Language Agents

From Next-Token Prediction to Digital Automation

Shunyu Yao

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Karthik Narasimhan

May 2024

Abstract

Building autonomous agents to interact with the world lies at the core of artificial intelligence (AI). This thesis introduces "language agents", a new category of agents that utilize large language models (LLMs) to reason to act, marking a departure from traditional agents via extensive rule design or learning. It is developed in three parts:

Part I motivates the necessity for language agents by introducing a new set of AI problems and benchmarks based on interaction with large-scale, real-world computer environments, such as the Internet or code interfaces. These "digital automation" tasks present tremendous value for alleviating tedious labor and improving our lives, yet pose significant challenges for prior agent or LLM methods in decision-making over open-ended natural language and long horizon, calling for new methodologies.

Part II lays the methodological foundation for language agents, where the key idea is to apply LLM reasoning for versatile and generalizable agent acting and planning, which also augments LLM reasoning to be more grounded and deliberate via external feedback and internal control. We show language agents can solve a diversity of language and agent tasks (especially digital automation tasks proposed in Part I), with notable improvements over prior LLM-based methods and traditional agents.

Part III consolidates insights from Parts I and II and outlines a principled framework for language agents. The framework provides modular abstractions to organize various LLM-based methods as agents, to understand their gaps from human cognition, and to inspire and develop new methods towards general-purpose autonomous agents.

From foundational empirical tasks and methods to a unifying conceptual framework, this thesis establishes the study of language agents as a distinct and rigorously defined field at the frontier of AI research.

Acknowledgements

Usually it starts with the advisor, but as a prelude I want to first thank my friend Kexin Yi. Back in 2018, when I was a visiting undergrad at MIT trying computer vision but unsure what to do or where to go next, he opened a webpage and said something like: “You should check out this Karthik guy, I heard he’s quite nice, and this text game thing is very interesting. Also, though Princeton AI isn’t very strong now, they are recruiting great young people. I’d go there if I’m younger.” Wow.

It turns out this Karthik guy is quite nice, and this text game thing is indeed very interesting. When I entered Princeton graduate school in 2019 but still unsure what to do or where to go next, I reached him and said, “This language model thing (GPT-2) looks quite promising and should just solve text games?” He’s like, “Sure.” This led to a paper called CALM, a PhD thesis on language agents, and five wonderful years of research and life with the best advisor possible and the best man in my wedding being this Karthik guy. (But Zork1 is still unsolved!) Thanks for taking a chance on a new grad not admitted by you and without any experience in your research fields, for supporting him when guidance was needed and trusting him when freedom was needed, and for all the walks in random Princeton locations and Salesforce Park talking random things and grand vision. It feels great.

It also turns out Princeton recruits great people, though not all young. In late 2018, I was thrilled to learn that Tom Griffith moved to Princeton, but we didn’t get to work together until five years later. Although we only worked together on three papers with very sparse meetings, they turned out to be some of my best work. Thanks for pointing me to Newell, Simon, General Problem Solver, Cognitive Architectures, the great classes in my junior years, and showing me what the most professional professor looks like. On the younger side, thanks to Danqi, the third member of my general exam committee. Although we haven’t worked together, you set a practical and complementary tone for the NLP group and recruited great students who became my great friends, and I enjoyed our fond chats in random places: IAS, Abu Dhabi, and Hawley, PA. I also want to thank the other members of my thesis committee, Ben, Sanjeev, and Tatsu, for their great help on my job talk, as well as Jia and Olga, one or two of whom admitted me to Princeton and made everything possible.

Besides Princeton professors (and Tatsu), I also want to thank other great mentors: Jiajun for starting my research career, Jun-Yan for the work ethic and passion that still inspire me today, Josh for being the first academic grandmaster figure in my life and deeply influencing my research, other undergrad mentors and teachers (Denny, Lihong, Chongjie, Ran) for your continuing care till today, Matthew for unlocking my career with Jericho, and Yuan for the solid and constant support during the ReAct and ToT time (including the free Google meals that made my life much easier in London). In addition, thanks to Bill, Antonio, Liz, Tomer, Kevin, Chuang, Mo, Christos, Sham, and all other senior researchers who have helped me along the way.

Princeton recruits not just great professors, but also talented, kind, and diverse students who played a principle component in my happiness in the last five years. I want to thank my NLP friends (Alex, Ameet, Austin, Ben, Carlos, Dan, Howard, Jane, Jens, Jimmy, John, Mengzhou, Michael, Ofir, Runzhe, Sadhika, Tianyu, Vishvak, Zexuan, among more), basketball friends (Chengyu, Yuxiao, Kexin, Sinong, Ryan, among more), roommate friends (Xiaoqi, Fan, Kehan, Tianshu), non-of-the-above friends (Ted, Allen, Gong, Chen, Xindi, among more), and non-Princeton friends too many to count. You know who you are. In particular, thank my mentees (John, Ben, Michael, Noah, among more) for giving me a chance to positively impact others' lives.

Usually it ends with family, and let me finally respect the norms. Doing a PhD abroad has been a lonely journey (special thanks to COVID), with countless video chats on WeChat. But the love and support have not faded as a result of distance, they evolved and strengthened. I want to thank my parents (Guoping and Feng), wife (Sixuan), and many other family members. I hope to have made you all proud.

To my family.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Building autonomous agents to interact with various environments is the core problem of artificial intelligence (AI) [266]. At a high level, this thesis proposes a fundamentally new kind of agent, and a fundamentally new kind of environment (Figure 1.1):

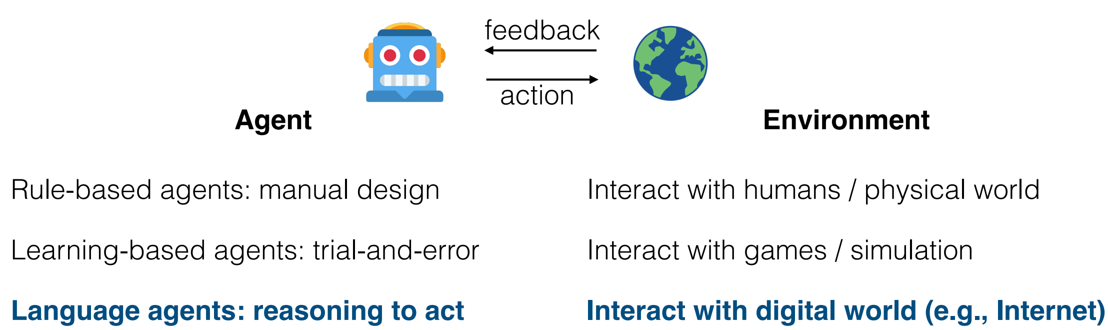


Figure 1.1: This thesis proposes a new way to build and benchmark AI agents.

- Existing agents either mainly follow domain-specific rules to act (rule-based agents, such as DeepBlue [38], Eliza [272], or Shaky the robot [229]) or mainly train on domain-specific data to act (learning-based agents, such as AlphaGo [281], Atari DQN [206], or ADR for hand manipulation [8]). This thesis introduces **language agents** that leverage language models to reason to act, which alleviates the intensive domain-specific efforts needed to build traditional agents, with few-shot generalization across various domains. This represents a major step toward the goal of building general-purpose autonomous agents.

- Existing agents either interact with humans or the physical world (practical but not scalable) or interact with games or simulations (scalable but not practical). This thesis introduces **digital automation**, a new kind of task where agents interact with large-scale real-world digital environments, such as the Internet. This provides new challenges for agents to make decision over open-ended actions and long horizon, as well as tremendous opportunities to alleviate our digital labor and discover new knowledge.

What is wrong with traditional agents and environments? What is the definition of "language agents" given traditional rule-based or learning-based agents might also perceive and act in language? Why do we have to move to large-scale real-world digital environments to make further progress, instead of using traditional agent testbeds like games? I will briefly use the domain of text adventure games to illustrate these points and motivate the rest of the thesis.

1.1 Prelude: Text Games

Text adventure games [106] such as Zork1 (Figure 1.4 (a)) have been one of the earliest domains for developing agents that receive textual observations and issue textual actions. In such games, agents receive sparse scalar rewards upon major progress (such as moving the rug, opening the trap door, and entering the underground), and aim to achieve high rewards.

