by five key pillars.

Part I Realistic Environment for AI Agent Evaluation First, to bridge the gap in agent evaluation, we built an environment for AI agents that is *highly realistic* and *reproducible*. Specifically, we focus on agents that perform tasks on the web and create an environment with fully functional websites from four common domains: e-commerce, social forum discussions, collaborative software development, and content management. Our environment is enriched with tools (*e.g.*, a map) and external knowledge bases (*e.g.*, user manuals) to encourage human-like task-solving. Building upon our environment, we release a set of benchmark tasks focusing on evaluating the *functional correctness* of task completions. The tasks in our benchmark are diverse, long-horizon, and designed to emulate tasks that humans routinely perform on the internet. We experiment with several baseline agents, integrating recent techniques such as reasoning before acting. The results demonstrate that solving complex tasks is challenging: sthe performance of strong models such as GPT-4-based agent is significantly lower than the human performance. These results highlight the need for further development of robust agents, that current state-of-the-art LLMs are far from perfect performance in these real-life tasks, and that WebArena can be used to measure such progress.

Part II More Versatile“Language” for AI Agents Further, we propose an alternative "language" for AI agents. We opted to make use of the programs in a high-level programming language (*e.g.*, Python) to represent procedures owing to natural compatibility between programs and procedures. Programs are flexible in encoding hierarchies (through nested functions), conditions (through control flow), and the ability to call other modules (through API calls). We name this formalism as "*Procedures as Programs*", or PaP in short. Moreover, PaP is comprehensible and curatable, which allows for fast creation of seed demonstrations and development on various tasks. Finally, the ubiquitous code corpus has been incorporated in the training of large language models (LLMs). PaP could effectively elicit the knowledge in an LLM and benefit agents with an LLM as the backbone model. We instantiate PaP in 15 benchmarks ranging from controlling a robot to perform household tasks and embodied question answering (QA) to mathematical reasoning and symbolic reasoning. PaP demonstrates strong generalization to unseen tasks while being more data-efficient.

Part III Knowledge Base of Hierarchical Procedures Third, we create an open-domain hierarchical knowledge base (H-KB) that stores the procedures in a hierarchical fashion. That is, a procedure is represented as a tree where the children nodes are the decomposed lower-level steps (in text) of their parent node. A procedure tree can have arbitrary depth depending on how concrete a human would like to write about a procedure. While a human user might issue an abstract NL intent corresponding to a node towards the root of a procedure tree, an agent could leverage our H-KB to roll out concrete executions by traversing the tree. We evaluate our KB both intrinsically on the quality of hierarchies and extrinsically on modeling concrete executions. While our H-KB encodes more accurate hierarchical knowledge than strong baselines, the extrinsic evaluation suggests that expressing a procedure with our H-KB is closer to the concrete executions demonstrated in videos. This provides evidence that our KB can bridge the high-level instructions and the low-level executions of procedures.

Part IV New Knowledge Acquisition without Direct Demonstrations Finally, we answer the question of how to further advance an AI agent's capabilities *without* relying on the massive collection of direct human demonstrations. Taking inspiration from humans, who often search for and read information online to solve tasks that they are not initially capable of, we propose two approaches to mimic human behaviors, with a focus on textual knowledge that explains *how-to* perform tasks. First, we propose to *retrieve knowledge* prior to the task performance. We exemplify our approach DocPrompting on the natural language to code generation task, in which code documentation is a key source of knowledge. DocPrompting first retrieves the relevant documentation pieces given an NL intent and then generates code based on the NL intent and the retrieved documentation. DocPrompting is general: it can be applied to any programming language and is agnostic to the underlying neural model. We demonstrate that DocPrompting consistently improves NL-to-code models. Second, we propose to derive hypothetical execution trajectories from plain textual knowledge. When humans read how-to articles, they often engage in mental simulation of the described scenarios. By leveraging the language processing and coding capabilities of LLMs, we can transform how-to articles on web navigation tasks into direct demonstrations. Our finetuned model demonstrates strong performance on web-based tasks compared to other models of similar size. In addition, while such synthetic demonstrations is only 3% the cost of human demonstrations, we show that the synthetic demonstrations can be more effective than a similar number of human demonstrations.

Chapter 2

Background

This section reviews three main topics in AI agent research that are highly relevant to this thesis.

2.1 Evaluation of AI Agents

One key aspect of AI agent evaluation is the evaluation of task success, which typically falls into three categories.

The first category is **reference-based evaluation**, which compares human annotations with agent executions [46, 97, 103, 138, 151, 178]. For instance, Deng et al. [46] collects demonstrations of web-based tasks and uses them as ground-truth references. The evaluation compares a model's predictions with these demonstrations at both the step and task levels. At the step level, the evaluation compares the action type (e.g., `click`) and the action argument (e.g., the location of an element) with the annotation. Task-level success is determined by combined success at the step level. All comparisons are conducted at the string level, so reference-based evaluation can falsely penalize alternative solutions. Collecting multiple references can alleviate this issue; however, the possible solution space may not be covered by just a few examples. For instance, a smart search input box may require only one or two keywords to trigger the correct search, but there can be many keyword options. Ultimately, smart AI agents do not necessarily need to follow human problem-solving patterns due to their silicon nature. For example, an agent that memorizes the full URL of useful websites does not need to perform any navigation or filtering selections.

The second category is **outcome-based evaluation** within executable environments. These environments range from simulated household environments [153], simple web pages [151, 182], to sandbox real-world web applications [82, 212]. Outcome-based evaluation assesses whether a task is truly completed by using verifiers that examine key aspects post-execution. These verifiers are either defined by templates, such as whether the selected product falls into certain categories [153, 182], or annotated by humans, such as whether certain contents are typed into the targeted fields [82, 212]. Outcome-based evaluation mirrors the trend in evaluating model-generated programs. Chen et al. [30] observed that the functional correctness of programs does not necessarily correlate with the surface-form similarity measured by reference-based evaluation, as small changes in programs can significantly alter their semantics. Nevertheless, outcome-based evaluation can be labor-intensive. It requires significant human effort and sometimes extensive expertise.

To address these limitations, many works investigate the third category, **model-based evaluation**, which leverages neural models, especially large language models, for task completion judgment. For example, Pan et al. [125] and He et al. [66] explore using vision-language models to determine whether a trajectory successfully fulfills a given task. Pan et al. [125] demonstrates that strong models such as GPT-4V achieve high accuracy in judging task success, especially when supplemented by auxiliary information such as observation captions. The development of model-based evaluation is particularly interesting not only because it does not rely on extensive human annotations but also because the developed metric can be used as a reward model or automatic feedback module that assists agent task completions (e.g., reject sampling) [16, 125]. Undoubtedly, the evaluation of AI agents goes beyond task success to include task completion efficiency [126, 153] and overall human preference [51], among others.

The benchmark described in Chapter 3 primarily utilizes outcome-based evaluation due to its novel setup and the necessity of establishing ground-truth annotation for future research efforts. Our benchmark also incorporates model-based evaluation in a portion of examples to accommodate alternative answers.

2.2 The "Language" in Procedural Tasks

Primitive actions are the lowest-level actions that can be directly executed in an environment. In the current research, the most common primitive actions are defined as application programming interfaces (APIs) that have various effects on the digital environment (*e.g.*, an API to simulate a mouse click) or on embodied machines (*e.g.*, an API to control the velocity of a robot arm).

Existing works aim to design domain-specific languages tailored to different tasks, defining primitive actions. Early works manually designed non-executable meaning representations (MRs) such as lambda calculus [11, 12] and variable-free logic [40]. Due to the sparse presence of these meaning representations on the Internet, LLMs may lack sufficient knowledge of them, causing the generation of such MRs to lose the benefits of large-scale pretraining. Later works utilize more powerful languages such as SQL, which can effectively query large-scale databases [188, 189, 204], and spreadsheet languages that can manipulate spreadsheet tables [35, 175, 187]. Although these pre-existing languages avoid hand-engineering, they are limited to tasks where the environment containing the required contents (e.g., a database with student information) already exists. Overall, domain-specific languages lack generalizability.

While primitive actions are pre-defined, agents have the flexibility to determine the intermediate steps leading to these actions. These intermediate steps, along with the primitive actions, form a broader language. The most common intermediate steps are **rationales written in natural language (CoT)** that explain why a primitive action should be issued [100, 170, 183]. While incorporating CoT into a task can improve performance, it does not extend a model beyond the natural language reasoning regime. Indeed, natural language reasoning is limited. LLMs are known to perform poorly in tasks such as arithmetic [95, 207], and they are inherently incapable of many tasks, such as accessing real-time information. Some approaches attempt to augment LLMs' capabilities by instructing them to use a more general language—**high-level programming languages** such as Python. Programs natively have the ability to call tools via APIs and execute dynamically through interpreters or compilers. LLMs can thus offload tasks they are not proficient into these tools, while their sole responsibility remains natural language reasoning [14, 33, 112]. Speaking programming languages can effectively extend text-based LLMs to multimodal without further training [62, 149]. Additionally, functions can serve as long-term memory, which, once constructed, can be reused easily [163].

Programming languages are not only suitable for describing the procedure of performing tasks; many works find that writing down declarative knowledge[1] in **program format** is beneficial. Madaan et al. [108] propose describing structured commonsense knowledge (*e.g.*, entity state changes over time) with built-in concepts in programs (*e.g.*, a class and its properties) to facilitate more accurate commonsense reasoning [199]. Given their versatility in both procedural and declarative knowledge, programming languages show promise as the unified comprehensive language for diverse real-world tasks.

The superior performance of writing programs for non-coding tasks has inspired many in-

[1]The knowledge about facts

depth studies. Zhang et al. [198] find that simply changing the format from natural language to programs yields mixed results for various tasks. For instance, text summarization does not benefit from the program format, while conference resolution does. Kim and Schuster [81] observe that coding-proficient models, extensively trained on coding corpora and tasks, demonstrate more accurate entity state tracking capabilities. Finally, models pretrained exclusively on code corpora in the initial stage show improved performance in mathematical reasoning compared to general natural language pretraining.

Chapter 4 presents a proof-of-concept demonstrating the benefit of representing procedures as programs, with human-in-the-loop for constructing and revising these programs. Chapter 5 introduces our approach that leverages LLMs to automatically write programs for broader reasoning tasks. In Chapter 8, we show that LLMs benefit from finetuning with trajectories in program format, compared to natural language CoT.

2.3 Learning without Extensive Human Annotations

Teaching models to perform new tasks without extensive human annotations is a long-standing interest in machine learning and deep learning communities, and AI agents are no exception. Unlike classical machine learning problems where the context is often static (*e.g.*, an image or a dialogue), agents interacting with dynamic environments not only incur additional costs and efforts to set up environments (*e.g.*, setting up multiple parallel environments to collect data), but also increase the curse of dimensionality, where the possible combination of states grows exponentially with the number of time steps. To alleviate the reliance on human annotations, existing works investigate the **source of knowledge** and **learning algorithms**.

Text-based knowledge is extensively studied due to its abundance and coverage across various domains. This knowledge includes environmental dynamics that describe the effects of actions in different states [21, 22, 116, 205]. Text can also convey knowledge in fine granularity. For instance, documentation of APIs often includes detailed explanations of individual arguments. This is beneficial for compositionality (hence alleviating the curse of dimensionality), as multiple pieces of information can be freely combined instead of enumerating all possible combinations as in demonstrations.

The most straightforward way to use text-based knowledge is through **continuous pretraining** with vanilla next-token prediction loss [64]. This strategy can gradually adapt a general model to specific domains [1]. However, Jiang et al. [77] argue that simple continued pretraining has limited effects compared to finetuning on more task-aware data constructed from the corpus. Hence, existing works also explore other approaches to utilize these resources. The first approach is to use the resources through **retrieval**, where the knowledge is presented as additional input to assist task completions. Additionally, many works perform **data synthesize** from the resources [27, 177]. For example, Xu et al. [177] use code documentation to synthesize additional NL-code pairs for code generation. Finally, we can leverage text knowledge as additional input to **construct more accurate functions** such as transition functions or reward functions in reinforcement learning [21, 22, 116, 205].

While natural language is versatile for conveying various types of information, it is not necessarily the most efficient channel for certain types, particularly low-level behaviors. For example, describing how to knead dough using only natural language can be cumbersome. Consequently, some knowledge is absent in text-based resources and requires **input from other modalities**. Fan et al. [52] train a reward model based on a video and caption parallel corpus, while Baker et al. [17] learn low-level keyboard and mouse controls for Minecraft from videos. Such learning from Internet-scale videos resembles research on learning from demonstrations in robotics [60, 109].

Besides learning from existing knowledge created by humans, research also explores learning from existing experiences to reduce reliance on additional human annotations. Here, existing experiences are loosely defined as episodes performed by the same agent in the past or datasets created in the same environment. Simply training the model on trajectories consisting of state and expected actions is data-hungry. Hence, works explore **augmenting existing experiences with additional information**. Chen et al. [28] and Yin et al. [184] add LLM-generated chain-of-thought reasoning and planning to execution trajectories. Fu et al. [54] and Sarch et al. [148] ask (multimodal) LLMs to summarize useful knowledge, such as environment dynamics and workflows, from existing experiences and dynamically provide the summarized knowledge in new examples. The assumption in this thread is that generating such information is easier than performing the task itself. Additionally, existing works study **relabeling of existing experiences** to effectively use failed examples. This includes adding task descriptions to arbitrary trajectories with LLMs [9, 115].

Chapter 6 discusses the limitation that text-based procedural knowledge lacks hierarchical structures. Consequently, how-to knowledge of the same task at different levels of granularity may be dispersed across various sources, resulting in inefficiency when consuming data. We proposed an approach to link such isolated knowledge together. Furthermore, Chapter 7 and Chapter 8 explore utilizing human-authored knowledge through retrieval and data synthesis.

Part I

Realistic Environment for AI Agent Evaluation

Chapter 3

Realistic Web Environment for Building Autonomous AI Agents

To fully leverage the power of autonomous agents, it is crucial to understand their behavior within an environment that is both *authentic* and *reproducible*. This will allow measurement of the ability of agents on tasks that human users care about in a fair and consistent manner. In this section, we present our work on building the first comprehensive benchmark with real-world complexities, reliable evaluation metrics and easily extensibility for AI agent development. This work first appears in:

- Shuyan Zhou*, Frank F. Xu*, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*, Vienna, Austria, 2024

3.1 Overview

Current environments for evaluate agents tend to *over-simplify* real-world situations. As a result, the functionality of many environments is a limited version of their real-world counterparts, leading to a lack of task diversity [5, 59, 113, 151, 153, 154, 182]. In addition, these simplifications often lower the complexity of tasks as compared to their execution in the real world [132, 153, 182]. Finally, some environments are presented as a static resource [46, 151] where agents are confined to accessing only those states that were previously cached during data collection, thus limiting the breadth and diversity of exploration. For evaluation, many en-

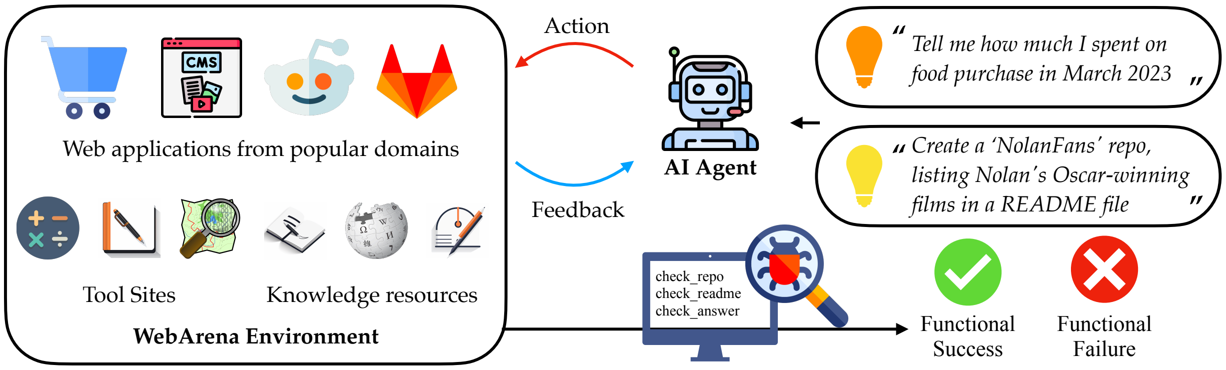

Web applications from popular domains
Tool Sites
Knowledge resources
WebArena Environment
Action
Feedback
AI Agent
“Tell me how much I spent on food purchase in March 2023”
“Create a ‘NolanFans’ repo, listing Nolan’s Oscar-winning films in a README file”
check_repo
check_readme
check_answer
Functional Success
Functional Failure

Figure 3.1: WebArena is a standalone, self-hostable web environment for building autonomous agents. WebArena creates websites from four popular categories with functionality and data mimicking their real-world equivalents. To emulate human problem-solving, WebArena also embeds tools and knowledge resources as independent websites. WebArena introduces a benchmark on interpreting *high-level realistic* natural language command to concrete web-based interactions. We provide validators to programmatically validate the functional correctness of each task.

vironments focus on comparing the textual *surface form* of the predicted action sequences with reference action sequences, disregarding the *functional correctness* of the executions and possible alternative solutions [46, 75, 97, 132, 178]. These limitations often result in a discrepancy between simulated environments and the real world, and can potentially impact the generalizability of AI agents to successfully understand, adapt, and operate within complex real-world situations.

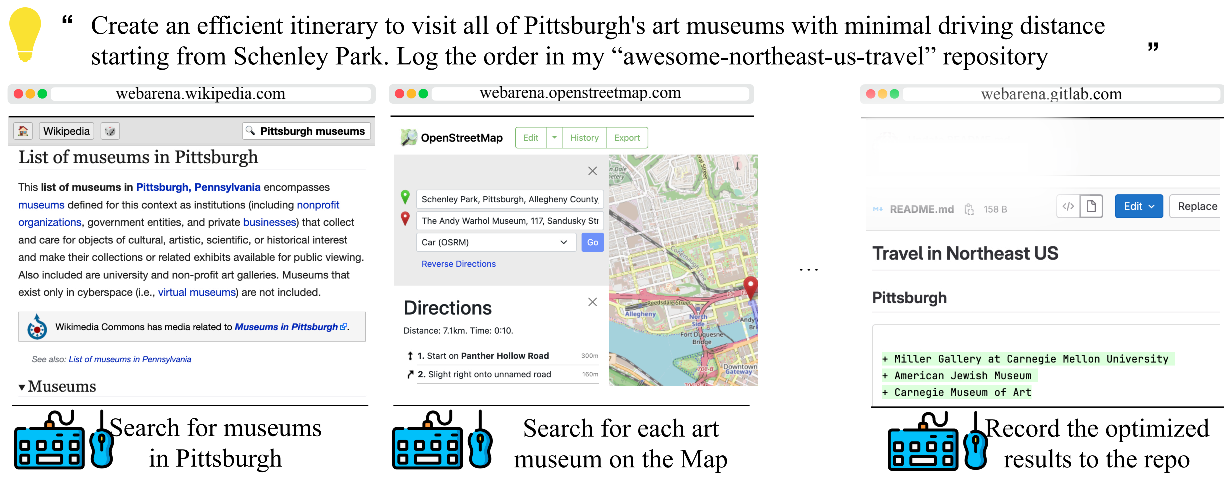
We introduce WebArena, a *realistic* and *reproducible* web environment designed to facilitate the development of autonomous agents capable of executing tasks (§3.2). An overview of WebArena is in Figure 3.1. Our environment comprises four fully operational, self-hosted web applications, each representing a distinct domain prevalent on the internet: online shopping, discussion forums, collaborative development, and business content management. Furthermore, WebArena incorporates several utility tools, such as map, calculator, and scratchpad, to best support possible human-like task executions. Lastly, WebArena is complemented by an extensive collection of documentation and knowledge bases that vary from general resources like English Wikipedia to more domain-specific references, such as manuals for using the integrated development tool [52]. The content populating these websites is extracted from their real-world counterparts, preserving the authenticity of the content served on each platform. We deliver the hosting services using Docker containers with `gym`-APIs [23], ensuring both the usability and the reproducibility of WebArena.

Along with WebArena, we release a ready-to-use benchmark with 812 long-horizon web-based tasks (§3.3). Each task is described as a high-level natural language intent, emulating the abstract language usage patterns typically employed by humans [19]. Two example intents are shown in the upper left of Figure 3.1. We focus on evaluating the *functional correctness* of these tasks, *i.e.*, does the result of the execution actually achieve the desired goal (§3.3.2). For instance, to evaluate the example in Figure 3.2, our evaluation method verifies the concrete contents in the designated repository. This evaluation is not only more reliable [31, 168, 203] than comparing the textual surface-form action sequences [46, 132] but also accommodate a range of potential valid paths to achieve the same goal, which is a ubiquitous phenomenon in sufficiently complex tasks.

We use this benchmark to evaluate several agents that can follow NL command and perform web-based tasks (§3.4). These agents are implemented in a few-shot in-context learning fashion with powerful large language models (LLMs) such as GPT-4 and PALM-2. Experiment results show that the best GPT-4 agent performance is somewhat limited, with an end-to-end task success rate of only 14.41%, while the human performance is 78.24%. We hypothesize that the limited performance of current LLMs stems from a lack of crucial capabilities such as active exploration and failure recovery to successfully perform complex tasks (§3.5.1). These outcomes underscore the necessity for further development towards robust and effective agents [91] in WebArena.

3.2 Web App as an Environment for Autonomous Agents

Our goal is to create a *realistic* and *reproducible* web environment. We achieve reproducibility by making the environment standalone, without relying on live websites. This circumvents technical challenges such as bots being subject to CAPTCHAs, unpredictable content modifications, and configuration changes, which obstruct a fair comparison across different systems over time. We achieve realism by using open-source libraries that underlie many in-use sites from several popular categories and importing data to our environment from their real-world counterparts.


“ Create an efficient itinerary to visit all of Pittsburgh's art museums with minimal driving distance starting from Schenley Park. Log the order in my “awesome-northeast-us-travel” repository ”
webarena.wikipedia.com
Wikipedia
Pittsburgh museums
List of museums in Pittsburgh
Wikimedia Commons has media related to Museums in Pittsburgh.
See also: List of museums in Pennsylvania
Museums
Search for museums in Pittsburgh
webarena.openstreetmap.com
OpenStreetMap
Edit
History
Export
Schenley Park, Pittsburgh, Allegheny County
The Andy Warhol Museum, 117, Sandusky Str
Car (OSRM)
Go
Reverse Directions
Directions
Distance: 7.1km. Time: 0:10.
1. Start on Panther Hollow Road
2. Slight right onto unnamed road
Search for each art museum on the Map
webarena.gitlab.com
README.md
Edit
Replace
Travel in Northeast US
Pittsburgh
Record the optimized results to the repo

This list of museums in Pittsburgh, Pennsylvania encompasses museums defined for this context as institutions (including nonprofit organizations, government entities, and private businesses) that collect and care for objects of cultural, artistic, scientific, or historical interest and make their collections or related exhibits available for public viewing. Also included are university and non-profit art galleries. Museums that exist only in cyberspace (i.e., virtual museums) are not included.

```
+ Miller Gallery at Carnegie Mellon University
+ American Jewish Museum
+ Carnegie Museum of Art
```

Figure 3.2: A high-level task that can be fully executed in WebArena. Success requires sophisticated, long-term planning and reasoning. To accomplish the goal (top), an agent needs to (1) find Pittsburgh art museums on Wikipedia, (2) identify their locations on a map (while optimizing the itinerary), and (3) update the README file in the appropriate repository with the planned route.

3.2.1 Controlling Agents through High-level Natural Language

The WebArena environment is denoted as $\mathcal{E} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T} \rangle$ with state space $\mathcal{S}$, action space $\mathcal{A}$ (§3.2.4) and observation space $\mathcal{O}$ (§3.2.3). The transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \longrightarrow \mathcal{S}$ is deterministic, and it is defined by the underlying implementation of each website in the environment. Given a task described as a natural language intent $\mathbf{i}$, an agent issues an action $a_t \in \mathcal{A}$ based on intent $\mathbf{i}$, the current observation $o_t \in \mathcal{O}$, the action history $\mathbf{a}_1^{t-1}$ and the observation history $\mathbf{o}_1^{t-1}$. Consequently, the action results in a new state $s_{t+1} \in \mathcal{S}$ and its corresponding observation $o_{t+1} \in \mathcal{O}$. We propose a reward function $r(\mathbf{a}_1^T, \mathbf{s}_1^T)$ to measure the success of a task execution, where $\mathbf{a}_1^T$ represents the sequence of actions from start to the end time step T, and $\mathbf{s}_1^T$ denotes all intermediate states. This reward function assesses if state transitions align with the expectations of the intents. For example, with an intent to place an order, it verifies whether an order has been placed. Additionally, it evaluates the accuracy of the agent's actions, such as checking the correctness of the predicted answer.

3.2.2 Website Selection

To decide which categories of websites to use, we first analyzed approximately 200 examples from the authors' actual web browser histories. Each author delved into their browsing histories, summarizing the goal of particular segments of their browser session. Based on this, we classified the visited websites into abstract categories. We then identified the four most salient categories and implemented one instance per category based on this analysis: (1) E-commerce platforms supporting online shopping activities (*e.g.*, Amazon, eBay), (2) social forum platforms for opinion exchanges (*e.g.*, Reddit, StackExchange), (3) collaborative development platforms for software development (*e.g.*, GitLab), and (4) content management systems (CMS) that manage the creation and revision of the digital content (*e.g.*, online store management).

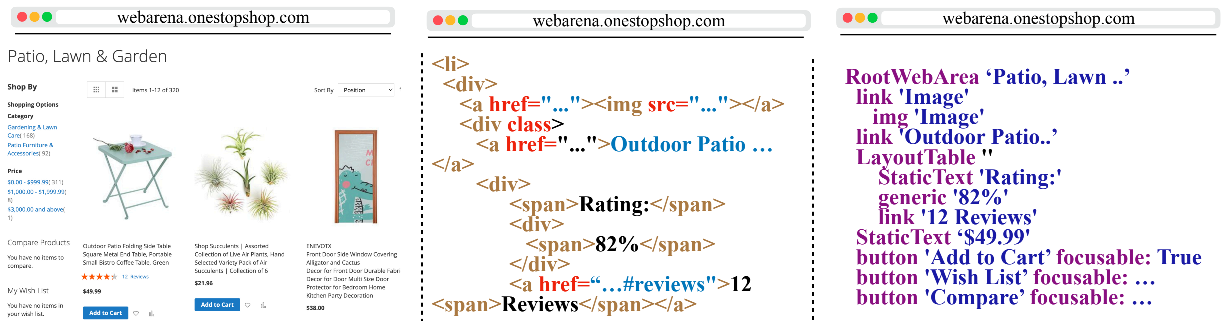
In addition to these platforms, we selected three utility-style tools that are frequently used in web-based tasks: (1) a map for navigation and searching for information about points of interest (POIs) such as institutions or locations (2) a calculator, and (3) a scratchpad for taking notes. As information-seeking and knowledge acquisition are critical in web-based tasks, we also incorporated various knowledge resources into WebArena. These resources range from general information hubs, such as the English Wikipedia, to more specialized knowledge bases, such as the website user manuals.

Implementation We leveraged open-source libraries relevant to each category to build our own versions of an E-commerce website (OneStopShop), GitLab, Reddit, an online store content management system (CMS), a map, and an English Wikipedia. Then we imported sampled data from their real-world counterparts. As an example, our version of GitLab was developed based on the actual GitLab project.[1] We carefully emulated the features of a typical code repository by including both popular projects with many issues and pull requests and smaller, personal projects. Details of all websites in WebArena can be found in Appendix A.1. We deliver the environment as dockers and provide scripts to reset the environment to a deterministic initial state (See Appendix A.2).

3.2.3 Observation Space

We design the observation space to roughly mimic the web browser experience: a web page URL, the opened tabs , and the web page content of the focused tab. WebArena is the first web

[1] https://gitlab.com/gitlab-org/gitlab


webarena.onestopshop.com

Patio, Lawn & Garden

Shop By

Items 1-12 of 320 Sort By Position

Shopping Options
Category
Gardening & Lawn Care(168)
Patio Furniture & Accessories(92)

Price
$0.00 - $999.99(311)
$1,000.00 - $1,999.99(8)
$3,000.00 and above(1)

Compare Products
You have no items to compare.

My Wish List
You have no items in your wish list.

Outdoor Patio Folding Side Table Square Metal End Table, Portable Small Bistro Coffee Table, Green
12 Reviews
$49.99
Add to Cart

Shop Succulents | Assorted Collection of Live Air Plants, Hand Selected Variety Pack of Air Succulents | Collection of 6
$21.96
Add to Cart

ENEVOTX
Front Door Side Window Covering Alligator and Cactus Decor for Front Door Durable Fabri Decor for Door Multi Size Door Protector for Bedroom Home Kitchen Party Decoration
$38.00

webarena.onestopshop.com

```
<li>
 <div>
   <a href="..."><img src="..."></a>
   <div class>
     <a href="...">Outdoor Patio ...
</a>
       <div>
         <span>Rating:</span>
         <div>
           <span>82%</span>
         </div>
         <a href="...#reviews">12
<span>Reviews</span></a>
```

webarena.onestopshop.com

```
RootWebArea 'Patio, Lawn ..'
 link 'Image'
   img 'Image'
 link 'Outdoor Patio..'
 LayoutTable ''
     StaticText 'Rating:'
     generic '82%'
     link '12 Reviews'
 StaticText '$49.99'
 button 'Add to Cart' focusable: True
 button 'Wish List' focusable: ...
 button 'Compare' focusable: ...
```

Figure 3.3: We design the observation to be the URL and the content of a web page, with options to represent the content as a screenshot (left), HTML DOM tree (middle), and accessibility tree (right). The content of the middle and right figures are trimmed to save space.

environment to consider multi-tab web-based tasks to promote tool usage, direct comparisons and references across tabs, and other functionalities. The multi-tab functionality offers a more authentic replication of human web browsing habits compared to maintaining everything in a single tab. We provide flexible configuration to render the page content in many modes: (see Figure 3.3 for an example): (1) the raw web page HTML, composed of a Document Object Model (DOM) tree, as commonly used in past work [46, 97, 151]; (2) a screenshot, a pixel-based representation that represents the current web page as an RGB array and (3) the accessibility tree of the web page.[2] The accessibility tree is a subset of the DOM tree with elements that are *relevant* and *useful* for displaying the contents of a web page. Every element is represented as its role (*e.g.*, a link), its text content, and its properties (*e.g.*, whether it is focusable). Accessibility trees largely retain the *structured* information of a web page while being more compact than the DOM representation.

We provide an option to limit the content to the contents within a viewport for all modes. This ensures that the observation can be input into a text-based model with limited context length or an image-based model with image size or resolution requirements.

3.2.4 Action Space

Following previous work on navigation and operation in web and embodied environments [101, 151], we design a compound action space that emulates the keyboard and mouse operations available on web pages. Figure 3.4 lists all the available actions categorized into three distinct groups. The first group includes element operations such as clicking, hovering, typing, and key

[2] https://developer.mozilla.org/en-US/docs/Glossary/Accessibility_tree

combination pressing. The second comprises tab-related actions such as opening, closing, and switching between tabs. The third category consists of URL navigation actions, such as visiting a specific URL or navigating forward and backward in the browsing history.

Building on these actions, WebArena provides agents with the flexibility to refer to elements for operation in different ways. An element can be selected by its on-screen coordinates, (x, y), or by a unique element ID that is prepended to each element. This ID is generated when traversing the Document Object Model (DOM) or accessibility tree. With element IDs, the element selection is transformed into an n-way classification problem, thereby eliminating any disambiguation efforts required from the agent or the underlying implementation. For example, issuing the action `click [1582]` clicks the button given the observation of `[1582] Add to Cart`. This flexible element selection allows WebArena to support agents designed in various ways (*e.g.*, accepting input from different modalities) without compromising fair comparison metrics such as step count.

User Role Simulation Users of the same website often have disparate experiences due to their distinct *roles*, *permissions*, and *interaction histories*. We emulate this scenario by generating unique user profiles on each platform. The details can be found in Appendix A.3.

3.3 Benchmark Suite of Web-based Tasks

We provide a benchmark with 812 test examples on grounding high-level natural language instructions to interactions in WebArena. Each example has a metric to evaluate the functional correctness of the task execution. In this section, we first formally define the task of controlling an autonomous agent through natural language. Then we introduce the annotation process of our benchmark.

3.3.1 Intent Collection

We focus on curating *realistic* intents to carry out *complex* and *creative* tasks within WebArena. To start with, our annotators were guided to spend a few minutes exploring the websites to familiarize themselves with the websites' content and functionalities. As most of our websites are virtually identical to their open-web counterparts, despite having sampled data, most annotators can quickly comprehend the websites.

Next, we instructed the annotators to formulate intents based on the following criteria:

1. The intent should be *abstract* and *high-level*, implying that the task cannot be fulfilled with merely one or two actions. As an example, instead of "*click the* `science` *subreddit*", we encouraged annotators to come up with something more complex like "*post a greeting message on* `science` *subreddit*", which involves performing multiple actions.
2. The intent should be *creative*. Common tasks such as account creation can be easily thought of. We encouraged the annotators to add constraints (*e.g.*, "*create a Reddit account* ***identical to my GitLab one***") to make the intents more unique.
3. The intent should be formulated as a *template* by making replaceable elements as variables. The annotators were also responsible for developing several instantiations for each variable. For example, the intent "*create a Reddit account identical to my GitLab one*" can be converted into "*create a {{site1}} account identical to my {{site2}} one*", with an instantiation like "*{site1: Reddit, site2: GitLab}*" and another like "*{site1: GitLab, site2: OneStopShopping}*". Notably, tasks derived from the same template can have distinct execution traces. The similarity resides primarily in the high-level semantics rather than the specific implementation.

We also provided a prompt for the annotators to use with ChatGPT[3] for inspiration, that contains an overview of each website and instructs the model to describe potential tasks to be performed on these sites. Furthermore, we offered a curated list of examples for annotators to reference.

Intent Analysis In total, we curated 241 templates and 812 instantiated intents. On average, each template is instantiated to 3.3 examples. The intent distribution is shown in Figure A.1. Furthermore, we classify the intents into three primary categories with examples shown in Figure 3.5:

1. **Information-seeking** tasks expect a textual response. Importantly, these tasks in WebArena often require navigation across multiple pages or focus on *user-centric* content. This makes them distinct from open-domain question-answering [88, 181], which focuses on querying general knowledge with a simple retrieval step. For instance, to answer "*When was the last time I bought the shampoo*", an agent traverses the user's purchase history, checking order details to identify the most recent shampoo purchase.
2. **Site navigation**: This category is composed of tasks that require navigating through web pages using a variety of interactive elements such as search functions and links. The

[3] https://chat.openai.com/

Action Type	Description
`noop`	Do nothing
`click(elem)`	Click at an element
`hover(elem)`	Hover on an element
`type(elem, text)`	Type to an element
`press(key_comb)`	Press a key comb
`scroll(dir)`	Scroll up and down
`tab_focus(index)`	focus on i-th tab
`new_tab`	Open a new tab
`tab_close`	Close current tab
`go_back`	Visit the last URL
`go_forward`	Undo `go_back`
`goto(URL)`	Go to URL

Figure 3.4: Action Space of WebArena

Category	Example
Information Seeking	When was the last time I bought shampoo
	Compare walking and driving time from AMC Waterfront to Randyland
Site Navigation	Checkout merge requests assigned to me
	Show me the ergonomic chair with the best rating
Content & Config	Post to ask "whether I need a car in NYC"
	Delete the reviews from the scammer Yoke

Figure 3.5: Example intents from three categories.

objective is often to locate specific information or navigate to a particular section of a site.

3. **Content and configuration operation**: This category encapsulates tasks that require operating in the web environment to create, revise, or configure content or settings. This includes adjusting settings, managing accounts, performing online transactions, generating new web content, and modifying existing content. Examples range from updating a social media status or README file to conducting online purchases and configuring privacy settings.

3.3.2 Evaluation Annotation

Evaluating Information Seeking Tasks To measure the correctness of information-seeking tasks where a textual answer is expected, we provide the annotated answer a^* for each intent. The a^* is further compared with the predicted answer $\hat{a}$ with one of the following scoring functions $r_{\text{info}}(\hat{a}, a^*)$.

First, we define `exact_match` where only $\hat{a}$ that is identical with a^* receives a score of one. This function is primarily applicable to intent types whose responses follow a more standardized format, similar to the evaluation on question answering literature [135, 181].

Second, we create `must_include` where any $\hat{a}$ containing a^* receives a score of one. This function is primarily used in when an unordered list of text is expected or where the emphasis of evaluation is on certain key concepts. In the second example in Table 3.1, we expect both the correct name and the email address to be presented, irrespective of the precise wording used to convey the answer.

Finally, we introduce `fuzzy_match` where we utilize a language model to assess whether $\hat{a}$ is semantically equivalent to a^*. Specifically, in this work, we use `gpt-4-0613` to perform this evaluation. The corresponding prompt details are provided in Appendix A.7. The `fuzzy_match` function applies to situations where the format of the answer is diverse. For instance, in responding to "*Compare the time for walking and driving route from AMC Waterfront to Randyland*", it is essential to ensure that driving time and walking time are accurately linked with the correct terms. The `fuzzy_match` function could also flexibly match the time "2h58min" with different forms such as "2 hour 58 minutes", "2:58" and others. We demonstrate a language model can achieve nearly perfect performance on this task in §A.8.