A key challenge of such text games is the heterogeneous, combinatorial, yet semantic **action space** (Figure 1.2): unlike chess or Atari with a small and fixed action space, there is a different set of valid actions at each step of the text game that are semantically meaningful to change the game state (Figure 1.2). If the agent

Observation: You are in the living room. There is a doorway to the east, a wooden door with strange gothic lettering to the west, which appears to be nailed shut, a trophy case, and a large oriental rug in the center of the room. You are carrying: A brass lantern ...

Random Actions: close door, north a, eat troll with egg, egg troll, ...
CALM (n-gram): enter room, leave room, open door, close door, ...
CALM (GPT-2): east, turn on lantern, move rug, unlock case with key, ...

Next Observation: With a great effort, the rug is moved to one side of the room, revealing the dusty cover of a closed trap door...

Figure 1.2: Sample gameplay from Zork1 along with action sets generated by two variants of CALM. The game recognizes a vocabulary size of 697, resulting in more than $697^4 \approx 200$ billion potential 4-word actions. '*move rug*' is the optimal action to take here and is generated by our method as a candidate.

samples a random action in the space of language, the chance of the action being "valid" is near zero, and exploration is nearly impossible. This characterizes the nature of high-level human decision-making, as major decisions in life are often semantic and open-ended, whether to decide which house to buy, or how to plan travel.

In light of the action space challenge, previous approaches either build rule-based agents [107] with manually designed actions (e.g., issue "turn on lantern" if "lantern" and "dark" both mentioned in observation text), or rely on a game handicap [106] to provide ground truth valid actions at each step for learning-based agents to effectively explore. However, these rule or learning-based agents do not possess language knowledge beyond the game(s) they are designed for or trained on, which poses the second key challenge of **generalization** to novel games or domains. In contrast, humans can easily play a new game based on our prior understanding of language and commonsense knowledge about the world. How can we inject such language and world priors into text game agents?

1.1.1 CALM: Applying language models for agents

Pre-trained language models have rich priors about language and commonsense, but they are trained to write, not to act. How can we use them for agents? **Contextual Action Language Models (CALM) [356] is the first work that applies a language model to build an agent.** We fine-tune GPT-2 [258] on human gameplay trajectories, where the task is to predict the action given a context of previous game observations and actions. Once trained, we use CALM to sample a set of actions as a reduced action space for a reinforcement learning agent, DRRN [108], to explore and learn to choose the most rewarding actions.

We apply CALM to 28 *unseen* games outside its training distribution, and find it can generalize to these novel domains and generate reasonable actions for RL exploration, thanks to the general language knowledge obtained from pre-training and general game commonsense knowledge obtained from fine-tuning. As shown in Table 1.3, CALM (GPT-2) paired with DRRN achieves an average normalized game score of 9.4%, significantly outperforming the previous best agent without game handicap, NAIL [107]. We also find that replacing GPT-2 to smaller and simpler language models (n-gram), ablating GPT-2 pre-training (w/o PT), reducing GPT-2 fine-tuning (20% FT), or ablating reinforcement learning (w/o RL) all lead to worse performances.

Agent	**Score**
CALM (GPT-2)	**9.4%**
CALM (n-gram)	5.5%
NAIL [107]	5.6%
CALM (w/o PT)	6.8%
CALM (20% FT)	8.1%
CALM (w/o RL)	1.8%

Figure 1.3: Normalized score across 28 games.

CALM shows the potential of language models for building autonomous agents: they can empower agents with general prior knowledge useful for various environments and tasks, and generate open-ended actions for decision-making given context. However, CALM still relies on game-specific reinforcement learning to optimize for game scores. Does the RL part also learn generalizable language understanding and reasoning?

1.1.2 Hash: Re-thinking semantics in text games and agents

(a) Zork I	(b) **hash**
Observation 21: You are in the living room. There is a doorway to the east, a wooden door with strange gothic lettering to the west, which appears to be nailed shut…	*Observation 21:* 0x6fc2204
Action 21: move rug	*Action 21*: 0x3a04222
Observation 22: With a great effort, the rug is moved to one side of the room, revealing the dusty cover of a closed trap door... Living room... You are carrying: ...	*Observation 22*: 0x103ba12
Action 22: open trap	*Action 22*: 0x16bb110

Figure 1.4: (a): Sample gameplay from Zork I, and (b) **hash** replaces observation and action texts by their string hash values.

To answer the question, in [354], we find that if we replace the observation and action texts for their hash strings, the DRRN agent performance does not degrade, but even slightly improves from 21% to 25% across 12 games (Figure 1.4). Note that in this hash scheme, all language semantics are lost, and even a word change would lead to a completely different hash representation. In this sense, the RL agent is not learning to *understand* the game via language semantics but to *memorize* it via language as markers.

This is fundamentally different from the way humans use language to represent and reason about the world in a general and generalizable way. For example, when we play the Zork1 game for the first time, instead of millions of interactions to overfit the game, we *reason* over the observation in our mind: "The door is nailed shut so I should focus on exploring the current room, but there is no apparent object that is interesting. Under the rug, there might be more things, so I can try to move it and see what happens." Thinking a *thought* is a special *action* for humans, as it does not affect the external world but instead updates the internal context and informs future decision-making. Also, the space of thought is infinite (anything can be thought of), so strong language priors are needed to effectively reason. In this sense, we define language agents not as agents with textual observation and action spaces, but **agents that process observations to actions via internal mechanisms of natural language reasoning**. Under this definition, the external observation and action spaces do not even have to be textual, as various environments can be turned into text games via off-the-shelf perception and control modules, e.g., image captioning models or symbolic controllers. What matters is the internal information processing mechanism, not external modalities, and the manually designed symbolic representations in rule-based agents or learned neural embeddings in learning-based agents cannot achieve human-like open-ended language reasoning across general domains.

But when the environment is **small, closed, and synthetic** like text games, which is often the case for academic research, manual rule design or reinforcement learning can overfit the environment without the need for open-ended language reasoning or generalization to new scenarios. As a result, the developed methods or agents often prove hard to transfer to real-world scenarios and deliver practical values. While interacting with humans or the physical world is open-ended and practical, it is also slow, expensive, and noisy, so collecting scalable data or reward signals has been challenging. Thus, we need fundamentally new domains that are **scalable, open-ended, and practical** to challenge traditional agents and motivate new methods for agents with language reasoning.

These reflections lead to the rest of the thesis outlined below.

1.2 Approach and Outline

My approach to language agents is holistic, which starts by constructing practical problems with scalable benchmarks that challenge existing agents and language models in decision-making over open-ended actions and long horizon (Part I, Chapters 2 and 3). Solving these problems motivates new methodology for agents that can reason in language, to which my work has made foundational contributions by designing simple, general methods that connect language model reasoning to agent acting and planning, with the key idea that **reasoning can be seen as internal actions for agents** (Part II, Chapters 4 and 5). Lastly, I synthesize the empirical insights from my problems, methods, and experiments into a principled conceptual framework for language agents, which inspires various future directions (Part III, Chapters 6 and 7).

The thesis is based on my following work [356, 354, 352, 349, 274, 164, 129, 360, 41, 276, 358, 292], whose content and appendices contain more details to be checked, and associated data and code all publicly released.

Part I

Benchmarks

Chapter 2

WebShop: Benchmarking Agents via Web Interaction

2.1 Introduction

A general-purpose autonomous agent should tackle the following key challenges: (1) reasoning over complex textual, visual, and other multimodal observations, (2) decision-making over open-ended actions, and (3) exploration over long horizon.

To build language agents towards these strong capabilities, we first need to have a research benchmark that reflects these key challenges. However, existing benchmarks for agents often feature environments with small action spaces, synthetic text (or pixels), and short-horizon tasks (Figure 2.1). On the other hand, practical applications for agents, such as dialogue or robotics, feature these research challenges but prove challenging in building scalable benchmarks, as it is slow, expensive, and noisy to collect interactions and reward signals from humans or physical environments.

Therefore, in order to make progress in building language agents, we believe there is a need for scalable interactive environments that contain: (1) language elements that reflect rich, real-world usage and are collectible at scale, and (2) task feedback

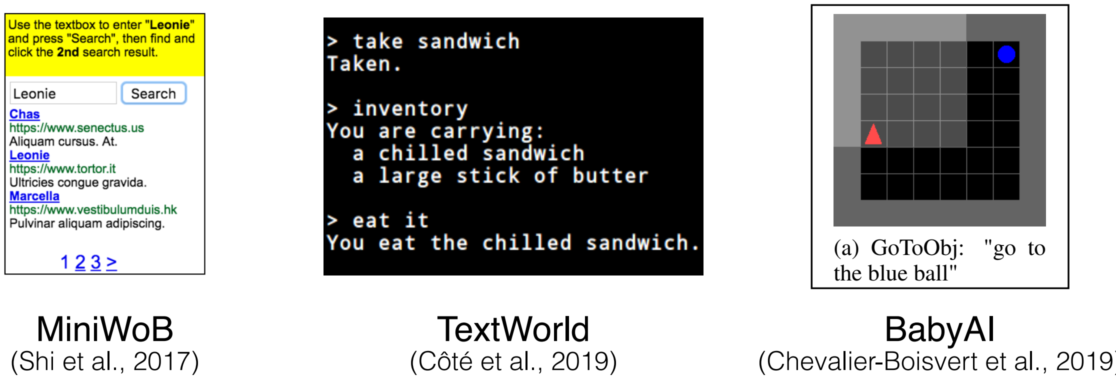


Figure 2.1: Typical benchmarks for agents that perceive or generate language usually feature synthetic text and environments, small action spaces, and short-horizon tasks.

that is well-defined and automatically computable to facilitate interactive learning, without the constant need for expensive feedback from humans.

The world wide web (WWW) is a massive open-domain interactive environment that inherently satisfies the first aforementioned requirement through its interconnected set of pages with natural text, images and interactive elements. By being simultaneously scalable, semantic, interactive, dynamic and realistic, the web is uniquely different from existing environments for autonomous agents like games or 3D navigation. Moreover, the web also provides a practical environment to deploy trained agents, with great potential for alleviating human efforts in tedious tasks (e.g. buying products, booking appointments). While there has been prior work on building web-based tasks, they either lack depth in the transition and action spaces, or prove difficult to scale up. Some benchmarks only contain either a single classification task [243, 287, 200] or interactions containing only a handful of different pages in each episode [275]. Others propose tasks with longer horizons but are either limited to following hyperlinks for web navigation [230] or require human-in-the-loop feedback due to the lack of an automated reward function [209].

In this chapter, we introduce WebShop (Figure 2.2) – a large-scale interactive web-based environment for language understanding and decision making – and train autonomous agents to complete tasks on this benchmark. With the goals of being

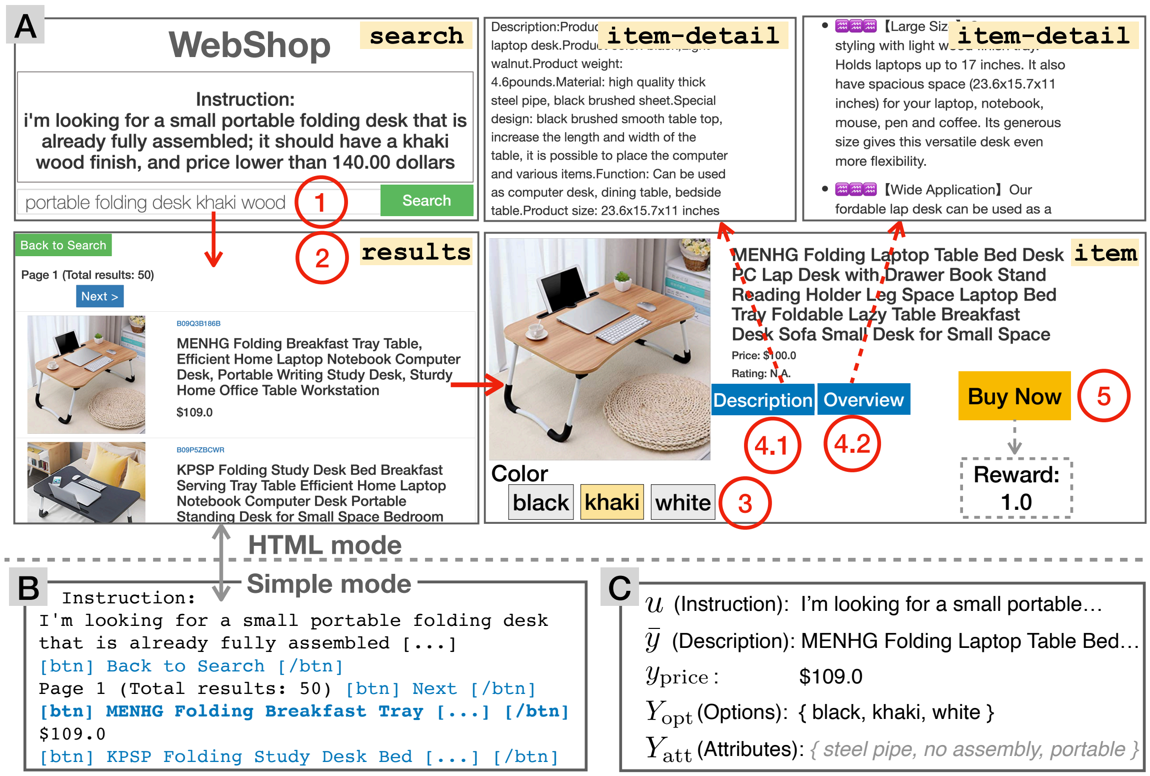


Figure 2.2: The WebShop environment. **A**: An example task trajectory in `HTML` mode, where a user can (1) search a query in a `search` page, (2) click a product item in a `results` page, (3) choose a color option in a `item` page, (4) check `item-detail` pages and go back to the item page, and (5) finally buy the product to end the episode and receive a reward $r \in [0, 1]$ (§2.3.2). **B**: the `results` page in `simple` mode for agent training and evaluation. The blue text indicates clickable actions and bold text indicates an action selected by the agent. **C**: The product notation used in §2.3 with corresponding examples from the product in **A**. The attributes Y_{att} are hidden from the task performer.

scalable and containing realistic language and visual elements, WebShop emulates the task of online shopping on an e-commerce website, where the agent's goal is to understand a human-provided text instruction and *purchase* a product to match the specifications. To do so, the agent needs to query the website's search engine, choose items to explore from search results, open and read their description and details, and select the necessary options (e.g. 32 oz., red color) before clicking the 'Buy' button. In order to pick the optimal product that matches user requirements, the agent may need to view and compare various products (including backtracking between pages), and potentially perform multiple searches. WebShop contains over one million products scraped from `amazon.com`, over 12 thousand crowdsourced instructions, and a diverse semantic action space of searching text queries and choosing text buttons. It is packaged into a convenient OpenAI Gym [26] environment and can be rendered in two modes (`HTML` or `simple`) with parallel observation spaces that are easy for human and model respectively. Rewards are automatically computed using a combination of programmatic matching functions that consider the attributes, type, options and price of the chosen product, alleviating the need for human evaluation and providing a path to scaling up interactive learning.

We develop several agents to perform this task, using both reinforcement learning (RL) and imitation learning (IL). We also leverage the latest pre-trained language models [161, 67] for representing and generating text. Our modular architecture includes a factorized processing of state observations and action choices using ResNets (visual) and Transformers (text), followed by an attention fusion layer that helps the agent contextually score each action. Our best agent achieves an average score of 62.4 (out of 100) and successfully completes the task 28.7% of the time, significantly higher than a heuristic baseline that achieves 45.6 and 9.6%, respectively. While this demonstrates the potential for IL and RL, the agents are still much lower than human experts, who can achieve 82.1 and 59.6% on this task.[1]

We perform several analyses and ablation studies to identify the cause of this gap and find several avenues for agent improvement in the future including more robust search generation, explicit memory modules, and better handling of noisy web text. Finally, we also demonstrate an instance of *sim-to-real* transfer by deploying agents trained with WebShop to operate on `amazon.com` and `ebay.com`, and find that they can achieve similar performances despite search engine and product differences,

[1]In our analysis (§2.5.3), we observe that the task requires patience and consistency, which is lacking in some crowdsource workers, leading to lower scores. Even with this caveat, the gap between human performance and the model remains significant.

and consistently outperform the rule baseline of using the first result returned by the commercial search engines when directly searching the instruction texts. This demonstrates the practical potential of our work towards developing agents that can operate autonomously on the world wide web (WWW).