Evaluating Site Navigation and Content & Config Tasks The tasks in these categories require accessing web pages that meet certain conditions or performing operations that modify the underlying data storage of the respective websites. To assess these, we establish reward functions $r_{\text{prog}}(\mathbf{s})$ that programmatically examine the intermediate states $\mathbf{s}$ within an execution trajectory to ascertain whether the outcome aligns with the intended result. These intermediate states are often the underlying databases of the websites, the status, and the content of a web page at each step of the execution.

Evaluating each instance involves two components. First, we provide a `locator`, tasked with retrieving the critical content pertinent to each intent. The implementation of this locator varies from a database query, a website-supported API call, to a JavaScript element selection on the relevant web page, depending on implementation feasibility. For example, the evaluation process for the intent of the fifth example in Table 3.1, first obtains the URL of the latest post by examining the last state in the state sequence $\mathbf{s}$. Then it navigates to the corresponding post page and obtains the post's content by running the Javascript "`document.querySelector('.submission__`

Subsequently, we annotate `keywords` that need to exist within the located content. For example, the evaluation verifies if the post is correctly posted in the "nyc" subreddit by examining the URL of the post and if the post contains the requested content by examining the post content. We reuse the `exact_match` and `must_include` functions from information-seeking tasks for this purpose.

Function	ID	Intent	Eval Implementation
$r_{\text{info}}(a^*, \hat{a})$	1	Tell me the name of the customer who has the most cancellations in the history	`exact_match`($\hat{a}$, "Samantha Jones")
	2	Find the customer name and email with phone number 8015551212	`must_include`($\hat{a}$, "Sean Miller") `must_include`($\hat{a}$, "sean@gmail.com")
	3	Compare walking and driving time from AMC Waterfront to Randyland	`fuzzy_match`($\hat{a}$, "walking: 2h58min") `fuzzy_match`($\hat{a}$, "driving: 21min")
$r_{\text{prog}}(\mathbf{s})$	4	Checkout merge requests assigned to me	`url=locate_current_url(`$\mathbf{s}$`)` `exact_match(URL,` "gitlab.com/merge_requests?assignee_username=byteblaze")

	5	Post to ask "whether I need a car in NYC"	`url=locate_latest_post_url(`$\mathbf{s}$`)`
			`body=locate_latest_post_body(`$\mathbf{s}$`)`
			`must_include(URL,` "/f/nyc")
			`must_include(body,` "a car in NYC")

Table 3.1: We introduce two evaluation approaches. r_{info} (top) measures the correctness of performing information-seeking tasks. It compares the predicted answer $\hat{a}$ with the annotated reference a^* with three implementations. r_{prog} (bottom) programmatically checks whether the intermediate states during the executions possess the anticipated properties specified by the intent.

Unachievable Tasks Due to constraints such as inadequate evidence, user permissions (§A.3), or the absence of necessary functional support on the website, humans may ask for tasks that are not possible to complete. Inspired by previous work on evaluating question-answering models on unanswerable questions [137], we design unachievable tasks in WebArena. For instance, fulfilling an intent like "*Tell me the contact number of OneStopShop*" is impracticable in WebArena, given that the website does not provide such contact information. We label such instances as "N/A" and expect an agent to produce an equivalent response. These examples allow us to assess an agent's ability to avoid making unfounded claims and its adherence to factual accuracy.

Annotation Process The intents were contributed by the authors following the annotation guideline in §3.3.1. Every author has extensive experience with web-based tasks. The reference answers to the information-seeking tasks were curated by the authors and an external annotator. To ensure consistency and accuracy, each question was annotated twice. If the two annotators disagreed, a third annotator finalized the annotation. The programs to evaluate the remaining examples were contributed by three of the authors who are proficient in JavaScript programming. Difficult tasks were often discussed collectively to ensure the correctness of the annotation. The annotation required the annotator to undertake the full execution and scrutinize the intermediate states.

Human Performance We sample one task from each of the 170 templates and ask five computer science graduate students to perform these tasks. The human performance is on the right. Overall, the human annotators complete 78.24% of the tasks, with lower performance on information-seeking tasks. Through examining the recorded trajectories, we found that 50% of the failures are due to misinterpreting the intent (*e.g.*, providing travel distance when asked for travel time), incomplete answers (*e.g.*, providing only name when asked for name and email), and incomplete executions (*e.g.*, partially filling the product information), while the remaining instances have more severe failures, where the executions are off-target. More discussions on human annotations can be found in §A.5.

Avg. Time	110s
Success Rate$_{\text{info}}$	74.68%
Success Rate$_{\text{others}}$	81.32%
Success Rate$_{\text{all}}$	78.24%

3.4 Baseline Web Agents

We experiment with three LLMs using two prompting strategies, both with two examples in the context. In the first setting, we ask the LLM to directly predict the next action given the current observation, the intent and the previously performed action. In the second setting, with the same information, the model first performs chain-of-thought reasoning steps in the text before the action prediction (CoT, Wei et al. [170], Yao et al. [183]). Before the examples, we provide a detailed overview of the browser environment, the allowed actions, and many rules. To make the model aware of the unachievable tasks, the instruction explicitly asks the agent to stop if it believes the task is impossible to perform. We refer to this directive as Unachievable hint, or **UA hint**. This introduction is largely identical to the guidelines we presented to human annotators to ensure a fair comparison. We use an accessibility tree with element IDs as the observation space. The agent can identify which element to interact with by the ID of the element. For instance, the agent can issue `click [1582]` to click the "Add to Cart" button with the ID of 1582. The full prompts can be found in Appendix A.9. The detailed configurations of each model can be found in Appendix A.6.

CoT	UA Hint	Model	SR	SR_{AC}	SR_{UA}
✓	✓	TEXT-BISON-001	5.05	4.00	27.78
✗	✓	GPT-3.5	6.41	4.90	38.89
✓	✓	GPT-3.5	8.75	6.44	58.33
✓	✓	GPT-4	11.70	8.63	**77.78**
✗	✗	GPT-3.5	5.10	4.90	8.33
✓	✗	GPT-3.5	6.16	6.06	8.33
✓	✗	GPT-4	**14.41**	**13.02**	44.44
-	✓	Human	78.24	77.30	100.00

Table 3.2: The end-to-end task success rate (SR %) on WebArena with different prompting strategies. **CoT**: the model performs step-by-step reasoning before issuing the action. **UA hint**: ask the model to stop when encountering unachievable questions.

3.5 Results

The main results are shown on the top of Table 3.2. GPT-4 [121] with CoT prompting achieves a modest end-to-end task success rate of 11.70%, which is significantly lower than the human performance of 78.24%. GPT-3.5 [120] with CoT prompting is only able to successfully perform 8.75% of the tasks. The explicit reasoning procedure is somewhat helpful, it brings 2.34% improvement over the version without it. Further, TEXT-BISON-001 [10] underperforms GPT-3.5, with a success rate of 5.05%. These results underline the inherent challenges and complexities of executing tasks that span long horizons, particularly in realistic environments such as WebArena.

3.5.1 Analysis

Do models know when to stop? In our error analysis of the execution trajectories, we observe a prevalent error pattern of early stopping due to the model's conclusion of unachievability. For instance, GPT-4 erroneously identifies 54.9% of feasible tasks as impossible. This issue primarily stems from the UA hint in the instruction, while this hint allows models to identify unachievable tasks, it also hinders performance on achievable tasks. To address this, we conduct an ablation study where we remove this hint. We then break down the success rate for

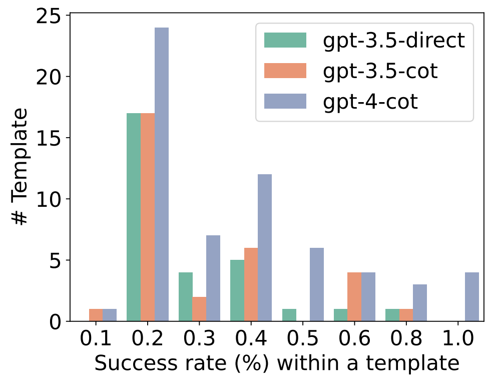


Figure 3.6: Distribution of success rates on templates with ≥ 1 successful executions on GPT models (no UA hint).

both achievable and unachievable tasks. As shown in Table 3.2, eliminating this instruction led to a performance boost in achievable tasks, enhancing the overall task success rate of GPT-4 to 14.41%. Despite an overall decline in identifying unachievable tasks, GPT-4 retains the capacity to recognize 44.44% of such tasks. It does so by generating *reasons of non-achievability*, even without explicit instructions. On the other hand, GPT-3.5 rarely exhibits this level of reasoning. Instead, it tends to follow problematic patterns such as hallucinating incorrect answers, repeating invalid actions, or exceeding the step limits. This result suggests that even subtle differences in instruction design can significantly influence the behavior of a model in performing interactive tasks in complex environments.

Can a model maintain consistent performance across similar tasks? Tasks that originate from the same template usually follow similar reasoning and planning processes, even though their observations and executions will differ. We plot a histogram of per-template success rates for our models in Figure 3.6. Of the 61 templates, GPT-4 manages to achieve a 100% task success rate on only four templates, while GPT-3.5 fails to achieve full task completion for any of the templates. In many cases, the models are only able to complete one task variation with a template. These observations indicate that even when tasks are derived from the same template, they can present distinct challenges. For instance, while "*Fork metaseq*" can be a straightforward task, "*Fork all repos from Facebook*" derived from the same template requires more repetitive operations, hence increasing its complexity. Therefore, WebArena provide a testbed to evaluate more sophisticated methods. In particular, those that incorporate memory components, enabling the *reuse* of successful strategies from past experiments [163, 208]. More error analysis with examples can be found in Appendix A.10.

Benchmark	Dynamic Interaction?	Realistic Environment?	Diverse Human Tasks?	Functional Correctness?
Mind2Web [46]	✗	✓	✓	✗
Form/QAWoB [151]	✗	✓	✓	✗
MiniWoB++ [101]	✓	✗	✗	✓
Webshop [182]	✓	✗	✗	✓
ALFRED [153]	✓	✗	✗	✓
VirtualHome [132]	✗	✗	✓	✗
AndroidEnv [161]	✓	✓	✗	✗
WebArena	✓	✓	✓	✓

Table 3.3: The comparison between our benchmark and existing benchmarks on grounding natural language instructions to concrete executions. Our benchmark is implemented in our fully interactable highly-realistic environment. It features diverse tasks humans may encounter in their daily routines. We design evaluation metrics to assess the functional correctness of task executions.

3.6 Comparison with Other Agent Benchmarks

Controlling agents via natural language in the digital world have been studied in the literature [21, 46, 97, 101, 151, 161, 178]. However, the balance between *functionality*, *authenticity*, and *support for environmental dynamics* remains a challenge. Existing benchmarks often compromise these aspects, as shown in Table 3.3. Some works rely on static states, limiting agents' explorations and functional correctness evaluation [46, 151], while others simplify real-world complexities, restricting task variety [101, 182]. While AndroidEnv [161] replicates an Android setup, it does not guarantee the reproducibility since live Android applications are used. [83, 132, 153] and extends to gaming environments [52, 87], where the environment mechanisms often diverge from human objectives.

Author Contributions Shuyan Zhou proposed the evaluation framework for AI agents on web-based tasks, and was responsible for designing and executing the data collection pipeline, developing the evaluation metrics, and building the codebase. Frank Xu designed, implemented or supervised the implementation of the sandbox web environments, and provided significant input into the overall design and development of the other components of the project.

Part II

More Versatile“Language” for AI Agents

Chapter 4

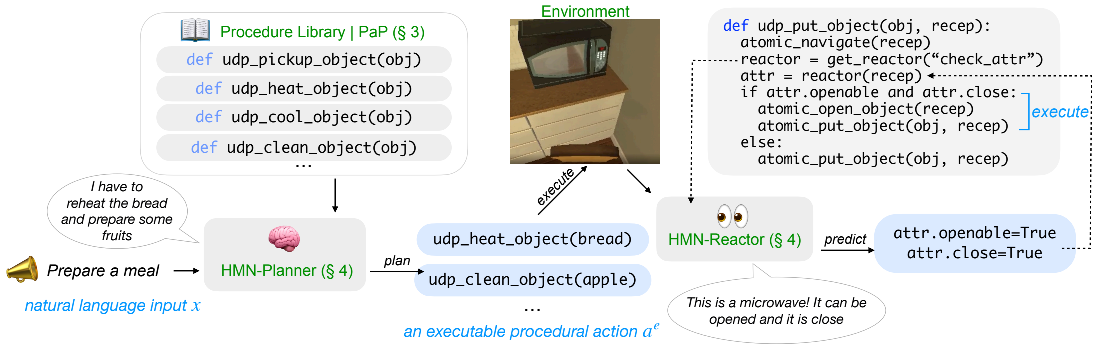
Representing Procedures as Programs

Large language models are proficient in producing natural language text. However, free-form text lacks the expressiveness to represent the versatile behaviors in the human problem-solving process, such as hierarchical planning and tool use. This chapter presents our initial work on exploring the appropriate "language" for AI agents to use, which could describe the task execution process in a versatile, accurate and user-friendly way. This work first appears in:

- Shuyan Zhou, Pengcheng Yin, and Graham Neubig. Hierarchical control of situated agents through natural language. In *Workshop on Structured and Unstructured Knowledge Integration (SUKI)*, Seattle, USA, July 2022. URL https://arxiv.org/abs/2109.08214

4.1 Overview

As discussed in many examples before, procedures are inherently hierarchical; high-level procedures consist of many lower-level procedures. There has been significant prior work on benchmarks and methods for complex task completion using situated agents given natural language (NL) instructions, such as agents trained to navigate the web and mobile UIs [97, 178] or solve household tasks [153]. However, most methods applied to these tasks use a *reactive* strategy that makes decisions on the low-level atomic actions available to the agent while making steps through the environment [61, 214], or define procedures in a shallow way where there only exists one level of hierarchy [8, 44, 59, 188].These approaches are often data-inefficient due to the semantic gap between abstract natural language instructions and concrete executions. In contrast, several works have demonstrated that using specially designed intermediate representations tailored to individual tasks [11, 29, 113] can help reduce this expense and improve performance, albeit at the cost of significant effort on the part of the researchers devising these


Procedure Library | PaP (§ 3)
Environment
I have to reheat the bread and prepare some fruits
Prepare a meal
natural language input x
HMN-Planner (§ 4)
plan
execute
an executable procedural action a^e
HMN-Reactor (§ 4)
predict
execute
This is a microwave! It can be opened and it is close

```
def udp_pickup_object(obj)
def udp_heat_object(obj)
def udp_cool_object(obj)
def udp_clean_object(obj)
...
```

```
udp_heat_object(bread)
udp_clean_object(apple)
...
```

```
def udp_put_object(obj, recep):
  atomic_navigate(recep)
  reactor = get_reactor("check_attr")
  attr = reactor(recep)
  if attr.openable and attr.close:
    atomic_open_object(recep)
    atomic_put_object(obj, recep)
  else:
    atomic_put_object(obj, recep)
```

```
attr.openable=True
attr.close=True
```

Figure 4.1: The proposed framework, containing a hierarchical library of procedures written as Python functions (§4.4). Coupled with this library is a hierarchical neural network (HMN, §4.5) with a PLANNER that constructs an executable procedure and REACTORS that react to the environment to resolve control flow.

methods.

In this chapter, we propose a framework to improve the execution of complex natural language commands (example in Fig. 4.1) by expressing *procedures as programs* (PaP) written in a high-level programming language like Python (§4.4). This makes it easy for human engineers to *express and leverage their hierarchical procedural knowledge*, and the execution of each program yields actions to accomplish a task described in NL. There are several merits to this approach. First, programs are inherently hierarchical; they apply nested function calls to realize higher-level functionality with multiple calls to lower-level functionality. Second, programs have built-in control-flow operators, making it possible to deal with multiple divergent situations without the loss of higher-level abstraction. Third, programs provide a flexible way to define, share and call different machine-learned components to perceive the environment through an embodied agent's executions. Finally, programs in a familiar high-level programming language are comprehensible and curatable, allowing for fast development on various tasks. These four features remain largely unexplored in the existing representations [11, 29, 113], as discussed further in §4.2.

Coupled with this representation, we propose a modeling paradigm of *hierarchical modular networks* (HMN; §4.5) that has (1) a learnable PLANNER that maps NL to the corresponding executable programs and (2) a collection of REACTORS that perceive the environment and provide context-sensitive feedback to decide the further execution of the program. Such modular design can facilitate training efficiency and improve the performance of each individual component [7].

We instantiate our framework on two task settings: the IQA dataset [59] where an agent explores the environment to answer questions regarding objects; and the ALFRED dataset [153], in which an agent must map natural language instructions to actions to complete household tasks (§4.6). In experiments (§4.7), we find that our framework outperforms the reactive baseline by a significant margin on both datasets, and is significantly more data-efficient. We also demonstrate the flexibility of our framework for fast iterative development of program libraries. We end with a discussion of the limitations of the framework and the potential solutions, paving the way for future works that scale our framework to more open-domain tasks (§7).

4.2 Contrast to Previous Formalisms

While designing intermediate representations that stand between NL and low-level actions *for individual tasks* has been studied in the literature, our goal is to design a framework that makes it simple to design such representations *for new tasks*, with a particular focus on capturing the hierarchical nature of procedures. In contrast to most previous works in this area, which employ relatively esoteric representation methods such as lambda calculus [11, 12], PaP uses widely-adopted general-purpose programming languages (*e.g.*, Python) to specify and represent hierarchical procedures. These are comprehensible to most engineers and do not require system designers to learn a new task-specific language. PaP also enable easy creation of more hierarchical procedures with *reusable sub-routines*. Existing works either do not model such sub-procedures as reusable components [113], or define procedures as a flat sequence of actions without any hierarchy [11, 34]. The hierarchical procedures with reusable sub-routines is also reminiscent of works in semantic parsing, which compose programs from idiomatic program structures [72, 152].

Additionally, PaP uses control flow with divergent branches to handle environment-specific variations of a high-level procedure. A single procedure could therefore dynamically adapt to a variety of environments following the branches triggered by the environments. This makes our representations more compact. This feature also allows developers to easily inject human priors of executions traces under different conditions, which might be challenging to learn in a data-efficient manner. To our best knowledge, this feature is largely unexplored in the literature on designing intermediate representations for agent control.

Finally, PaP provides a convenient interface for procedures to query and interact with task-specific situated components (*e.g.*, a visual component). Under PaP, situated components are exposed as pre-defined APIs, and can be easily called by high-level procedures. In contrast, existing works either require separate mechanisms to call such components [113], or the environment where they are expected to work is less complex, and thus the flexible use of a collection of situated components is not a necessity [29].

We can also view the PaP formalism as a way to construct behavior trees [43], which have been used in robotic planning and game design literature. We can use the off-the-shelf tools to convert the programs to abstract syntax trees (AST) which resemble these trees. Previous works on robotics also leverage planning domain definition language (PDDL) and answer set planners (ASP) for task planning [76], which is conceptually different from our formalism. PDDL+ASP searches for an action sequences based on the initial and the final states, while our formalism focuses on describing the actual procedure used to accomplish a task.

4.3 Task: Controlling Situated Agents

First, we define the task of controlling an agent in some situated environment $\mathbb{E}$ through natural language. The environment $\mathbb{E}$ provides a set of atomic actions $\mathcal{A}^a = \{a_1^a, a_2^a, \ldots\}$ to interact with the environment. Each atomic action can take zero or more arguments that specify which parts of the environment to which it is to be applied. We denote action a_i^a's jth argument as $r_{i,j}$. The specific type of each argument will depend on the action and environment; it could be discrete symbols, scalar values, tensors describing regions of the visual space, etc. Given a user intent $\boldsymbol{x}$, the control system aims at creating an atomic action sequence consisting of a sequence of actions $\boldsymbol{a} = [a_1, a_2, \ldots]$ $(a_i \in \mathcal{A}^a)$ and concrete assignments vfor each of these n actions. This action sequence is executed against the environment to achieve a result $\hat{y} = \mathbb{E}(\boldsymbol{a}, \mathrm{v})$, which is compared against a gold-standard result y using a score function $s(y, \hat{y})$. Action sequences realizing the intent will receive a high score, and those that do not will receive a low score.

4.4 Representing Procedures as Programs

Next, we introduce the main components of our formalism. A few examples are listed in Table 4.1.[1]

Interface to Atomic Actions $\mathcal{A}^a$ (C1) Atomic actions provide a medium for direct interaction with the environment. The call of an atomic action with proper argument types will invoke the corresponding execution in the environment.

[1]Since actions are implemented as functions, we use "action" and "function" interchangeably.

```python
# C1: an atomic action to toggle on an appliance
def atomic_toggle_on(obj):
    env.call("toggle_on", obj)
# C2: a procedural action to pick and then put an object
def udp_pick_and_put_object(obj, dst):
    udp_pickup_object(obj)
    udp_put_object(obj, dst)
# C3: an emptying receptacle procedure with for-loop
def udp_empty_recep(recep, dst):
    reactor = get_reactor("find_all_obj")
    obj_list = reactor(recep)
    for obj in obj_list:
        udp_pick_and_put_object(obj, dst)
# C4: a pickup object procedure with control flow
def udp_pickup_object(obj):
    atomic_navigate(obj)
    reactor1 = get_reactor("find_recep")
    reactor2 = get_reactor("check_obj_attr")
    recep = reactor1(obj)
    attr = reactor2(recep)
    if attr.openable and attr.close:
        atomic_open_object(recep)
        atomic_pickup_object(obj)
        atomic_close_object(recep)
    else: atomic_pickup_object(obj)
```

Table 4.1: Atomic and procedural action functions in Python, starting with `atomic` and udp respectively.

Procedural Actions $\mathcal{A}^p$ (C2-C4) Procedural actions describe abstractions of higher-level procedures composed of either lower-level procedures or atomic actions. Notably, lower-level procedures can be *re-used* across many higher-level procedures without re-definition. Formalizing the hierarchies in this compact way can not only facilitate the procedure library curation process but also potentially benefit automatic library induction (e.g. through minimal description length [50]).

Control-flow of $\mathcal{A}^p$ (C3-C4) There can be multiple execution traces to accomplish the same goal under different conditions. For example, picking up an object from inside a closed receptacle requires opening the receptacle first, while the open action is not required for objects not in a receptacle. To improve the *coverage* of procedural functions we leverage the built-in control flow of the host programming language to allow for conditional execution of environment-specific actions (**C4**). To deal with the repeated calls of the same routine, we further introduce for/while-loops. For example, **C3** works for emptying receptacles with variable number of objects without repeatedly writing down the `udp_pick_put_object`. Leveraging control flows to describe divergent procedural traces remains largely unexplored in previous works.

Call of Situated Components (C3-C4) The dynamic trigger of a control flow often remain unknown before the agent interacts with the environment. We introduce situated components to probe the environment and gather state information to guide program execution. In **C4**, the agent uses two different reactors to find the potential holder of an object (`reactor1`) and exam the holder's properties (`reactor2`). A reactor can be implemented in many ways (*e.g.*, using a neural network).

4.5 Hierarchical Modular Networks

This section introduces how to use the procedure library $\mathcal{A}$ to generate executable programs to complete tasks described in natural language $\boldsymbol{x}$. We propose a modeling method of *hierarchical modular networks* (HMN) that consists of two main components. First, there is a HMN-Planner that convert $\boldsymbol{x}$ to an executable procedural action $\boldsymbol{a}^e = \{a_1, a_2, ..., a_n\}$ where a_i either belongs to atomic functions $\mathcal{A}^a$ or procedural functions $\mathcal{A}^p$. We model the HMN-Planner as a sequence-to-sequence model where the encoder takes $\boldsymbol{x}$ as input, and the decoder generates one function a_i at a time from a constrained vocabulary $\mathcal{A}^p \bigcup \mathcal{A}^a$, conditioned on $\boldsymbol{x}$ and the action history $\{a_1, ..., a_{i-1}\}$.

Next, we define the collection of situated components, “reactors,” as HMN-Reactors. Each reactor is a classifier that predicts one or many labels given the observed information (*e.g.*, the NL input, the visual observation. For example, `reactor2` in **C4** in Table 4.1 probes the status of a receptacle based on receptacle name and the visual input. HMN-Reactors allows us to flexibly share the same reactor among different functions and design separated reactors to serve different purposes. For example in **C4**, we use two reactors to find the possible receptacle of an object (`reactor1`) and to perceive the open/closed status of a receptacle (`reactor2`) since these two tasks presumably require more mutually exclusive information. At the same time, we share `reactor2` to also probe the related `openable` property of a receptacle for more efficient parameter sharing. This sort of modular design leads to efficient training and improved performance [7].

4.6 Instantiations

In this section, we introduce two concrete realizations of the proposed framework over the IQA dataset [59] and the ALFRED dataset [153]. Both are based on egocentric vision in a high-fidelity simulated environment THOR [45].

4.6.1 IQA

IQA is a dataset for situated question answering with three types of questions querying (1) the existence of an object (*e.g.*, *Is there a mug?*), (2) the count of an object (*e.g.*, *How many mugs are there?*) and (3) whether a receptacle contains an object (*e.g.*, *Is there a mug in the fridge?*).

There are seven atomic actions in IQA, *i.e.*, `Moveahead`, `RotateLeft`, `RotateRight`, `LookDown`, `LookUp`, `Open` and `Close`; and all arguments are expressed through the unique object IDs (*e.g.*, `apple_1`). We further process the atomic navigation actions to a single atomic action `Navigate` with one argument `destination`, which moves the agent directly to the destination. This replacement is done by searching the scene and recording the coordinates of unmovable objects (*e.g.*, cabinet)

Procedure Library We design a procedure for each of the three types of questions in IQA, as shown in Table 4.2. Generally speaking, those procedures first search all or a subset of the receptacles (*e.g.*,`table`, `fridge`) in a scene for the target object (*e.g.*,`mug`), and then execute a question-specific intent (*e.g.*, existence-checking, counting). Table 4.2 shows the procedure for

```
# check existence of an object in the scene
def udp_check_obj_exist(obj):
    all_recep = udp_grid_search_recep()
    for recep in all_recep:
        rel = udp_check_relation(obj, recep)
        if rel == OBJ_IN_RECEP:
            return True
    return False
# check object inside receptacle
def udp_check_relation(obj, recep):
    atomic_navigate(recep)
    r1 = get_reactor("check_obj_attr")
    r2 = get_reactor("check_obj_recep_rel")
    attr = r1(recep)
    if attr.is_openable and attr.is_closed:
        atomic_open_object(recep)
        rel = r2(obj, recep)
        atomic_close_object(recep)
    else:
        rel = r2(obj, recep)
    return rel
```

Table 4.2: The procedural actions to answer the existence questions of the IQA dataset.

answering existence questions. Since the target object can be inside a receptacle (*e.g.*, fridge), we introduce control flow to decide whether to open and close a receptacle before and after checking its contents in sub-procedure `udp_check_relation`. Following the paper author's understanding of the three types of questions, these procedural functions were created without looking into any actual trajectories that answer these questions.

HMN The natural language questions x in IQA are generated with a limited number of templates. There are only seven receptacles, and three of them are openable. We thus use a rule-based HMN-PLANNER to map a template to one of the three high-level procedural actions (*i.e.*, existence, count and contain). Then, we design two reactors, each as a multi-classes classifier: ATTRCHECKER, which examines the properties (whether the object is openable) and the status (whether the object is opened) of an object, and RelChecker, which checks the spatial relation between two objects. Notably, we use *zero* IQA training data to build the HMN. Instead, it is made up of a few heuristic components based on the predictions of a pre-trained perception component.

4.6.2 ALFRED

ALFRED is a benchmark for mapping NL instructions to actions to accomplish household tasks in the situated environment (*e.g.*, heat an egg). Examples in ALFRED come with both single-sentence high-level intents describing a goal (*e.g.*, the NL input in Fig. 4.1), and more fine-grained, step-by-step instructions. In this paper we only use the high-level intents, a more realistic yet more challenging setting to study the effectiveness of our framework in encoding extra procedural knowledge for under-specified intents. Besides the seven atomic actions in the IQA dataset, ALFRED also introduces `Pickup`, `Put`, `ToggleOn`, `ToggleOff` for object interactions. ALFRED uses 2D binary tensor describing regions of the visual space as arguments. Similarly to IQA, we replace the navigation action with an atomic action `Navigate destination`. Previous works also apply similar replacement [79, 154] to allow the agent to proceed to a location without fail.

Procedure Library We create a procedure library for ALFRED by identifying idiomatic control flow and operations from a small set of randomly sampled examples. The library is designed with two goals in mind as discussed in §4.4: *reusability*, where a single function can be applied to multiple similar scenarios, and *coverage*, where a function should cover different execution trajectories under different conditions For instance, many tasks consist of a sub-routine to obtain an object by first navigating to the object and then picking up the object by hand, calling for a reusable procedure adaptable to those scenarios. Moreover, if an object is positioned inside a receptacle, picking up the object would require opening the receptacle first, an edge case that should be covered by relevant procedures (*e.g.*, **C4** in Table 4.1). Notably, we constrain the conditions of the control flow to the logic operation of the property values of objects (*e.g.*, `fridge.is_openable=True`).

In total, we define ten such procedural actions. This creation process was done by the first author, a graduate student proficient in Python, and took about two hours. This modest amount of time is partially due to PaP's intuitive interface that allows for quick summarization of complex procedures and partially due to ALFRED's relative simplicity; it has a limited number

```
# C1, heat an object with microwave
def udp_heat_object(obj):
    udp_pick_and_put_object(obj, microwave)
    atomic_toggleon_object(microwave)
    atomic_toggleoff_object(microwave)
# C2, prepare the receptacle for future interactions
def udp_prepare_recep(obj):
    reactor = get_reactor("check_obj_attr")
    attr = reactor(obj)
    if attr.is_openable and attr.is_closed:
        atomic_open_object(obj)
```

Table 4.3: Two procedural actions for ALFRED

of task types and consistent execution traces. A sanity check of an initial version of the library uncovered some mismatches. For example, a laptop should be closed before picking up, which was not captured by our library. We thus added a `udp_close_if_needed` function call before the `atomic_pick_object` in `udp_pick_object`. On one hand this increases the complexity of the library design process, but on the other hand it also demonstrates the flexibility of the PaP framework, as the necessary fixes could be done entirely by modifying the procedure library itself. §4.7.1 provides an end-to-end comparison with different procedural libraries.

To investigate the scalability of our annotation process, we also provided a similar guideline and the 21 examples to a separate programmer who does not have any prior knowledge to the dataset. We found that the programmer could quickly understand the PaP Python interface and issue reasonable procedural functions that highly resemble our own creations. This indicates the possibility to curate the procedure libraries with crowd-sourcing efforts.

HMN As discussed in §4.5, HMN-Planner generates an executable procedural action $\boldsymbol{a}^e$, given the natural language instruction $\boldsymbol{x}$. We implement our planner with a sequence-to-sequence model with attention [15].

Based on the construction of the procedure library and the required argument type, we design three reactors: AttrChecker, which has the same functionality as in IQA, ReFinder, which probes where the desired object lies by predicting a receptacle name from all available receptacles to the dataset, and MGenerator, which generates the 2D binary tensor representing the interaction region. Since ALFRED has much richer scene configurations and more diverse

		EX	CNT	CT
A3C	*seen*	-	-	-
	unseen	48.6	24.5	49.9
HIMN	*seen*	73.7	36.3	60.7
	unseen	68.5	30.4	58.7
Reactive	*seen*	50.0	25.1	49.6
	unseen	18.9	9.1	30.6
PaP-HMN	*seen*	**82.8**	**43.8**	**82.2**
	unseen	**83.8**	**45.2**	**83.1**
PaP$_{v0.1}$-HMN	*seen*	80.3	41.5	75.7

Table 4.4: The answer accuracy (%) over IQA dataset on existence (EX), counting (CNT) and contain (CT) questions. The results of AC3 and HIMN are from Gordon et al. [59]. **Bold** shows the best performance

objects than IQA, the reactors are fully implemented with neural networks.This demonstrates the flexibility of our framework to share, add and replace components to suit different situations.

4.7 Experiments

We compare our proposed framework with the baseline reactive agents that predicts a single atomic action at each time step. Notably, we apply the same pretrained vision models, pre-searched map and the `Navigate` atomic action used in PaP-HMN to the baseline to ensure a fair comparison. We attempt to answer the following research questions: (1) Does our framework performs better in complex tasks with inherent hierarchical structures, comparing to a purely reactive system? If so, in what way? (2) Can our framework leverage the procedural knowledge encoded in the procedure library and the modularity of its HMN to learn more efficiently? And (3) Can our framework accelerate the development of the task of interest?

4.7.1 Results on IQA

Results in Table 4.6 show that our framework yields the best performance across all models over different question types. Through error analysis, we observe that while the reactive model can generate reasonable action sequences *seen*, its answers are no better than a random guess. This indicates the inability of a reactive model to book-keep the observed objects in the memory. For *unseen*, we find that the baseline model skips predicting some receptacles or even generates syntactically invalid sequences (*e.g.*, functions without required arguments). This is surprising, since the reactive baseline is trained using the *canonicalized* action sequences according to the roll-out of the for-loops in the procedure library, which are quite regular. This indicates that even simple repeated procedures can be easily represented with a for/while-loop can still be challenging to a reactive agent implemented with a sequence-based backbone. The strong performance of PaP might seem unsurprising given that the library is tailored carefully to the domain. However, sophisticated models like HIMN [59] still struggle to capture such simple patterns, and there is not a straightforward way to plug the simple rules that we were easily able to describe in PaP in to improve its performance; PaP solves the easy problems so that an ML model can focus its effort on the more challenging problems that truly require learning (*e.g.*, object grounding).

Procedure Library Manipulation One advantage of our approach is that it decouples the reactors from the creation of the procedural knowledge, thus allowing plug-in update of the procedure library without time-consuming redesigning or retraining the reactors. Table 4.5 lists two versions of the procedure that decides the list of receptacles to enumerate, and the results of v0.1 are shown at the bottom of Table 4.4. In v0.1, the agent stands in its randomly initialized position, looks around, and detects receptacles. Only the detected receptacles are checked to answer the question. However, since not all receptacles are visible to the agent at the agent's initial point, such checking could be incomplete. We upgraded this function to the new version where the agent searches all possible positions of the scene and memorizes the unmovable receptacle positions. This process only happens once for a scene, and the searched map is stored for future uses. In this way, most receptacles are covered. This simple modification without changing the remaining parts of the framework improved the CT answer accuracy by 6.6% and improvement of around 2.5% over the other two question types.

[1] *unseen* features the out-of-distribution visual appearances and arrangements of objects, same for ALFRED

```
# v0.1: only scan at the start position
def udp_search_recep():
    r = get_reactor("detect_recep")
    receps = []
    for rotation in range(0, 360, 90):
        atomic_rotate(rotation)
        for horizon in [-30, 0, 30]:
            atomic_look(horizon)
            receps += r()
    return receps
# now: navigate to every reachable point and scan
def udp_grid_search_recep():
    if not done_search:
        all_receps = [] # global var
        for pos in reachable_pos:
            atomic_navigate_pos(pos)
            all_receps += udp_search_recep()
    return all_receps
```

Table 4.5: Two versions for getting receptacles.

4.7.2 Results on ALFRED

Table 4.6 lists the results. Our model yields a consistent gain over the baseline system on both splits.[2] In our analysis, we find that the Mask R-CNN vision model is the main bottleneck of both end-to-end systems, which we hypothesis is due to the sub-optimal transfer from the MSCOCO [98] to the ALFRED data. It frequently misclassifies the object types or does not recognize the object in the scene at all. This results in the failure of object grounding and thus the failure of the task completion. Since the development of a better object detector is somewhat orthogonal to our main contributions, to isolate the impact of using a weak object detector on the end-to-end performance, we replace the Mask R-CNN with an oracle object mask generator, which always localize and interact with the provided object name if the object is in view for all experiments below. We observe a larger performance gap using this oracle mask generator

[2]Singh et al. [156] predicts atomic navigation sequences (*e.g.*,`MoveAhead`) instead of `Navigate`. The agent struggles to navigate to destination with only high-level goal. This shows the difficulty of navigation under our experiment setting.

	seen	*unseen*
Singh et al. [156]	5.4	0.2
Reactive	21.0	5.6
PaP-HMN	**27.0**	**11.7**
Reactive + Oracle MG	40.7 (48.6)	36.4 (45.0)
PaP-HMN + Oracle MG	**54.5 (61.0)**	**51.3 (61.1)**

Table 4.6: The full task success rate SR (the partial task success rate, SSR, %) of the baseline reactive model and our model. MG represents the mask generator. **bold** shows the best performance for each setting.

as shown in the bottom half of Table 4.6. This gap suggests that procedural knowledge that could be summarized as several functions describable within a short period of time (in this case, ten functions in two hours) can still be difficult for a reactive system to capture. While the same procedural knowledge can be used in many cases with different environment dynamics, a reactive system struggle to distill such knowledge when interacting with highly diverse and dynamic environments.

Performance w.r.t. Action Length In Fig. 4.2, we further break down the results to buckets w.r.t the length of atomic action sequences (without arguments), which roughly represents the difficulty of a task. We observe consistent improvements over all buckets, This difference is even more obvious for challenging tasks with over 21 atomic actions. Our model maintains similar performance for such cases on *seen*, and being able to accomplish 30% tasks successfully on *unseen*, while the baseline can barely complete any task. These suggest our framework's stronger capacity to solve long-horizon tasks of deeper hierarchies.

Data Efficiency The hierarchical procedural knowledge could potentially allow the system to learn task completion in a data-efficient manner. We benchmark HMN with varying amounts of training data. As shown in Fig. 4.2, with 20% of the training data, our method exceeds the baseline with the full training set by a large margin (7.7% and 17.3% respectively). Furthermore, for *seen*, the baseline only obtains less than 60% SR with 20% training data, compared to the full data; our method could maintain around 90% SR of the full data setting. These strongly demonstrate the data efficiency of our method.

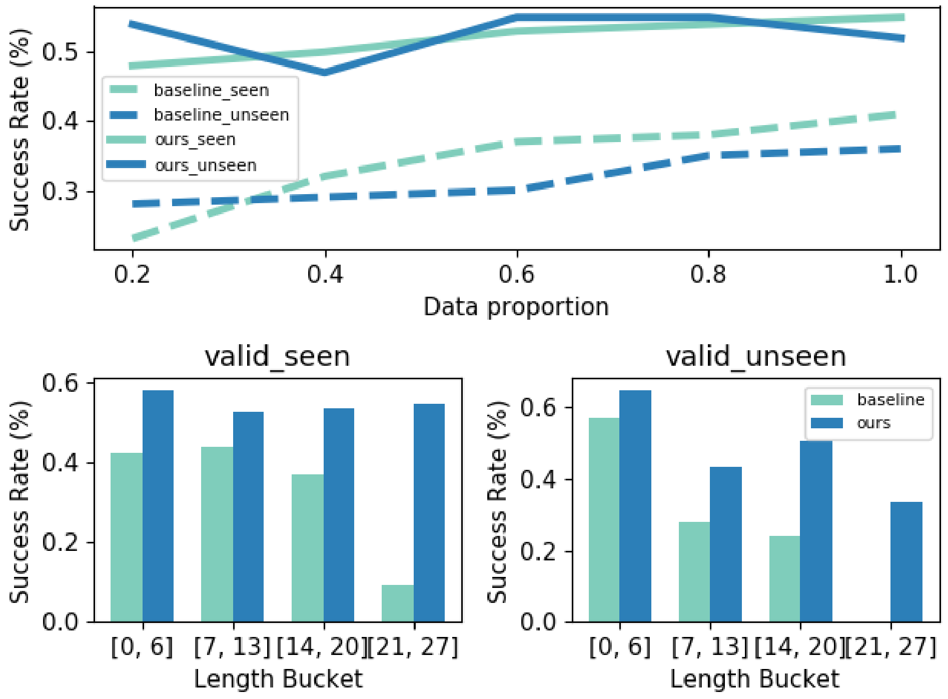


Figure 4.2: The SR (%) with proportions of the full training set (top) and on each length bucket of the *seen,unseen* (bottom).

Few-shot Generalization Next, we test if our framework can generalize to novel compositional procedures with relatively supervised examples. We design the few-shot experiments where a subset of the executable procedural actions ($\boldsymbol{a}^e$) are held out, and we sample at most 20 samples of each $\boldsymbol{a}^e$ and add them to the training set. We evaluate the model on these held-out $\boldsymbol{a}^e$. We use two strategies to choose the held-out set; the first randomly selects n $\boldsymbol{a}^e$; the other selects the longest n $\boldsymbol{a}^e$ ($n = 4/19$). PaP-HMN achieves **33.1** and **44.9** SR with these two strategies while the reactive baseline only reaches 13.9 and 3.3 respectively[3]. Our method consistently outperforms the baseline by a large margin on both settings, which strongly demonstrates our method's generalization ability in the few-shot scenario. The significant gain under the short to long setting shows our method's strong capacities in completing long-horizon tasks in a data-efficient way compared to the baseline.

Analysis Our framework brings several advantages. First, compared to low-level actions, the high-level procedural functions are better aligned with abstract NL inputs. This thus benefits the learning and the prediction of PLANNER. Second, programs maintain the consistency of the actions, while a reactive agent might make inconsistent predictions, especially arguments,

[3]For *random split*, we average over four different splits.

between actions. Finally, the modular design of Planner and the Reactors improve the robust behavior of the agent.

Next, we investigate failure cases. First, our ablation study shows that Planner correctly predicts 80% of executable procedural actions $\boldsymbol{a}^e$, and the failures are mainly due to rare words (*e.g.*, *soak a plate*). In addition, we manually annotated 50 failed examples whose $\boldsymbol{a}^e$ are correct. We found that 26 failures are due to the sub-optimal interaction positions of the receptacles that we compute during the pre-search phase. This causes the interaction with a visible object or receptacle to fail. The pre-search map also missed some objects, and navigating to these objects always failed. Besides, the Reactor prediction errors fail on 18 examples; ambiguous annotations caused two errors, and the wrong argument prediction of the Planner caused four errors.

Chapter 5

Few-shot Program Writing for Broader Reasoning Tasks

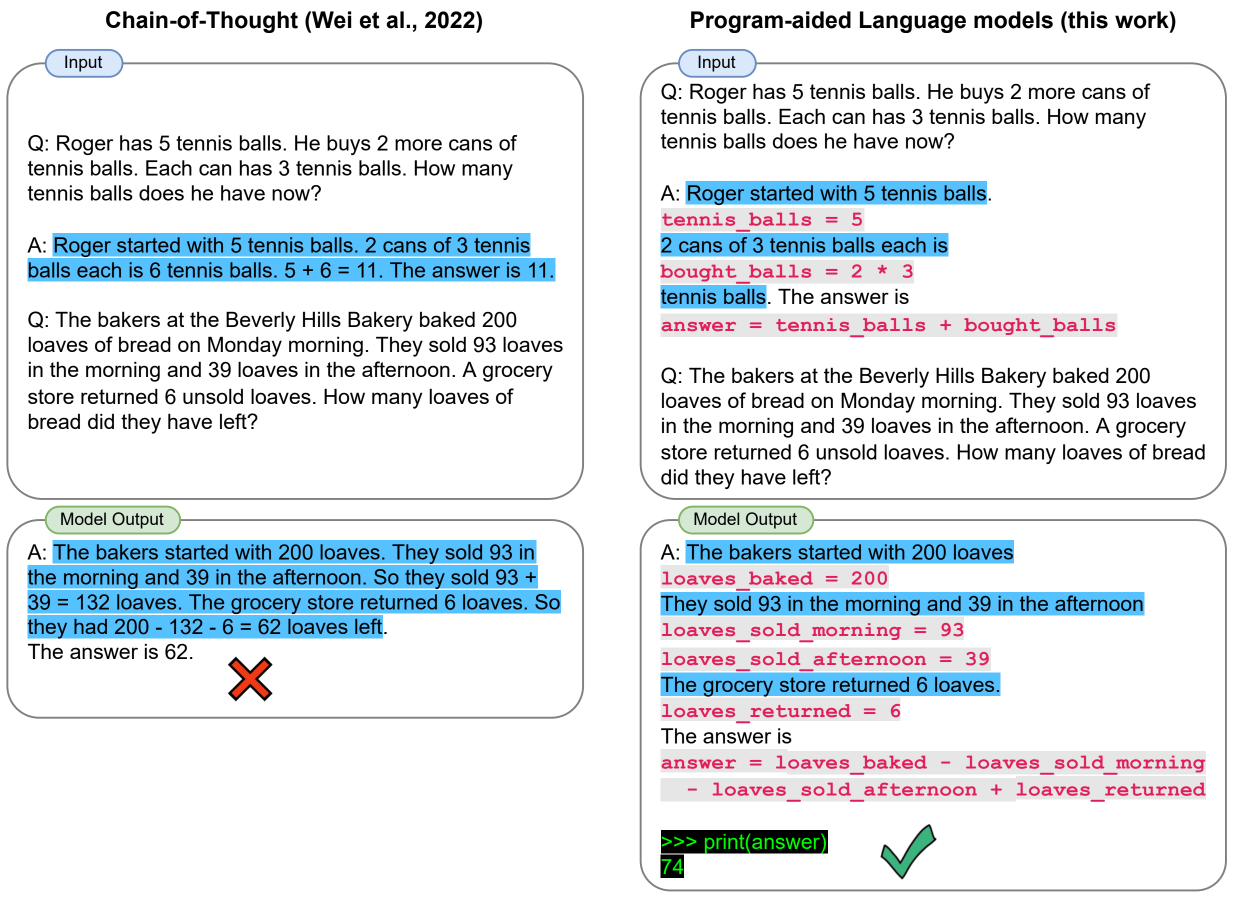
So far, we have demonstrated that human could quickly construct and effective refine the programs to increase the task success rate of situated agents, thanks to the readability of programs. However, high-quality program annotations often require a comprehensive understanding of the task, including its sub-tasks and potential branching situations. In this chapter, we investigate how to *automatically* write program solutions for reasoning tasks that require multiple steps. This work first appears in:

- Luyu Gao*, Aman Madaan*, Shuyan Zhou*, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR, 2023

5.1 Overview

Recent studies have shown that LLMs can achieve impressive success in various tasks, including commonsense [107, 147, 169], mathematical [94, 112, 174], and symbolic reasoning [2, 183], through few-shot prompting [26]. Therefore, we aim to explore the effectiveness of LLMs in writing programs for complex tasks with*only a few human demonstrations on specific instances.*

We propose Program-Aided Language model (PaL): a novel method that uses an LLM to read natural language problems and generate *programs* as reasoning steps, but offloads the *solution* step to a Python interpreter, as illustrated in Figure 5.1. This offloading leverages an LLM that can decompose a natural language problem into programmatic steps, which is fortunately



Chain-of-Thought (Wei et al., 2022)

Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 tennis balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?

Model Output

A: The bakers started with 200 loaves. They sold 93 in the morning and 39 in the afternoon. So they sold 93 + 39 = 132 loaves. The grocery store returned 6 loaves. So they had 200 - 132 - 6 = 62 loaves left.
The answer is 62.

Program-aided Language models (this work)

Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 tennis balls.
```
tennis_balls = 5
```
2 cans of 3 tennis balls each is
```
bought_balls = 2 * 3
```
tennis balls. The answer is
```
answer = tennis_balls + bought_balls
```

Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?

Model Output

A: The bakers started with 200 loaves
```
loaves_baked = 200
```
They sold 93 in the morning and 39 in the afternoon
```
loaves_sold_morning = 93
loaves_sold_afternoon = 39
```
The grocery store returned 6 loaves.
```
loaves_returned = 6
```
The answer is
```
answer = loaves_baked - loaves_sold_morning
  - loaves_sold_afternoon + loaves_returned
```

```
>>> print(answer)
74
```

Figure 5.1: A diagram illustrating PaL: Given a mathematical reasoning question, Chain-of-thought (left) generates intermediate reasoning steps of free-form text. In contrast, Program-aided Language models (PaL, right) generate intermediate steps *and* Python code. This shifts the role of *running* the reasoning steps from the language model to the Python interpreter. The final answer is obtained by running the generated reasoning chain. Chain-of-thought reasoning is highlighted in blue; PaL steps are highlighted in gray and pink; the Python interpreter run is highlighted in black and green.

available using contemporary state-of-the-art LLMs that are pre-trained on both natural language and programming languages [26, 30, 37]. While natural language understanding and decomposition require LLMs, solving and reasoning can be done with the external solver. This bridges an important gap in chain-of-thought-like methods, where reasoning chains can be correct but produce an incorrect answer.

We demonstrate the effectiveness of PaL across **13** arithmetic and symbolic reasoning tasks. In all these tasks, PaL using Codex [30] outperforms much larger models such as PaLM-540B using chain-of-thought prompting. For example, on the popular GSM8K benchmark, PaL achieves state-of-the-art accuracy, surpassing PaLM-540B with chain-of-thought by absolute 15% top-1 accuracy. When the questions contain large numbers, a dataset we call GSM-HARD, PaL outperforms CoT by an absolute 40%. We believe that this seamless synergy between a neural LLM and a symbolic interpreter is an essential step towards general and robust AI reasoners.

5.2 Background: Few-shot Prompting

Few-shot prompting leverages the strength of large-language models to solve a task with a set of k examples that are provided as part of the test-time input [26, 37, 102], where k is usually a number in the low single digits. These input-output examples $\{(x_i, y_i)\}_{i=1}^{k}$ are concatenated in a prompt $p \equiv \langle x_1 \cdot y_1 \rangle \parallel \langle x_2 \cdot y_2 \rangle \parallel \ldots \parallel \langle x_k \cdot y_k \rangle$. where "$\cdot$" denotes the concatenation of an input and output, and "$\parallel$" indicate the concatenation of different examples. During inference, a test instance x_{test} is appended to the prompt, and $p \parallel x_{test}$ is passed to the model which attempts to complete $p \parallel x_{test}$, and thereby generate an answer y_{test}. Note that such few-shot prompting does not modify the underlying LLM.

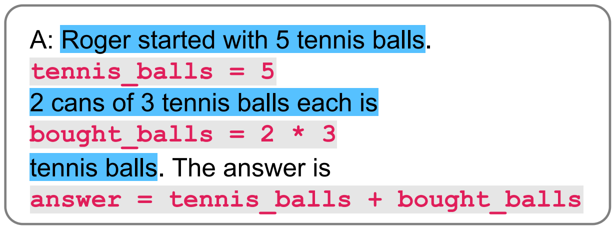
Wei et al. [170] additionally augment each in-context example with *chain of thought* (CoT) intermediate steps. Specifically, each in-context example in the CoT setup is a triplet $\langle x_i, t_i, y_i \rangle$, where x_i and y_i are input-output pair as before, and t_i is a natural language description of the steps that are needed to arrive at the output y_i from the input x_i. See Figure 5.1 for an example. With the additional "thoughts" t_i, the prompt is set to $p \equiv \langle x_1 \cdot t_1 \cdot y_1 \rangle \parallel \langle x_2 \cdot t_2 \cdot y_2 \rangle \parallel \ldots \parallel \langle x_k \cdot t_k \cdot y_k \rangle$.

During inference, the new question x_{test} is appended to the prompt as before and supplied to the LLM. Crucially, the model is tasked with generating *both* the thought t_{test} and the final answer y_{test}. This approach of prompting the model to first generate a reasoning process t_{test} improves the accuracy of the answer y_{test} across a wide range of tasks [165, 166, 170, 206].

5.3 Program-aided Language Models

In a Program-aided Language model, we propose to generate the thoughts t for a given natural language problem x as interleaved natural languagea () and programming language (PL) statements. Since we delegate the solution step to an interpreter, we do not provide the final answers to the examples in our prompt. That is, every in-context example in PaL is a *pair* $\langle x_i, t_i \rangle$, where $t_j = [s_1, s_2, \ldots, s_N]$ with each $s_i \in \text{NL} \cup \text{PL}$, a sequence of tokens in eithear or PL. The complete prompt is thus $p \equiv \langle x_1 \cdot t_1 \rangle \parallel \langle x_2 \cdot t_2 \rangle \parallel \ldots \parallel \langle x_k \cdot t_k \rangle$.

Given a test instance x_{test}, we append it to the prompt, and $p \parallel x_{test}$ is fed to the LM. We let the LM generate a prediction t_{test}, which contains both the intermediate steps *and* their corresponding programmatic statements.



```
A: Roger started with 5 tennis balls.
tennis_balls = 5
2 cans of 3 tennis balls each is
bought_balls = 2 * 3
tennis balls. The answer is
answer = tennis_balls + bought_balls
```

Figure 5.2: A close-up of a single example from a PaL prompt. Chain-of-thought reasoning is highlighted in blue, and PaL programmatic steps are highlighted in gray and pink

Example A close-up of the example from Figure 5.1 is shown in Figure 5.2. While chain-of-thought only decomposes the solution in the prompt into natural language steps such as Roger started with 5 tennis balls and 2 cans of 3 tennis balls each is 6, in PaL we also augment each such NL step with its corresponding programmatic statement such as tennis_balls = 5 and bought_balls = 2 * 3. This way, the model learns to generate a *program* that will provide the answer for the test question, instead of relying on LLM to perform the calculation correctly. We prompt the language model to generate NL intermediate steps using comment syntax (e.g. " # . . . " in Python) such they will be ignored by the interpreter. We pass the generated program t_{test} to its corresponding solver, we run it, and obtain the final run result y_{test}. In this work we use a standard Python interpreter, but this can be any solver, interpreter or a compiler.

Crafting prompts for PaL In our experiments, we leveraged the prompts of existing work whenever available, and otherwise randomly selected the same number (3-6) of examples as previous work for creating a fixed prompt for every benchmark. In all cases, we augmented the free-form text prompts into PaL-styled prompts, leveraging programming constructs such as `for` loops and dictionaries when needed. Generally, writing PaL prompts is easy and quick.

We also ensure that variable names in the prompt meaningfully reflect their roles. For example, a variable that describes the *number of apples in the basket* should have a name such as `num_apples_in_basket`. This keeps the generated code linked to the entities in the question. In Section 5.6 we show that such meaningful variable names are critical. Notably, it is also possible to incrementally run the PL segments and feed the execution results back to the LLM to generate the following blocks. For simplicity, in our experiments, we used a single, post-hoc, execution.

5.4 Experimental Setup

Data and in-context examples We experiment with three broad classes of reasoning tasks: (1) mathematical problems (§5.4.1) from a wide range of datasets including GSM8K [41], SVAMP [130], ASDIV [110], and MAWPS [84]; (2) symbolic reasoning (§5.4.2) from BIG-Bench Hard [159]; (3) and algorithmic problems (§5.4.3) from BIG-Bench Hard as well. For all of the experiments for which CoT prompts were available, we use the same in-context examples as used by previous work. Otherwise, we randomly sampled a fixed set of in-context examples, and used the same set for PaL and CoT.

```
# Q: Olivia has $23. She bought five bagels for $3 each.
# How much money does she have left?

money_initial = 23
bagels = 5
bagel_cost = 3
money_spent = bagels * bagel_cost
money_left = money_initial - money_spent
answer = money_left
```

Figure 5.3: Example prompt for the mathematical reasoning tasks, from the GSM8K benchmark.

Baselines We consider three prompting strategies: Direct prompting – the standard prompting approach using pairs of questions and immediate answers (e.g., 11) as in Brown et al. [26]; chain-of-thought (CoT) prompting [170]; and our PaL prompting. We performed greedy decoding from the language model using a temperature of 0. Unless stated otherwise, we used Codex (code-davinci-002) as our backend LLM for both PaL, Direct, and CoT. In datasets where results for additional base LMs, such as PaLM-540B, were available from previous work, we included them as CoT $_{\text{PaLM-540B}}$.

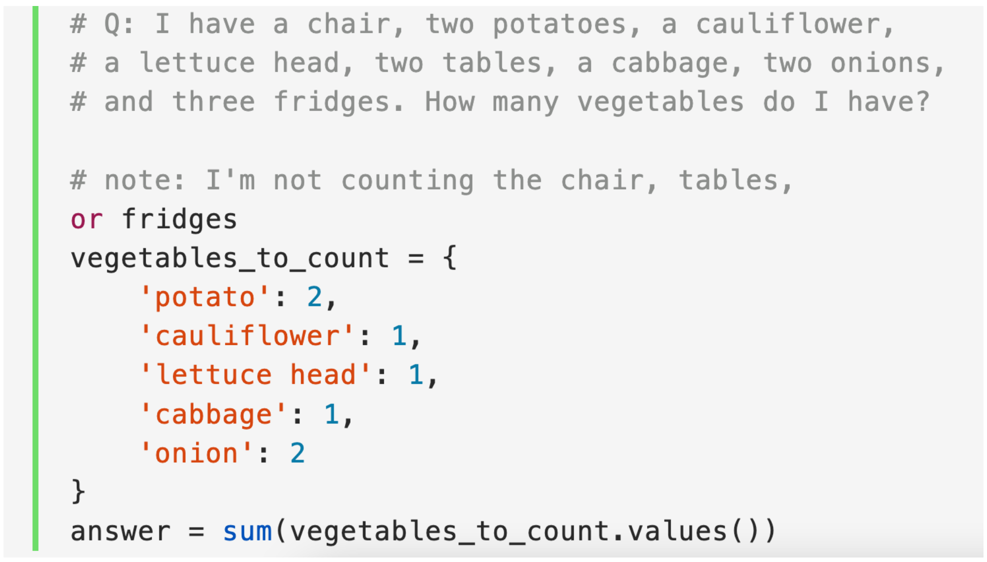
```
# Q: On the table, you see a bunch of objects arranged in a row:
# a purple paperclip, a pink stress ball, a brown keychain,
# a green scrunchiephone charger, a mauve fidget spinner,
# and a burgundy pen. What is the color of the object
# directly to the right of the stress ball?

# Put objects into a list to record ordering
objects = []
objects += [('paperclip', 'purple')] * 1
objects += [('stress ball', 'pink')] * 1
objects += [('keychain', 'brown')] * 1
objects += [('scrunchiephone charger', 'green')] * 1
objects += [('fidget spinner', 'mauve')] * 1
objects += [('pen', 'burgundy')] * 1

# Find the index of the stress ball
stress_ball_idx = None
for i, object in enumerate(objects):
    if object[0] == 'stress ball':
        stress_ball_idx = i

        break
# Find the directly right object
direct_right = objects[stress_ball_idx+1]
# Check the directly right object's color
answer = direct_right[1]
```

Figure 5.4: An example for a PaL prompt in the Colored Objects task. For space considerations, we omit the code that creates the list `objects`.

5.4.1 Mathematical Reasoning

We evaluate PaL on eight mathematical word problem datasets. Each question in these tasks is an algebra word problem at grade-school level. An example for a question and PaL example prompt is shown in Figure 5.3. We found that using explicit NL intermediate steps does not further benefit these math reasoning tasks, hence we kept only the meaningful variable names in the prompt.

GSM-HARD LLMs can perform simple calculations with *small* numbers. However, Madaan and Yazdanbakhsh [106] found that 50% of the numbers in the popular GSM8K dataset of math reasoning problems are *integers between 0 and 8*. This raises the question of whether LLMs



```
# Q: I have a chair, two potatoes, a cauliflower,
# a lettuce head, two tables, a cabbage, two onions,
# and three fridges. How many vegetables do I have?

# note: I'm not counting the chair, tables,
or fridges
vegetables_to_count = {
    'potato': 2,
    'cauliflower': 1,
    'lettuce head': 1,
    'cabbage': 1,
    'onion': 2
}
answer = sum(vegetables_to_count.values())
```

Figure 5.5: An example for a PaL prompt in the Object Counting task. The base LM is expected to convert the input into a dictionary where keys are entities and values are their quantities, while filtering out non-vegetable entities. Finally, the answer is the sum of the dictionary values.

can generalize to larger and non-integer numbers? We constructed a harder version of gsm8k, which we call gsm-hard, by replacing the numbers in the questions of gsm8k with larger numbers. Specifically, one of the numbers in a question was replaced with a random integer of up to 7 digits.

5.4.2 Symbolic Reasoning

We applied PaL to three symbolic reasoning tasks from BIG-Bench Hard [159], which involve reasoning about objects and concepts: (1) Colored Objects requires answering questions about colored objects on a surface. This task requires keeping track of relative positions, absolute positions, and the color of each object. Figure 5.4 shows an example for a question and example PaL prompt. (2) Penguins describes a table of penguins and some additional information in natural language, and the task is to answer a question about the attributes of the penguins, for example, "*how many penguins are less than 8 years old?*". While both Penguins and Colored Object tasks require tracking objects, Penguins describes *dynamics* as well, since the penguins in the problem can be added or removed. (3) Date is a date understanding task that involves inferring dates from natural language descriptions, performing addition and subtraction of relative periods of time, and having some global knowledge such as "how many days are there in February", and performing the computation accordingly.

	GSM8K	GSM-HARD	SVAMP	ASDIV	SINGLEEQ	SINGLEOP	ADDSUB	MULTI
Direct $_{\text{Codex}}$	19.7	5.0	69.9	74.0	86.8	93.1	90.9	44.0
CoT $_{\text{UL2-20B}}$	4.1	-	12.6	16.9	-	-	18.2	10.7
CoT $_{\text{LaMDA-137B}}$	17.1	-	39.9	49.0	-	-	52.9	51.8
CoT $_{\text{Codex}}$	65.6	23.1	74.8	76.9	89.1	91.9	86.0	95.9
CoT $_{\text{PaLM-540B}}$	56.9	-	79.0	73.9	92.3	94.1	91.9	94.7
CoT $_{\text{Minerva 540B}}$	58.8	-	-	-	-	-	-	-
PaL	**72.0**	**61.2**	**79.4**	**79.6**	**96.1**	**94.6**	**92.5**	**99.2**

Table 5.1: Problem solve rate (%) on mathematical reasoning datasets. The highest number on each task is in **bold**. The results for Direct and PaLM-540B are from Wei et al. [170], the results for LaMDA and UL2 are from Wang et al. [166], and the results for Minerva are from Lewkowycz et al. [94].

5.4.3 Algorithmic Tasks

Finally, we compare PaL and CoT on algorithmic reasoning. These are tasks where a human programmer can write a deterministic program with prior knowledge of the question. We experiment with two algorithmic tasks: Object Counting and Repeat Copy. Object Counting involves answering questions about the number of objects belonging to a certain type. For example, as shown in Figure 5.5: *"I have a chair, **two potatoes**, **a cauliflower**, **a lettuce head**, two tables, ... How many vegetables do I have?"*). Repeat Copy requires generating a sequence of words according to instructions.

5.5 Results

5.5.1 Math Results

Table 5.1 shows the following results: across all tasks, PaL using Codex sets a new few-shot state-of-the-art top-1 decoding across all datasets, outperforming CoT$_{\text{Codex}}$, CoT$_{\text{PaLM-540B}}$, and CoT$_{\text{Minerva 540B}}$ which was fine-tuned on explicit mathematical content.

Interestingly, CoT also benefits from Codex over PaLM-540B in some of the datasets such as asdiv, but performs worse than PaLM-540B in others such as svamp. Yet, using PaL further improves the solve rate across all datasets.

	Colored Object	Penguins	Date	Repeat Copy	Object Counting
Direct Codex	75.7	71.1	49.9	81.3	37.6
CoT LaMDA-137B	-	-	26.8	-	-
CoT PaLM-540B	-	65.1	65.3	-	-
CoT Codex	86.3	79.2	64.8	68.8	73.0
PaL Codex	**95.1**	**93.3**	**76.2**	**90.6**	**96.7**

Table 5.2: Solve rate on three symbolic reasoning datasets and two algorithmic datasets, In all datasets, PaL achieves a much higher accuracy than chain-of-thought. Results with closed models LaMDA-137B and PaLM-540B are included if available to public [159, 170].

gsm-hard On gsm-hard (Table 5.1), the accuracy of Direct drops dramatically from 19.7% to 5.0% (a relative drop of 74%), the accuracy of CoT drops from 65.6% to 20.1% (a relative drop of almost 70%), while PaL remains stable at 61.5%, dropping by only 14.3%. The results of CoT on gsm-hard did not improve even when we replaced its prompts with prompts that include large numbers. This shows how PaL provides not only better results on the standard benchmarks, but is also much more *robust*. In fact, since PaL offloads the computation to the Python interpreter, any complex computation can be performed accurately given the correctly generated program.

Large Numbers or Incorrect Reasoning? Are the failures on gsm-hard primarily due to the inability of LLMs to do arithmetic, or do the large numbers in the question "confuse" the LM which generates irrational intermediate steps? To investigate this, we evaluated the outputs generated by CoT for the two versions of the same question (with and without large numbers). We find that in 16 out of 25 cases we analyzed, CoT generates nearly identical natural language "thoughts", indicating that the primary failure mode is the inability to perform arithmetic accurately.

Multi-sample Generation As found by Wang et al. [166], chain-of-thought-style methods can be further improved by sampling $k > 1$ outputs, and selecting the final answer using majority voting. We thus repeated the greedy-decoding experiments using nucleus sampling [68] with $p = 0.95$ and $k = 40$ as in Lewkowycz et al. [94] and temperature of 0.7. As shown in Table 5.3, this further increases the accuracy of PaL from 72.0% to 80.4% on gsm8k, obtaining 1.9% higher accuracy than Minerva-540B using the same number of samples.

	GSM8K
CoT $_{\text{UL2-20B}}$	7.3
CoT $_{\text{LaMDA-137B}}$	27.7
CoT $_{\text{Codex}}$	78.0
CoT $_{\text{PaLM-540B}}$	74.4
CoT $_{\text{Minerva 540B}}$	78.5
PaL $_{\text{Codex}}$	**80.4**

Table 5.3: Problem solve rate (%) on GSM8K using `majority@40` [166]

5.5.2 Symbolic Reasoning & Algorithmic Tasks Results

Results for symbolic reasoning and algorithmic tasks are shown in Table 5.2. In Colored Objects, PaL improves over the strong CoT by 8.8%, and by 19.4% over the standard direct prompting. In Penguins, PaL provides a gain of absolute 14.1% over CoT. In Date, PaL further provides 11.4% gain over both CoT $_{\text{Codex}}$, $_{\text{PaLM-540B}}$, and $_{\text{LaMDA-137B}}$.

The two rightmost columns of Table 5.2 show that PaL is close to solving Object Counting, reaching 96.7% and improving over CoT by absolute 23.7%. Similarly, PaL vastly outperforms CoT by absolute 21.8% on Repeat Copy. Surprisingly, Direct prompting performs better than CoT on Repeat Copy. Yet, PaL improves over Direct by 9.3% in Repeat Copy.

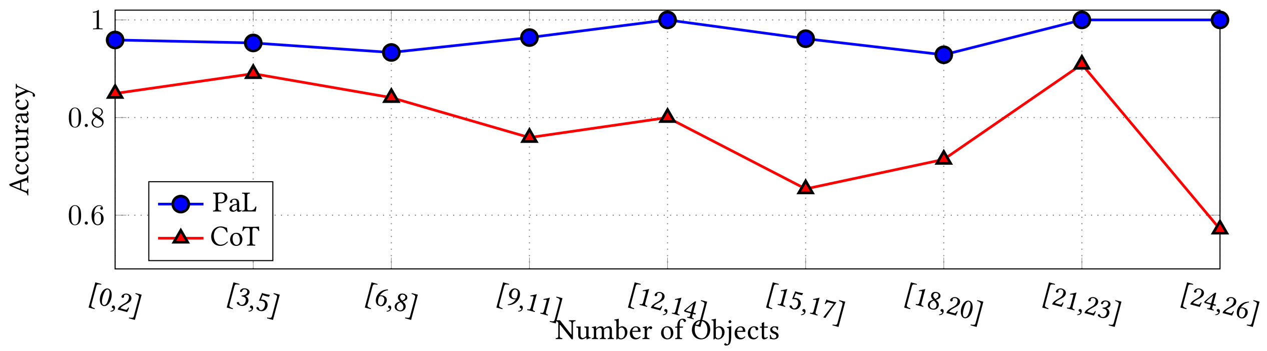


Figure 5.6: The solve rate on Colored Objects with respect to the number of objects included in the test question.

Is PaL sensitive to the complexity of the question? We examined how the performance of PaL and CoT changes as the complexity of the input question grows, measured as the number

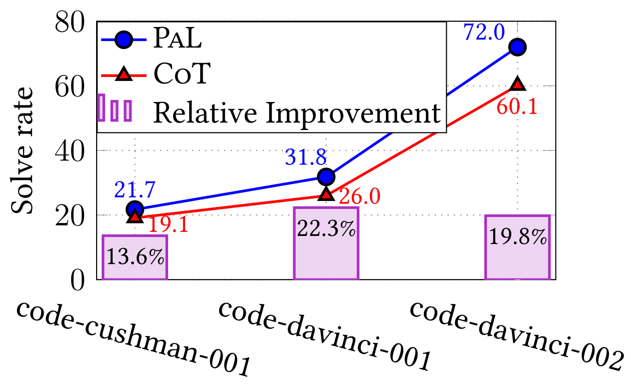


Figure 5.7: PaL with different models on GSM8K: though the absolute accuracies with code-cushman-001 and code-davinci-001 are lower than code-davinci-002, the relative improvement of PaL over CoT is consistent across models.

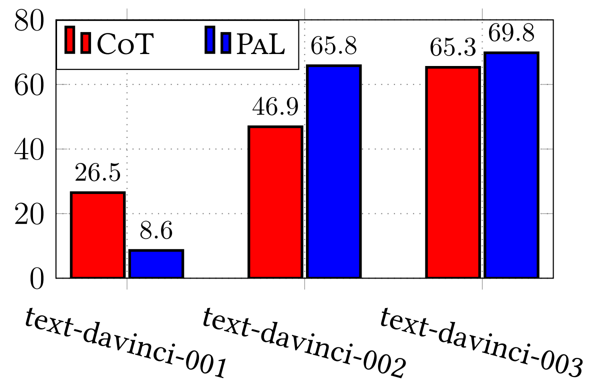


Figure 5.8: PaL with NL LMs on GSM8K: though CoT outperforms PaL with text-davinci-001, once the base LM is sufficiently strong, PaL is beneficial with text-davinci-002 and text-davinci-003 as well. That is, PaL is not limited to code-LMs only.

of objects in the question of Colored Objects. As shown in Figure 5.6, PaL is superior CoT across all input lengths. As the number of objects in the question increases, CoT's accuracy is unstable and drops, while PaL remains consistently close to 100%.

5.6 Analysis

Does PaL work with weaker LMs? In all our experiments in Section 5.5, PaL used the code-davinci-002 model; but can PaL work with weaker models of code? We compared PaL with CoT when both prompting approaches use the same weaker base LMs code-cushman-001 and code-davinci-001. As shown in Figure 5.7, even though the absolute accuracies of code-cushman-001 and code-davinci-001 are lower, the relative improvement of PaL over CoT remains consistent across models. This shows that PaL can work with weaker models, while its benefit scales elegantly to stronger models as well.

Does PaL work with LMs of natural language? We also experimented with PaL using the text-davinci series. Figure 5.8 shows the following interesting results: when the base LM's "code modeling ability" is weak (using text-davinci-001), CoT performs better than PaL.

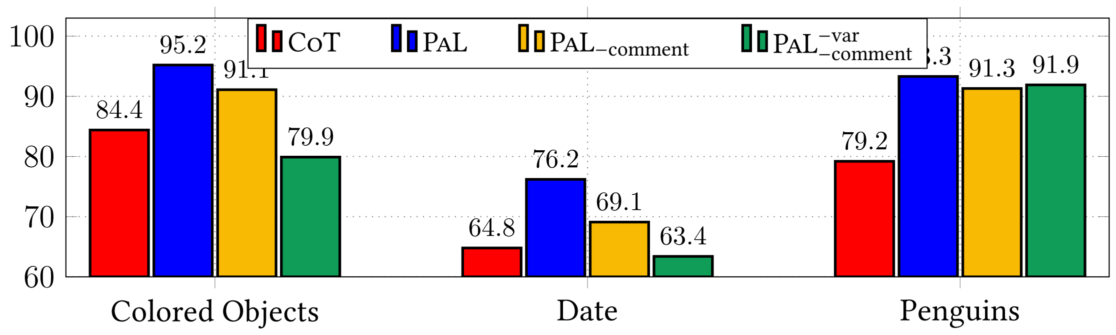


Figure 5.9: Ablation study of PaL prompt formats. We consider the original PaL prompt, it with natural language comments removed (PaL$_{-\text{comment}}$), and further variable names replaced with random character (PaL$^{-\text{var}}_{-\text{comment}}$). As a reference, we also show the CoT performance (blue).

However, once the LM's code modeling ability is sufficiently high (using `text-davinci-002` and `text-davinci-003`), PaL outperforms CoT, and PaL $_{\texttt{text-davinci-003}}$ performs almost as PaL $_{\texttt{code-davinci-002}}$. This shows that PaL is not limited to LMs of code, but it can work with LMs that were mainly trained for natural language, if they have a sufficiently high coding ability.

Is PaL better because of the Python prompt or because of the interpreter? We experimented with generating Python code, while requiring the neural LM to "execute" it as well, without using an interpreter, following Madaan et al. [107], Nye et al. [118]. We created prompts that are similar to PaL's, except that they *do include* the final answer. This resulted in a 23.2 solve rate on GSM8K, much lower than PaL (72.0), and only 4.5 points higher than Direct. These results reinforce our hypothesis that the main benefit of PaL comes from the synergy with the interpreter, and not only from having a better prompt.

Do variable names matter? In all our experiments, we used meaningful variable names in the PaL prompts, to ease the model's grounding of variables to the entities they represent. For the Python interpreter, however, variable names are meaningless. To measure the importance of meaningful variable names, we experimented with two prompts variants:

1. PaL$_{-\text{comment}}$ – the PaL prompt without intermediate NL comments.
2. PaL$^{-\text{var}}_{-\text{comment}}$ – the PaL prompt without intermediate NL comments *and* with variable names substituted with random characters.

The results are shown in Figure 5.9. In Colored Objected and Date, removing intermediate NL comments but keeping meaningful variable names (PaL$_{-\text{comment}}$) – slightly reduces the results compared to the full PaL prompt, but it still achieves higher accuracy than the baselines CoT. Removing variable names as well (PaL$^{-\text{var}}_{-\text{comment}}$) further decreases accuracy, and performs worse than CoT. Since variable names have an important part in code quality [58, 160], meaningful variable names are only expected to ease reasoning for Codex, which was trained on mostly meaningful names, as was also found by Madaan et al. [107].