2.2 Related Work

Reinforcement learning on the web. WikiNav [230] is a benchmark for RL agents navigating webpages, but the task is purely navigational with the actions restricted to either choosing a hyperlink to follow or deciding to stop. The World of Bits (WoB) benchmark [275] enables training of RL agents to complete tasks on webpages using pixel and Document Object Model (DOM) observations. Several follow-up papers have tackled MiniWoB using techniques like workflow-guided exploration [182], curriculum and meta-learning [97], DOM tree representation [127], adversarial environment generation [96] and large-scale behavioral cloning [121]. However, MiniWoB lacks long-range decision making across multiple different pages and does not scale easily in terms of difficulty or size due to its use of low-level mouse clicks and keystrokes as actions. In contrast, WebShop requires navigating longer paths with context-based action selection and backtracking, and it uses high-level *search* and *choose* actions that are more scalable and transferable to real settings. While not directly operating on web pages, AndroidEnv [305] and MoTIF [36] provide environments to train agents for interacting with apps and services on mobile platforms.

Non-interactive web-based tasks. Various supervised classification tasks on webpages have been proposed, including predicting web elements [243], generating API calls [287, 288, 337] and semantic parsing into concept-level navigation actions [200]. Perhaps most similar content-wise to our work is the Klarna product page dataset [115] which contains over $50,000$ product pages labeled with different element categories for supervised classification. All these works only consider supervised settings with a single decision, and may require the definition of web APIs or command templates for each domain. Our benchmark, WebShop, combines webpages with realistic text and image content with a rich and diverse interaction space for long-range sequential decision making.

Leveraging the web for traditional NLP tasks. Several papers have explored the use of the web for information extraction [211] and retrieval [3], question answering [366, 156], dialog [279], and training language models on webtext [6]. These approaches primarily use web search engines as a knowledge retriever for gathering additional evidence for the task at hand. Perhaps most similar to our work is WebGPT [209], which uses a web interface integrated with a search engine to train RL agents to navigate the web and answer questions. However, our environment has a more diverse action and observation space (including images) and does not require human-in-the-loop evaluation.

2.3 The WebShop Environment

We create WebShop as a large-scale web-based interactive environment with over 1.1 million real-world products scraped from amazon.com. In this environment, an agent needs to find and purchase a product according to specifications provided in a natural language instruction. WebShop is designed in a modular fashion which disentangles the website transitions from the task-specific aspects like instructions and reward, allowing for easy extension to new tasks and domains.

2.3.1 Task formulation

WebShop can be formulated as a partially observable Markov decision process (POMDP) $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{U}, \mathcal{O})$ with state space $\mathcal{S}$, action space $\mathcal{A}$, deterministic

Type	Argument	State → Next State
`search`	[*Query*]	Search → Results
`choose`	Back to search	$*$ → Search
`choose`	Prev/Next page	Results → Results
`choose`	[*Product title*]	Results → Item
`choose`	[*Option*]	Item → Item
`choose`	Desc/Overview	Item → Item-Detail
`choose`	Previous	Item-Detail → Item
`choose`	Buy	Item → Episode End

Table 2.1: Actions in WebShop.

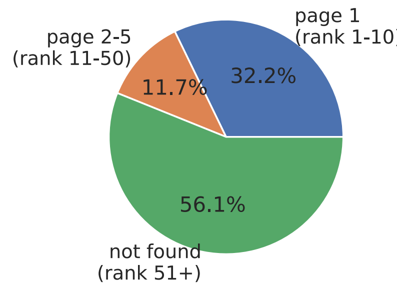


Figure 2.3: Item rank in search results when the instruction is directly used as search query.

transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, instruction space $\mathcal{U}$, and a state observation space $\mathcal{O}$.

State and action. A state $s \in \mathcal{S}$ represents a web page, which falls into one of the four types – the *search* page that contains a search bar, the *results* page that lists a set of products returned by a search engine, the *item* page that describes a product, or the *item-detail* page that shows further information about the product (Figure 2.2A(1-4) respectively). We define the following notations for a product y. We denote $\bar{y}$ to be the aggregation of the various text fields including product title, description, and overview. We denote y_{price} to be the price, Y_{opt} to be a set of buying options, and I to be a set of images, each corresponding to a specific option. Finally, each product is associated with Y_{att}, a set of attributes hidden from the agent which is extracted from the title and the *item-detail* pages (§2.3.2). The attributes are used for the automatic reward calculation.

An action $a \in \mathcal{A}(s)$ can either be searching a text query (e.g. `search`[Red shoes]) or choosing a text button (e.g. `choose`[Size 9]) as shown in Table 2.1. These two action types are not available simultaneously – search is only allowed when the agent is at the search page; on all other pages, click is the only action choice. The chosen action argument (button) will be clicked as a web link as opposed to the low-level mouse-click actions in previous environments such as World of Bits [275]. The transitions initiated by clicks deterministically redirect the web page to one of the four page types (Table 2.1). The transition initiated by search is based on a deterministic search engine (§2.3.2).

Observation. Using Flask [263] and OpenAI Gym [26], we provide two parallel observation modes to render the state and instruction $\mathcal{S} \times \mathcal{I} \rightarrow \mathcal{O}$: (1) `HTML` mode that contains the HTML of the web page, allowing for interaction in a web browser(Figure 2.2A), and (2) `simple` mode which strips away extraneous meta-data from raw HTML into a simpler format (Figure 2.2B). The human performance scores in §2.4.2 are collected in the `HTML` mode, while all models are trained and evaluated in the `simple` mode. Note that while the environment allows for training reinforcement learning agents on raw pixels in `HTML` mode (like in [275]), we believe that it provides a very low-level non-semantic action space. Moreover, it is straightforward to write a translator that converts any new HTML page into `simple` format for use with trained agents, which enables sim-to-real transfer.

Instruction and reward. Each natural language instruction $u \in \mathcal{U}$ contains the following information: a non-empty set of attributes U_{att}, a set of options U_{opt}, and a price u_{price}. The instruction is generated based on a target product y^* by human annotators. The instruction collection process is lightweight and scalable (§2.3.2). Concretely, $U_{\text{att}} \subseteq Y^*_{\text{att}}$ is a subset of the product attributes, $U_{\text{opt}} \subseteq Y^*_{\text{opt}}$ is a subset of the product option field-value pairs, $u_{\text{price}} > y^*_{\text{price}}$ is a price set to be higher than the target product price. For example, the instruction "Can you find me a pair of *black-and-blue* sneaker that is *good in rain weather*? I want it to have *puffy soles*, and price less than 90 dollars." contains the aforementioned attributes $U_{\text{att}} = \{$"waterproof", "soft sole"$\}$ and option $U_{\text{opt}} = \{$"color": "black and blue"$\}$. In each episode, the agent receives a reward $r = \mathcal{R}(s_T, a)$ in the end at timestep T, where

$a = \texttt{choose[buy]}$, y is the product chosen by the agent in the final state s_T, and Y_{att} and Y_{opt} are its corresponding attributes and options. The reward is defined as:

$$r = r_{\text{type}} \cdot \frac{|U_{\text{att}} \cap Y_{\text{att}}| + |U_{\text{opt}} \cap Y_{\text{opt}}| + \mathbf{1}[y_{\text{price}} \leq u_{\text{price}}]}{|U_{\text{att}}| + |U_{\text{opt}}| + 1} \tag{2.1}$$

where the type reward $r_{\text{type}} = \texttt{TextMatch}(\bar{y}, \bar{y}^*)$ is based on text matching heuristics to assign low reward when y and y^* have similar attributes and options but are obviously different types of products. For example, "butter" and "plant-based meat" differ in types but may both contain attributes "cruelty-free", "non-GMO", and an option "size: pack of 2".