Author Contributions The key idea of this work emerged during a discussion between the co-first authors. The three co-first authors then experimented with this idea on various datasets, including those in BigBench-Hard, other math-related datasets and multi-hop QA datasets. Shuyan Zhou was later responsible for conducting a more in-depth analysis of why the method works, through both qualitative and quantitative studies.

Part III

Knowledge Base of Hierarchical Procedures

Chapter 6

Discovering Hierarchies of Procedures from Semi-structured Web Data

The third goal of this thesis is to teach AI agents to perform new tasks without direct demonstrations from humans. The key idea is to *leverage the ubiquitous human-authored knowledge on the Internet*. This chapter examines what knowledge resources AI agents can use, discusses why these resources may not be readily usable, and explores how to mitigate these deficiencies. This work first appears in:

- Shuyan Zhou, Li Zhang, Yue Yang, Qing Lyu, Pengcheng Yin, Chris Callison-Burch, and Graham Neubig. Show me more details: Discovering hierarchies of procedures from semi-structured web data. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2998–3012, Dublin, Ireland, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.214. URL https://aclanthology.org/2022.acl-long.214

6.1 Overview

A procedure includes some *steps* needed to achieve a particular *goal* [114]. Procedures are inherently hierarchical: a high-level procedure is composed of many lower-level procedures. For example, a procedure with the goal *make videos* consists of steps like *purchase a camera*, *set up lighting*, *edit the video*, and so on, where each step itself is a procedure as well. Such hierarchical relations between procedures are recursive: the lower-level procedures can be further decomposed into even more fine-grained steps: one may need to *arrange the footage* in order to *edit the video*.

Relatively little attention has been paid to hierarchical relations in complex procedures in the field of NLP. Some work performed a shallow one-level decomposition and often required costly resources such as human expert task-specific annotation [38, 194, 201]. More attention has been paid in fields adjacent to NLP. For example, Lagos et al. [89] and Pareti et al. [127] both create hierarchical structures in how-to documents by linking action phrases in one procedure to another procedure or by linking steps in how-to articles to resources like DBPedia [13]. This kind of linking is helpful for explaining complex steps to readers who do not have prior knowledge of the topic being explained.

In this chapter, we revisit this important but understudied task to develop a simple and effective algorithm (Figure 6.1) to construct a hierarchical knowledge-base (KB) for over $110k$ complex procedures spanning a wide range of topics from wikiHow, a large-scale how-to website that has recently become a widely-used resource in NLP [191, 195, 196, 213].[1] From each wikiHow article which represents a procedure, we follow Zhang et al. [196] and extract the title as the goal (*e.g.*, g_1 in Figure 6.1), and the paragraph headlines as steps (*e.g.*, $s_1 \dots s_n$). Next, we decompose the steps by linking them to articles with the same or a similar goal (*e.g.*, s_1 to g_2). The steps of the linked article are treated as the finer-grained steps (s_i to s_j) of the linked step (s1). In this way, the procedural hierarchies go from shallow (B1) to deep (B4).

To link steps and article goals, we employ a retrieve-then-rerank approach, a well-established paradigm in related tasks [70, 173]. Our hierarchy discovery model (§6.3) first *independently* encodes each step and goal in wikiHow and searches the k nearest goals of similar meaning for each step (B2). Then, it applies a dedicated *joint* encoder to calculate the similarity score between the step and each candidate goal, thus reranking the goals (B3). This pipeline can efficiently search over a large candidate pool while accurately measuring the similarity between steps and goals. With each step linked to an article goal, a hierarchical KB of procedures is thus constructed.

We evaluate our KB both intrinsically and extrinsically. Intrinsically, the discovered links can be directly used to complete missing step-goal hyperlinks in wikiHow, which have been manually curated (B5). Our proposed method outperforms strong baselines (*e.g.*, Lagos et al. [89]) according to both automatic and human evaluation, in terms of recall and usefulness respectively (§6.4, §6.5). Extrinsically, we consider the task of retrieving instructional videos given textual queries. We observe that queries that encode deeper hierarchies are better than those

[1] www.wikihow.com

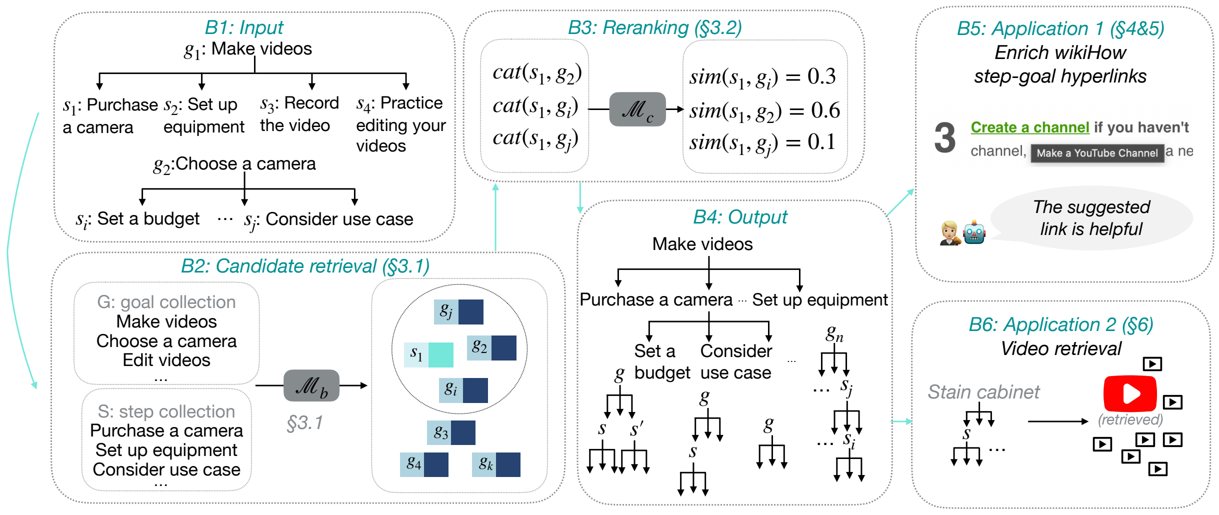


Figure 6.1: The overview of our proposed method. The input (Block1) and output (B4) of the hierarchy discovery model (B2, B3) and the applications (B5, B6) of the hierarchical knowledge base.

that do not (§6.6). This provides evidence that our KB can bridge the high-level instructions and the low-level executions of procedures, which is important for applications such as robotic planning.

6.2 Problem Formulation

We represent a procedure as a tree where the root node n represents a goal and its children nodes $\texttt{Ch}(n)$ represent the steps of n. We formulate the hierarchy discovery task as identifying the steps among $\texttt{Ch}(n)$ that can themselves be a goal of some other finer-grained steps (sub-steps), which are inserted into the tree.

While this formulation could potentially be used on any large collection of procedures, we specifically focus on wikiHow. As shown in B1 of Figure 6.1, each article comprises a goal (g), and a series of steps ($\texttt{Ch}(g)$). Therefore, each article forms a procedure tree of depth one.

We denote the collection of all goals and steps in wikiHow as G and S respectively. Our hierarchy discovery algorithm aims to link a step $s_i \in S$ to a goal $g \in G$ such that g has the same meaning as s_i. It then treats $\texttt{Ch}(g)$ as $\texttt{Ch}(s_i)$. Given that g and s_i are both represented by textual descriptions, the discovery process can be framed as a *paraphrase detection task*. This discovery process can be applied recursively on the leaf nodes until the resulting leaf nodes reach the desired granularity, effectively growing a hierarchical procedure tree (B4 of Figure 6.1).

6.3 Hierarchy Discovery Model

For each of the 1.5 million steps in the wikiHow corpus, we aim to select one goal that expresses the same procedure as the step from over $110k$ goals. We propose a simple and efficient method to deal with such a large search space through a two-stage process. First, we perform *retrieval*, encoding each step and goal *separately* in an unsupervised fashion and select the k most similar goals for each step s. This process is fast at the expense of accuracy. Second, we perform *reranking*, *jointly* encoding a step with each of its candidate goals in a supervised fashion to allow for more expressive contextualized embeddings. This process is more accurate at the expense of speed, since calculating each similarity score requires a forward pass in the neural network. The goal with the highest similarity score is selected and the step is expanded accordingly, as in B4 of Figure 6.1.

6.3.1 Retrieval

In the first stage, we independently encode each step $s \in S$ and goal $g \in G$ with a model $\mathcal{M}_b$, resulting in embeddings $\boldsymbol{e}_{s_1}, \boldsymbol{e}_{s_2}, ..., \boldsymbol{e}_{s_n}$ and $\boldsymbol{e}_{g_1}, \boldsymbol{e}_{g_2}, ..., \boldsymbol{e}_{g_m}$. The similarity score between s and g is calculated as the cosine similarity between $\boldsymbol{e}_s$ and $\boldsymbol{e}_g$. We denote this first-stage similarity score as $\text{sim}_1(s, g)$. Using this score, we can obtain the top-k most similar candidate goals for each step s, and we denote this candidate goal list as $\mathbb{C}(s) = [g_1, ..., g_k]$. To perform this top-$k$ search, we use efficient similarity search libraries such as FAISS [78].

We instantiate $\mathcal{M}_b$ with two learning-based paraphrase encoding models. The first is the SP model [171, 172], which encodes a sentence as the average of the sub-word unit embeddings generated by SentencePiece [86]. The second is SBERT [139], which encodes a pair of sentences with a siamese BERT model that is finetuned on paraphrase corpus. For comparison, we additionally experiment with search engines as $\mathcal{M}_b$, specifically Elasticsearch with the standard BM25 weighting metric [140]. We index each article with its title only or with its full article. We also experiment with Bing Search API where we limit the search to wikiHow website only[2]. The BM25 with the former setting resembles the method proposed by Lagos et al. [89].

[2]www.bing.com

6.3.2 Reranking

While efficient, encoding steps and goals independently is likely sub-optimal as information in the steps cannot be used to encode the goals and vice-versa. Therefore, we concatenate a step with each of its top-k candidate goals in $\mathtt{C}(s)$ and feed them to a model $\mathcal{M}_c$ that jointly encodes each step-goal pair. Concretely, we follow the formulation of Wu et al. [173] to construct the input of each step-goal pair as:

`[CLS]` *ctx* `[ST]` *step* `[ED]` *goal* `[SEP]`

where `[ST]` and `[ED]` are two reserved tokens in the vocabulary of a pretrained model, which mark the location of the step of interest. *ctx* is the context for a step (*e.g.*, its surrounding steps or its goal) that could provide additional information. The hidden state of the `[CLS]` token is taken as the final contextualized embedding. The second-stage similarity score is calculated as follows:

$$\text{sim}_2(s, g_i) = \text{proj}(\mathcal{M}_c(s, g_i)) + \lambda \text{sim}_1(s, g_i) \tag{6.1}$$

where $\text{proj}(\cdot)$ takes an d-dimension vector and turns it to a scalar with weight matrix $W \in \mathcal{R}^{d \times 1}$, and λ is the weight for the first-stage similarity score. Both W and λ are optimized through backpropagation (see more about labeled data in §6.4.1).

With labeled data, we finetune $\mathcal{M}_c$ to minimize the negative log-likelihood of the correct goal among the top-k candidate goal list, where the log-likelihood is calculated as:

$$ll(s, g_i) = -\log\left(\text{softmax}\left(\frac{\text{sim}_2(s, g_i)}{\sum_{g_j \in \mathtt{C}(s)} \text{sim}_2(s, g_j)}\right)\right) \tag{6.2}$$

Compared to the randomly sampled in-batch negative examples, the top-k candidate goals are presumably harder negative examples [80] and thus the model must work harder to distinguish between them. We will explain the extraction of the labeled step-goal pairs used to train this model in §6.4.1.

Concretely, we experiment with two pretrained models as $\mathcal{M}_c$, specifically BERT-base [47] and DeBERTa-large finetuned on the MNLI dataset [67]. We pick them due to their high performance on various tasks [200]. [3]

In addition, we consider including different *ctx* in the reranking input. For each step, we experiment with including no context, the goal of the step, and the surrounding steps of the step within a window-size n (n=1).

[3] https://cutt.ly/oTx5gMM. BERTScore measures the semantic similarity between a pair of texts, similar to the objective of our reranking.

6.3.3 Unlinkable Steps

Some steps in wikiHow could not be matched with any goal. Such steps are *unlinkable* because of several reasons. First, the step itself might be so fine-grained that further instructions are unnecessary (e.g. *Go to a store*). Second, although wikiHow spans a wide range of complex procedures, it is far from comprehensive. Some goals simply do not exist in wikiHow.

Hence, we design a mechanism to predict whether a step is linkable or not explicitly. More specifically, we add a special token `unlinkable`, taken from the reserved vocabulary of a pre-trained model, as a placeholder "goal" to the top-k candidate goal list $\texttt{C}(s)$, and this placeholder is treated as the gold-standard answer if the step is determined to be unlinkable. The similarity score between a step and this placeholder goal follows Equation 6.1 and $\text{sim}_1(s, \texttt{unlinkable})$ is set to the lowest first-stage similarity score among the candidate goals retrieved by the first-stage model. Accurately labeling a step as `unlinkable` is non-trivial – it requires examining whether the step can be linked to any goal in G. Instead, we train the model to perform this classification by assigning `unlinkable` to steps that have a ground-truth goal but this goal does not appear in the top-k candidate goal list. The loss follows Equation 6.2.

6.4 Automatic Step Prediction Evaluation

We evaluate the quality of our construct hierarchical KB both intrinsically and extrinsically. Intrinsically, we perform human evaluation and automatic evaluation using the manually curated step-goal hyperlinks in wikiHow. Extrinsically, we conduct retrieving instructional videos using queries that encode hierarchical knowledge. In the following sections, we demonstrate the qualitative evaluations and defer the in-depth analysis and case study to appendix.

6.4.1 Labeled Step-goal Construction

In wikiHow, there are around $21k$ steps that already have a hyperlink redirecting it to another wikiHow article, populated by editors. We treat the title of the linked article as the ground-truth goal for the step. For example, as in B5 of Figure 6.1, the ground-truth goal of the step *Create a channel* is *Make a Youtube Channel.* We build the training, development and test set with a 7:2:1 ratio.

Model	**R@1**	**R@10**	**R@30**
SP	35.8	64.4	72.5
SBert	30.6	53.3	63.4
BM25 (goal only)	30.5	51.6	61.1
BM25 (article)	9.3	35.3	49.2
Bing Search	28.0	47.9	-
BERT	50.7	69.4	-
DeBERTa	**55.4**	**71.9**	-
− surr	54.3	71.6	-
− goal	55.0	71.5	-
− both	52.4	71.0	-
+ unlinkable	50.4	71.6	-
+ $\lambda = 0$	51.9	71.4	-

Table 6.1: The recall@n for different models on the test set. The top half are with paraphrase retrieval only and the bottom half are with taking the top-30 candidate goals generated by the best model (SP) and adding the reranking model. The best performance recall is **bold**. "surr" denotes the surrounding steps of the query step.

6.4.2 Results

Table 6.1 lists the recall of different models without or with the reranking. Precision is immaterial here since each step has *only one* linked article.

Candidate Retrieval The SP model achieves the best recall of all models, outperforming SBert by a significant margin. Models based on search engines with various configurations, including the commercial Bing Search, are less effective. In addition, BM25 (goal only), which does not consider any article content, notably outperforms BM25 (article) and Bing Search, implying that the full articles may contain undesirable noise that hurts the search performance. This interesting observation suggests that while commercial search engines are powerful, they may not be the best option for specific document retrieval tasks such as ours.

[3]We are unable to get the top-30 results from Bing search because the web queries only return top-10 search results.

Reranking We select the top-30 candidate goals predicted by the SP model as the input to the reranking stage. The recall@30 of the SP model is 72.5%, which bounds the performance of any reranker.[4] As seen in the bottom half of Table 6.1, reranking is highly effective, as the best configuration brings a 19.6% improvement on recall@1, and the recall@10 almost reaches the upper bound of this stage. We find that under the same configuration, DeBERTa-large finetuned on MNLI [67] outperforms BERT by 1.7% on recall@1, matching the reported trends from BERTScore.[5]

In §6.5, we will demonstrate that this explicit unlinkable prediction is overall informative to distinguish steps of the two types through crowdsourcing annotations. We empirically find that setting the weight of $\text{sim}_1(s, g)$ (λ) to 0 is beneficial in the unlinkable prediction setting.

6.5 Manual Step Prediction Evaluation

The automatic evaluation strongly indicates the effectiveness of our proposed hierarchy discovery model. However, it is not comprehensive because the annotated hyperlinks are not exhaustive. We complement our evaluation with crowdsourced human judgments via Amazon Mechanical Turk (MTurk).

Each example of annotating is a tuple of a step, its original goal from wikiHow, and the top-ranked goal predicted by one of our models. For each example, we ask three MTurk workers to judge whether the steps in the article of the linked goal are exact, helpful, related, or unhelpful with regard to accomplishing the queried step.

We select SP, DeBERTa, and DeBERTa with unlinkable prediction and $\lambda = 0$ (DeBERTa-ul) for comparison. We attempt to answer the following questions. First, does the performance trend shown in automatic evaluation hold in human evaluation? Second, can the unlinkable predictions help avoid providing users with misleading information [136]?

For the purpose of the second question, we separate the examples into two groups. One contains linkable examples. Namely, those whose top-1 prediction is not predicted as `unlinkable` by the DeBERTa-ul model. Ideally, the linked articles from these examples should be helpful. The other group contains unlinkable examples. For these, we evaluate the second-highest ranked prediction of the DeBERTa-ul model. Ideally, the linked articles from these examples should be unhelpful.

The corresponding crowd judgment is shown in Figure 6.2. Comparing the models, the

[4] We only experiment with SP because it is the best retrieval model, providing a larger improvement headroom.

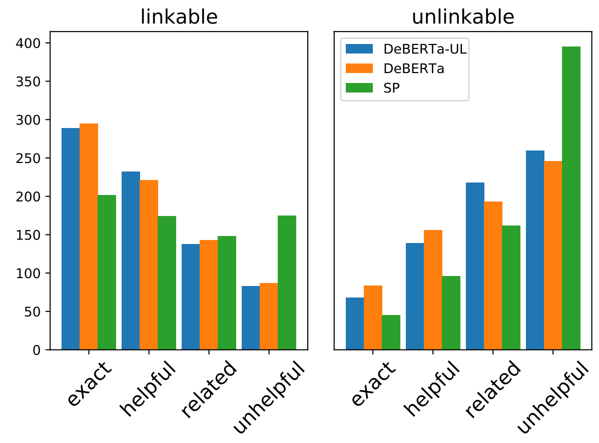


Figure 6.2: Crowd workers' ratings of step-goal links predicted by our models. The left graph shows steps linked to *some* goals by the DeBERTa-UL model, while the right shows steps those predicted as `unlinkable`.

DeBERTa model and the DeBERTa-UL model have similar performance, while greatly outperforming the SP model. This shows that our proposed model decomposes much more helpful finer-grained steps to assist users with tasks, similar to the trend observed in our automatic evaluation. Comparing the two graphs, it is apparent that when the DeBERTa-UL model predicts `unlinkable` for a step, the suggested decompositions of all models are more likely to be unhelpful. This implies the high precision of the `unlinkable` prediction, effectively avoiding misleading predictions. Note that our study does not explicitly require subjects to carry out the task, but only annotates whether they find the instructions helpful.

6.6 Application to Video Retrieval

In addition to intrinsic evaluation, we take a further step to study the usefulness of our open-domain hierarchical KB to downstream tasks. We select video retrieval as the extrinsic evaluation task, which aims at retrieving relevant how-to videos for a textual goal to visually aid users. More formally, given a textual goal g, the task is to retrieve its relevant videos v_g from the set of all videos, with a textual query q. Intuitively, our KB can be useful because videos usually contain finer-grained steps and verbal descriptions to accomplish a task. Therefore, the extra information presented in decomposed steps could benefit retrieving relevant videos.

Query	R/P@1	R/P@10	R/P@25	R/P@50	MR
L$_0$	2.2/89.2	19.2/78.1	39.9/66.0	56.6/48.2	79.49
L$_1$	2.2/88.0	19.2/78.0	40.1/66.4	58.1/49.6	75.79
FIL-L$_1$	2.2/**89.9**	20.2/81.7	43.1/71.2	63.2/53.8	66.32
FIL-L$_2$	2.2/89.4	**20.3/82.7**	**43.9/72.3**	**65.0/55.2**	**63.38**
L$_0$	12.1/81.7	59.8/42.8	71.9/20.8	77.9/11.3	41.60
L$_1$	11.8/79.7	61.2/43.9	74.1/21.4	80.5/11.6	36.70
FIL-L$_1$	12.4/83.7	66.0/47.3	77.4/22.4	82.9/**12.0**	33.35
FIL-L$_2$	**12.5/84.4**	**66.1/47.7**	**78.0/22.5**	**83.3/12.0**	**32.30**
L$_0$	11.4/82.6	59.2/45.2	71.8/22.1	77.8/12.0	43.11
L$_1$	11.2/81.3	60.4/46.2	73.8/22.7	79.9/12.3	38.19
FIL-L$_1$	**11.7/85.1**	64.8/49.5	77.2/23.8	82.2/**12.7**	34.76
FIL-L$_2$	11.6/84.5	**65.5/50.0**	**77.9/24.0**	**82.7/12.7**	**34.13**

Table 6.2: The Recall/Precision@N (%, ↑) and mean rank (MR, ↓) with different queries on the relevant video retrieval task on the training (top), development (middle) and the test set (bottom). The best performance on each set is **bold**.

6.6.1 Dataset Construction

We use Howto100M [111] for evaluation. It is a dataset of millions of instructional videos corresponding to over $23k$ goals. We construct our video retrieval corpus by randomly sampling $1,000$ goals (*e.g.*, *record a video*) with their relevant videos. The relevant videos $\boldsymbol{v}_g = \{v_1, v_2, ..., v_n\}$ of each goal g in the dataset are obtained by selecting the top 150 videos among the search results of the goal on YouTube.[5] For each goal g, we randomly split its relevant videos $\boldsymbol{v}_g$ into three sub-sets $\boldsymbol{v}_g^{\text{tr}}$, $\boldsymbol{v}_g^{\text{dev}}$ and $\boldsymbol{v}_g^{\text{test}}$ with a ratio of 7.5:1.25:1.25, as the training, development, and testing sets.

[5]Although the relevance between a goal and a video is not explicitly annotated in the Howto100M dataset, we argue that with the sophisticated engineering of the YouTube video search API and hundreds of thousands user clicks, the highly ranked videos likely demonstrate the queried goal.

6.6.2 Setup

Since our KB is fully textual, we also represent each video textually with its automatically generated captions. For the search engine, we use Elasticsearch with the standard BM25 metric [140].[6] We denote the relevance score calculated by BM25 between the query q and a textually represented video v as $\text{Rel}(q, v)$. We experiment with four different methods, which differ in how they construct the query q:

$\textsc{l}_0$: Goal only. The query is the goal g itself. This is the minimal query without any additional hierarchical information. The relevance score is simply $\text{Rel}(q, v) = \text{Rel}(g, v)$.

$\textsc{l}_1$: Goal + Children. The query is a concatenation of the goal g and its immediate children steps $\texttt{Ch}(g)$. This query encodes hierarchical knowledge that already exists in wikiHow. The relevance score is then defined as a weighted sum, $\text{Rel}(q, v) = w_g \text{Rel}(g, v) + w_s \sum_{s \in \texttt{Ch}(g)} \text{Rel}(s, v)$. The weights w_g and w_s are tuned on a development set and set to 1.0 and 0.1 respectively.

$\textsc{Fil-l}_1$: Goal + Filtered children. The query is a concatenation of the goal g and a filtered sequence of its children $\texttt{Ch}(g)$. Intuitively, decomposing a goal introduces richer information but also introduces noise, since certain steps may not visually appear at all (*e.g.*, *enjoy yourself*). Therefore, we perform filtering and only retain the most informative steps, denoted by $\texttt{Ch}'(g)$. Specifically, to construct $\texttt{Ch}'(g)$ for a goal g, we use a hill-climbing algorithm to check each step s from $\texttt{Ch}(g)$, and include s into the query only if it yields better ranking results for the ground-truth videos in the training set $\boldsymbol{v}_g^{\text{train}}$.The relevance score is defined as $\text{Rel}(q, v) = w_g \text{Rel}(g, v) + w_s \sum_{s \in \texttt{Ch}'(g)} \text{Rel}(s, v)$, where w_g is set to 1.0 and w_s is set to 0.5 after similar tuning.

$\textsc{Fil-l}_2$: Goal + Filtered children + Filtered grand-children. The query is the concatenation of the goal g and a filtered sequence of its immediate children $\texttt{Ch}(g)$ and grandchildren $\texttt{Ch}(s)$ $(s \in \texttt{Ch}(g))$. These filtered steps are denoted by $\texttt{Ch}'(g + \texttt{Ch}(g))$. This two-level decomposition uses the knowledge from our KB, therefore including lower-level information about the execution of the goal. We perform the same filtering algorithm as in $\textsc{Fil-l}_1$, and we define $\text{Rel}(q, v) = w_g \text{Rel}(g, v) + w_s \sum_{s \in \texttt{Ch}'(g + \texttt{Ch}(g))} \text{Rel}(s, v)$. w_g is set to 1.0 and w_s is set to 0.5.

[6] We find the performance of a neural model (BERT finetuned on query/video caption pairs) significantly lower than BM25 and therefore, we only report the results with BM25.

6.6.3 Results

We report the precision@N, recall@N and mean rank (MR) following existing work on video retrieval [104]. Table 6.2 lists the results. Many steps do not have corresponding executions in the videos and become noisy steps in the L_1 queries. More interestingly, we observe that queries using deeper hierarchies (FIL-L_2) outperform the shallower ones (FIL-L_1) in most cases. This is probably due to the fact that how-to videos usually contain detailed (verbal) instructions of a procedure, which are better aligned with more fine-grained steps found in FIL-L_2.

In our qualitative study, we investigate how FIL-L_2 queries with deeper hierarchies help retrieval. Table 6.3 list FIL-L_1 and FIL-L_2 queries for two goals. We find that the FIL-L_2 queries are more informative and cover more aspects. For example, the FIL-L_2 queries for *stain cabinet* and *make avocado fries* consist of the preparation, actual operations, and the post-processing steps, while the FIL-L_1 query only contains the first one. In addition, we search the goals on Google and list the key moments of some randomly sampled videos.[7] These key moments textually describe the important clips of the videos, and therefore they presumably also serve as the query for the goal. We find that the FIL-L_2 query of *make avocado fries* explains a few necessary steps to accomplish this goal, while the key moment is mostly composed of the ingredients of this dish. This comparison suggests the potential integration of our induced hierarchical knowledge to identify key moments in videos in the future.

6.7 Decomposition Analysis

In this section, we study the properties of the hierarchies. First, what kind of steps are likely to be linked to another goal and are thus decomposed? Second, what do the decomposed steps look like?

We group steps into two clusters. The first contains the immediate steps of a goal ($s \in \texttt{Ch}(g)$) whose prediction is not `unlinkable`. The second contains the decomposed steps of the steps in the first cluster ($s' \in \texttt{Ch}(s)$). We use spaCy [69] to extract and lemmatize the verb in each step and rank the verbs by their frequency in each cluster. Next, the top-100 most frequent verbs in each cluster are selected and we measure the rank difference of these verbs in the two clusters. Figure 6.3 plots the verbs with largest rank difference. We observe that verbs that convey complex actions and intuitively consist of many other actions become

[7]Key moments are either identified manually or are extracted automatically by YouTube. https://cutt.ly/qTcxSi6

Goal	Stain Cabinet
FIL-L1	Purchase some stain colors to test
FIL-L2	FIL-L1 + Buy cloth with which to apply the stain Unscrew the cabinet from the wall Clean your workspace
KM	Remove the doors Sanding the front Top coat Finished look
Goal	**Make Avocado Fries**
FIL-L1	Bake the avocado fries until they are golden Dip the avocado wedges into the egg and then the breadcrumbs
FIL-L2	FIL-L1 + Preheat the oven Peel and pit the avocados Cut your avocado in half and remove the stone Let rise Finished, cool and enjoy
KM	2 large avocados ... pinch of salt, pinch of pepper two eggs, beaten ... bake at 425F 20 min until golden bros ...

Table 6.3: The queries and the key moments (KM) for two goals. "..." represents the omission of steps that describe the ingredients to save space. The first selected video is `h9k0T25_NxA` and the second is `o7uVUmPph6I`.

less frequent after the decomposition (*e.g.*, decorate). On the other hand, verbs that describe the action itself gain in frequency after the decomposition (*e.g.*, push, hold, press). This observation follows our assumption that the decomposition would lead to more fine-grained realizations of a complex procedure. Some other more abstract actions such as "learn" and "decide" also increase

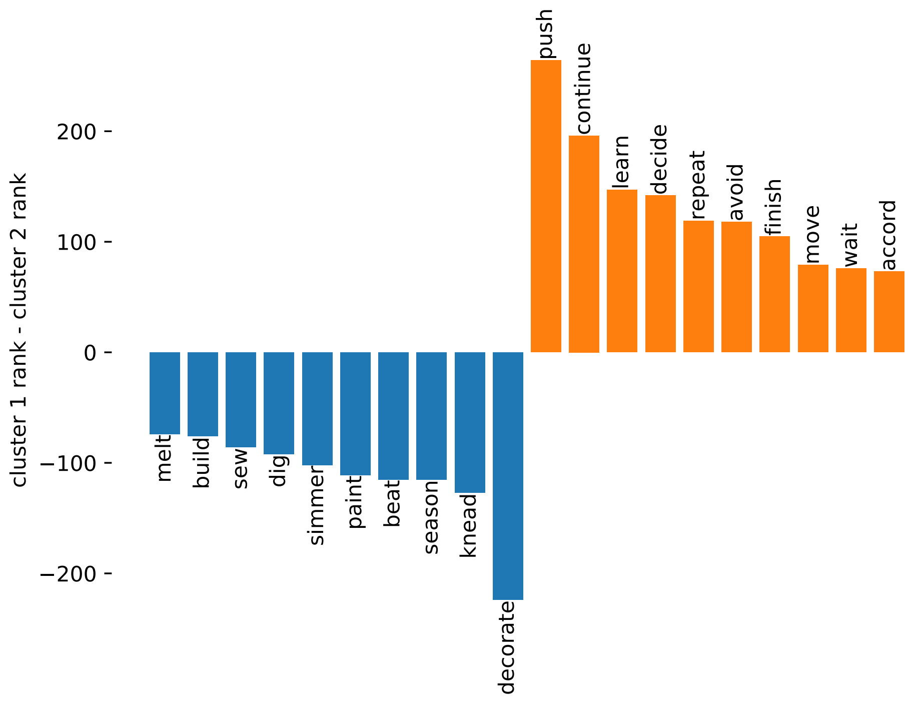


Figure 6.3: The verbs with largest rank difference in two clusters. The blue bars are words becoming less frequent in cluster 2 (decomposed steps) and the orange bars are words becoming more frequent.

in frequency, as some low-level goals are explained with more complex steps.

Author Contributions Shuyan Zhou proposed the idea of defining the hierarchies of procedures from wikiHow, designed and implemented the two-stage method and performed the automatic and downstream task evaluations. Li Zhang provides key insights from his earlier work [196] and conducted the human evaluation of our method.

Part IV

New Knowledge Acquisition without Direct Demonstrations

Chapter 7

Generating Code with Unseen Usages by Retrieving the Docs

Regardless of the impressive capabilities of LLMs in learning from only a few examples, LLMs are limited by a knowledge cutoff, as they can only be trained with knowledge that exists in a specific snapshot of the training corpus. As the world evolves, new knowledge emerges and existing knowledge may be updated, pretrained models inherently cannot generalize and effectively use new knowledge that was not included in their training. Luckily, new knowledge are likely to be recorded in some form of text, and it can be easily added to an unstructured text knowledge base using a single insertion action. In light of this, we propose a solution to combat knowledge cutoff problem based on retrieval, where an agent retrieves relevant text information from the knowledge base to supplement their existing knowledge and improve performance on a given task. Specifically, we apply this approach to the natural language to code generation task, where the relevant knowledge corresponds to libraries and functions. This work first appears in:

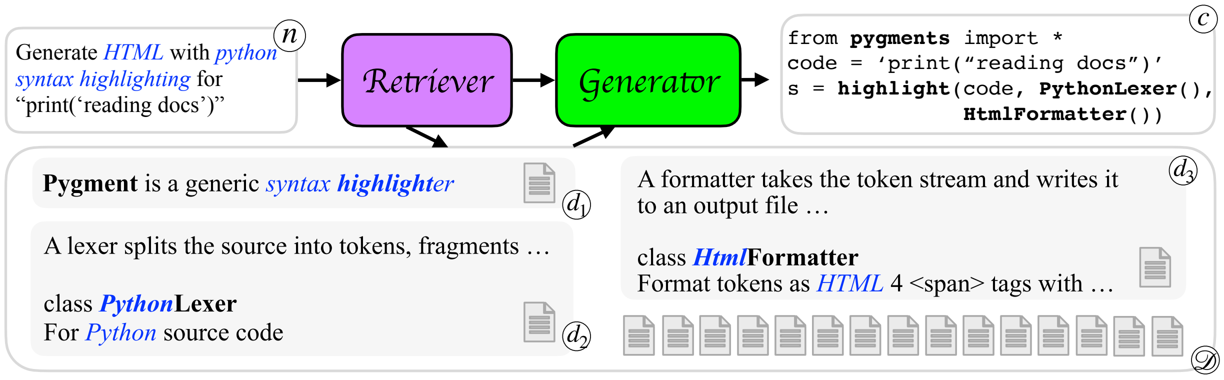
- Shuyan Zhou, Uri Alon, Frank F. Xu, Zhengbao Jiang, and Graham Neubig. Docprompting: Generating code by retrieving the docs. In *International Conference on Learning Representations (ICLR)*, 2023

7.1 Overview

Many existing code generation models either learn directly from input-output pairs provided as training data [3, 4, 24, 73, 167, 177, 185], or learn the mapping between input and output implicitly from naturally occurring corpora of intertwined natural language and code [14, 117]. Nevertheless, all these works assume that *all libraries and function calls were seen in the training data*; and that at test time, the trained model will need to generate only *seen* libraries and function calls. However, new functions and libraries are introduced all the time, and even a seen function call can have unseen arguments. Thus, these existing models *inherently cannot* generalize to generate such unseen usages.

In contrast to these existing models, human programmers frequently refer to manuals and documentation when writing code [92, 119]. This allows humans to easily use functions and libraries they have never seen nor used before. Inspired by this ability, we propose DocPrompting: a code generation approach that learns to retrieve code documentation before generating the code. An overview of our approach is illustrated in Figure 7.1: First, a document *retriever* uses the NL intent $\textcircled{n}$ to retrieve relevant code documentation $\{\textcircled{d_1}, \textcircled{d_2}, \textcircled{d_3}\}$ from a documentation pool $\textcircled{\mathcal{D}}$. Then, a code *generator* uses these docs in its prompt to generate the corresponding code $\textcircled{c}$. The documentation pool serves as an external data store that can be updated frequently with new contents (e.g., documentation of newly released libraries), without re-training any model component. This way, DocPrompting can leverage newly added documentation, and it can generate code containing unseen and unused functions and libraries. DocPrompting is general and applicable to any programming language and underlying base architecture. To the best of our knowledge, this is the *first* demonstration of leveraging documentation in models of code explicitly and effectively.

We demonstrate the effectiveness of DocPrompting on two NL→code benchmarks and tasks, across two programming languages, and using several base models: GPT-Neo [20], T5 [134], CodeT5 [167], Fusion-in-Decoder [74]), and Codex [31]. Further, we experiment with both sparse retrievers such as BM25 [141] and dense retrieval models such as SimCSE [57]. Finally, we introduce *two new benchmarks* for retrieval-based code generation: (a) in Bash, we curate a new benchmark by crawling the `tldr` repository, and constructing the training/development/test splits without overlapping commands; (b) in Python, we re-split the popular `CoNaLa` benchmark [186] by making every test example contain at least one Python function that is not seen in the training data. Models that use DocPrompting consistently outperform their base models that generate code solely based on the NL intents. Using DocPrompting improves strong base models such as CodeT5 by 2.85% in pass@1 (52% relative gain) and 4.39% in pass@10 (30% relative gain) in execution-based evaluation in `CoNaLa`; on the new `tldr` dataset, DocPrompting improves CodeT5 and GPT-Neo-1.3B by up to absolute 6.9% exact match. We release our new benchmarks, including annotation of oracle documents for each


Generate HTML with python syntax highlighting for "print('reading docs')"
n
Retriever
Generator
c
Pygment is a generic syntax highlighter
d1
A lexer splits the source into tokens, fragments …
class PythonLexer
For Python source code
d2
A formatter takes the token stream and writes it to an output file …
class HtmlFormatter
Format tokens as HTML 4 <span> tags with …
d3
D

```
from pygments import *
code = 'print("reading docs")'
s = highlight(code, PythonLexer(),
              HtmlFormatter())
```

Figure 7.1: DocPrompting: given an NL intent ⓝ, the retriever retrieves a set of relevant documentation {ⓓ1,ⓓ2, ⓓ3} from a documentation pool Ⓓ. Then, the generator generates the code ⓒ based on the NL and retrieved docs. DocPrompting allows the model to generalize to previously unseen usages by reading those docs. *Italic blue* highlights the shared tokens between NL and docs; **Bold** shows shared tokens between docs and the code snippet.

example and pools of documentation, to serve as a test-bed for future retrieval-based code generation models.

7.2 Code Generation by Reading the Docs

Our underlying assumption is that code documentation is the most exhaustive yet succinct resource for most libraries and programming languages [143], and that documentation allows to effectively generalize to unseen libraries and functions [53]. We follow the retrieve-then-generate paradigm [65, 93], focusing on retrieving *documentation*. In this section, we describe the general approach of DocPrompting; in §7.3 and §7.6.2, we elaborate and experiment with practical implementations of DocPrompting.

Formulation Given NL intent n, our goal is to generate a corresponding code snippet c written in some programming language (PL) such as Python. We assume that a model has access to a collection of code documentation $\mathcal{D}$. Each document $d_i \in \mathcal{D}$ describes the usage of a library, a function, or an argument in that PL. The construction of $\mathcal{D}$ is flexible: it can either be a comprehensive set of all available libraries and functions in a PL, or a customized subset for the scope of a specific project.

7.2.1 Background: Retrieval-Conditioned Generation

Although a model may use the entire collection of documents $\mathcal{D}$, only a few documents in $\mathcal{D}$ are relevant for any particular intent. Further, it is usually computationally infeasible to directly condition on the entire, unbounded, collection of documents while making predictions. Thus, we first let the model *select* a subset of documents $\mathcal{D}_n = \{d_1, d_2, .., d_k\} \subseteq \mathcal{D}$ that are potentially relevant given n, and refer to this subset while generating c.

Overall, we decompose the probability of generating c into the probability of choosing a particular subset of documents $P(\mathcal{D}_n \mid \mathcal{D}, n)$, and the probability of generating the code conditioned on the intent and the selected documents $P(c \mid \mathcal{D}_n, n)$; finally, we marginalizing over all $\mathcal{D}_n \subseteq \mathcal{D}$:

$$P(c \mid \mathcal{D}, n) = \sum_{\mathcal{D}_n \subseteq \mathcal{D}} P(c \mid \mathcal{D}_n, n) \cdot P(\mathcal{D}_n \mid \mathcal{D}, n) \tag{7.1}$$

assuming that c is independent of $\mathcal{D}$ given $\mathcal{D}_n$ (that is, $(c \perp\!\!\!\perp \mathcal{D} \mid \mathcal{D}_n)$). Since enumerating all possible subsets $\mathcal{D}_n$ is computationally infeasible, we follow the common practice and approximate the marginalization over $\mathcal{D}_n$ in Equation (7.1) by taking the most probable subset of retrieved documents $\hat{\mathcal{D}}_n$, and then conditioning the prediction of c on these most likely documents:

$$\hat{\mathcal{D}}_n := \text{argmax}_{\mathcal{D}_n \subseteq \mathcal{D}} P(\mathcal{D}_n \mid \mathcal{D}, n) \qquad P(c \mid \mathcal{D}, n) \approx P(c \mid \hat{\mathcal{D}}_n, n) \cdot P(\hat{\mathcal{D}}_n \mid \mathcal{D}, n) \tag{7.2}$$

7.2.2 DocPrompting: Generating Code by Retrieving the Docs

Equation 7.2 implies that DocPrompting relies of two main components: A *retriever* $\mathcal{R}$ retrieves relevant documents $\hat{\mathcal{D}}_n$ given the intent n; and a *generator* $\mathcal{G}$ generates the code snippet c conditioned on the retrieved documents $\hat{\mathcal{D}}_n$ and the intent n, which compose a new prompt. Specifically, $\mathcal{R}$ computes a similarity score $s(d_i, n)$ between a intent n and every document $d_i \in \mathcal{D}$. Thus, the subset $\hat{\mathcal{D}}_n \subseteq \mathcal{D}$ is the top-k documents with the highest similarity scores: $\hat{\mathcal{D}}_n = \textit{top-}k_{d_i \in \mathcal{D}}(s(d_i, n))$.

An overview of our approach is illustrated in Figure 7.1: given the intent *Generate HTML with python syntax highlighting for "print('reading docs')"*, the retriever $\mathcal{R}$ retrieves three relevant documents: d_1 describes the syntax highlighting library `pygments`, d_2 describes the class `PythonLexer`, and d_3 describes the `HtmlFormatter` class. Given these docs and the intent, the generator $\mathcal{G}$ generates the code snippet c, which uses `PythonLexer` and `Html-Formatter` from the `pygment` library.

7.3 Practical Instantiations of DocPrompting

DocPrompting is a general approach that is not bound to any specific model choices, and it can be instantiated with any base retriever and generator. This section presents the concrete instantiations of $\mathcal{R}$ and $\mathcal{G}$ that we found to provide the best performance in our experiments.

7.3.1 Retriever Instantiation

We experiment with two main types of retrievers: *sparse* retrievers and *dense* retrievers. As our sparse retriever, we use Elasticsearch[1] with the standard BM25 [141]. This retriever represents documents using sparse features that rely on word frequencies, such as BM25 and TF-IDF.

As our dense retriever, we follow prior work [32, 57, 80]: given a triplet $(n, c, \mathcal{D}_n^*)$, where $\mathcal{D}_n^*$ are the oracle docs for n, each $d_i^+ \in \mathcal{D}_n^*$ and n form a *positive* pair (n, d_i^+), while each $d_j^- \notin \mathcal{D}_n^*$ and n form a *negative* pair $\left(n_i, d_j^-\right)$. We train the retriever in a contrastive fashion where the similarity score of a positive pair is maximized while that of in-batch negative pairs is minimized. For a pair (n_i, d_i^+), the loss function is defined as:

$$\mathcal{L}^r = -\log \frac{\exp\left(\text{sim}(\boldsymbol{h}_n, \boldsymbol{h}_{d_i^+})\right)}{\exp\left(\text{sim}(\boldsymbol{h}_n, \boldsymbol{h}_{d_i^+})\right) + \sum_{d_j^- \in \mathcal{B}/\mathcal{D}_n^*} \exp\left(\text{sim}(\boldsymbol{h}_n, \boldsymbol{h}_{d_j^-})\right)} \tag{7.3}$$

where $\boldsymbol{h}_x$ is the representation of x computed by a neural encoder, and $\mathcal{B}$ are positive docs for other examples in the batch. We define $\text{sim}(\boldsymbol{h}_x, \boldsymbol{h}_y)$ as the cosine similarity between $\boldsymbol{h}_x$ and $\boldsymbol{h}_y$.

We use all (n_i, d_i^+) in the training set as our supervised training dataset. Additionally, we use all sentences in the documentation pool for weak supervision: Following Chen et al. [32] and Gao et al. [57], representations of the same sentence with different dropout masks are treated as a positive example. Instead of using either supervised or weakly supervised training as in Gao et al. [57], we simply mix the two resulting supervision signals, and examples are randomly distributed into batches. This mixture of tasks not only facilitates the learning process (§7.6.2), but also reduces the engineering effort required to store and reload models for separate supervised and unsupervised training phases. We initialize the retriever encoder with either the best model of Gao et al. [57] or the encoder of CodeT5-base [167].

[1] https://github.com/elastic/elasticsearch

7.3.2 Generator Instantiation

We experimented with a variety of generator models. We used GPT-Neo-125M, GPT-Neo-1.3B [20] and Codex [31], where we concatenate the retrieved documents and the NL intent as a single, long, prompt. T5-base [133] and CodeT5-base [167] have a shorter input size of 512 tokens, which is sometimes too short for the concatenation of multiple docs. Thus, for T5 and CodeT5 we apply the fusion-in-decoder approach [FiD; 74]: we first concatenate the intent n with each retrieved $d_i \in \hat{\mathcal{D}}_n$ and encode each (n, d_i) pair independently. Then, the decoder attends to *all* encoded NL-document pairs. We finetune the generator to maximize the log-likelihood of the reference code c given n and $\hat{\mathcal{D}}_n$.

With Codex [31], we performed few-shot learning rather than finetuning because the model parameters are not publicly available. We constructed the prompt with three static examples, each of which is a concatenation of retrieved documentation, an NL intent and the reference code snippet. We then appended the test example and its retrieved documentation to the few-shot examples. We used the *code-davinci-001* version because we suspect potential leakage of the test set into the training set of *code-davinci-002*.

7.4 Experimental Setup

We evaluate DocPrompting on two NL→code tasks: shell scripting (§7.4.1), in which we generate complex shell commands given an intent, and Python programming (§7.4.2), where we generate answers in Python for NL questions. In this section, we first introduce a *newly curated* benchmark `tldr`; we then describe our re-split of the popular `CoNaLa` benchmark [186]. For each benchmark, we provide a global documentation pool $\mathcal{D}$ that is shared for all examples and oracle documents $\mathcal{D}_n^*$ which we use to train the retriever. We release our newly curated benchmarks to serve as test-bed for future retrieval-based code generation models.

7.4.1 Shell Scripting

`tldr` is a community-driven project that maintains easily-readable help pages with examples for over 2.5k Bash commands in over 25 natural languages[2]. We collected pairs of English intents and Bash command lines. The NL intents are written by human users, and the Bash commands range from popular ones like `cat` and `tar`, to uncommon commands such as `toilet`

[2] https://github.com/tldr-pages/tldr

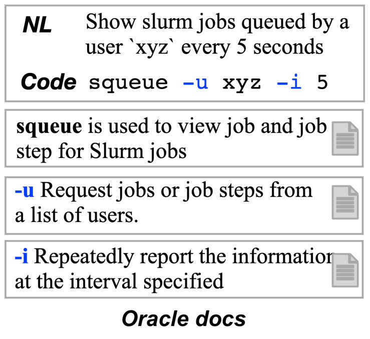

NL
Show slurm jobs queued by a user `xyz` every 5 seconds
Code
squeue -u xyz -i 5
squeue is used to view job and job step for Slurm jobs
-u Request jobs or job steps from a list of users.
-i Repeatedly report the information at the interval specified
Oracle docs

Figure 7.2: An example NL-code pair from `tldr`, along with three oracle documentation items.

and `faketime`. Our resulting `tldr` benchmark contains 1,879 unique Bash commands and 9,187 NL→Bash pairs. We constructed the training, development and the test set with *completely disjoint commands* to test the generalizability of a code generation model. The shared documentation pool $\mathcal{D}$ is made up of the $400k$ paragraphs from the 1,879 Bash manuals. Each paragraph describes a single concept such as an argument flag. We further curated the oracle documents $\mathcal{D}_n^*$ for each example using simple string matching. An example from `tldr` is shown in Figure 7.2. To the best of our knowledge, this is the first work to leverage `tldr` as an NL→code benchmark. In `tldr`, each NL intent results in a single Bash command with a combination of argument flags. We therefore first retrieve an entire Bash manual; then, we take the top manual and retrieve the top-10 paragraphs from that manual.

Evaluation metrics We measure: (a) command name accuracy (CMD Acc) – whether the command name (*e.g.*, `cat`) is an exact match; (b) exact match (EM) – exact match between the reference and the generation; (c) token-level F1; and (d) character-level BLEU [charBLEU; 99, 150]. In all metrics, we disregard user-specific variable names in the references and the models outputs. For example, "`mycli -u [user] -h [host] [database]`" is evaluated as "`mycli -u $1 -h $2 $3`".

7.4.2 Python Programming

`CoNaLa` [186] is a popular benchmark for NL→Python generation. NL intents are StackOverflow questions, and code snippets are their answers. Both intents and code snippets are rewritten by human annotators. We re-split the dataset to test models' generalization to unseen Python functions. In our re-split, we verifed that every example in the development or the test set uses at least one Python function (*e.g.*, `plt.plot`) that was *not* seen in the training data. In addition, we make sure that the examples from the same StackOverflow posts are in the same set to prevent leakage. This re-split results in 2,135/201/543 examples in the training/development/test sets, respectively.

The `CoNaLa` documentation pool $\mathcal{D}$ contains 35,763 documents, each describing a single function, from all Python libraries available on `DevDocs` (https://devdocs.io). These include built-in libraries and other popular libraries such as `numpy`. We constructed the oracle docs $\mathcal{D}_n^*$ for each example by matching all function names in the target code c with docs.

Evaluation metrics We follow Yin et al. [186] and measure BLEU-4. Since we focus on generalization to unseen functions, we additionally report function name recall (*recall*) and unseen function recall (*recall*$_{unseen}$), which measures recall among function calls that do not appear in the training set. Finally, following Austin et al. [14], Chen et al. [31], we used the manually written unit tests from Wang et al. [168] for 100 examples from `CoNaLa`'s test set and measure pass@k. We followed Chen et al. [31] and performed nucleus sampling [68] with $p = 0.95$. For each k, we searched for the best temperature for each model from $\{0.2, 0.4, 0.6, 0.8, 1.0\}$. On average, each example has 2.03 tests. The concatenation of multiple Python docs often exceeded the length limit of GPT-Neo, we hence experimented in this dataset with FiD, which allows longer inputs.

7.5 Results

In all following results, all models with DocPrompting use the top-10 retrieved docs from the best retriever on that dataset (Table 7.4). Every baseline uses the exact same setup as its "+DocPrompting" version, except for not using the documentation.

7.5.1 Shell Scripting Results

Results for `tldr` are shown in Table 7.1. DocPrompting consistently improves the base models. For example, T5+DocPrompting achieves more than *twice* higher accuracy in predicting the command name, more than 16 charBLEU points on the entire prediction, and almost 9% of absolute exact match gain, compared to the vanilla T5. In the few-shot learning setting with Codex, DocPrompting brings gains of 6.7 charBLEU points, and consistent improvement across all metrics over the baseline that observes only NL-code pairs in its prompt. These results show that retrieving documentation also benefits strong models such as Codex, and with only few examples in the context.

Table 7.1: Results on shell scripting, using a BM25 retriever with top-10 retrieved docs, on the test set of `tldr`. For the "oracle command name" experiments, we selected the best model of each type.

Model		CMD Acc (%)	EM (%)	Token F1	charBLEU
GPT-Neo-125M	-	11.96	1.94	28.75	19.99
	+DocPrompting	**25.32**	**3.56**	**31.23**	**24.43**
GPT-Neo-1.3B	-	14.55	3.12	32.46	24.70
	+DocPrompting	**27.59**	**9.05**	**37.24**	**30.57**
T5	-	10.02	0.76	19.90	25.48
	+DocPrompting	**30.28**	**9.16**	**37.58**	**31.97**
CodeT5	-	14.60	2.18	30.00	21.50
	+DocPrompting	**30.72**	**9.15**	**36.71**	**33.83**
Codex *3-shots*	-	27.48	8.94	36.04	16.94
	+DocPrompting	**31.21**	**9.29**	**36.77**	**23.72**
With the oracle command name					
T5	-	-	12.96	59.36	45.05
	+DocPrompting	-	**22.55**	**64.84**	**54.28**
Codex *3-shots*	-	-	22.44	62.26	50.29
	+DocPrompting	-	**32.43**	**69.73**	**55.21**

Code generation with oracle command names In realistic settings, a human programmer may know the command name they need to use (*e.g.*, `awk`), but not know the exact usage and flags. In fact, better understanding of the usage of *known* commands is the purpose of Unix `man` pages and the `tldr` project. We conducted an oracle experiment where we provided T5 (which was the strongest model using DocPrompting) and Codex with the oracle command name (*e.g.*, `awk`). This oracle information is provided to both the baseline and the model that uses DocPrompting. The results are shown on the bottom part of Table 7.1. When the oracle command is given, DocPrompting further improves over the base models. For example, when providing Codex with the ground truth command name, DocPrompting improves its exact match from 22.44% to 32.43%.

Should we retrieve documentation or examples? All existing retrieval-based models of

Table 7.2: Comparison to approaches that retrieve examples [128, 129]

.

Model		CMD Acc (%)	EM (%)	Token F1	charBLEU
GPT-Neo-125M	+`ExPrompting`	6.68	0.32	20.49	11.15
	+$\mathcal{M}$	**25.32**	**3.56**	**31.23**	**24.43**
GPT-Neo-1.3B	+`ExPrompting`	14.01	2.8	30.07	22.11
	+$\mathcal{M}$	**27.59**	**9.05**	**37.24**	**30.57**

Table 7.3: Results on `CoNaLa`, using a CodeT5 retriever with top-10 retrieved docs. Function recall (Recall) measures how many functions in the reference code are correctly predicted, and unseen function recall (Recall$_{\text{unseen}}$) only considers the subset held out from the training data.

Model		BLEU	Recall	Recall$_{unseen}$
Codex *3-shots*	-	43.16	39.52	-
	+ $\mathcal{M}$	**43.47**	**39.87**	-
	+ $\mathcal{M}_o$racle docs	50.59	57.84	-
T5	-	28.07	14.36	2.57
	+ $\mathcal{M}$	**30.04**	**21.34**	**8.24**
CodeT5	-	34.57	24.24	9.03
	+ $\mathcal{M}$	**36.22**	**27.80**	**18.30**
	+ $\mathcal{M}_o$racle docs	49.04	72.20	63.91

code retrieve NL-code pairs or code snippets, rather than documentation. To simulate this scenario, we followed Parvez et al. [128] and Pasupat et al. [129] to retrieve NL-code pairs from the training set of `tldr`, and refer to this baseline as `ExPrompting`. We finetuned the best retriever RoBERTa and two generators, and retrieved the top-30 NL-code pairs for every example. As shown in Table 7.2, *retrieving documentation* (DocPrompting) provides much higher gains than retrieving examples (`ExPrompting`). Theoretically, adding examples of unseen commands can help `ExPrompting` generalize to them as well. However, new libraries and functions may not have available examples on the web yet, while documentation often *does* becomes available when the library is released.

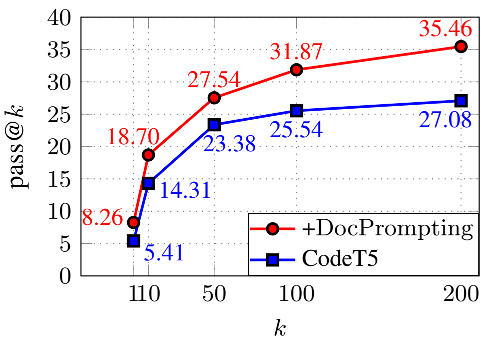


Figure 7.3: Pass@k of CodeT5 with and without DocPrompting on 100 CoNaLa examples.

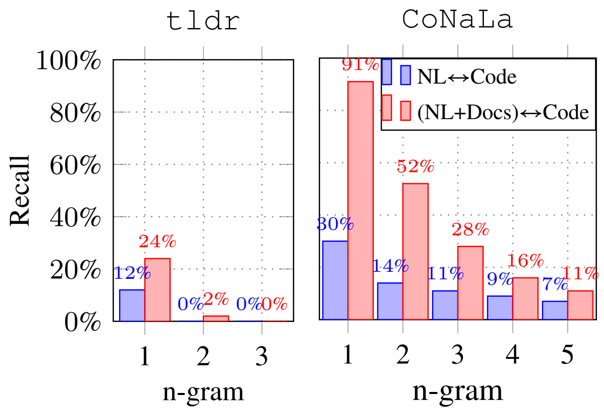


Figure 7.4: Using documentation significantly increases the n-gram overlap recall between the input and the output, in tldr and CoNaLa.

7.5.2 Python Programming Results

Table 7.3 shows the results on CoNaLa. CodeT5+DocPrompting yields a 1.65 BLEU improvement over the state-of-the-art baseline that was initialized with CodeT5.[3] When measuring the recall of the generated function names, the benefit of DocPrompting is especially higher for *unseen* functions (*recall*$_{unseen}$). For example, DocPrompting achieves 18.30 compared to only 9.03 of the base CodeT5 in unseen functions. Additionally, DocPrompting improves in-context learning setting with Codex. We hypothesis that the minor gain is mainly due to the potential data leakage of Codex, which violates the split of seen and unseen functions. Another reason is that a strong generator such as Codex may require an equally strong retriever as well. We find that Codex can achieve even higher results with an oracle retriever, which shows the potential further improvement by improving the retrievers. Finally, CodeT5 performs better than T5, with and without using DocPrompting. This emphasizes the importance of using code-specific pretrained models.

Execution-based evaluation The results are shown in Figure 7.3. Using DocPrompting consistently outperforms the baseline CodeT5 for all values of pass@k. For example, DocPrompting yields 2.85% improvement on pass@1 and 4.45% improvement on pass@5, which are realistic numbers of completions that can be suggested in an IDE. When $k = 200$, DocPrompting

[3]In a separate experiment on the original split of CoNaLa, this baseline achieved a BLEU score of 39.12, which outperforms the previous state-of-the-art [18] by 4.92 BLEU points.

Table 7.4: Retrieval performance of multiple models on the dev set of `tldr` (top) and `CoNaLa` (bottom). RoBERTa is the best model taken from from Gao et al. [57], and CodeT5 is the encoder of CodeT5-base [167]. Models with the subscript "off-shelf" are the off-the-shelf models, and the other models were finetuned with the objective in Equation 7.3. The last column is the best model (RoBERTa for `tldr` and CodeT5 for `CoNaLa`) trained without the weak supervision corpus.

	n	BM25	$\text{RoBERTa}_{\text{off-shelf}}$	RoBERTa	$\text{CodeT5}_{\text{off-shelf}}$	CodeT5	Best w/o weak sup.
`tldr`	1	**32.81**	17.53	30.03	10.45	18.10	28.30
	5	51.73	37.89	**52.50**	20.26	38.52	50.50
	10	59.86	46.80	**60.33**	25.73	51.03	59.84
	20	62.01	56.11	**64.30**	33.65	57.26	62.30
`CoNaLa`	1	3.01	4.46	13.49	4.60	**16.54**	10.51
	5	7.16	7.58	26.38	8.63	**42.35**	21.15
	10	9.73	10.93	34.86	12.25	**55.81**	29.34
	20	11.46	13.89	45.46	18.46	**66.79**	42.21

widens the gap to 8.38%. These results demonstrate that DocPrompting does not only improve the quality of the generated code in its surface form, but also increase its functional correctness.

7.6 Analysis

7.6.1 Why does reading the documentation help?

We believe that one of the major reasons is that *documentation eases the mapping between NL intents and code*, since the documentation contains both NL descriptions *and* function signatures. We calculated the n-gram overlap between the NL intents and their corresponding code snippets (NL↔code), and the overlap between the NL intents with their top-10 retrieved documents and their code snippets ((NL+docs)↔code). As shown in Figure 7.4, adding documentation *significantly increases* the overlap across n-grams, and increase, for example, the unigram overlap from 12% to 24% in `tldr`. That is, one of the reasons that retrieving documentation helps generating accurate code is that documentation bridges the gap between the "intent terminology" and the "code terminology".

7.6.2 Ablation Study

We compared different configurations of the retriever, to gather more insights for effective DocPrompting. Table 7.4 shows a comparison between different retrievers and their setups. First, the performance of BM25 varies among datasets: In `tldr`, BM25 matches the recall of trained dense retrievers; however in `CoNaLa`, BM25 achieves only recall@10 of 9.73%, and strong dense retrievers such as the encoder of CodeT5 achieve recall@10 of 55.81. We hypothesize that this difference between datasets stems from the ways these datasets were created: `tldr` intents were written based on existing Bash commands and manuals; while `CoNaLa` examples were mined from StackOverflow posts, where users ask questions with limited or no context. Thus, NL intents in `CoNaLa` require a better semantic alignment with the documents, and thus benefit from dense retrievers. The gap resulting from different data curation processes was also observed by Rodriguez and Boyd-Graber [142] in open-domain question answering (QA).

Second, retrievers that were pretrained on the target programming language are generally stronger. For example in `CoNaLa`, CodeT5 which was pretrained on Python, is both a better off-the-shelf retriever and a better finetuned-retriever than RoBERTa, which was pretrained mainly on text. In contrast, `tldr` is based on Bash, which neither CodeT5 nor RoBERTa were explicitly pretrained on. Thus, `tldr` benefits mostly from BM25 and RoBERTa rather than CodeT5 as retrievers.

Finally, training the retriever using weak supervision on the documentation pool (Section 7.3.1) dramatically improves the retriever. The recall of the best retrievers of each dataset without this corpus is shown in the last column of Table 7.4 ("Best w/o weak sup."). On `CoNaLa`, removing this corpus results in severe performance degradation. One possible explanation is that this weak supervision helps the retriever perform domain adaptation more effectively.

7.6.3 Case study

We examine the models' outputs and show two representative examples in Figure 7.5. In the first example, `Image.open` was not seen in the training set, and the baseline CodeT5 incorrectly predicts `os.open`. In contrast, using DocPrompting allows to retrieve the docs and to correctly predict `Image.open`. In the second example, `df.to_csv` was not seen in training, and the baseline CodeT5 fails to correctly predict it. In contrast, DocPrompting *does* predict most of the `df.to_csv` call correctly, thanks to the retrieved docs. Nevertheless,

NL Intent: **Open image "picture.jpg"**	
Ground truth:	`img = Image.open('picture.jpg') \n Img.show`
CodeT5:	`os.open('picture.jpg', 'r')`
CodeT5+DocPrompting:	`image = Image.open('picture.jpg', 'rb')`
NL Intent: **Exclude column names when writing dataframe 'df' to a csv file 'filename.csv'**	
Ground truth:	`df.to_csv ('filename.csv', header=False)`
CodeT5:	`df.drop(['col1', 'col2'], axis=1, inplace=True)`
CodeT5+DocPrompting:	`df.to_csv('filename.csv', skiprows=1)`

Figure 7.5: Examples of predictions from `CoNaLa`, of the base CodeT5 compared to CodeT5+$\mathcal{M}$. Unseen functions are underscored.

DocPrompting generates an incorrect argument `skiprows=1`, instead of `header=False`. The reason is that along with the retrieved documentation of `df.`**`to`**`_csv`, the retriever also retrieved the documentation of `df.`**`read`**`_csv`, which has a `skiprows` argument. That is, the generator uses an argument of `df.`**`read`**`_csv` with the function `df.`**`to`**`_csv`. Further improving the retrievers and the generators, and post-filtering based on the validity of argument names, may mitigate such mistakes.

Chapter 8

Turning Indirect Knowledge into Direct Demonstrations at Scale

While documentation explains how to perform a task, the knowledge is still indirect, requiring an agent to comprehend the information and take corresponding actions. How can we derive knowledge that directly teaches models what to do? This chapter continues the discussion on leveraging widely available human-authored knowledge through data synthesis. This work is currently under review:

- Tianyue Ou, Frank F. Xu, Aman Madaan, Jiarui Liu, Robert Lo, Abishek Sridhar, Sudipta Sengupta, Dan Roth, Graham Neubig, and Shuyan Zhou. Synatra: Turning indirect knowledge into direct demonstrations for digital agents at scale. In *submission to NeurIPS*, 2024

8.1 Overview

For AI agents, demonstrations typically involve specifying the desired next action under certain states to successfully complete tasks, as shown on the right of Figure 8.1. Existing works that automatically collect demonstrations (1) set up environments for an agent to interact with, (2) run a baseline agent within this environment, and (3) employ heuristically designed filtering mechanisms to remove low quality demonstrations [55] or perform relabeling [9, 115]. All three of these requirements limit applicability to a wide variety of practical applications. Setting up an environment that is representative of the actual environments in which we would like agents to act is a difficult task, and existing environments are generally limited in scope to a few web sites [49, 212] or digital apps [176]. Even within these constrained settings, strong LLMs such as `GPT-4` struggle on tackling digital tasks [212], making collecting successful demonstration with LLMs inefficient. In addition, collecting demonstrations of the tasks that we would like to solve from human is costly and the scope is not exhaustive [46, 71]. For example, gathering a demonstration for canceling a PayPal order requires an actual PayPal account with a legitimate subscription history.

In this work, we propose Synatra, a data generation approach that synthesizes high-quality execution trajectories for complex digital tasks at scale. This approach is based on the intuition that there is a rich trove of existing knowledge that encodes *indirect* supervision about how to perform digital tasks (§8.2.2). An example indirect knowledge is a tutorial that details the sequential breakdown of a complex task on a web site, such as "how to cancel a recurring payment on Paypal" for human readers (Figure 8.1 upper left) [197, 210]. While this outlines some procedural knowledge, it does not directly map these steps onto the actual actions within concrete observations (e.g. on the Paypal web site). Given this indirect knowledge, we leverage an LLM to *re-purpose* it into more usable form that directly demonstrates the exact action to take under different observations (§8.3). This process involves leveraging LLMs's language processing capability for knowledge paragraphing, coding capability for generative digital environment, and general web knowledge for hypothetical executionsTherefore, our approach can scale with the availability of the indirect knowledge, rather than the reliance on direct human annotations or the human-level performance of LLMs digital tasks.

We carefully study the sources of indirect knowledge, the design of demonstration formats, and the mechanisms for iterative refinement in order to synthesize high-quality demonstrations using an LLM (§8.3, §8.4). We generate demonstrations of 50k tasks from 21 domains, and finetune a 7b `Codellama-instruct` model with this synthetic data. The resulting agent, `Synatra-CodeLlama`, surpasses existing open-source models of similar size on three popular web-based task benchmarks: Mind2Web, MiniWoB++, and WebArena. Moreover, it consistently outperforms models that are ten times larger and have been fine-tuned with interactive data (§8.6). Our findings also indicate that our model is a more accurate option for predicting single-step actions or for performing tasks with fewer steps in browser copilot settings, in comparison to `GPT-3.5`. Importantly, while each synthetic example incurs only approximately 3% of the cost of a human-generated demonstration, we demonstrate that synthetic data with good domain coverage can be *more* effective than an identical quantity of limited-domain human demonstrations in real-world web-based tasks (§8.7).

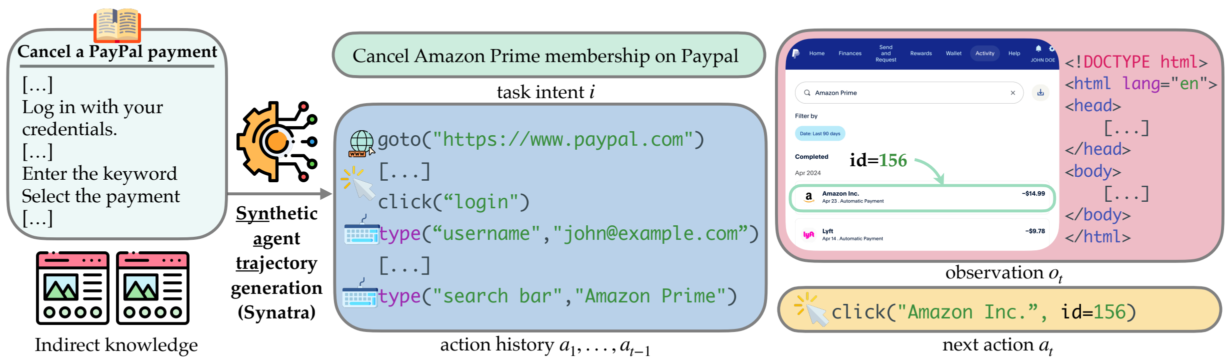

Cancel a PayPal payment
[...]
Log in with your credentials.
[...]
Enter the keyword
Select the payment
[...]
Indirect knowledge
Synthetic agent trajectory generation (Synatra)
Cancel Amazon Prime membership on Paypal
task intent i
goto("https://www.paypal.com")
[...]
click("login")
type("username","john@example.com")
[...]
type("search bar","Amazon Prime")
action history $a_1, \ldots, a_{t-1}$
id=156
<!DOCTYPE html>
<html lang="en">
<head>
[...]
</head>
<body>
[...]
</body>
</html>
observation o_t
click("Amazon Inc.", id=156)
next action a_t

Figure 8.1: Our approach aims to synthesize direct demonstrations (right of the arrow) that specify the immediate next actions based on previous actions and current observations. The sources comprise only indirect knowledge (left of the arrow), such as tutorials designed for human consumption and randomly sampled observations that lack associated tasks and actions.

8.2 Problem Formulation

8.2.1 Controlling Digital Agents through Natural Language

An agent interacts with a computer environment $\mathcal{E} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T} \rangle$ with state space $\mathcal{S}$, action space $\mathcal{A}$, observation space $\mathcal{O}$ and the environment dynamics $\mathcal{T} : \mathcal{S} \times \mathcal{A} \longrightarrow \mathcal{S}$. While this framework can be applied to different types of task, in this work, we consider a *web browser* as the unified entry to access different web applications. We follow WebArena [212] to define the observation space as the contents on screen, represented as the text-based accessibility trees. We use the same universal action space as in WebArena. This action space resembles the keyboard and mouse operations of a computer (*e.g.*, `click`, `type`), and is applicable to *arbitrary* web applications. Appendix 3.2.4 lists all valid actions. The environment dynamics (*e.g.*, the effect of clicking a button) and the states (*e.g.*, the database status) are decided by the implementations of the web applications.

Given the natural language intent $\boldsymbol{i}$, at each time step t, an agent issues an action $a_t \in \mathcal{A}$ based on s_t. The environment state is updated to s_{t+1} with new observation o_{t+1}. This process ends when the agent predicts the `stop` action. We follow existing works [46, 183, 202, 212] to represent s_t as $(\boldsymbol{i}, a_1, ..., a_{t-1}, o_t)$. A benchmark (*e.g.*, WebArena) supplies a scoring function $r(\boldsymbol{i}, s_n)$ that examines the final state and returns 1 if the desired goal state is satisfied, and 0 otherwise.

8.2.2 Definition of Direct Demonstrations and Indirect Knowledge

We consider the expected action a_t given s_t as a form of direct demonstration, *i.e.*, (s_t, a_t). This allows an agent to directly learn how to predict the next action under a given state. On the other hand, indirect knowledge is broadly defined as resources that can benefit the task execution, but is not in the format of state and expected action tuple. We mainly focus on three types of indirect knowledge:

1. **Procedural knowledge** details the sequence of steps $\langle a'_1, a'_2, ..., a'_n \rangle$ required to complete a specific task $\boldsymbol{i}$. Unlike an action a_t in trajectories, the steps in procedural knowledge are ungrounded, they lack a direct association with any particular observation and are not tied to specific action spaces. For instance, the tutorial in Figure 8.1 instructs "*login with your credentials*" without providing the concrete PayPal login page and the input fields to `type`.
2. **Environment knowledge** $\mathcal{T}$ that describe the effects of applying different actions in some hypothetical states. Example knowledge includes verbal descriptions such as "*... after clicking the cancel button, you will see a pop up window ...*".
3. **Ungrounded observations** o that are not associated with particular tasks or trajectories. In the context of web-based tasks, an observation is a random web page with different contents and status (*e.g.*, a product page with a query in the search field).

8.3 Scalable Demonstration Synthesis for Digital Agents

In this section, we first introduce our design choices on the canonical formalization of trajectories, which account for the *structural* nature of procedures. Then, we delve into the sources for acquiring indirect knowledge, and the mechanisms for re-purposing this knowledge into direct supervisions.

8.3.1 Trajectories as Programs

Existing works demonstrate that representing the task solving procedure as writing programs is beneficial due to the structural natural of programs compared to free-form text [108, 208], and the flexibility of using tools [33, 56, 164]. Inspired by these observations, we conceptualize a trajectory as a Python function that interleaves between natural language planning articulated in comments and actions API calls on the right, as shown in Figure 8.2. The planning includes both

```
website = "<url>"
observation = "<AXtree of the page>"
objective = "[...]"

# past actions
def solve():
    objective = "[...]"
    # sub-task 1: [...]
    # <NL explanation for the step>
    action(arg=value)
    [...]

    # sub-task 2: [...]
    [...]
    stop(ans=value)
```

Figure 8.2: The template of formulating trajectories as programs.

task-level planning, which decomposes the task into multiple sub-tasks, and action-level planning, which can be viewed as the natural language translation of the code. A concrete example can be found in appendix B.1. We note that while existing works also study using program formalization for web-based tasks [63, 179], they focus on action-level interventions without incorporating task-level planning. We study the empirical effect of program formalization in §8.8.

8.3.2 Synthesizing from Text Procedural Knowledge with Generative Environment

The Internet offers fairly extensive procedural knowledge that describes *how* to perform high-level tasks by breaking down the task into detailed lower-level steps, such as how-tos and tutorials.

Source We use wikiHow[1] as our main source for these tutorials due to its comprehensive coverage of diverse tasks, and its consistent format. Each article is consist of a high-level task description and the step-by-step instructions. We performed a filtering step and only kept the qualified articles that only involving navigation through the graphical user interface (GUI) of a computer or a mobile phone. We prompted `GPT-3.5-turbo` with six examples mixing

[1]https://www.wikihow.com/Main-Page

qualified articles (*e.g.*, How to redeem an Amazon gift card online[2]) and unqualified articles (*e.g.*, How to make a pizza) to perform the classification of all wikiHow articles. The prompt is shown in appendix B.2. As a result, we obtained $25k$ qualified articles. We further sample $3k$ articles to perform data synthesis.