Evaluation metrics. We use two evaluation metrics: (1) **Task Score**: defined as $(100 \times \text{avg. reward})$, which captures the average reward obtained across episodes; and (2) **Success Rate (SR)** defined as the portion of instructions where $r = 1$. Note that it is possible to obtain $r = 1$ for an episode even if the final product is not y^* — for example, there could be many items that satisfy the goal "I want a red shirt", even if the goal is generated from a specific red shirt item.

2.3.2 Environment implementation

Data scraping We use ScraperAPI [227] to scrape $1,181,436$ products from `amazon.com` across 5 categories (fashion, makeup, electronics, furniture, and food) using 113 sub-category names as queries. The product texts (title and item details) have an average length of 262.9 and a vocabulary size $224,041$ (word frequency higher than 10). In addition, the products have a total of $842,849$ unique options, reflecting the scale and complexity of the data.

Search engine We use Pyserini [177] for the search engine, where indices are built offline using a BM25 sparse retriever with text for each product concatenated from the title, description, overview, and customization options. The search engine is deterministic, which eases imitation learning and result reproducibility.

Attribute mining and annotation. Each product is annotated with a set of hidden *attributes*, which are used to represent its latent characteristics as well as to calculate the reward as detailed in §2.3. An attribute is a short natural language phrase that describes the property of the product (see examples in Figure 2.2). We mine the attributes by calculating TF-IDF scores for all bi-grams in the concatenated titles and descriptions based on each product category. We review the top 200 bi-grams for each category, remove the noisy ones by inspection (decide based on whether the bi-gram is human understandable), and assign them to the products. We consolidate a pool of 670 attributes.

Natural language instructions. We use Amazon Mechanical Turk (AMT) to collect natural language instructions that specify goal products with appropriate options. Specifically, an AMT worker is presented with a sampled goal product, including the product title, category, attributes, and the buying options, and asked to write a command to instruct an automatic shopping agent to find the target. Workers are instructed to avoid being too specific such as including the entire title in the instruction, but stay faithful to describing the target product. We collect a total of $12,087$ linguistically diverse instructions with an overall vocabulary size of $9,036$ words and an average length of 15.9 words.

Human demonstrations. We collect trajectories from humans performing the task in the `HTML` mode of WebShop to understand the task difficulty for humans and to analyze how humans would solve the task. We use qualification tests to train and select motivated workers to perform the task. We recruit and train a total of 13 workers for data collection, and among them we select the top 7 performing workers to be "experts". We also leverage this data to perform imitation learning.

2.3.3 Research challenges

WebShop brings together several research challenges for autonomous systems from various subfields in NLP and RL into a single benchmark. These include: 1) generation of good search queries [143, 388] and reformulation [231, 320], 2) strategic exploration for navigating through the website [356, 354, 182], 3) robust language understanding for textual state and action spaces [12, 34, 106, 277], and 4) long-term memory for comparing items or backtracking [330, 82, 154] (Figure 2.2). While we believe individual advances in each of these will improve agent performance, WebShop also provides an ideal testbed for the development of interdisciplinary techniques that tackle more than one of the above mentioned challenges simultaneously. For example, external memory modules may be very effective if combined with strategic exploration, or exploration could be helpful in information query reformulation. Further analysis based on human and model trajectories is in §2.5.3.

2.4 Methods

We propose various models that combine language and image pre-training with imitation learning (IL) and reinforcement learning (RL).

2.4.1 Rule baseline

A simple rule baseline is to search the exact instruction text, then choose and buy the first item in the results page without choosing any options. The heavy lifting of the lexical search engine makes it also a simple non-learnable information retrieval (IR) baseline, and would lead to a non-trivial attribute reward. However, simple heuristic

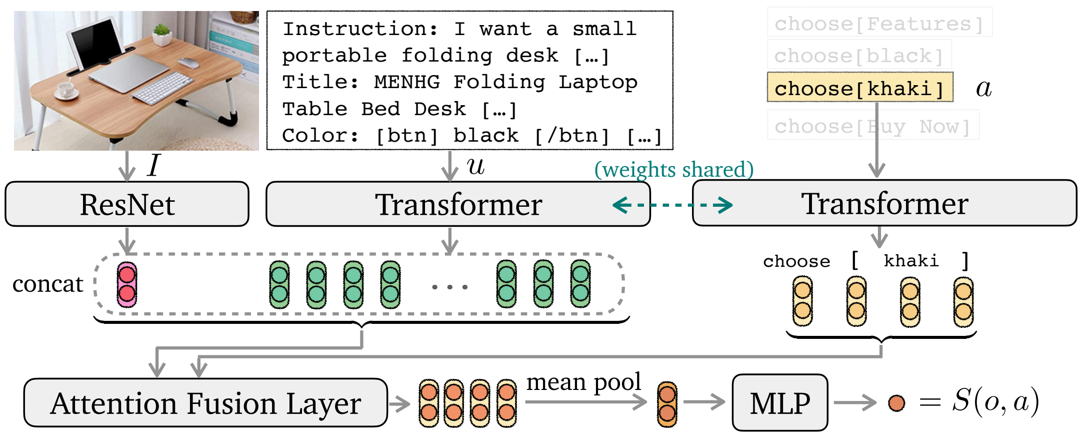


Figure 2.4: Architecture of our choice-based imitation learning (IL) model. The image I is passed to a ResNet to obtain the image representation. The instruction text u is passed to a transformer (initialized with BERT) to obtain the text representations. The concatenated bi-modal representations are fused with the action representations using the Attention Fusion Layer. The resulting fused-action representations are mean-pooled and reduced by an MLP layer to a scalar value $S(o, a)$ denoting the logit value of the action `choose`[khaki].

rules cannot resolve noisy natural language options, strategically explore, or learn to generate what to search, so the total reward and task success rate should be low.

2.4.2 Imitation learning (IL)

For the text generation and choice problems presented in WebShop, we propose using two pre-trained language models to separately learn how to search and choose from human demonstrations.

Imitating human search generation. We frame searching as a sequence-to-sequence text-generation problem: the agent generates a search action $a =$ `search`[...] given an instruction u without considering any other context (e.g. past searches, visited items). We use $M = 1,421$ instruction-search pairs from $1,012$ training human trajectories to construct a dataset $\mathcal{D} = \{(u, a)\}_{i=1}^{M}$ and fine-tune a BART model [161] parameterized by ϕ to perform conditional language modeling:

$$\mathcal{L}_{\text{search}} = \mathbb{E}_{u,a\sim\mathcal{D}}\left[-\log \pi_\phi(a \mid u)\right] \tag{2.2}$$

Imitating human choice. The choice-based imitation model (Figure 2.4) predicts a probability distribution over all the available click actions $\mathcal{A}(o)$ in observation o and maximizes the likelihood of the human clicked button $a^* \in \mathcal{A}(o)$. We construct a dataset $\mathcal{D}' = \{(o, \mathcal{A}(o), a^*)\}_{i=1}^{M'}$ of $M' = 9,558$ samples from the training human trajectories. We use a 12-layer pre-trained BERT model [68] parameterized by θ to encode the o into an observation representation of contextualized token embeddings, and we similarly encode each action. Each action representation is passed into a cross-attention layer with the observation representation, then mean pooled into a single vector and multiplied with a matrix W to obtain a scalar score $S(o, a)$. The policy $\pi_\theta\left(a \mid o, \mathcal{A}(o)\right)$ is the softmax distribution over action scores $S(o, a)$:

$$\mathcal{L}_{\text{choose}} = \mathbb{E}_{o,\mathcal{A}(o),a^*\sim\mathcal{D}'}\left[-\log \pi_\theta\left(a^* \mid o, \mathcal{A}(o)\right)\right] \tag{2.3}$$

$$\pi_\theta\left(a \mid o, \mathcal{A}(o)\right) \sim \exp\Big(W^\top \text{mean}\big[\text{cross-attn}\big(\text{BERT}(o;\theta), \text{BERT}(a;\theta)\big)\big]\Big) \tag{2.4}$$

Handling Images. We use a pre-trained ResNet-50 [109] to pre-process images across different products and options into a 512 dimensional feature vector, which is then transformed into 768 dimensions with a learned linear layer and concatenated to $\text{BERT}(o)$ as the observation representation.