Synthesis Approach We want to bridge two gaps to re-purpose $\langle a'_1, a'_2, ..., a'_n \rangle$ for task $\boldsymbol{i}$ into $\langle a_1, ..., a_{t-1}, o_t \rangle$. When re-purposing a sequence of actions $\langle a'_1, a'_2, ..., a'_n \rangle$ for task $\boldsymbol{i}$ into a new sequence $\langle a_1, ..., at - 1, o_t \rangle$, many challenges arise. First, the action descriptions provided in tutorials are not constrained to specific action spaces. Instead, they are presented as free-form natural language (NL) expressions, which can lead to ambiguity. For instance, various verbs such as "enter," "input," and others may all correspond to the same underlying action, `type`. Second, NL descriptions are often abstract, omitting concrete actions. For example, the process of "logging in" involves a series of actions, including typing in a username and password, but these specific actions may not be explicitly mentioned. Finally, the steps outlined in tutorials are ungrounded, meaning they are not directly associated with observable states or outcomes. Tutorials typically employ generic descriptions to accommodate various instances of conceptually similar tasks. For example, as illustrated in Figure 8.1, the tutorial merely instructs to “enter the keyword” without addressing any specific scenario.

Based on these findings, we propose an *iterative* approach that first use an LLM to rewrite an article into a hypothetical trajectory in the format shown in §8.3.1, then we leverage a generative model to synthesize the intermediate observation between two consecutive actions. First, in the rewriting step, we ask `GPT-4` to perform: (1) propose a hypothetical concrete scenario relevant to the task (2) perform basic parsing such as translating “enter the keyword [...]” into `type("search bar", "Amazon Prime")`; (3) categorize actions into groups that reflect the sub-task structures outlined by coding blocks. These tasks mainly demand a LLMs’s creativity, language processing ability and event understanding respectively. An example of rewriting a how-to article into a trajectory in program format is showed in appendix B.4. The detailed prompt for the rewriting step is shown in appendix B.3.

Next, we leveraged `GPT-4` to generate the observations between randomly sampled consecutive actions. We use the consecutive actions of `type("search bar", "Amazon Prime")` and `click("Amazon Inc", id=156)` in Figure 8.1 as the example. There are mainly two requirements for the generated observation. First, the observation reflects the *outcomes* of past actions. In the example, it corresponds to a page with a user logged in, and a search input

[2] https://www.wikihow.com/Apply-a-Gift-Card-Code-to-Amazon#Online

field filled with "Amazon Prime". Second, the observation *encodes* the necessary elements to perform the next action. In the example, it corresponds a payment history list with a payment to Amazon. We prompt `GPT-4` with the action sequence to generate a HTML snippet that fulfills the above requirements. Since the next action requires the concrete element to interact with, we ask the model to insert a tag of `id="next-action-target-element"` in the corresponding HTML node to indicate the grounding. This step mainly requires a model's coding capabilities, particularly in front-end development. We do not require the LLM to generate HTMLs with high fidelity and complexity, which is a open research question [155]. The full prompt is in appendix B.3 and an example for this step is in Figure B.1.

It is notable that there are other resources that share similar traits, such as the captions of YouTube how-to videos [111], our transformation mechanism is generally applicable to such resources and we leave the empirical study as our future work.

8.3.3 Synthesize from Random Observations

To compensate for the loss in simplified observations, we also perform data synthesis with real web pages. We show that these two sources can compensate each other by examining the generated data in §8.4 and comparing the actual web-based task performance in §8.8.

Source We utilize ClueWeb [124] as our data source, which comprises HTML snapshots of more than 10 billion web pages. Our initial analysis indicates that a random sampling approach would likely lead to a homogeneous distribution dominated by less interactive pages, such as news articles. In contrast, more complex web-based tasks typically requires interaction with various web elements to advance the task. To diversify the sampled web pages, we employed a temperature sampling approach to select pages based on their content categories. Web pages from ClueWeb exhibits the distribution that domains with higher frequency are typically more interactive, such as Amazon and Reddit, while domains appearing in lower frequency are less interactive. We use a temperature sampling with $t = 0.6$ to up-sample more interactive sites while maintaining diversity on more rare sites. More details are listed in § B.7.

Synthesis Approach We treat each sampled web page as an intermediate observation at time step t, aiming to synthesize the specific task $\boldsymbol{i}$, the previous actions $a_1, ..., a_{t-1}$ and the subsequent action a_t consisting of an action, and a corresponding interacting element in the observation. We first convert a web page into its corresponding accessibility tree at the begin-

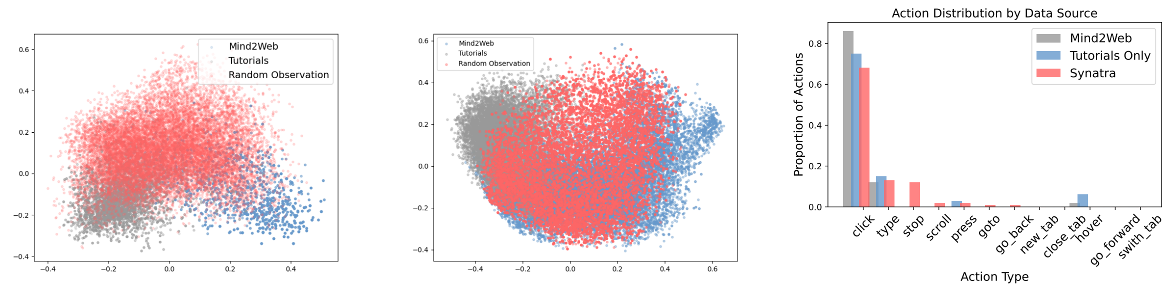


Figure 8.3: Left: t-SNE of task intent embedding; middle: t-SNE plots of accessibility tree embeddings of the synthetic data source and Mind2Web; right: Distribution of Action Types.

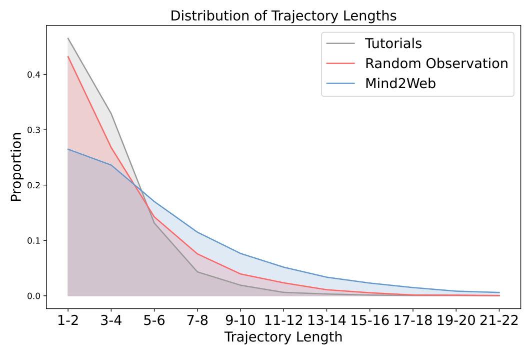


Figure 8.4: Distribution of trajectory lengths

ning of each node, and sample a segment to present to `GPT-4`. We follow WebArena to assign a unique ID to each node in the tree to easy the challenging in referencing to the nodes. To increase the diversity of the tasks, we first instruct the model to brainstorm k task categories relevant to the web domain. Then the model randomly selects three of these categories and develops them into concrete scenarios with past actions leading up to the current observation and the next action to take. The prompt is in § B.5 and the response is in § B.6.

8.4 Data Statistics

We inspect on the number of action steps, action types, and semantic embedding of agent trajectories, while using Mind2Web's human collected trajectories as a reference. We ask the question: how does our synthetic data compared with real human annotated data? To answer this question, we look into it in two dimensions: how close we are to real data, and how different we are from real data.

How close are we? For length of trajectories, our synthesized data share similar distribution with human annotated ones, as displayed in Figure 8.4. Majority of the histories are shorter than six steps. The distribution of action types are also similar between synthesized data and human annotated ones. As shown on the right of Figure 8.3. Both data sources work on similar tasks too. As shown on the left of Figure 8.3, semantic embedding of task objectives are displayed with t-SNE reduction. The task semantics are highly overlapped between human collected trajectories and synthetic trajectories.

How different are we? One key advantage of synthesized data is its low turn around cost, particularly in adjusting action space and action distribution, which are key for adapting to new environments. Synthetic trajectories completes the action space of existing human annotated training data through providing necessary yet lacking actions such as `stop`, `scroll`, and so on. There is no inherent limitations to include additional action types, our flexible synthesizing approach could expand to any action space. Further more, the combine use of ClueWeb and wikiHow generation pipeline allows us to have fine-grained control over which action types we want the agent to prioritize in learning, and relax on those that have been learned heavily in the past.

Synthetic trajectories provides additional diversity on top of human collected ones. We embedded our synthetic data's accessibility trees along with those of Mind2Web's training set with embedding model all-mpnet-base-v2. We visualize the embeddings with t-SNE and display the results in the middle of Figure 8.3 [162]. Accessibility trees from our synthetic data has a good proportion of overlap with human annotated Mind2Web dataset while providing many diversity at the same time.

8.5 Experimental Setup

Agent training We fine-tune `CodeLlama-instruct-7b` [145] with $49,373$ of web-navigation instruction data, which is sourced from WikiHow articles and ClueWeb web pages.

Evaluation tasks We select 3 evaluation tasks in the domain of web-navigation. 1) We test with Mind2Web's test set. Mind2Web's test set contains three categories, marked by their degree of deviation from its training data: cross-task, cross-website, cross-domain in the order of increasing difference from training data [46]. However, since we do not train at all on its training set, every category are held-out set for us. Our setting is also different from the original

Mind2Web, where we remove the DeBerta filter. To feed in the most information in our limited context window, we input in-viewport accessibility trees instead of whole page HTML. 2) We test with WebArena. WebArena is an execution based web agent benchmark that allows agents to freely choose which path to take in completing their given tasks. There could be multiple correct next actions to take at any given stage, using a result based benchmark here would account for all correct actions [212]. 3) We test with MiniWoB++, another dynamic interactive benchmark. Different from WebArena, MiniWoB++ consists of easier tasks that involve only one web page, one or a few steps to complete, and more specific, low level task directions [71].

8.6 Results

Table 8.1 presents the performance of various models across three web-based task benchmarks. Overall, `Synatra-CodeLlama` achieves the best performance among models of comparable size. Notably, `Synatra-CodeLlama` significantly outperforms its base model `CodeLlama-instruct-7b`. While `CodeLlama-instruct-7b` fails to complete any tasks in WebArena, `Synatra-CodeLlama` successfully executes 4.8% of them. Furthermore, `Synatra-CodeLlama` elevates the performance of `CodeLlama-instruct-7b` from 6.62% to 17.26% in Mind2Web (a 160% relative improvement) and from 23.04% to 39.57% in MiniWoB++ (a 71% relative improvement). More encouragingly, `Synatra-CodeLlama` demonstrates superior performance on Mind2Web and MiniWoB++ compared to `GPT-3.5`. It also outperforms `Lemur-chat-70b`, which is finetuned with interactive data and is ten times larger, across all three benchmarks. The results suggest that our data synthesis approach is effective in helping the model predict the next action (as in Mind2Web) and performing simple tasks with a few steps (as in MiniWoB++). The synthesized data can also guide the model towards executing real-world complex tasks more accurately. Consequently, `Synatra-CodeLlama` has potential applications in suggesting individual steps in browser copilot scenarios.

`Synatra-CodeLlama` surpassed the performance of all open source model finetuned with interactive data. Among these models, `AgentLM`, `CodeActAgent` and `AgentFlan` includes demonstrations to perform web-based tasks in their instruction finetuning dataset. However, we find that these models may not serve as capable agents to perform web-based tasks due to the special design choice encoded in the finetuned models. For instance, `AgentLM` and `CodeActAgent` use Regex expression to match interactive element on a web page and require carefully selected in-context examples to showcase which are the proper Regex expression for different examples. However, Regex expressions only work for simple web pages with a few elements as in MiniWoB++, while it is prohibitive to do pattern matching in complex web pages as in Mind2Web and WebArena. As a result, when we experiment with the more generic action space which is suitable for all three benchmarks without in-context examples, we see these models have a significant performance degradation. On the other hand, `Synatra-CodeLlama` targets at the generic web-based tasks and does not encode any dataset artifacts during training. Even though all three benchmarks are completely held-out during data generation, `Synatra-CodeLlama` achieves consistent superior performance on all benchmarks.

8.7 Analysis

What does the model learn from the synthetic data? Basic Interactive Pattern: Synatra training data enables models to learn how to interact with web pages. Trained models are xxx percent less likely to click on non-interactive elements compared to `GPT-4-turbo`, and xx percent less likely to issue a type action to web-components that do not contain a textbox.

Planning: Synatra trained models can better keep track of and utilize histories.

Can Synatra be as effective as human annotations? The Mind2Web dataset [46] includes human-annotated trajectories for approximately $1k$ web-based tasks in its training set, with each trajectory costing $0.8. We process these $1k$ trajectories into $9k$ direct demonstrations ((s_t, a_t)) to match the format of Synatra. We then compare the model trained with these $9k$ human demonstrations (human only) and the model trained with $9k$ demonstrations generated by Synatra. The results are shown on the right of Figure 8.5. Simply training `CodeLlama` with the Mind2Web human annotations provides modest improvement of 3.96% on MiniWoB++ , while failing entirely on WebArena. In contrast, Synatra led to a substantial improvement of 17.81% on MiniWoB++ and 4.56% on WebArena. The limited performance of the human annotations can be partially attributed to their restricted task coverage; notably, Mind2Web lacks information seeking tasks that require a string answer. Furthermore, the human trajectories did not specify conditions for terminating task execution. Despite adding such trajectories from Synatra (human + synthsis), the model still underperformed. We hypothesize the diversity of tasks plays a role in this discrepancy, since many tasks cannot be covered without pre-set environment (*e.g.*, return an order) (Figure 8.3). These findings underscore the efficacy of Synatra, which also exempts from the complexities of developing recording tools and managing communication with annotators. However, it is important to recognize that the quality of human demonstrations is presumably higher, but they require meticulous design and control during the data collection process.

8.8 Ablation

In this section, we perform an ablation to validate the design choices of our data synthesis approach.

Representing trajectories as programs is beneficial To verify if the programs format is helpful, we convert $30k$ trajectories to the NL format similar to setting in WebArena [212] and we compare its performance with the model trained with the exact data, but in our program format. The results are shown in left of Figure 8.5. We can see that performance drops on both MiniWoB++ and WebArena when using the NL format. We hypothesize that program is potentially a more natural format to represent a sequence of actions in the form of API calls, and the chain-of-thought reasoning can be embedded as code comments. Our observation also echo the observations of using program representation for non-programming tasks [108, 131, 164], while our experiments further contributes insights towards finetuning setups for interactive tasks.

Difference sources of indirect knowledge complement each other Our indirect knowledge primarily comes from two sources: tutorials and randomly sampled web pages. In the former source, the procedural knowledge of how-to perform the task are accurate, since the tutorials are written and verified by human. However, there is no guarantee on the authenticity of the generated observations of web pages. Indeed, generating web pages with real-world complexity is a open research problem [155]. In the contrast, the observations from the latter sources are completely real, while there is no guarantee of the trajectory accuracy – they are simply hallucinated by an LLM. Therefore, we hypothesize that the two sources can compensate each other. To test this hypothesis, we trained three models: one using $9k$ synthetic data mixed from both sources, and two others each using $9k$ data exclusively from one of the sources and the results are shown in the middle of Figure 8.5. We observe a noticeable performance degradation when models are trained with data from only one source. This indicates that utilizing multiple sources yields a more comprehensive dataset by integrating the precise procedural knowledge from tutorials with the realistic observations of web snapshot data.

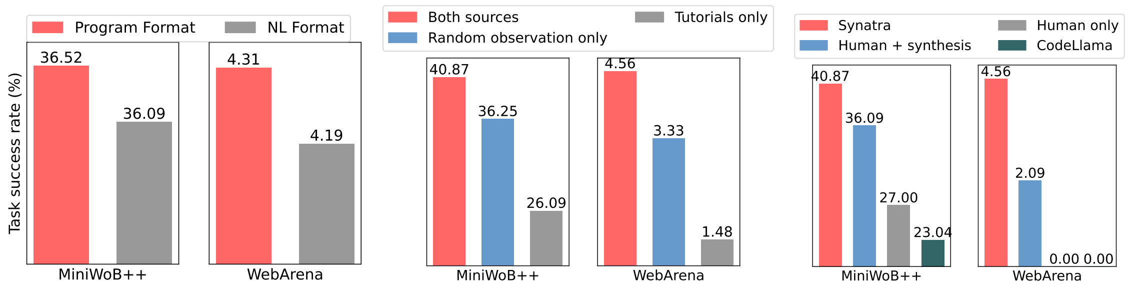
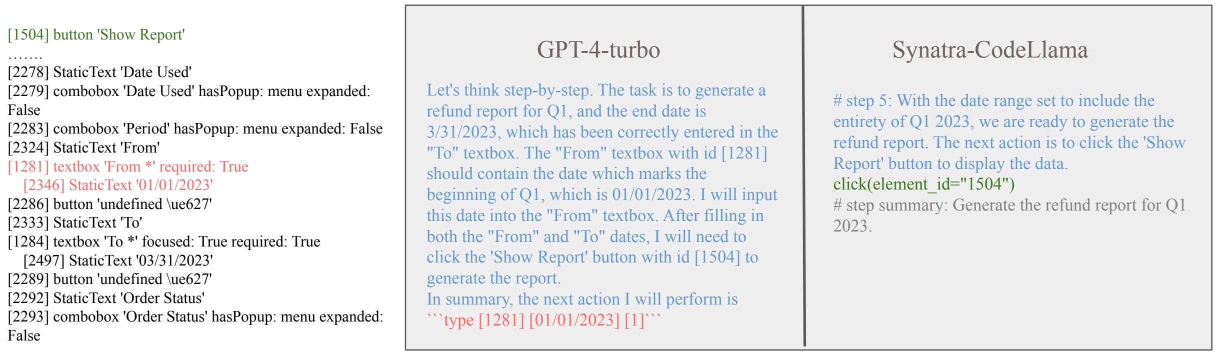


Figure 8.5: Left: the comparison between different trajectory formats. Middle: the comparison between different sources of indirect knowledge. Right: comparison between the models trained with trajectories generated by our approach and the data collected from human.

8.9 Case Study

We conducted a detailed examination of instances where `Synatra-CodeLlama` successfully completes tasks that `GPT-4-turbo` fails to accomplish on WebArena. We present a specific example in Figure 8.6, where both agents are required to create a refund report for Q1 2023 in the admin portal of an online store. Up to this point, both agents had correctly entered the start and end dates. However, `GPT-4` incorrectly predicts that the next step is to re-enter the starting date, whereas `Synatra-CodeLlama` correctly interprets the status of the web page ("with the date range set to include the entirety of Q1 2023"), plans appropriately ("we are ready to generate the refund report"), and executes the correct action. This example illustrates that data generated by Synatra could enhance the capability of a model in accurately understanding and responding to the context, thereby enhancing the efficiency and reliability of task execution in dynamic settings.



```
[1504] button 'Show Report'
.......
[2278] StaticText 'Date Used'
[2279] combobox 'Date Used' hasPopup: menu expanded:
False
[2283] combobox 'Period' hasPopup: menu expanded: False
[2324] StaticText 'From'
[1281] textbox 'From *' required: True
    [2346] StaticText '01/01/2023'
[2286] button 'undefined \ue627'
[2333] StaticText 'To'
[1284] textbox 'To *' focused: True required: True
    [2497] StaticText '03/31/2023'
[2289] button 'undefined \ue627'
[2292] StaticText 'Order Status'
[2293] combobox 'Order Status' hasPopup: menu expanded:
False
```