Full pipeline. Combining the above during environment interaction, we use the BART model in the search page to generate the top-5 search queries via beam search and choose a random one. For other pages, we sample one action from $\pi_\theta\left(a \mid o, \mathcal{A}(o)\right)$ using the BERT model. We find these methods useful to encourage diverse actions. In contrast, an ineffective strategy that uses only the top generated search query or the button with the highest probability might lead to limited product candidates or being stuck (e.g. bouncing back and forth between pages).

2.4.3 Reinforcement learning (RL)

We also fine-tune the choice-based IL model with online RL (i.e. IL+RL). Prior work suggests that directly fine-tuning text generation via RL might lead to language drifting [157] and deteriorated performance. Therefore, we freeze the BART model to provide the top-10 search generations as a refined action space for the choice-based IL model to learn to pick – an inspiration borrowed from previous work in text games [356] and referential games [157]. We use the policy gradient method [205] with return-to-go $R_t = \mathbb{E}_\pi[r_t + \gamma R_{t+1}]$ and a learned value baseline $V(o) = W_v^\top \text{BERT}(o; \theta)$ parameterized by $\{W_v, \theta\}$ (the BERT weights are tied with the policy):

$$\mathcal{L}_{\text{PG}} = \mathbb{E}_\pi \left[-\left(R_t - V(o_t) \right) \log \pi \left(a_t \mid o_t, \mathcal{A}(o_t) \right) \right] \tag{2.5}$$

The value $V(o)$ is learned with an L2 loss $\mathcal{L}_{\text{value}} = (R_t - V(o_t))^2$. We also add an entropy loss $\mathcal{L}_{\text{entropy}} = \sum_{a \in \mathcal{A}(o_t)} \pi_\theta\big(a_t \mid o_t, \mathcal{A}(o_t)\big) \log \pi_\theta\big(a_t \mid o_t, \mathcal{A}(o_t)\big)$ to prevent premature convergence. Our full RL model minimizes the total loss $\mathcal{L}_{\text{RL}} = \mathcal{L}_{\text{PG}} + \mathcal{L}_{\text{value}} + \mathcal{L}_{\text{entropy}}$.

2.5 Experiments

2.5.1 Setup and task verification

We split a total of $12,087$ instructions into an i.i.d. distributed train / development / test split of $10,587$ / $1,000$ / 500 instances for all models. While future work can investigate splits with more generalization gaps (e.g. split by product category), we will show the i.i.d. split is already challenging for current models. We randomly sample a subset of the $10,587$ training instructions, then collect $1,012$ human demonstrations

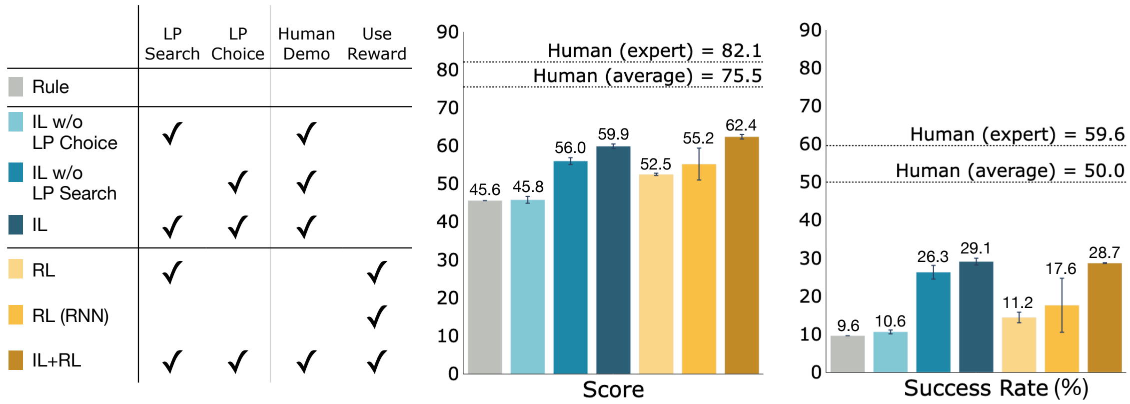


Figure 2.5: Task scores and Success Rate (%) for our models on the test split of WebShop over 3 trials. LP Search uses a pre-trained BART model to generate the search query and IL w/o LP Search uses the rule-based heuristic. LP Choice uses pre-trained BERT weights to initialize the choice action model and IL w/o LP Choice trains a Transformer from scratch.

for task verification and imitation learning (IL) and a further 54 demonstrations from instances in the development set for IL hyperparameter tuning and checkpoint selection. We also collect human trajectories for all 500 test instructions and report human and model performances averaged across these 500 instructions.

2.5.2 Results

Task performance. From Figure 2.5, we observe that the rule baseline obtains a low score of 45.6 and a very low success rate of 10% since it cannot resolve options specified in language or explore more products, empirically demonstrating the non-trivial nature of the task. The IL model significantly outperforms the rule baseline on both metrics, achieving a score of 59.9. Further RL finetuning improves the score to 62.4 while slightly hurting the success rate (29.1% → 28.7%) (analyzed further in §2.5.3). We also observe a significant gap between models and humans – our best model's success rate (29.1%) is less than half of expert humans (59.6%) and only 60% of the average human (50%). This indicates a great room for model improvement by tackling reseach challenges in WebShop.

IL ablations. Figure 2.5 also contains several ablations that confirm important design choices for models. When the choice action model for the IL agent is randomly initialized (**IL (w/o LP Choice)**; LP = language-pretraining), the success rate drops by nearly two-thirds, indicating the importance of language pre-training for our task. When the search query generator in the IL agent is replaced by a simple rule, which always uses the instruction text (**IL (w/o LP Search)**), both reward and success rate drop by around 3 points. This suggests the importance to explore by expanding the search space for exploration, but it is not as critical as learning to choose the right options. We experiment with incorporating history of one past observation and the last five actions into the model and find a slight degradation in the score from 59.9 to 57.3, suggesting more advanced techniques are needed to leverage past information.

RL ablations. When we directly train an RL agent (**RL**) from pre-trained BERT parameters, the performance is even worse than the rule baseline. This suggests that IL warm-starting is critical, possibly because of the significant domain shift from traditional language tasks. We also consider a simple RL model with RNN text encoders instead of the Transformer (**RL (RNN)**), which has a success rate more than 10% worse than the IL + RL model with a much larger variance. We hypothesize that RL with a more powerful architecture could help boost and stabilize the performance if the model is initialized with better language and task priors.

2.5.3 Analysis

To better understand the differences between the agents and human experts, we perform several fine-grained analyses. We first break down the overall score into its four sub-parts according to Eq. (2.1): 1) attribute score ($|U_{\text{att}} \cap Y_{\text{att}}|/|U_{\text{att}}|$), 2) option score ($|U_{\text{opt}} \cap Y_{\text{opt}}|/|U_{\text{opt}}|$), 3) price score ($\mathbf{1}[y_{\text{price}} \leq u_{\text{price}}]$), and 4) type score ($r_{\text{type}}$). We report trajectory statistics such as the average number of states, unique

	Score					Count		
	All	Att	Opt	Type	Price	State	Item	Search
Rule	45.6	66.6	0.0	80.5	86.0	3.0 (3 / 3)	1.0 (1 / 1)	1.0 (1 / 1)
IL	59.9	69.3	**45.2**	86.4	84.0	9.4 (90 / 3)	1.6 (11 / 1)	1.3 (17 / 1)
IL+RL	**62.4**	**74.0**	38.9	**89.7**	**88.7**	4.5 (5 / 1)	1.0 (1 / 1)	1.0 (1 / 1)
Human Expert	82.1	81.8	73.9	94.4	97.7	11.3 (114 / 4)	1.9 (16 / 1)	1.4 (16 / 1)

Table 2.2: Left: Score breakdown. Right: average, maximum, and minimum number of states visited, items checks, and searches in a trajectory.

Instruction 1	**Instruction 2**
I want to find white blackout shades that are 66 inches in width and 66 inches in height. They need to be easy to install [...]	I need a gingko light and 20"x20" pillow cover that is hand painted [...]