GPT-4-turbo

Let's think step-by-step. The task is to generate a refund report for Q1, and the end date is 3/31/2023, which has been correctly entered in the "To" textbox. The "From" textbox with id [1281] should contain the date which marks the beginning of Q1, which is 01/01/2023. I will input this date into the "From" textbox. After filling in both the "From" and "To" dates, I will need to click the 'Show Report' button with id [1504] to generate the report.
In summary, the next action I will perform is
```type [1281] [01/01/2023] [1]```

Synatra-CodeLlama

# step 5: With the date range set to include the entirety of Q1 2023, we are ready to generate the refund report. The next action is to click the 'Show Report' button to display the data.
click(element_id="1504")
# step summary: Generate the refund report for Q1 2023.

Figure 8.6: An example task where `Synatra-CodeLlama` is successful while `GPT-4-turbo` is not.

Author Contributions Tianyue Ou and Shuyan Zhou were the primary contributors to this work. Shuyan Zhou proposed the idea, designed the data generation paradigm, conducted the major part of the data generation and analysis. Tianyue Ou performed detailed investigations and revisions of the data generation pipeline, fine-tuned the agent models, and conducted the evaluations.

Table 8.1: Performance of various models in different tasks. We measure step accuracy (%) for Mind2Web, and task success rate (%) for MiniWoB++ and WebArena. The numbers of `FireAct-7b` is taken from [36]; `AutoWebGLM-7b(S1)` represents the model trained with only synthetic data in [90]. All other numbers are reproduced by our work under with the same configurations.

Model	**Mind2Web**	**MiniWoB++**	**WebArena**
	Single step Reference-based	Short Execution-based	Long Execution-based
API-based Models			
`GPT-3.5`	12.79	39.57	6.16
`GPT-4`	29.09	53.04	14.41
Open Source Instructed Models			
`CodeLlama-instruct-7b`	6.62	23.04	0.00
`Llama3-chat-8b`	11.50	31.74	3.32
`Llama3-chat-70b`	22.27	48.70	7.02
Open Source Interactive Data Finetuned Models			
`FireAct-7b` [28]	-	-	0.25*
`AgentLM-7b` [192]	2.99	15.65	0.86
`CodeActAgent-7b` [164]	3.13	9.78	2.34
`AutoWebGLM-7b(S1)` [90]	-	-	2.50*
`AgentFlan-7b` [36]	3.80	20.87	0.62
`Lemur-chat-70b` [179]	14.28	21.30	3.33
`AgentLM-70b` [192]	10.61	36.52	3.07
`Synatra-CodeLlama-7b`	17.26	39.57	4.80

Chapter 9

Conclusion and Future Work

This thesis contributes to the ecosystem for creating general-use agents for real-world tasks. Part I introduces our efforts to build WebArena that features complex real-world tasks and accurate outcome-based evaluations. We believe that good benchmarks and evaluation setups are key steps for research innovations. WebArena uncovers the fundamental limitations of LLM-powered AI agents, and provides a shared playground for future development.

In Part II, we propose instructing AI agents to use programming languages, even for non-coding tasks. Programming languages inherently encode structures and contain many built-in concepts useful for complex problem-solving. Utilizing programming languages as the medium for AI agents leverages both the design of these languages and the benefits of pretraining on large-scale coding corpora. Our findings, particularly those presented in PaL (Chapter 5), have inspired many follow-up works on LLMs tool use. The idea has been widely applied in products such as the ChatGPT code interpreter[1] and Bard's implicit code executions.[2]

Part III presents large-scale resources encoding the hierarchies of procedures previously absent in human-authored knowledge. Observations from this part inspired the choice of procedural knowledge sources and the demonstration conversion mechanism described in Chapter 8.

Finally, in Part IV, we explore leveraging human-authored indirect knowledge for agent task learning, thus circumventing the limitations of human annotations for data scaling. In Chapter 7, we propose a method that involves first retrieving relevant knowledge and then generating corresponding actions. This approach mirrors the human workflow of learning from documentation, enabling AI agents to utilize knowledge originally intended for human use. This method obviates the need for specific annotations, such as exact demonstrations for

[1]https://openai.com/blog/chatgpt-plugins#code-interpreter
[2]https://blog.google/technology/ai/bard-improved-reasoning-google-sheets-export/

agents, which are often unnecessarily labor-intensive for humans. In Chapter 8, we introduce Synatra, which capitalizes on the strengths of LLMs in language processing and code generation to synthesize demonstrations and enhance model performance on tasks where they typically underperform.

9.1 Open Problems and Future Directions

9.1.1 Safeguard against Execution Risks

AI agents pose *execution risks* as they can take actions that directly impact the environment or user assets. For example, they may access credit card information and make irrational transactions, resulting in financial distress. A future direction is to **develop terminologies, metrics, and methodologies** for evaluating execution risks, similar to current benchmarks for assessing risks such as hallucination and factuality in LLMs.

Measuring these risks requires real executions in controlled environments. As we anticipate increasing AI agent capabilities, the features of the testing environment must evolve accordingly, becoming more resource-intensive in both time and cost than current environments like WebArena (Chapter 3). Thus, an interesting direction is to explore **generative agent benchmarks** that use generative technologies to enhance the authenticity and controllability of the environments in a cost-efficient way [146, 180]. For instance, can we develop approaches that use LLMs as an auxiliary environment engine, describing specific and intricate environment dynamics that are otherwise difficult to implement manually?

9.1.2 Personalized AI Agents

Personalized AI agents are a vital cornerstone for developing generally useful agents. We view **software engineering automation** with AI agents as a tangible next step, enabling individuals to develop customized applications effortlessly, even without programming experience. The works presented in this thesis primarily focus on single-round code generation. Expanding this to more dynamic scenarios with executions and feedback loops will be interesting.

In addition, as AI agents become more adept at specific tasks and involved in mission-critical, multi-task jobs, the need for enhanced **agent memory** will increase. We explored creating hierarchical procedural memory with programs and retrieving memory with goals in this thesis. Moving forward, open questions include investigating novel memory architectures to store diverse types of memory (*e.g.*, semantic memories of facts and concepts, procedural memories of processes [85]). Learning algorithms that enable agents to refine their memory based on experiences is another intriguing topic.

Finally, our focus has been activating AI agents through natural language commands, where agents act as passive executors. It will be interesting to investigate **new modes of agent-human interaction** where agents can proactively offer suggestions, even before being explicitly asked [96].

9.1.3 Physical Robots

While this thesis primarily focuses on digital agents (except for Chapter 4, which discusses robots in simulated environments), the expansion of AI agent research into physical robots to reduce *manual* labor in home environments is a natural progression. A crucial step is to develop advanced **large multi-modal models (LMMs)**, as robotics tasks often require multi-modal inputs. Specifically, procuring high-quality multi-modal data and innovating more efficient model architectures for LMMs are essential [25]. Additionally, developing **algorithms that bridge LMMs with robotic systems** to enhance their *semantic* understanding for tasks such as precise scene interpretation, task planning, and efficient tool utilization is necessary [42]. Finally, it will be exciting to explore **LMMs for low-level tasks** such as dexterous manipulations [105], addressing Moravec's Paradox concerning their computational demands[3].

[3]https://en.wikipedia.org/wiki/Moravec's_paradox

Bibliography

[1] Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. Unified pre-training for program understanding and generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2655–2668, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.211. URL https://aclanthology.org/2021.naacl-main.211. (page 10)

[2] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. *ArXiv preprint*, abs/2204.01691, 2022. URL https://arxiv.org/abs/2204.01691. (page 49)

[3] Miltiadis Allamanis, Daniel Tarlow, Andrew D. Gordon, and Yi Wei. Bimodal modelling of source code and natural language. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2123–2132. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/allamanis15.html. (page 81)

[4] Uri Alon, Roy Sadaka, Omer Levy, and Eran Yahav. Structural language models of code. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 245–256. PMLR, 2020. URL http://proceedings.mlr.press/v119/alon20a.html. (page 81)

[5] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian D. Reid, Stephen Gould, and Anton van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 3674–3683. IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00387. URL http://openaccess.thecvf.com/content_cvpr_2018/html/

Anderson_Vision-and-Language_Navigation_Interpreting_CVPR_2018_paper.html. (page 15)

[6] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian D. Reid, Stephen Gould, and Anton van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 3674–3683. IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00387. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Anderson_Vision-and-Language_Navigation_Interpreting_CVPR_2018_paper.html. (page 2)

[7] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 39–48. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.12. URL https://doi.org/10.1109/CVPR.2016.12. (pages 34 and 39)

[8] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 166–175. PMLR, 2017. URL http://proceedings.mlr.press/v70/andreas17a.html. (pages 3 and 33)

[9] Marcin Andrychowicz, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5048–5058, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/453fadbd8a1a3af50a9df4df899537b5-Abstract.html. (pages 11 and 95)

[10] 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, 2023. (page 27)

[11] Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1:49–62, 2013. doi: 10.1162/tacl_a_00209. URL https://aclanthology.org/Q13-1005. (pages 3, 9, 33, 34, and 35)

[12] Yoav Artzi, Dipanjan Das, and Slav Petrov. Learning compact lexicons for CCG semantic parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1273–1283, Doha, Qatar, 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1134. URL https://aclanthology.org/D14-1134. (pages 9 and 35)

[13] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007. (page 66)

[14] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *ArXiv preprint*, abs/2108.07732, 2021. URL https://arxiv.org/abs/2108.07732. (pages 9, 82, and 88)

[15] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.0473. (page 42)

[16] Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *ArXiv preprint*, abs/2406.11896, 2024. URL https://arxiv.org/abs/2406.11896. (page 8)

[17] Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654, 2022. (page 11)

[18] Nathanaël Beau and Benoit Crabbé. The impact of lexical and grammatical processing on generating code from natural language. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2204–2214, Dublin, Ireland, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.173. URL https://aclanthology.org/2022.findings-acl.173. (page 91)

[19] Yonatan Bisk, Jan Buys, Karl Pichotta, and Yejin Choi. Benchmarking hierarchical script knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4077–4085, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1412. URL https://aclanthology.org/N19-1412. (page 17)

[20] Sid Black, Leo Gao, Phil Wang, Connor Leahy, and Stella Biderman. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow, 2021. URL https://doi.org/10.5281/zenodo.5297715. If you use this software, please cite it using these metadata. (pages 82 and 86)

[21] S.R.K. Branavan, Harr Chen, Luke Zettlemoyer, and Regina Barzilay. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 82–90, Suntec, Singapore, 2009. Association

for Computational Linguistics. URL https://aclanthology.org/P09-1010. (pages 10, 11, and 29)

[22] S.R.K. Branavan, Luke Zettlemoyer, and Regina Barzilay. Reading between the lines: Learning to map high-level instructions to commands. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1268–1277, Uppsala, Sweden, 2010. Association for Computational Linguistics. URL https://aclanthology.org/P10-1129. (pages 10 and 11)

[23] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. URL https://arxiv.org/abs/1606.01540. (page 16)

[24] Marc Brockschmidt, Miltiadis Allamanis, Alexander L. Gaunt, and Oleksandr Polozov. Generative code modeling with graphs. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bke4KsA5FX. (page 81)

[25] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *ArXiv preprint*, abs/2307.15818, 2023. URL https://arxiv.org/abs/2307.15818. (page 113)

[26] 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. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html. (pages 49, 50, 51, and 53)

[27] Sahil Chaudhary. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca, 2023. (page 11)

[28] Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu

Yao. Fireact: Toward language agent fine-tuning. *ArXiv preprint*, abs/2310.05915, 2023. URL https://arxiv.org/abs/2310.05915. (pages 11 and 109)

[29] David L. Chen and Raymond J. Mooney. Learning to interpret natural language navigation instructions from observations. In Wolfram Burgard and Dan Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press, 2011. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3701. (pages 33, 34, and 36)

[30] 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*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374. (pages 8 and 50)

[31] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, Harri Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *ArXiv preprint*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374. (pages 17, 82, 86, and 88)

[32] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR, 2020. URL http://proceedings.mlr.press/v119/chen20j.html. (page 85)

[33] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *ArXiv preprint*, abs/2211.12588, 2022. URL https://arxiv.org/abs/2211.12588. (pages 9 and 98)

[34] Xinyun Chen, Chen Liang, Adams Wei Yu, Dawn Song, and Denny Zhou. Compositional generalization via neural-symbolic stack machines. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/12b1e42dc0746f22cf361267de07073f-Abstract.html. (page 35)

[35] Xinyun Chen, Petros Maniatis, Rishabh Singh, Charles Sutton, Hanjun Dai, Max Lin, and Denny Zhou. Spreadsheetcoder: Formula prediction from semi-structured context. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 1661–1672. PMLR, 2021. URL http://proceedings.mlr.press/v139/chen21m.html. (pages 3 and 9)

[36] Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. Agent-flan: Designing data and methods of effective agent tuning for large language models. *ArXiv preprint*, abs/2403.12881, 2024. URL https://arxiv.org/abs/2403.12881. (page 109)

[37] 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*, abs/2204.02311, 2022. URL https://arxiv.org/abs/2204.02311. (pages 50 and 51)

[38] Cuong Xuan Chu, Niket Tandon, and Gerhard Weikum. Distilling task knowledge from

how-to communities. In Rick Barrett, Rick Cummings, Eugene Agichtein, and Evgeniy Gabrilovich, editors, *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, pages 805–814. ACM, 2017. doi: 10.1145/3038912.3052715. URL https://doi.org/10.1145/3038912.3052715. (page 66)

[39] Cuong Xuan Chu, Niket Tandon, and Gerhard Weikum. Distilling task knowledge from how-to communities. In Rick Barrett, Rick Cummings, Eugene Agichtein, and Evgeniy Gabrilovich, editors, *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, pages 805–814. ACM, 2017. doi: 10.1145/3038912.3052715. URL https://doi.org/10.1145/3038912.3052715. (page 3)

[40] James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. Driving semantic parsing from the world's response. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, pages 18–27, Uppsala, Sweden, 2010. Association for Computational Linguistics. URL https://aclanthology.org/W10-2903. (page 9)

[41] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training Verifiers to Solve Math Word Problems. *ArXiv preprint*, abs/2110.14168, 2021. URL https://arxiv.org/abs/2110.14168. (page 53)

[42] OXE Collaboration, A Padalkar, A Pooley, A Jain, A Bewley, A Herzog, A Irpan, A Khazatsky, A Rai, A Singh, et al. Open x-embodiment: Robotic learning datasets and rt-x models. *ArXiv preprint*, abs/2310.08864, 2023. URL https://arxiv.org/abs/2310.08864. (page 113)

[43] Michele Colledanchise and Petter Ögren. *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018. (page 36)

[44] Abhishek Das, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Neural Modular Control for Embodied Question Answering. *ArXiv preprint*, abs/1810.11181, 2018. URL https://arxiv.org/abs/1810.11181. (pages 3 and 33)

[45] Matt Deitke, Winson Han, Alvaro Herrasti, Aniruddha Kembhavi, Eric Kolve, Roozbeh Mottaghi, Jordi Salvador, Dustin Schwenk, Eli VanderBilt, Matthew Wallingford, Luca Weihs, Mark Yatskar, and Ali Farhadi. Robothor: An open simulation-to-real embodied AI platform. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 3161–3171. IEEE, 2020. doi: 10.1109/CVPR42600.2020.00323. URL https://doi.org/10.1109/CVPR42600.2020.00323. (page 39)

[46] 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, 2023. (pages 3, 7, 15, 16, 17, 20, 29, 96, 97, 103, 105, and 149)

[47] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423. (page 69)

[48] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000. (page 3)

[49] 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?, 2024. (page 95)

[50] Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sable-Meyer, Luc Cary, Lucas Morales, Luke Hewitt, Armando Solar-Lezama, and Joshua B Tenenbaum. Dreamcoder: Growing generalizable, interpretable knowledge with wake-sleep bayesian program learning. *ArXiv preprint*, abs/2006.08381, 2020. URL https://arxiv.org/abs/2006.08381. (page 38)

[51] Mikhail Evtikhiev, Egor Bogomolov, Yaroslav Sokolov, and Timofey Bryksin. Out of the bleu: how should we assess quality of the code generation models? *Journal of Systems and Software*, 203:111741, 2023. (page 8)

[52] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL https://openreview.net/forum?id=rc8o_j8I8PX. (pages 3, 11, 16, and 29)

[53] Andrew Forward and Timothy C Lethbridge. The relevance of software documentation, tools and technologies: a survey. In *Proceedings of the 2002 ACM symposium on Document engineering*, pages 26–33, 2002. (page 83)

[54] Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran,

Kyunghoon Bae, and Honglak Lee. Autoguide: Automated generation and selection of state-aware guidelines for large language model agents. *ArXiv preprint*, abs/2403.08978, 2024. URL https://arxiv.org/abs/2403.08978. (page 11)

[55] 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*, abs/2305.11854, 2023. URL https://arxiv.org/abs/2305.11854. (page 95)

[56] Luyu Gao*, Aman Madaan*, Shuyan Zhou*, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR, 2023. (page 98)

[57] Tianyu Gao, Xingcheng Yao, and Danqi Chen. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.552. URL https://aclanthology.org/2021.emnlp-main.552. (pages 82, 85, and 92)

[58] Edward M Gellenbeck and Curtis R Cook. An investigation of procedure and variable names as beacons during program comprehension. In *Empirical studies of programmers: Fourth workshop*, pages 65–81. Ablex Publishing, Norwood, NJ, 1991. (page 61)

[59] Daniel Gordon, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, Dieter Fox, and Ali Farhadi. IQA: visual question answering in interactive environments. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4089–4098. IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00430. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Gordon_IQA_Visual_Question_CVPR_2018_paper.html. (pages 2, 3, 15, 33, 35, 39, 43, and 44)

[60] Kristen Grauman, Andrew Westbury, Eugene Byrne, Zachary Chavis, Antonino Furnari, Rohit Girdhar, Jackson Hamburger, Hao Jiang, Miao Liu, Xingyu Liu, Miguel Martin, Tushar Nagarajan, Ilija Radosavovic, Santhosh Kumar Ramakrishnan, Fiona Ryan, Jayant Sharma, Michael Wray, Mengmeng Xu, Eric Zhongcong Xu, Chen Zhao, Siddhant Bansal, Dhruv Batra, Vincent Cartillier, Sean Crane, Tien Do, Morrie Doulaty, Akshay Erapalli, Christoph Feichtenhofer, Adriano Fragomeni, Qichen Fu, Abrham Gebreselasie, Cristina González, James Hillis, Xuhua Huang, Yifei Huang, Wenqi Jia, Weslie Khoo, Jáchym Kolár, Satwik Kottur, Anurag Kumar, Federico Landini, Chao Li, Yanghao Li, Zhenqiang Li, Karttikeya Mangalam, Raghava Modhugu, Jonathan Munro, Tullie Mur-

rell, Takumi Nishiyasu, Will Price, Paola Ruiz Puentes, Merey Ramazanova, Leda Sari, Kiran Somasundaram, Audrey Southerland, Yusuke Sugano, Ruijie Tao, Minh Vo, Yuchen Wang, Xindi Wu, Takuma Yagi, Ziwei Zhao, Yunyi Zhu, Pablo Arbeláez, David Crandall, Dima Damen, Giovanni Maria Farinella, Christian Fuegen, Bernard Ghanem, Vamsi Krishna Ithapu, C. V. Jawahar, Hanbyul Joo, Kris Kitani, Haizhou Li, Richard A. Newcombe, Aude Oliva, Hyun Soo Park, James M. Rehg, Yoichi Sato, Jianbo Shi, Mike Zheng Shou, Antonio Torralba, Lorenzo Torresani, Mingfei Yan, and Jitendra Malik. Ego4d: Around the world in 3, 000 hours of egocentric video. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 18973–18990. IEEE, 2022. doi: 10.1109/CVPR52688.2022.01842. URL https://doi.org/10.1109/CVPR52688.2022.01842. (page 11)

[61] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 7272–7281. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.769. URL https://doi.org/10.1109/CVPR.2017.769. (page 33)

[62] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962, 2023. (page 9)

[63] 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*, abs/2307.12856, 2023. URL https://arxiv.org/abs/2307.12856. (page 99)

[64] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.740. URL https://aclanthology.org/2020.acl-main.740. (page 10)

[65] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training. *ArXiv preprint*, abs/2002.08909, 2020. URL https://arxiv.org/abs/2002.08909. (page 83)

[66] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. *ArXiv preprint*, abs/2401.13919, 2024. URL https://arxiv.org/abs/2401.13919. (page 8)

[67] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: decoding-enhanced bert with disentangled attention. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=XPZIaotutsD. (pages 69 and 72)

[68] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=rygGQyrFvH. (pages 57 and 88)

[69] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spaCy: Industrial-strength Natural Language Processing in Python, 2020. URL https://doi.org/10.5281/zenodo.1212303. (page 76)

[70] Samuel Humeau, Kurt Shuster, Marie-Anne Lachaux, and Jason Weston. Poly-encoders: Transformer architectures and pre-training strategies for fast and accurate multi-sentence scoring. *ArXiv preprint*, abs/1905.01969, 2019. URL https://arxiv.org/abs/1905.01969. (page 66)

[71] Peter C. Humphreys, David Raposo, Tobias Pohlen, Gregory Thornton, Rachita Chhaparia, Alistair Muldal, Josh Abramson, Petko Georgiev, Adam Santoro, and Timothy P. Lillicrap. A data-driven approach for learning to control computers. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 9466–9482. PMLR, 2022. URL https://proceedings.mlr.press/v162/humphreys22a.html. (pages 96 and 104)

[72] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973, Vancouver, Canada, 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1089. URL https://aclanthology.org/P17-1089. (page 35)

[73] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Mapping language to code in programmatic context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1643–1652, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1192. URL https://aclanthology.org/D18-1192. (page 81)

[74] Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.74. URL https://aclanthology.org/2021.eacl-main.74. (pages 82 and 86)

[75] Yacine Jernite, Kavya Srinet, Jonathan Gray, and Arthur Szlam. CraftAssist Instruction Parsing: Semantic Parsing for a Minecraft Assistant. *ArXiv preprint*, abs/1905.01978, 2019. URL https://arxiv.org/abs/1905.01978. (pages 2, 3, and 16)

[76] Yu-qian Jiang, Shi-qi Zhang, Piyush Khandelwal, and Peter Stone. Task planning in robotics: an empirical comparison of pddl-and asp-based systems. *Frontiers of Information Technology & Electronic Engineering*, 20(3):363–373, 2019. (page 36)

[77] Zhengbao Jiang, Zhiqing Sun, Weijia Shi, Pedro Rodriguez, Chunting Zhou, Graham Neubig, Xi Victoria Lin, Wen-tau Yih, and Srinivasan Iyer. Instruction-tuned language models are better knowledge learners. *ArXiv preprint*, abs/2402.12847, 2024. URL https://arxiv.org/abs/2402.12847. (page 10)

[78] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *ArXiv preprint*, abs/1702.08734, 2017. URL https://arxiv.org/abs/1702.08734. (page 68)

[79] Siddharth Karamcheti, Dorsa Sadigh, and Percy Liang. Learning adaptive language interfaces through decomposition. In *Proceedings of the First Workshop on Interactive and Executable Semantic Parsing*, pages 23–33, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.intexsempar-1.4. URL https://aclanthology.org/2020.intexsempar-1.4. (page 41)

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

emnlp-main.550. (pages 69 and 85)

[81] Najoung Kim and Sebastian Schuster. Entity tracking in language models. *ArXiv preprint*, abs/2305.02363, 2023. URL https://arxiv.org/abs/2305.02363. (page 10)

[82] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Annual Conference of the Association for Computational Linguistics (ACL)*, Bangkok, Thailand, August 2024. URL https://arxiv.org/abs/2401.13649. (page 8)

[83] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*, 2017. (page 29)

[84] Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. MAWPS: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157, San Diego, California, 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1136. URL https://aclanthology.org/N16-1136. (page 53)

[85] Iuliia Kotseruba and John K Tsotsos. 40 years of cognitive architectures: core cognitive abilities and practical applications. *Artificial Intelligence Review*, 53(1):17–94, 2020. (page 113)

[86] Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-2012. URL https://aclanthology.org/D18-2012. (page 68)

[87] Heinrich Küttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/569ff987c643b4bedf504efda8f786c2-Abstract.html. (pages 3 and 29)

[88] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026. (page 22)

[89] Nikolaos Lagos, Matthias Gallé, Alexandr Chernov, and Ágnes Sándor. Enriching how-to guides with actionable phrases and linked data. In *Web Intelligence*, volume 15, pages 189–203. IOS Press, 2017. (pages 66 and 68)

[90] Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, et al. Autowebglm: Bootstrap and reinforce a large language model-based web navigating agent. *ArXiv preprint*, abs/2404.03648, 2024. URL https://arxiv.org/abs/2404.03648. (page 109)

[91] Yann LeCun. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, 62, 2022. (page 17)

[92] Timothy C Lethbridge, Janice Singer, and Andrew Forward. How software engineers use documentation: The state of the practice. *IEEE software*, 20(6):35–39, 2003. (page 82)

[93] Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html. (page 83)

[94] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. *ArXiv preprint*, abs/2206.14858, 2022. URL https://arxiv.org/abs/2206.14858. (pages 49, 56, and 57)

[95] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al.

Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857, 2022. (pages 3 and 9)

[96] Belinda Z Li, Alex Tamkin, Noah Goodman, and Jacob Andreas. Eliciting human preferences with language models. *ArXiv preprint*, abs/2310.11589, 2023. URL https://arxiv.org/abs/2310.11589. (page 113)

[97] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. Mapping natural language instructions to mobile UI action sequences. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8198–8210, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.729. URL https://aclanthology.org/2020.acl-main.729. (pages 2, 4, 7, 16, 20, 29, and 33)

[98] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. (page 45)

[99] Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D. Ernst. NL2Bash: A corpus and semantic parser for natural language interface to the linux operating system. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, 2018. European Language Resources Association (ELRA). URL https://aclanthology.org/L18-1491. (page 87)

[100] Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program Induction by Rationale Generation: Learning to Solve and Explain Algebraic Word Problems. *ArXiv preprint*, abs/1705.04146, 2017. URL https://arxiv.org/abs/1705.04146. (page 9)

[101] Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=ryTp3f-0-. (pages 20, 29, and 149)

[102] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *ArXiv preprint*, abs/2107.13586, 2021. URL https://arxiv.org/abs/2107.13586. (page 51)

[103] Xing Han Lù, Zdeněk Kasner, and Siva Reddy. Weblinx: Real-world website navigation with multi-turn dialogue. *ArXiv preprint*, abs/2402.05930, 2024. URL https://arxiv.org/

abs/2402.05930. (page 7)

[104] Huaishao Luo, Lei Ji, Ming Zhong, Yang Chen, Wen Lei, Nan Duan, and Tianrui Li. CLIP4Clip: An empirical study of clip for end to end video clip retrieval. *ArXiv preprint*, abs/2104.08860, 2021. URL https://arxiv.org/abs/2104.08860. (page 76)

[105] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *ArXiv preprint*, abs/2310.12931, 2023. URL https://arxiv.org/abs/2310.12931. (page 113)

[106] Aman Madaan and Amir Yazdanbakhsh. Text and patterns: For effective chain of thought, it takes two to tango. *ArXiv preprint*, abs/2209.07686, 2022. URL https://arxiv.org/abs/2209.07686. (page 54)

[107] Aman Madaan, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. Language models of code are few-shot commonsense learners. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1384–1403, Abu Dhabi, United Arab Emirates, 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.emnlp-main.90. (pages 49, 60, and 61)

[108] Aman Madaan, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. Language models of code are few-shot commonsense learners. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1384–1403, Abu Dhabi, United Arab Emirates, 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.emnlp-main.90. (pages 9, 98, and 106)

[109] Robert McCarthy, Daniel CH Tan, Dominik Schmidt, Fernando Acero, Nathan Herr, Yilun Du, Thomas G Thuruthel, and Zhibin Li. Towards generalist robot learning from internet video: A survey. *ArXiv preprint*, abs/2404.19664, 2024. URL https://arxiv.org/abs/2404.19664. (page 11)

[110] Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. A diverse corpus for evaluating and developing English math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 975–984, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.92. URL https://aclanthology.org/2020.acl-main.92. (page 53)

[111] Antoine Miech, Dimitri Zhukov, Jean-Baptiste Alayrac, Makarand Tapaswi, Ivan Laptev, and Josef Sivic. Howto100m: Learning a text-video embedding by watching hundred

million narrated video clips. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 2630–2640. IEEE, 2019. doi: 10.1109/ICCV.2019.00272. URL https://doi.org/10.1109/ICCV.2019.00272. (pages 74 and 101)

[112] Swaroop Mishra, Matthew Finlayson, Pan Lu, Leonard Tang, Sean Welleck, Chitta Baral, Tanmay Rajpurohit, Oyvind Tafjord, Ashish Sabharwal, Peter Clark, and Ashwin Kalyan. LILA: A unified benchmark for mathematical reasoning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5807–5832, Abu Dhabi, United Arab Emirates, 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.emnlp-main.392. (pages 9 and 49)

[113] Dipendra K Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. Tell me dave: Context-sensitive grounding of natural language to manipulation instructions. *The International Journal of Robotics Research*, 35(1-3):281–300, 2016. (pages 15, 33, 34, 35, and 36)

[114] Yoshio Momouchi. Control structures for actions in procedural texts and PT-chart. In *COLING 1980 Volume 1: The 8th International Conference on Computational Linguistics*, 1980. URL https://aclanthology.org/C80-1016. (page 65)

[115] Shikhar Murty, Christopher Manning, Peter Shaw, Mandar Joshi, and Kenton Lee. Bagel: Bootstrapping agents by guiding exploration with language. *ArXiv preprint*, abs/2403.08140, 2024. URL https://arxiv.org/abs/2403.08140. (pages 11 and 95)

[116] Karthik Narasimhan, Regina Barzilay, and Tommi Jaakkola. Grounding language for transfer in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 63: 849–874, 2018. (pages 10 and 11)

[117] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. A conversational paradigm for program synthesis. *arXiv preprint*, 2022. (page 82)

[118] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your Work: Scratchpads for Intermediate Computation with Language Models. *ArXiv preprint*, abs/2112.00114, 2021. URL https://arxiv.org/abs/2112.00114. (page 60)

[119] Janet Nykaza, Rhonda Messinger, Fran Boehme, Cherie L Norman, Matthew Mace, and Manuel Gordon. What programmers really want: results of a needs assessment for sdk

documentation. In *Proceedings of the 20th annual international conference on Computer documentation*, pages 133–141, 2002. (page 82)

[120] OpenAI. Chatgpt: Optimizing language models for dialogue. *OpenAI Blog*, 2022. (page 27)

[121] OpenAI. Gpt-4 technical report. *arXiv*, pages 2303–08774, 2023. (page 27)

[122] Tianyue Ou, Frank F. Xu, Aman Madaan, Jiarui Liu, Robert Lo, Abishek Sridhar, Sudipta Sengupta, Dan Roth, Graham Neubig, and Shuyan Zhou. Synatra: Turning indirect knowledge into direct demonstrations for digital agents at scale. In *submission to NeurIPS*, 2024. No citations.

[123] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022. (page 157)

[124] Arnold Overwijk, Chenyan Xiong, Xiao Liu, Cameron VandenBerg, and Jamie Callan. Clueweb22: 10 billion web documents with visual and semantic information. *ArXiv preprint*, abs/2211.15848, 2022. URL https://arxiv.org/abs/2211.15848. (page 101)

[125] Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. Autonomous evaluation and refinement of digital agents. *ArXiv preprint*, abs/2404.06474, 2024. URL https://arxiv.org/abs/2404.06474. (page 8)

[126] Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, et al. Webcanvas: Benchmarking web agents in online environments. *ArXiv preprint*, abs/2406.12373, 2024. URL https://arxiv.org/abs/2406.12373. (page 8)

[127] Paolo Pareti, Benoit Testu, Ryutaro Ichise, Ewan Klein, and Adam Barker. Integrating know-how into the linked data cloud. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 385–396. Springer, 2014. (page 66)

[128] Md Rizwan Parvez, Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. Retrieval augmented code generation and summarization. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2719–2734, Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.232. URL https://aclanthology.org/2021.findings-emnlp.232. (page 90)

[129] Panupong Pasupat, Yuan Zhang, and Kelvin Guu. Controllable semantic parsing via retrieval augmentation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7683–7698, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.607. URL https://aclanthology.org/2021.emnlp-main.607. (page 90)

[130] Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP Models Really Able to Solve Simple Math Word Problems? *ArXiv preprint*, abs/2103.07191, 2021. URL https://arxiv.org/abs/2103.07191. (page 53)

[131] Haritz Puerto, Martin Tutek, Somak Aditya, Xiaodan Zhu, and Iryna Gurevych. Code prompting elicits conditional reasoning abilities in text+ code llms. *ArXiv preprint*, abs/2401.10065, 2024. URL https://arxiv.org/abs/2401.10065. (page 106)

[132] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 8494–8502. IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00886. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Puig_VirtualHome_Simulating_Household_CVPR_2018_paper.html. (pages 3, 4, 15, 16, 17, and 29)

[133] 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. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020. URL http://jmlr.org/papers/v21/20-074.html. (page 86)

[134] 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. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020. URL http://jmlr.org/papers/v21/20-074.html. (page 82)

[135] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264. (page 23)

[136] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable

questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2124. URL https://aclanthology.org/P18-2124. (page 72)

[137] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2124. URL https://aclanthology.org/P18-2124. (page 25)

[138] Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Androidinthewild: A large-scale dataset for android device control. *Advances in Neural Information Processing Systems*, 36, 2024. (page 7)

[139] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410. (page 68)

[140] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, 2009. ISSN 1554-0669. doi: 10.1561/1500000019. URL https://doi-org.proxy.library.upenn.edu/10.1561/1500000019. (pages 68 and 75)

[141] Stephen E Robertson and K Sparck Jones. Relevance weighting of search terms. *Journal of the American Society for Information science*, 27(3):129–146, 1976. (pages 82 and 85)

[142] Pedro Rodriguez and Jordan Boyd-Graber. Evaluation paradigms in question answering. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9630–9642, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.758. URL https://aclanthology.org/2021.emnlp-main.758. (page 93)

[143] Tobias Roehm, Rebecca Tiarks, Rainer Koschke, and Walid Maalej. How do professional developers comprehend software? In *2012 34th International Conference on Software Engineering (ICSE)*, pages 255–265. IEEE, 2012. (page 83)

[144] Subhro Roy and Dan Roth. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1743–1752, Lisbon, Portugal, 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1202. URL https://aclanthology.org/D15-1202. (page 3)

[145] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *ArXiv preprint*, abs/2308.12950, 2023. URL https://arxiv.org/abs/2308.12950. (page 103)

[146] Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J Maddison, and Tatsunori Hashimoto. Identifying the risks of lm agents with an lm-emulated sandbox. *arXiv preprint*, 2023. (page 112)

[147] Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Stella Biderman, Leo Gao, Tali Bers, Thomas Wolf, and Alexander M. Rush. Multitask Prompted Training Enables Zero-Shot Task Generalization, 2021. URL https://arxiv.org/abs/2110.08207. (page 49)

[148] Gabriel Sarch, Lawrence Jang, Michael J Tarr, William W Cohen, Kenneth Marino, and Katerina Fragkiadaki. Ical: Continual learning of multimodal agents by transforming trajectories into actionable insights. *ArXiv preprint*, abs/2406.14596, 2024. URL https://arxiv.org/abs/2406.14596. (page 11)

[149] 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. *Advances in Neural Information Processing Systems*, 36, 2024. (page 9)

[150] Freda Shi, Daniel Fried, Marjan Ghazvininejad, Luke Zettlemoyer, and Sida I. Wang. Natural language to code translation with execution. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3533–3546, Abu Dhabi, United Arab Emirates, 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.emnlp-main.231. (page 87)

[151] Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3135–3144. PMLR, 2017. URL http://proceedings.mlr.press/v70/shi17a.html. (pages 7, 15, 20, 29, and 149)

[152] Eui Chul Richard Shin, Miltiadis Allamanis, Marc Brockschmidt, and Alex Polozov. Program synthesis and semantic parsing with learned code idioms. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 10824–10834, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/cff34ad343b069ea6920464ad17d4bcf-Abstract.html. (page 35)

[153] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. ALFRED: A benchmark for interpreting grounded instructions for everyday tasks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 10737–10746. IEEE, 2020. doi: 10.1109/CVPR42600.2020.01075. URL https://doi.org/10.1109/CVPR42600.2020.01075. (pages 2, 4, 7, 8, 15, 29, 33, 35, and 39)

[154] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn. (pages 15 and 41)

[155] Chenglei Si, Yanzhe Zhang, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code: How far are we from automating front-end engineering? *ArXiv preprint*, abs/2403.03163, 2024. URL https://arxiv.org/abs/2403.03163. (pages 101 and 106)

[156] Kunal Pratap Singh, Suvaansh Bhambri, Byeonghwi Kim, Roozbeh Mottaghi, and Jonghyun Choi. Moca: A modular object-centric approach for interactive instruction following. *ArXiv preprint*, abs/2012.03208, 2020. URL https://arxiv.org/abs/2012.03208. (pages 45 and 46)

[157] Alane Suhr, Claudia Yan, Jack Schluger, Stanley Yu, Hadi Khader, Marwa Mouallem, Iris

Zhang, and Yoav Artzi. Executing instructions in situated collaborative interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2119–2130, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1218. URL https://aclanthology.org/D19-1218. (pages 2 and 4)

[158] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112 (1-2):181–211, 1999. (page 3)

[159] Mirac Suzgun, Nathan Scales, Nathanael Scharli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed Chi, Denny Zhou, and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them. *ArXiv preprint*, abs/2210.09261, 2022. URL https://arxiv.org/abs/2210.09261. (pages 53, 55, and 57)

[160] Armstrong A Takang, Penny A Grubb, and Robert D Macredie. The effects of comments and identifier names on program comprehensibility: an experimental investigation. *J. Prog. Lang.*, 4(3):143–167, 1996. (page 61)

[161] Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafarali Ahmed, Tyler Jackson, Shibl Mourad, and Doina Precup. Androidenv: A reinforcement learning platform for android. *ArXiv preprint*, abs/2105.13231, 2021. URL https://arxiv.org/abs/2105.13231. (page 29)

[162] Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An Introduction to Probabilistic Programming. *ArXiv preprint*, abs/1809.10756, 2018. URL https://arxiv.org/abs/1809.10756. (page 103)

[163] 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*, abs/2305.16291, 2023. URL https://arxiv.org/abs/2305.16291. (pages 9 and 28)

[164] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents, 2024. (pages 98, 106, and 109)

[165] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. Rationale-Augmented Ensembles in Language Models. *ArXiv preprint*, abs/2207.00747,

2022. URL https://arxiv.org/abs/2207.00747. (page 51)

[166] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models. *ArXiv preprint*, abs/2203.11171, 2022. URL https://arxiv.org/abs/2203.11171. (pages 51, 56, 57, and 58)

[167] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8696–8708, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.685. URL https://aclanthology.org/2021.emnlp-main.685. (pages 81, 82, 85, 86, and 92)

[168] Zhiruo Wang, Shuyan Zhou, Daniel Fried, and Graham Neubig. Execution-based evaluation for open-domain code generation. *ArXiv preprint*, abs/2212.10481, 2022. URL https://arxiv.org/abs/2212.10481. (pages 17 and 88)

[169] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned Language Models are Zero-shot Learners. *ArXiv preprint*, abs/2109.01652, 2021. URL https://arxiv.org/abs/2109.01652. (page 49)

[170] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022. (pages 9, 26, 51, 53, 56, and 57)

[171] John Wieting, Kevin Gimpel, Graham Neubig, and Taylor Berg-Kirkpatrick. Simple and effective paraphrastic similarity from parallel translations. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4602–4608, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1453. URL https://aclanthology.org/P19-1453. (page 68)

[172] John Wieting, Kevin Gimpel, Graham Neubig, and Taylor Berg-kirkpatrick. Paraphrastic representations at scale. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 379–388, Abu Dhabi, UAE, 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.emnlp-demos.38. (page 68)

[173] Ledell Wu, Fabio Petroni, Martin Josifoski, Sebastian Riedel, and Luke Zettlemoyer. Zero-

shot entity linking with dense entity retrieval. corr abs/1911.03814 (2019). *ArXiv preprint*, abs/1911.03814, 2019. URL https://arxiv.org/abs/1911.03814. (pages 66 and 69)

[174] Yuhuai Wu, Albert Q Jiang, Wenda Li, Markus N Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with Large Language Models. *ArXiv preprint*, abs/2205.12615, 2022. URL https://arxiv.org/abs/2205.12615. (page 49)

[175] Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I. Wang, Victor Zhong, Bailin Wang, Chengzu Li, Connor Boyle, Ansong Ni, Ziyu Yao, Dragomir Radev, Caiming Xiong, Lingpeng Kong, Rui Zhang, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. UnifiedSKG: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 602–631, Abu Dhabi, United Arab Emirates, 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.emnlp-main.39. (page 9)

[176] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024. (page 95)

[177] Frank F. Xu, Zhengbao Jiang, Pengcheng Yin, Bogdan Vasilescu, and Graham Neubig. Incorporating external knowledge through pre-training for natural language to code generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6045–6052, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.538. URL https://aclanthology.org/2020.acl-main.538. (pages 11 and 81)

[178] Nancy Xu, Sam Masling, Michael Du, Giovanni Campagna, Larry Heck, James Landay, and Monica Lam. Grounding open-domain instructions to automate web support tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1022–1032, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.80. URL https://aclanthology.org/2021.naacl-main.80. (pages 7, 16, 29, and 33)

[179] Yiheng Xu, Hongjin Su, Chen Xing, Boyu Mi, Qian Liu, Weijia Shi, Binyuan Hui, Fan

Zhou, Yitao Liu, Tianbao Xie, et al. Lemur: Harmonizing natural language and code for language agents. *ArXiv preprint*, abs/2310.06830, 2023. URL https://arxiv.org/abs/2310.06830. (pages 99 and 109)

[180] Mengjiao Yang, Yilun Du, Kamyar Ghasemipour, Jonathan Tompson, Dale Schuurmans, and Pieter Abbeel. Learning interactive real-world simulators. *arXiv preprint*, 2023. (page 112)

[181] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259. (pages 22 and 23)

[182] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 20744–20757. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/82ad13ec01f9fe44c01cb91814fd7b8c-Paper-Conference.pdf. (pages 8, 15, 29, and 147)

[183] 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*, abs/2210.03629, 2022. URL https://arxiv.org/abs/2210.03629. (pages 9, 26, 49, and 97)

[184] Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. Lumos: Learning agents with unified data, modular design, and open-source llms. *ArXiv preprint*, abs/2311.05657, 2023. URL https://arxiv.org/abs/2311.05657. (page 11)

[185] Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver, Canada, 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1041. URL https://aclanthology.org/P17-1041. (page 81)

[186] Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. Learning to mine aligned code and natural language pairs from stack overflow. In *2018*

IEEE/ACM 15th international conference on mining software repositories (MSR), pages 476–486. IEEE, 2018. (pages 82, 86, 87, and 88)

[187] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. TaBERT: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8413–8426, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.745. URL https://aclanthology.org/2020.acl-main.745. (page 9)

[188] Licheng Yu, Xinlei Chen, Georgia Gkioxari, Mohit Bansal, Tamara L. Berg, and Dhruv Batra. Multi-target embodied question answering. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 6309–6318. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00647. URL http://openaccess.thecvf.com/content_CVPR_2019/html/Yu_Multi-Target_Embodied_Question_Answering_CVPR_2019_paper.html. (pages 3, 9, and 33)

[189] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1425. URL https://aclanthology.org/D18-1425. (page 9)

[190] John M Zelle and Raymond J Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055, 1996. (page 3)

[191] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472. (page 66)

[192] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. *ArXiv preprint*, abs/2310.12823, 2023. URL https://arxiv.org/abs/2310.12823. (page 109)

[193] Hongming Zhang, Muhao Chen, Haoyu Wang, Yangqiu Song, and Dan Roth. Analogous

process structure induction for sub-event sequence prediction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1541–1550, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.119. URL https://aclanthology.org/2020.emnlp-main.119. (page 3)

[194] Hongming Zhang, Muhao Chen, Haoyu Wang, Yangqiu Song, and Dan Roth. Analogous process structure induction for sub-event sequence prediction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1541–1550, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.119. URL https://aclanthology.org/2020.emnlp-main.119. (page 66)

[195] Li Zhang, Qing Lyu, and Chris Callison-Burch. Intent detection with WikiHow. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 328–333, Suzhou, China, 2020. Association for Computational Linguistics. URL https://aclanthology.org/2020.aacl-main.35. (page 66)

[196] Li Zhang, Qing Lyu, and Chris Callison-Burch. Reasoning about goals, steps, and temporal ordering with WikiHow. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4630–4639, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.374. URL https://aclanthology.org/2020.emnlp-main.374. (pages 66 and 78)

[197] Li Zhang, Qing Lyu, and Chris Callison-Burch. Reasoning about goals, steps, and temporal ordering with WikiHow. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4630–4639, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.374. URL https://aclanthology.org/2020.emnlp-main.374. (page 96)

[198] Li Zhang, Liam Dugan, Hainiu Xu, and Chris Callison-burch. Exploring the curious case of code prompts. In Bhavana Dalvi Mishra, Greg Durrett, Peter Jansen, Danilo Neves Ribeiro, and Jason Wei, editors, *Proceedings of the 1st Workshop on Natural Language Reasoning and Structured Explanations (NLRSE)*, pages 9–17, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.nlrse-1.2. URL https://aclanthology.org/2023.nlrse-1.2. (page 10)

[199] Li Zhang, Hainiu Xu, Yue Yang, Shuyan Zhou, Weiqiu You, Manni Arora, and Chris Callison-Burch. Causal reasoning of entities and events in procedural texts. In *Findings*

of the Association for Computational Linguistics: EACL 2023, pages 415–431, Dubrovnik, Croatia, 2023. Association for Computational Linguistics. URL https://aclanthology.org/2023.findings-eacl.31. (page 9)

[200] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with BERT. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SkeHuCVFDr. (page 69)

[201] Yi Zhang, Sujay Kumar Jauhar, Julia Kiseleva, Ryen White, and Dan Roth. Learning to decompose and organize complex tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2726–2735, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.217. URL https://aclanthology.org/2021.naacl-main.217. (page 66)

[202] Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v (ision) is a generalist web agent, if grounded. *ArXiv preprint*, abs/2401.01614, 2024. URL https://arxiv.org/abs/2401.01614. (page 97)

[203] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. arxiv 2017. *ArXiv preprint*, abs/1709.00103, 2017. URL https://arxiv.org/abs/1709.00103. (page 17)

[204] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *ArXiv preprint*, abs/1709.00103, 2017. URL https://arxiv.org/abs/1709.00103. (page 9)

[205] Victor Zhong, Tim Rocktäschel, and Edward Grefenstette. Rtfm: Generalising to novel environment dynamics via reading. *ArXiv preprint*, abs/1910.08210, 2019. URL https://arxiv.org/abs/1910.08210. (pages 2, 10, and 11)

[206] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. *ArXiv preprint*, abs/2205.10625, 2022. URL https://arxiv.org/abs/2205.10625. (page 51)

[207] Hattie Zhou, Azade Nova, Hugo Larochelle, Aaron Courville, Behnam Neyshabur, and Hanie Sedghi. Teaching algorithmic reasoning via in-context learning. *ArXiv preprint*, abs/2211.09066, 2022. URL https://arxiv.org/abs/2211.09066. (pages 3 and 9)

[208] Shuyan Zhou, Pengcheng Yin, and Graham Neubig. Hierarchical control of situated agents through natural language. In *Proceedings of the Workshop on Structured and Unstructured Knowledge Integration (SUKI)*, pages 67–84, Seattle, USA, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.suki-1.8. URL https://aclanthology.org/2022.suki-1.8. (pages 28 and 98)

[209] Shuyan Zhou, Pengcheng Yin, and Graham Neubig. Hierarchical control of situated agents through natural language. In *Workshop on Structured and Unstructured Knowledge Integration (SUKI)*, Seattle, USA, July 2022. URL https://arxiv.org/abs/2109.08214. No citations.

[210] Shuyan Zhou, Li Zhang, Yue Yang, Qing Lyu, Pengcheng Yin, Chris Callison-Burch, and Graham Neubig. Show me more details: Discovering hierarchies of procedures from semi-structured web data. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2998–3012, Dublin, Ireland, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.214. URL https://aclanthology.org/2022.acl-long.214. (page 96)

[211] Shuyan Zhou, Uri Alon, Frank F. Xu, Zhengbao Jiang, and Graham Neubig. Docprompting: Generating code by retrieving the docs. In *International Conference on Learning Representations (ICLR)*, 2023. No citations.

[212] Shuyan Zhou*, Frank F. Xu*, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*, Vienna, Austria, 2024. (pages 8, 95, 96, 97, 104, and 106)

[213] Yilun Zhou, Julie Shah, and Steven Schockaert. Learning household task knowledge from WikiHow descriptions. In *Proceedings of the 5th Workshop on Semantic Deep Learning (SemDeep-5)*, pages 50–56, Macau, China, 2019. Association for Computational Linguistics. URL https://aclanthology.org/W19-5808. (page 66)

[214] Fengda Zhu, Yi Zhu, Xiaojun Chang, and Xiaodan Liang. Vision-language navigation with self-supervised auxiliary reasoning tasks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 10009–10019. IEEE, 2020. doi: 10.1109/CVPR42600.2020.01003. URL https://doi.org/10.1109/CVPR42600.2020.01003. (page 33)

Appendix A

Appendix for Chapter 3

A.1 Website Implementation

Given the selected websites described in §3.2.2, we make the best attempt to reproduce the functionality of commonly used sites in a reproducible way. To achieve this, we utilized open-source frameworks for the development of the websites across various categories and imported data from their real-world counterparts. For the E-commerce category, we constructed a shopping website with approximately $90k$ products, including the prices, options, detailed product descriptions, images, and reviews, spanning over 300 product categories. This website is developed using Adobe Magento, an open-source e-commerce platform[1]. Data resources were obtained from data from actual online sites, such as that included in the Webshop data dump[182]. As for the social forum platform, we deployed an open-source software Postmill[2], the open-sourced counterpart of Reddit[3]. We sampled from the top 50 subreddits[4]. We then manually selected many subreddit for northeast US cities as well as subreddit for machine learning and deep learning-related topics. This manual selection encourages cross-website tasks such as seeking information related to the northeast US on both Reddit and the map. In total, we have 95 subreddits, 127390 posts, and 661781 users. For the collaborative software development platform, we choose GitLab[5]. We heuristically simulate the code repository characteristics by sampling at least ten repositories for every programming language: 80% of them are sampled from the set of top 90 percentile wrt stars repos using a discrete probability distribution weighted pro-

[1]https://github.com/magento/magento2
[2]https://postmill.xyz/
[3]https://www.reddit.com/
[4]https://redditlist.com/sfw.html
[5]https://gitlab.com/gitlab-org/gitlab

portional to their number of stars; the remaining are sampled from the bottom ten percentile set using similar weighted distribution. This is done to ensure fair representation of repos of all kinds, from popular projects with many issues and pull requests to small personal projects. In total, we have 300 repositories and more than 1000 accounts with at least one commit to a repository. For the content management system, we adapted Adobe Magento's admin portal, deploying the sample data provided in the official guide. We employ OpenStreetMap[6] for map service implementation, confining our focus to the northeast US region due to data storage constraints. We implement a calculator and a scratchpad ourselves.

Lastly, we configure the knowledge resources as individual websites, complemented with search functionality for efficient information retrieval. Specifically, we utilize Kiwix[7] to host an offline version of English Wikipedia with a knowledge cutoff of May 2023. The user manuals for GitLab and Adobe Commerce Merchant documentation are scraped from the official websites.

A.2 Environment Delivery and Reset

One goal for our evaluation environment is ease of use and reproducibility. As a result, we deploy our websites in separate Docker images [8], one per website. The Docker images are fully self-contained with all the code of the website, database, as well as any other software dependencies. They also do not rely on external volume mounts to function, as the data of the websites are also part of the docker image. This way, the image is easy to distribution containing all the pre-populated websites for reproducible evaluation. End users can download our packaged Docker images and run them on their systems and re-deploy the exact websites together with the data used in our benchmarks for their local benchmarking.

Since some evaluation cases may require the agent to modify the data contained in the website, *e.g.*, creating a new user, deleting a post, etc., it is crucial to be able to easily reset the website environment to its initial state. With Docker images, the users could stop and delete the currently running containers for that website and start the container from our original image again to fully reset the environment to the initial state. Depending on the website, this process may take from a few seconds to one minute. However, not all evaluation cases would require an environment reset, as many of the intents are information gathering and are read-only for the website data. Also, combined with the inference time cost for the agent LLMs, we argue that

[6] https://www.openstreetmap.org/

[7] https://www.kiwix.org/en/

[8] https://www.docker.com/

this environment reset method, through restarting Docker containers from the original images, will have a non-negligible but small impact on evaluation time.

A.3 User Roles Simulation

Users of the same website often have disparate experiences due to their distinct *roles*, *permissions*, and *interaction histories*. For instance, within an E-commerce CMS, a shop owner might possess full read and write permissions across all content, whereas an employee might only be granted write permissions for products but not for customer data. We aim to emulate this scenario by generating unique user profiles on each platform.

On the shopping site, we created a customer profile that has over 35 orders within a span of two years. On GitLab, we selected a user who maintains several popular open-source projects with numerous merge requests and issues. This user also manages a handful of personal projects privately. On Reddit, our chosen profile was a user who actively participates in discussions, with many posts and comments. Lastly, on our E-commerce CMS, we set up a user profile for a shop owner who has full read-and-write access to all system contents.

All users are automatically logged into their accounts using a pre-cached cookie. To our best knowledge, this is the first publicly available agent evaluation environment to implement such a characteristic. Existing literature typically operates under the assumption of universally identical user roles [46, 101, 151].

A.4 Intent Distribution

The distribution of intents across the websites are shown in Figure A.1.

A.5 Human Performance

We acknowledge that there may be a difference in human performance when annotators with different demographics are involved. In fact, many tasks in our dataset require domain-specific knowledge. For instance, an average user may not know what a git merge request is; or how to create a product in a complex content management system. We aim to design tasks that have easy-to-imagine outcomes (*e.g.*, a new product page is created) rather than those that are easily performed by an average user without significant domain knowledge.

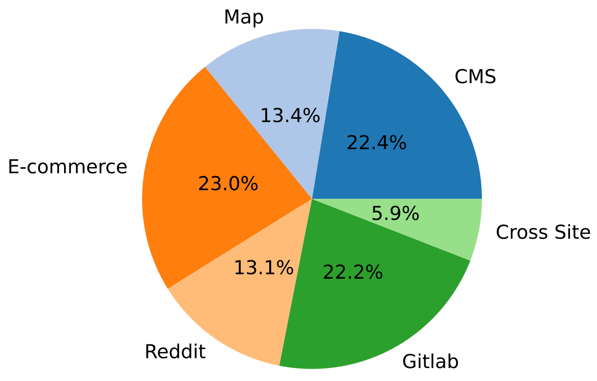


Figure A.1: The intent distribution across different websites. Cross-site intents necessitate interacting with multiple websites. Notably, regardless of the website, all user intents require interactions with multiple web pages.

CoT	UA Hint	Model	SR
✓	✗	GPT-3.5	6.28

Table A.1: The task success rate (SR %) of GPT-3.5-turbo-16k-0613 with temperature 0.0.

A.6 Experiment Configurations

We experiment with GPT-3.5-turbo-16k-0613, GPT-4-0613, and text-bison-001 with a temperature of 1.0 and a top-p parameter of 0.9. The maximum number of state transitions is set to 30. We halt execution if the same action is repeated more than three times on the same observation or if the agent generates three consecutive invalid actions. These situations typically indicate a high likelihood of execution failure and hence warrant early termination. For text-bison-001, we additionally allow ten retries until it generates a valid action.

Primarily, we use a high temperature of 1.0 to encourage the *exploration*. To aid replicating the results, we provide the results of GPT-3.5-turbo-16k-0613 with temperature 0.0 in Table A.1 and the execution trajectories in our code repository.

Dataset	`gpt-4-0613`	`gpt-4-1106-preview`
Date (900 examples)	100	100
Time duration (900 examples)	100	100

Table A.2: The accuracy (%) of two versions of GPT-4 on judging if dates and time duration of different formats are equivalent.

A.7 Prompt for `fuzzy_match`

```
Help a teacher to grade the answer of a student given a question. Keep in mind that the student may use different phrasing or wording to answer the question. The goal is to evaluate whether the answer is semantically equivalent to the reference answer.
question: {{intent}}
reference answer: {{reference answer}}
all the string 'N/A' that you see is a special sequence that means 'not achievable'
student answer: {{prediction}}
Conclude the judgement by correct/incorrect/partially correct.
```

Predictions that are judged as "correct" will receive a score of one, while all other predictions will receive a score of zero.

A.8 The Accuracy of Fuzzy Match Function

To evaluate this, we manually checked 40 examples and found that 39 of them are identical to our human judgment. In addition, among the 82 examples that require using GPT-4 for evaluation, the answer of 49 (60%) examples is a date (*e.g.*, 10/23/2022) or time duration (*e.g.*, 15 minutes). In these cases, GPT-4 is only used to judge the different *format* of the answers. We quantitatively evaluate the correctness of GPT-4 in this case by generating different formats of a date and time duration programmatically. We randomly sample negative examples. For instance, Nov 3, 2022, November 3, 2022, 3rd November 2022, 3 Nov 2022, 2022-11-03, and 3rd of November, 2022 are all correct variances of 2022/11/03. The accuracy of GPT-4 is shown in Table A.2. We can see that two versions of GPT-4 are extremely accurate, both achieving 100% accuracy.

A.9 The Prompts of the Baseline Web Agents

The system message of the reasoning agent for both GPT-3.5 and GPT-4 and two examples; the system message of the direct agent for GPT-3.5 and the two examples are shown below. **UA hint** refers to the instruction of " If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.". We remove this sentence in our ablation studies.