Human Actions ($r = 1.0$, length = 8)
`search[` 66 inches in width and 66 inches in height white shades `]`
`choose[` item : CALYX... `]`
`choose[` Back to Search `]`
`search[` 66 x 66 blackout shades `]`
`choose[` item : Milin... `]`
`choose[` opt : 66"w x 66"h `]`
`choose[` opt : cordless bottom up-blackout-white `]`
`choose[` Buy `]`
	Human Actions ($r = 1.0$, length = 17)
	`search[` gingko light 20"x20" pillow cover hand painted `]`
	`choose[` item : Maison... `]` [...]
	`choose[` Description `]`
	`choose[` ¡ Previous `]`
	`choose[` Overview `]`
	`choose[` ¡ Previous `]` [...]
	`choose[` item : Maison... `]`
	`choose[` opt : 20"x20" `]`
	`choose[` opt : nudes (gingko light) `]`
	`choose[` Buy `]`

IL+RL Actions ($r = 0.2$, length = 3)
`search[` white blackout shades 65 inches in width and 66 inches in height `]`
`choose[` item : Window... `]`
`choose[` Buy `]`
	IL+RL Actions ($r = 0.25$, length = 3)
	`search[` gingko light and 20x20 pillow cover hand painted `]`
	`choose[` item : UPOOS... `]`
	`choose[` Buy `]`

Table 2.3: Two example trajectories (showing only actions) from the human and the IL+RL model. We omit some human actions from instruction 2 for space and truncate the item names for readability. Red denotes options and blue denotes attributes.

items visited, and number of searches per episode in Table 2.2 and provide qualitative examples of the trajectories in Table 2.3.

Human expert vs. agents. Human experts outperform the agents on all score sub-parts (Table 2.2), but the most significant boost comes from the option score (a 28% gap), revealing that agents have trouble selecting the correct product options. Humans also have longer trajectories, explore more items and perform more searches

	Instr. text	IL BART	Human expert (first)	Human expert (last)
Score	94.9	94.5	94.5	**95.5**
Success Rate	85.4%	84.2%	85.6%	**87.8%**

Table 2.4: Task performance with the `Choice` oracle. *first* and *last* refer to the first and last search queries found in human demonstrations, respectively.

than the agents, with a higher variance, demonstrating their flexibility. Table 2.3 provides some samples trajectories. In the first example, the human decides to search again after removing 'inches', 'width', 'height', and 'white' from the query since product texts often contain abbreviated symbols for these terms like '"', 'w', and 'h'. Thus, **search generation** is challenging for models since it involves reasoning and adapting to grounded environments, and ideas from query reformulation [231, 3] could help alleviate this. Agents also struggle to perform robust **semantic matching**, which is important in choosing options that contain noisy paraphrases of instruction spans. In the second example, the human explores several products first, and decides to return to the first explored product, demonstrating long-term **memory** that is lacking in the IL+RL model.

Effect of RL fine-tuning after IL. Table 2.2 also shows that RL fine-tuning adapts the IL model to become more 'greedy' and less 'exploratory', as the average trajectory length drops from 9.4 to 4.8, and the model explores fewer items and search queries. As a result, the attribute, type, and price scores all increase, but option score drops from 45.2 to 38.9. This points to the need for a better balance exploration with exploitation during RL, e.g. by using intrinsic bonuses.

Results with at `Choice` oracle. To disentangle the effects of learning to search from choosing the right actions, we construct a `Choice` oracle that has access to the hidden reward function as well as hidden attributes and options underlying each product and instruction.[2] Given a search query, the `Choice` oracle will perform an exhaustive search over every result item, try out all combinations of options and finally choose the best item with options that maximize the reward — meaning each episode will take hundreds or thousands of steps, as opposed to 4.5 and 11.3 steps on average for the IL+RL model and human experts (Table 2.2). We use 500 test instructions and consider four types of search queries: the instruction text (used by rule baseline), top IL BART generated query (used by all learning models), and the first and last queries from human experts in each test trajectory.[3] `Choice` oracle improves the success rate of rule heuristics from 9.6% to 85.4%, and even the human expert success rate from 59.6% to 87.8% (Table 2.4), confirming that choosing the right actions is indeed a major bottleneck for current models with great room for improvement. However, using a better search query is still important even with such a strong `Choice` oracle, as the last human search query still outperforms other search queries. This also suggests human experts improve search query qualities over reformulations.

2.5.4 Zero-shot sim-to-real transfer

Finally, we conduct a '*sim-to-real*' transfer experiment where our models trained on WebShop are tested on the real-world Amazon (`amazon.com`) and eBay (`ebay.com`) shopping websites without any fine-tuning. We sample 100 test instructions and deploy 3 WebShop models (rule, IL, IL+RL) to interact with Amazon and eBay, and manually score each episode based on Eq. (2.1). As shown in Table 2.5, model performances on the two website are similar to WebShop performances in Figure 2.5, except for the rule baseline, likely due to the better search engine of Amazon than WebShop.

On `amazon.com`, IL+RL achieves a Score of 65.9 and SR of 25%, outperforming the Rule baseline's Score of 45.8 and SR of 19% by large margin. Similarly, on `ebay.com`,

[2] A similar search oracle is also possible but harder to design since the search space is infinite. One possible oracle is to search for the underlying product name for each instruction, but that makes choice trivial as the underlying product is then almost always the first search result.

[3] 74.8% of the time there is only one query in the trajectory.

	Amazon					eBay				
	Score / SR	Att	Opt	Type	Price	Score / SR	Att	Opt	Type	Price
Rule	45.8 / 19%	45.6	38.0	66.2	90.0	31.7 / 7%	62.3	25.9	49.0	67.0
IL	61.5 / 27%	60.7	**53.7**	85.6	96.0	58.2 / **21%**	60.2	**52.3**	85.1	96.9
IL+RL	**65.9** / **25%**	**71.6**	47.0	**87.8**	**100.0**	**62.3** / **21%**	**69.1**	39.5	**91.7**	**97.0**
Human	88.2 / 65%	86.2	76.3	99.0	100.0	79.7 / 40%	80.3	70.1	99.5	100.0

Table 2.5: Zero-shot sim-to-real transfer to Amazon and eBay over 100 test instructions. The Score / SR (Success Rate) column indicates the overall performance. The remaining breakdown are in Score.

IL+RL achieves a Score of 62.3 and SR of 21%, widely outperforming the Rule baseline's Score of 31.7 and SR of 7%. These results confirm positive sim-to-real values of trained agents for real-world web tasks despite domain shifts in data (products) and dynamics (search engine). We also obtain a human average score of 88.0 / 79.7 and success rate of 65% / 40% by asking turkers to find the instructed product on the Amazon and eBay websites respectively. While humans perform much better than agents, their web interactions are much slower — taking on average 815 seconds per episode as opposed to < 8 seconds per episode for our IL and IL+RL models on Amazon. This sim-to-real transfer only requires two minor coding additions, suggesting that environments like WebShop are suitable for developing *practical* grounded agents to reduce human effort on real-world web tasks.

2.6 Discussion

We have developed WebShop, a new web-based benchmark for sequential decision making and language grounding, modeled on interaction with an e-commerce website. We performed an empirical evaluation of autonomous agents trained using imitation and reinforcement learning, and demonstrated promising results on sim-to-real transfer to real-world shopping websites. Our qualitative and quantitative analysis of model and human trajectories (§2.5.3) identified several research challenges in WebShop and provided insights for future model development by incorporating multidisciplinary techniques. For example, pre-training with multi-modal data [169, 329], web hypertext [6], or web instruction-action mapping [244] could help agents better understand and leverage rich semantics of webpage content, actions, and instructions. Ideas from query (re)formulation [143, 388, 231, 320] may help agents expand the range of search exploration, and improved action exploration [246, 74, 308] and memory [330, 82, 154] mechanisms could help agents make better decisions over the long horizon and large action space. The modular design of WebShop also allows for new web tasks and domains to be easily incorporated, which we hope will help shape future research into grounded language agents with stronger capabilities for real-world web interaction.

Beyond web interaction, WebShop initiates and inspires a new direction to benchmark autonomous agents based on large-scale, real-world digital environments, such as the Internet, code terminals (Chapter 3), and other computer software. Compared to traditional agent setups that interact with humans, physical environments, or games, such **digital automation** tasks are both **scalable** to collect interactions and feedback and **practical** for alleviating our tedious digital labor and improving our life. They also present unique **challenges** for agents to reason over complex real-world text (e.g., webpages) and make open-ended language decisions (e.g., search query) over long horizon, which is not reflected in previous agent benchmarks. As we see in Section 2.5.2, imitation and/or reinforcement learning agents cannot solve such challenges yet require intensive training. This partly motivates the creation of language agents that reason to act, and we will see in Chapter 4 how language agents can significantly improve the WebShop performance using just one learning example.