```
You are an autonomous intelligent agent tasked with navigating a web browser. You will be given web-based
tasks. These tasks will be accomplished through the use of specific actions you can issue.

Here's the information you'll have:
The user's objective: This is the task you're trying to complete.
The current web page's accessibility tree: This is a simplified representation of the webpage, providing key
information.
The current web page's URL: This is the page you're currently navigating.
The open tabs: These are the tabs you have open.
The previous action: This is the action you just performed. It may be helpful to track your progress.

The actions you can perform fall into several categories:
Page Operation Actions
`click [id]`: This action clicks on an element with a specific id on the webpage.
`type [id] [content] [press_enter_after=0|1]`: Use this to type the content into the field with id. By default,
the "Enter" key is pressed after typing unless press_enter_after is set to 0.
`hover [id]`: Hover over an element with id.
`press [key_comb]`: Simulates the pressing of a key combination on the keyboard (e.g., Ctrl+v).
`scroll [direction=down|up]`: Scroll the page up or down.

Tab Management Actions:
`new_tab`: Open a new, empty browser tab.
`tab_focus [tab_index]`: Switch the browser's focus to a specific tab using its index.
`close_tab`: Close the currently active tab.

URL Navigation Actions:
`goto [url]`: Navigate to a specific URL.
`go_back`: Navigate to the previously viewed page.
`go_forward`: Navigate to the next page (if a previous `go_back` action was performed).
```

(cont'd)

Completion Action:
`stop [answer]`: Issue this action when you believe the task is complete. If the objective is to find a text-based answer, provide the answer in the bracket. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

Homepage:
If you want to visit other websites, check out the homepage at http://homepage.com. It has a list of websites you can visit.
http://homepage.com/password.html lists all the account names and passwords for the websites. You can use them to log in to the websites.

To be successful, it is very important to follow the following rules:
1. You should only issue an action that is valid given the current observation
2. You should only issue one action at a time.
3. You should follow the examples to reason step by step and then issue the next action.
4. Generate the action in the correct format. Start with a "In summary, the next action I will perform is“ phrase, followed by action inside ``````. For example, "In summary, the next action I will perform is ```click [1234]```".
5. Issue stop action when you think you have achieved the objective. Don't generate anything after stop."""

example_user
OBSERVATION:
[1744] link 'HP CB782A#ABA 640 Inkjet Fax Machine (Renewed)'
  [1749] StaticText '$279.49'
  [1757] button 'Add to Cart'
  [1760] button 'Add to Wish List'
  [1761] button 'Add to Compare'
URL: http://onestopmarket.com/office-products/office-electronics.html
OBJECTIVE: What is the price of HP Inkjet Fax Machine
PREVIOUS ACTION: None
example_assistant
Let's think step-by-step. This page lists the information of HP Inkjet Fax Machine, which is the product identified in the objective. Its price is $279.49. I think I have achieved the objective. I will issue the stop action with the answer. In summary, the next action I will perform is ```stop [$279.49]```
example_user
OBSERVATION:
[164] textbox 'Search' focused: True required: False
[171] button 'Go'

[174] link 'Find directions between two points'
[212] heading 'Search Results'
[216] button 'Close'
URL: http://openstreetmap.org
OBJECTIVE: Show me the restaurants near ABC
PREVIOUS ACTION: None
example_assistant
Let's think step-by-step. This page has a search box whose ID is [164]. According to the nominatim rule of openstreetmap, I can search for the restaurants near a location by ̈restaurants near ̈. I can submit my typing by pressing the Enter afterwards. In summary, the next action I will perform is ```type [164] [restaurants near ABC] [1]```

You are an autonomous intelligent agent tasked with navigating a web browser. You will be given web-based tasks. These tasks will be accomplished through the use of specific actions you can issue.

Here's the information you'll have:
The user's objective: This is the task you're trying to complete.
The current web page's accessibility tree: This is a simplified representation of the webpage, providing key information.
The current web page's URL: This is the page you're currently navigating.
The open tabs: These are the tabs you have open.
The previous action: This is the action you just performed. It may be helpful to track your progress.

The actions you can perform fall into several categories:
Page Operation Actions
`click [id]`: This action clicks on an element with a specific id on the webpage.
`type [id] [content] [press_enter_after=0|1]`: Use this to type the content into the field with id. By default, the "Enter" key is pressed after typing unless press_enter_after is set to 0.
`hover [id]`: Hover over an element with id.
`press [key_comb]`: Simulates the pressing of a key combination on the keyboard (e.g., Ctrl+v).
`scroll [direction=down|up]`: Scroll the page up or down.

Tab Management Actions:
`new_tab`: Open a new, empty browser tab.
`tab_focus [tab_index]`: Switch the browser's focus to a specific tab using its index.
`close_tab`: Close the currently active tab.

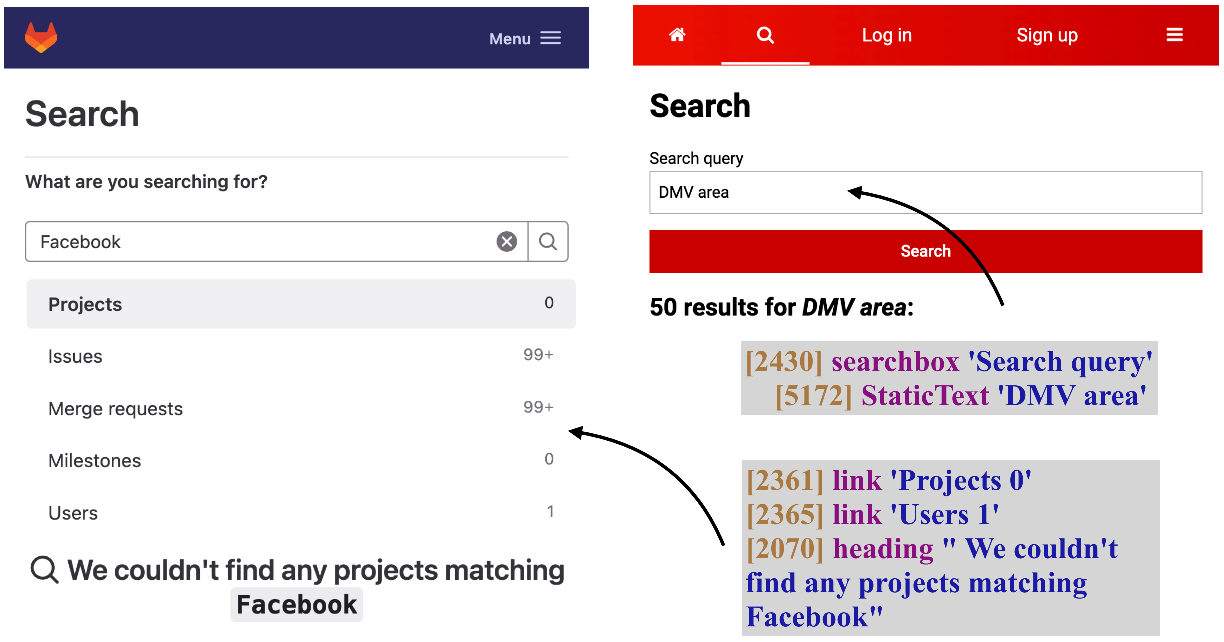
URL Navigation Actions:
`goto [url]`: Navigate to a specific URL.
`go_back`: Navigate to the previously viewed page.

`go_forward`: Navigate to the next page (if a previous
`go_back` action was performed).
Completion Action:
`stop [answer]`: Issue this action when you believe the task is complete. If the objective is to find a text-based answer, provide the answer in the bracket. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

Homepage:
If you want to visit other websites, check out the homepage at http://homepage.com. It has a list of websites you can visit.
http://homepage.com/password.html lists all the account name and password for the websites. You can use them to log in to the websites.

To be successful, it is very important to follow the following rules:
To be successful, it is very important to follow the following rules:
1. You should only issue an action that is valid given the current observation
2. You should only issue one action at a time.
3. Generate the action in the correct format. Always put the action inside a pair of ```. For example, ```click [1234]```
4. Issue stop action when you think you have achieved the objective. Don't generate anything after stop."""

OBSERVATION:
[1744] link 'HP CB782A#ABA 640 Inkjet Fax Machine (Renewed)'
  [1749] StaticText '$279.49'
  [1757] button 'Add to Cart'
  [1760] button 'Add to Wish List'
  [1761] button 'Add to Compare'
URL: http://onestopmarket.com/office-products/office-electronics.html
OBJECTIVE: What is the price of HP Inkjet Fax Machine
PREVIOUS ACTION: None
example_assistant
```stop [$279.49]```
example_user
OBSERVATION:
[164] textbox 'Search' focused: True required: False
[171] button 'Go'
[174] link 'Find directions between two points'
[212] heading 'Search Results'
[216] button 'Close'
URL: http://openstreetmap.org



[2430] searchbox 'Search query'
[5172] StaticText 'DMV area'

[2361] link 'Projects 0'
[2365] link 'Users 1'
[2070] heading '' We couldn't find any projects matching Facebook''

Figure A.2: Two examples where the GPT-4 agent failed, along with their screenshot and the accessibility tree of the relevant sections (grey). On the left, the agent fails to proceed to the "Users" section to accomplish the task of "Fork all Facebook repos"; on the right, the agent repeats entering the same search query even though the observation indicates the input box is filled.

~~~
OBJECTIVE: Show me the restaurants near ABC
PREVIOUS ACTION: None

example_assistant
```type [164] [restaurants near ABC] [1]```
~~~

A.10 Additional Error Analysis

Observation Bias Realistic websites frequently present information on similar topics across various sections to ensure optimal user accessibility. However, a GPT-4 agent often demonstrates a tendency to latch onto the first related piece of information it encounters without sufficiently verifying its relevance or accuracy. For instance, the homepage of the E-Commerce CMS displays the best-selling items based on *recent purchases*, while historical best-seller data is typically accessed via a separate report. Presented with the task of "*What is the top-1 best-*

selling product in 2022", the GPT-4 agent defaults to leveraging the readily available information on the homepage, bypassing the necessary step of generating the report to obtain the accurate data.

Failures in Observation Interpretation Interestingly, while GPT-4 is capable of summarizing the observations, it occasionally overlooks more granular information, such as the previously entered input. As in the right-hand example of Figure A.2, `[5172] StaticText` indicates that the search term "DMV area" has already been entered. However, the agent disregards this detail and continuously issues the command `type [2430] [DMV area]` until it reaches the maximum step limit. Furthermore, the agent often neglects the previous action information that is provided alongside the observation.

We hypothesize that these observed failures are related to the current pretraining and supervised fine-tuning on dialogues employed in GPT models [123]. These models are primarily trained to execute instructions given *immediate* observations (*i.e.*,, the dialogue history); thereby, they may exhibit a lack of explorations. Furthermore, in dialogue scenarios, subtle differences in NL expressions often have less impact on the overall conversation. As a result, models may tend to overlook minor variations in their observations.

Appendix B

Appendix for Chapter 8

B.1 Trajectory Representation

objective = "Find and review the estimated value of your property on the website."
# sub-task 1: Look up your property on Zillow
# step 1: Search for your address on Zillow's homepage search bar to open the property page.
type(element_id="6135", string="Main St")
# step 2: From the property details page, navigate to the "Zestimate" section.
scroll(down)
# sub-task 2: Begin adjusting the estimated value
# step 3: Click on 'Edit home facts' to adjust details that might affect the home's estimated value.
click(element_id="9945")

B.2 Prompt to Filter wikiHow Articles

Prompt to filter wikiHow articles

Given the title of an article, determine if it is about performing a task solely with computer or mobile phone's graphical user interface, and without any physical world configurations.

input: How to Set Up Chromecast WiFi (Using an Android Phone or Tablet)
output: Set Up Chromecast WiFi involves both a mobile application and physical interactions with the Chromecast device such as plug in the device, so the answer is "No"

input: How to Change Your Desktop Wallpaper on Linux Mint (Using the Linux Mint Wallpapers) output: Linux Mint is a desktop operating system, and changing the desktop wallpaper is typically done through the system settings or desktop environment's configuration tools, which are desktop applications, so the answer is "Yes"

input: How to delete a file using command line in Linux
output: Command line interface (CLI) in Linux is a text-based interface not a graphical user interface (GUI), so the answer is "No"

input: How to Reboot an iPad (Frozen iPads)
output: Rebooting an iPad usually involves physical actions like pressing and holding buttons on the iPad, so the answer is "No"

input: How to Connect the Kindle Fire to the
Internet (Connecting to an Existing Wi-Fi Network)
output: Kindle is neither a computer nor a mobile phone, so the answer is "No"

input: How to Pair AirPods to an iPhone (If Your AirPods Won't Connect)
output: Pairing AirPods with an iPhone typically includes physical actions such as opening the AirPods case near the iPhone and possibly pressing a button on the AirPods case, so the answer is "No"

input: {{Title of the article}}
output: {{Model prediction}}

B.3 Prompt to Generate Demonstrations from Tutorials

Prompt to rewrite an article to a trajectory in program format

# Task overview
You are given an article about performing a task in a web browser. Your goal is to make this article as accessible as possible to a user who is not familiar with the functionalities of the websites and the task at all.

# Guideline
Read the article carefully and follow the instructions below:

- Assume you start with the home page of the web application, skip the initial ‘goto‘ action.
- Break down the article into a sequence of steps.
- In every step, provide a concrete example that reflects a real execution. This example should clearly describe the element you are interacting with, the concrete value of an element you select, the precise content you type and other details. Never use broad descriptions. The example should be creative and realistic, avoid boilerplate text such as email@example.com. Make sure that the example is consistent across steps.
- Following the concrete example, provide the Python API call corresponding to the example.
- Group all API calls into multiple sub-sections, each section corresponds to a logical and actionable sub-task.

There are special scenarios and here are the ways to deal with them: - If the article describes multiple scenarios or multiple ways to approach the same goal, you can use your own judgement to choose the most common one to describe. - If there are repeated steps, make sure to unroll the steps and describe each of them canonically. - Always assume you perform this task using a web browser, if the original article uses a desktop app or mobile phone app, simply assume the corresponding web app exists. Hence, any steps regarding installation or login can be skipped.

# APIs
The APIs are as follows: ‘click(element_desc: str)‘ - Click on an element.
‘element_desc‘ is the the displayed text or the most representative attribute of the HTML element.
‘hover(element_desc: str)‘ - Hover over an element.
‘click_and_type(element_desc: str, content: str)‘ - Click an input element and type ‘content‘ into it.
‘key_press(key_comb: str)‘ - Press a key combination. ‘key_comb‘ is the combination of keys you want to press on. The default OS is MacOS if there is no explicit specification in the article.
‘goto(url: str)‘ - Navigate to ‘url‘
‘go_back()‘ - Go back to the previous page.
‘go_forward()‘ - Go forward to the next page.
‘new_tab()‘ - Open a new tab.
‘close_tab()‘ - Close the current tab.
‘switch_tab(tab_index: int)‘ - Switch to the tab with index ‘tab_index‘, starting from 0.

# Response format
Your response should follow the following format.
“‘python
sub-task <index>: <sub-task description>
# step <index>: <the real execution with concrete values for each argument>
<API, do not skip the keys in the API calls>

# step <index>: <the real execution with concrete values for each argument>
<API, do not skip the keys in the API calls>

<repeat for all sub tasks>

# task: <task command given to a smart assistant, only the necessary details on expectation are needed.>
“‘

# Article
{{Article here}}

Prompt to generate observation for two consecutive actions

# HTML Background Knowledge
Commonly used interactable elements in HTML:
[’a’, ’button’, ’input’, ’textarea’, ’select’, ’option’, ’label’, ’form’, ’details’, ’summary’, ’map’, ’area’, ’iframe’, ’embed’, ’object’, ’dialog’, ’menu’, ’fieldset’, ’legend’, ’datalist’, ’output’, ’progress’, ’meter’, ’keygen’]

# Task Overview
You are given:
- A browser-based task
- A seuqnece of past actions to perform the task and
- The next action to perform the task.

Your goal is to recover the HTML and the dynamic of a web application with the following requirements:
- The web page embodies a same level of content richness as advanced web applications on the internet. That is, the web page should have around 80 elements and at least 20 interactable elements. The depth of the DOM tree should be around 7. The length is at least 3000 tokens.
- Analyze the past actions and determine which of these actions have visible or functional impacts on the web page you design. Reflect the effects of these past actions in your HTML code. This may involve updating text, adding new elements, or modifying the layout or styles to represent the state of the web page after these actions.
- Design the interactable element that enables the next action. Make sure the choice of element type, attributes, and other essential characteristics are correct. For example, a text field is not interactable. Once the element is designed, assign the attribute id="next-action-target-element" to this interactable element.
- Please focus on making the static HTML visually rich. Ignore CSS animations & style and JavaScript functionality in your HTML code. - Provide the concrete reason to perform the next action.

# Response format
“‘html
<HTML that fullfils the requirements, make sure ‘next-action-target-element‘ is always included>

"'
<Summarize the progress by analyzing past actions. Provide a brief reason for performing the next action. Keep it short. Use imperative sentences.>

# Provided information
task: {{task description}}

past actions:
"'python
{{past actions}}
"'

next action:
"'python
{{next action}}
"'

B.4 Example Synthetic Demonstration from Tutorials from a wikiHow Article

A wikiHow article

How to Use Google Chat on iPhone (Enabling Google Chat in Gmail)

Open Gmail on your iPhone or iPad. This app icon looks like a white and red envelope. If you don't have it, you can download the Gmail app for free from the App Store.

Tap and tap Settings. The three-line menu icon is in the top left corner of your screen, and the option is generally at the bottom of the menu next to a gear icon. If you have multiple Gmail accounts, select the one you want to use Google Chats with. You can repeat these steps if you want to use Google Chats with multiple Gmail accounts.

Tap to check the box next to "Chat. A checked box indicates that Google Chat is active in your Gmail account and you'll see the Chat and Spaces tabs near the bottom of your screen. If Google Chat is enabled, you can tap these tabs to move between conversations in Chat and your emails in Gmail. If this is turned off, you'll need to use the Google Chats app to see your conversations in Chat.

The output from the rewriting step

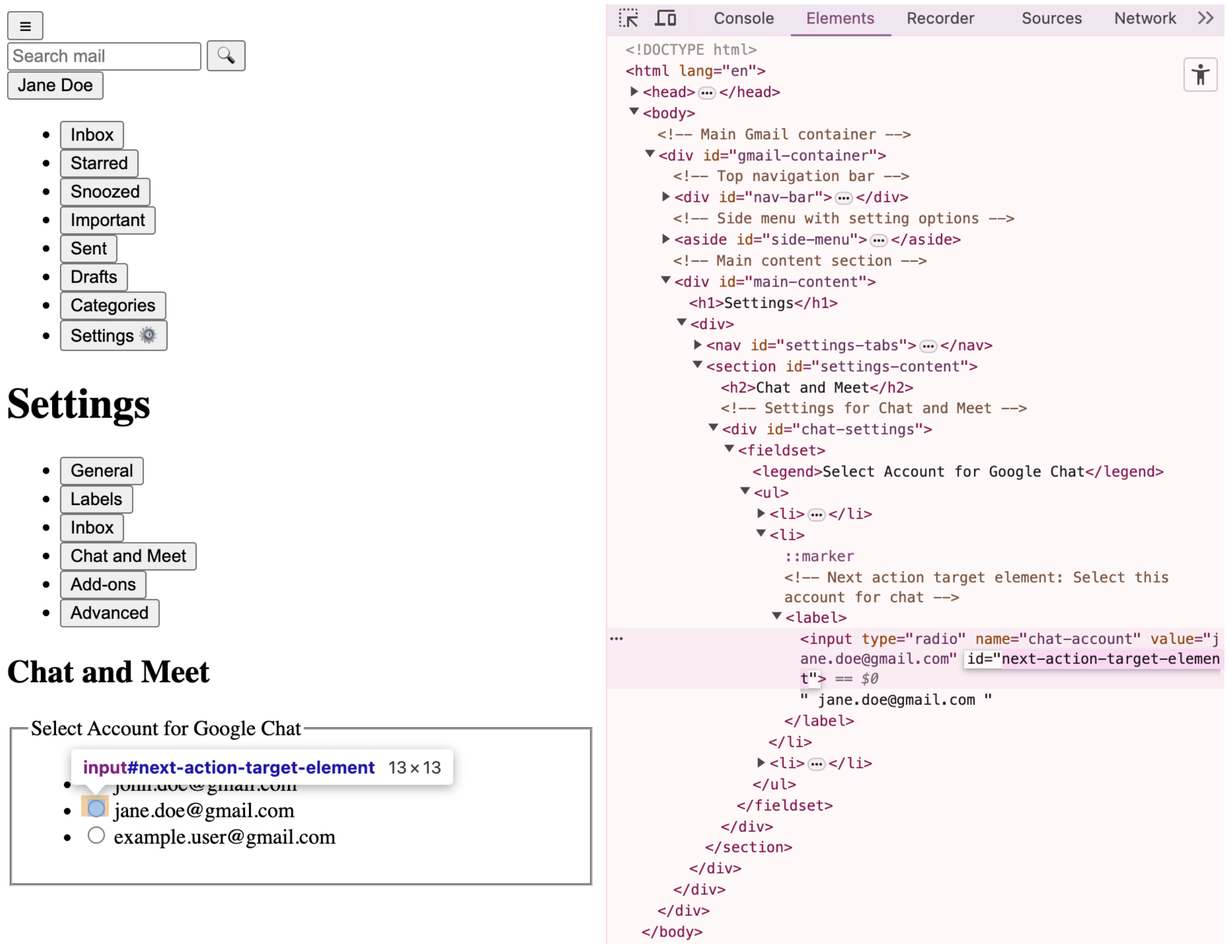
```
# sub-task 1: Navigate to Settings to activate Google Chat in another account
# step 1: Click on the menu icon to reveal options
click(element="Menu Icon")


# step 2: Go into Settings by clicking on it
click(element="Settings")


# step 3: Select another Gmail account, this time for jane.doe@gmail.com
click(element="jane.doe@gmail.com")


# sub-task 2: Activate Google Chat in the chosen account
# step 4: Click to enable Google Chat by checking the box
click(element="Check box next to Chat")


# step 5: Ensure the Chat feature is activated
stop(answer="Google Chat activated for jane.doe@gmail.com")


# task: Enable Google Chat on web version of Gmail for jane.doe@gmail.com
```

B.5 Prompt to Generate Direct Demonstrations from Random Observations

Prompt to Generate Direct Demonstrations from Random Observations

## Task overview
Given the accessibility tree of a web page, your goal is to propose creative and diverse browser-based tasks that involves interacting with this page, along with the previous actions that lead to the current state and the next action needed to be taken to accomplish the task.

## Action space



```
<!DOCTYPE html>
<html lang="en">
 ▶<head>⋯</head>
 ▼<body>
    <!-- Main Gmail container -->
   ▼<div id="gmail-container">
      <!-- Top navigation bar -->
     ▶<div id="nav-bar">⋯</div>
      <!-- Side menu with setting options -->
     ▶<aside id="side-menu">⋯</aside>
      <!-- Main content section -->
     ▼<div id="main-content">
        <h1>Settings</h1>
       ▼<div>
         ▶<nav id="settings-tabs">⋯</nav>
         ▼<section id="settings-content">
            <h2>Chat and Meet</h2>
            <!-- Settings for Chat and Meet -->
           ▼<div id="chat-settings">
             ▼<fieldset>
                <legend>Select Account for Google Chat</legend>
               ▼<ul>
                 ▶<li>⋯</li>
                 ▼<li>
                    ::marker
                    <!-- Next action target element: Select this
                    account for chat -->
                   ▼<label>
                      <input type="radio" name="chat-account" value="j
                      ane.doe@gmail.com" id="next-action-target-elemen
                      t"> == $0
                      " jane.doe@gmail.com "
                    </label>
                  </li>
                 ▶<li>⋯</li>
                </ul>
              </fieldset>
            </div>
          </section>
        </div>
      </div>
    </div>
  </body>
```

Figure B.1: The rendered generated HTML between step 2 and step 3. The concrete element to interact with is tagged with `id="next-action-target-element"`.

Here are the allowed actions that you can take to interact with the web page:
‘click(element: str, element_id: int=0)‘ - Click on an element. ‘element‘ is the displayed text or the most representative attribute of the HTML element. ‘element_id‘ is the index of the element at the beginning of the node.
‘click_and_type(element: str, content: str, element_id: int=0)‘ - Click and type ‘content‘ into an ‘element‘.
‘key_press(key_comb: str)‘ - Press a key combination. ‘key_comb‘ is the combination of keys you want to press on. The default OS is MacOS if there is no explicit specification in the article.
‘goto(url: str)‘ - Navigate to ‘url‘
‘go_back()‘ - Go back to the previous page.
‘go_forward()‘ - Go forward to the next page.
‘new_tab()‘ - Open a new tab.
‘close_tab()‘ - Close the current tab.
‘switch_tab(tab_index: int)‘ - Switch to the tab with index ‘tab_index‘, starting from 0.
‘scroll(up|down)‘ - Scroll the page up or down.
‘stop(answer: str=”)‘ - The task is completed. If the task is to seek information, include the answer as a string. Otherwise, leave it empty.

## Guidelines
You will follow the guidelines below to perform the task:
1. Examine the web page to understand the the domain of the web page.
2. Brainstorm 8 task categories that could be performed the website. Be creative.
3. For each task category, propose a concrete task that has this web page as one of its steps. You want the concrete task to be unambiguous and clear so that no further clarification is needed to perform the task.
4. Given a concrete task, you are ask to come up with the past actions that leads to the current page, as well as the next action.
* Requirement for past actions: You should write down each past action in the details. You want to group all actions into multiple sub-sections, each section corresponds to a logical and actionable sub-task. The next action could start with a new sub-task. You can omit the ‘elemement_id‘ if they are not in the current page. There should only be one action at each step. DO NOT give goto() or new_tab() as first step.
* Requirement for next action: Provide the reasoning behind your past actions and the progress in completing the task. Also, describe your understanding of the current page and the concrete reason to execute the next action. If the action takes an element as the argument, it is important that you understand the role and the attributes of that element so that the action can be appropriately applied. Make sure to always include the ‘element_id‘ in your next action if there is any. Any ‘element_id‘ must come from the given Accessibility Tree.

## Format of the response
You are asked to provide the action sequence for task #1 with roughly 7 past actions; task #2 with roughly 0 past actions; task #4 with roughly 6 past actions; task #5 with roughly 4 past actions; task #6 with roughly 10 past actions. Your answer should follow the following format:

<Analysis and understanding about the domain and the concrete content of the web page>
<The list of 8 creative task categories>
<The concrete tasks for task category #1 #2 #4 #5 #6. Remember, a concrete task needs to include concrete details so that no further clarification is required when performing the task. Use imperative sentences.>

“‘python
# task: <repeat concrete task #1>

———————
# past actions (history)
# sub-task 1: <sub-task description>
# step 1: <step description>
<action>
# step 2: <step description>
<action>
# sub-task 2: <sub-task description>
# step 3: <step description>
<action>
# step 4: <step description>
<action>
# sub-task 3: <sub-task description>
# step 5: <step description>
<action>
# step 6: <step description>
<action>
# sub-task 4: <sub-task description>
# step 7: <step description>
<action>

———————

# next action
# step <index>: <summarize the progress so far and analyze the current state of the web page. Provide the concrete reason to perform the next action>
<action>
# step summary: <brief step description>
“‘
“‘python
# task: <repeat concrete task #2>

```
# ———————–
# past actions (history)

# ———————–
# sub-task <index>: <sub-task description>
# next action
# step <index>: <summarize the progress so far and analyze the current state of the web page. Provide the concrete reason to perform the next action>
<action>
# step summary: <brief step description>
“‘
......
## The Accessibility Tree
[4812] link ’Shopbop Designer Fashion Brands’
  [4815] LineBreak
  [4819] LineBreak
[1742] LayoutTableCell ”
[1743] LayoutTableCell ’Amazon Warehouse Great Deals on Quality Used Products’
  [4822] link ’Amazon Warehouse Great Deals on Quality Used Products’
    [4825] LineBreak
    [4829] LineBreak
```

B.6 Example Generated Trajectories from Random Observation

Generated trajectories from random observation

The domain appears to be that of a major online retail website, likely Amazon, considering the references to "Prime," "Best Sellers," and "Amazon Basics," as well as services like "Magazine Subscriptions" and "Prime Video." The structure of the page includes a variety of product categories, a search textbox, and links to other services such as account sign-in and shopping cart. This suggests that users on this page can shop for a wide range of items, manage their account, or engage with support services.

Below are eight creative task categories that could be performed on this type of website:

1. Product Searching: Looking up products in a specific category or using a search query.
2. Account Management: Signing into an account, creating a new account, or managing existing account details.

3. Navigational Inquiry: Using the site's navigation to find different departments or special features.
4. Customer Support: Accessing customer service features or disability support services.
5. Deal Hunting: Searching for best sellers, deals of the day, or exclusive prime deals.
6. Media Consumption: Navigating to and consuming digital media such as eBooks, Prime Video, or music.
7. Educational Browsing: Exploring educational materials or subscribing to learning resources.
8. Wishlist Management: Adding items to a wishlist or gift registry for future purchase.

The concrete tasks for task category #1, #2, #6, #7, #8 are as follows:

### Concrete Task #1: Product Searching
Find and search for popular fitness trackers within the "Sports & Outdoors" category.

“‘python
# task: Find and search for popular fitness trackers within the "Sports & Outdoors" category.

# ————–
# past actions (history)
# sub-task 1: Open the site and navigate to the product categories.
# step 1: The user opens the main page of the retail website.
go_forward()
# sub-task 2: Navigate to the "Sports & Outdoors" category.
# step 2: The user accesses the menu to view the category list.
click(element='Open Menu')
# step 3: The user browses through the categories.
scroll(down)
# step 4: The user selects the "Sports & Outdoors" category.
click(element='Sports & Outdoors', element_id=7964)

————–

# next action
# step 5: The user now needs to search for fitness trackers within the selected category. Clicking and typing into the search bar is necessary to filter the results.
click_and_type(element='Search', content='fitness trackers', element_id=7657)
# step summary: Initiate a search for fitness trackers in the Sports & Outdoors category.
“‘

### Concrete Task #2: Account Management
Sign into your personal account to check on active subscriptions.

```
“‘python
# task: Sign into your personal account to check on active subscriptions.

# ————–
# past actions (history)

# ————–
# sub-task 1: Sign into the account.
# next action
# step 1: The user is already on a page where they can initiate the sign-in process, but the next step is to click and type the username in the search bar, which suggests that an error might have taken place, or this is being used as a workaround.
click_and_type(element=’Search’, content=’myusername@example.com’, element_id=7657)
# step summary: Try to sign in by using the search bar to enter the username.
“‘

### Concrete Task #6: Media Consumption
Navigate to Prime Video to browse the latest movie releases.

“‘python
# task: Navigate to Prime Video to browse the latest movie releases.

# ————–
# past actions (history)
# sub-task 1: Access the main page.
# step 1: The user starts by reaching the home page, where they can find various services.
go_back()
# step 2: The user considers different media options available on the site.
scroll(up)
# sub-task 2: Go to Prime Video.
# step 3: The user clicks on the "Prime Video" link.
click(element=’Prime Video’, element_id=7961)

# ————–

# next action
# step 4: To search for the latest movie releases within Prime Video, the user needs to use the search function.
click_and_type(element=’Search’, content=’latest movie releases’, element_id=7657)
# step summary: Search for the latest movie releases on Prime Video.
“‘
```

### Concrete Task #7: Educational Browsing
Subscribe to a magazine about technology within the "Magazine Subscriptions" category.

“‘python
# task: Subscribe to a magazine about technology within the "Magazine Subscriptions" category.

# ————–
# past actions (history)
# sub-task 1: Enter the site and locate the magazine section.
# step 1: The user starts on the home page where they can browse products.
go_forward()
# step 2: The user opens the categories menu to look for subscriptions.
click(element=’Open Menu’)
# step 3: The user scrolls through the menu to find the magazine subscriptions.
scroll(down)
# sub-task 2: Select the "Magazine Subscriptions" category.
# step 4: The user selects the "Magazine Subscriptions" option from the categories.
click(element=’Magazine Subscriptions’, element_id=7955)
# step 5: The user is now presented with different types of magazines but wants to find technology-related ones.
scroll(down)

————–

# next action
# step 6: With the magazine subscriptions displayed, it’s time to search for technology magazines specifically.
click_and_type(element=’Search’, content=’technology magazine’, element_id=7657)
# step summary: Search for technology magazines in the Magazine Subscriptions section.
“‘

### Concrete Task #8: Wishlist Management
Add a popular sci-fi novel to your wishlist for future purchasing.

“‘python
# task: Add a popular sci-fi novel to your wishlist for future purchasing.

————–
# past actions (history)
# sub-task 1: Access the website’s book section.

```
# step 1: The user has started on the homepage and is looking for books.
click(element='Books', element_id=6007)

# ————————

# next action
# step 2: The user wants to find and add a sci-fi novel to their wishlist, and the next logical action involves
searching for the item.
click_and_type(element='Search', content='popular sci-fi novels', element_id=7657)
# step summary: Begin the search for popular sci-fi novels to add to the wishlist.
```

B.7 Data Selection from Random Observations in ClueWeb

We inspect a random sample of ClueWeb on its domain distribution. As shown in Figure B.2, the majority of domains appear only once, making up 69.1% of all the web pages.

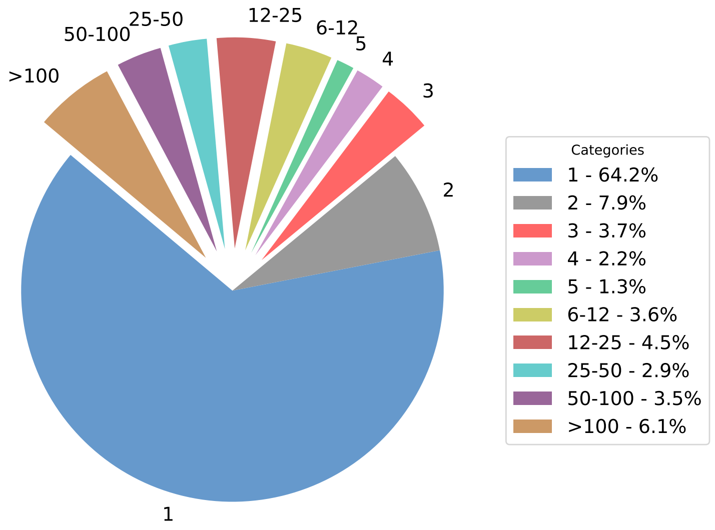


Figure B.2: Frequencies of domains and proportion of each frequency in ClueWeb

B.8 Training Settings

The `CodeLlama` checkpoints are fine-tuned with A100 GPUs with deepspeed [1] acceleration framework. We set the context length to 4096 tokens. To train on 50k dataset, we train with 6 x A100 80G GPUs for about 20 hours. We use a batch size of 48, and learning rate of 5e-5. We use cosine annealing and a warm-up ratio of 0.03.

[1] https://github.com/microsoft/DeepSpeed