Chapter 3

InterCode: Benchmarking Agents via Code Interaction

3.1 Introduction

Chapter 2 opens up the direction of benchmarking agents via **digital automation**, i.e., real-world computer-based tasks. Besides web browsing, another major computer-based task is coding, and it has two differences: (1) while the web environment features natural language, images, video, and other multimodal elements, coding mainly involves the interplay of natural and programming languages; (2) while web tasks are harder to evaluate, coding naturally has unit tests as a systematic and reliable means to evaluation. In this chapter, we establish interactive coding with execution feedback as a new problem for evaluating and developing agents.

For humans, programming is a naturally interactive process, but existing coding benchmarks are often not interactive. When a human programmer writes code, she relies on several iterations of a 'write-execute-test' loop in order to iteratively refine solutions, plan changes, test sub-modules, and solve ambiguities by checking execution behavior. While this is reminiscent of other human endeavors like writing, code compilation and execution produce exact results that provide a deterministic form of feedback to make the refinement process more straightforward. Depending on the observed results, programmers perform various levels of debugging and rewriting, and continue the process until their code satisfies the requirements.

There has been increasing interest in recent years around the development of models that can automatically generate code given a specification in natural language [80, 325, 55, 170, 158]. Powered by large-scale pre-training over thousands of codebases [5, 122, 86], these models have shown solid performance on static benchmarks like HumanEval [46], APPS [110], MBPP [15], CodeXGLUE [190]. However, generating code in a static, sequence-to-sequence or auto-regressive fashion has several drawbacks: 1) simple errors (even typos) can propagate and there is no chance for recovery or revision, 2) there is a disconnect between the code generation process and its downstream execution on the desired software and hardware environment, and 3) there is little room for human intervention or collaboration in the code generation process.

Recently, some works have proposed the use of execution feedback or interaction [326] to benefit code generation models [146, 117, 327, 110]. However, these papers consider their own individual setup and are difficult to compare with one other due to the use of different compilers, execution environments, feedback signals, and assumptions on the interactive process such as human participation to create task descriptions or provide natural language feedback. This makes it difficult to compare existing methods for code generation and to clearly understand the benefits of interactive generation.

To address these issues, we propose InterCode, the first standard coding benchmark designed natively with an interactive execution environment. Closely mimicking the human decision-making process, InterCode allows a coding agent to interactively receive feedback from compilers/interpreters that execute its code, and to submit further

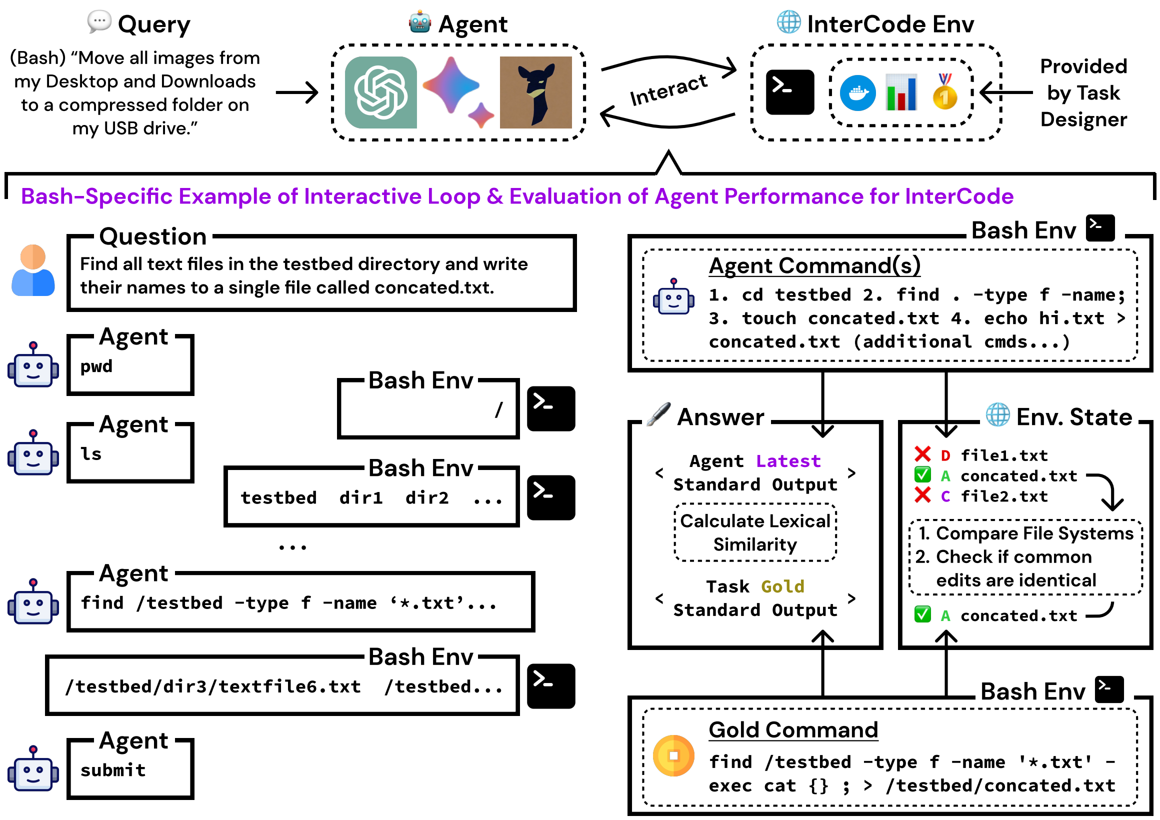


Figure 3.1: Overview of InterCode. Setting up an interactive code environment with InterCode requires a Dockerfile, dataset, reward function definition, and a small amount of subclass implementation. The interactive loop between agent and environment closely mirrors real world software development processes. While InterCode task performance is generally quantified as a binary 0/1 completion score, InterCode allows for the design of more complex evaluation criteria that can incorporate execution output and the effects of interaction on the state space.

refinements. We design InterCode to be like a standard reinforcement learning (RL) environment that requires minimal human intervention and one in which generated code is treated as actions, which are executed to reveal observations. Our framework is (1) language and platform agnostic and can easily be used for new coding problems, (2) uses self-contained Docker environments to provide safe execution, and (3) compatible out-of-the-box with traditional seq2seq generation methods, while also enabling and empowering the development of new interactive techniques.

We demonstrate the power of the framework by implementing Bash, SQL, and Python tasks within InterCode, building on pre-existing static datasets [378, 178, 15].

We perform experiments across diverse models and prompting methods, including ReAct [360] and Plan & Solve [316]. Our findings concretely showcase the benefits of interaction towards solving coding tasks, discuss the distribution of distinct code understanding challenges across different task settings, and explore the ease with which new tasks and datasets can be defined using InterCode.

3.2 Related Work

Interactive environments for coding. Most coding benchmarks (e.g. SQL - Spider [364], KaggleDBQA [160]; Bash - NLC2CMD [4], NL2Bash [178]; Python - HumanEval [46], APPS [110], MBPP [15], CodeXGLUE [190], CodeNet [251]) frame the coding problem as a sequence transduction problem (from instruction to code), rather than an interactive decision making problem with an execution environment. Attempts have been made to simulate interaction by developing conversational, dialogue-style [365, 363], multi-step problem solving [228] datasets, which involve pre-annotated human-designed queries. The work closest to InterCode has been recent explorations of Python Jupyter Notebooks as a natural choice for interactive coding [117, 146, 361]. However, task data and settings often constrain allowed actions to a closed domain of code and libraries [146, 361], use evaluation procedures or metrics that may not generalize [117], require human-in-the-loop participation (i.e. create task contexts, write problems, evaluate execution per task instance) [146], or are Python-exclusive [117, 146, 361, 327]. InterCode provides a more general purpose foundation defining interactive coding tasks that enables easy construction of diverse task settings, can have any programming language(s) as the action space, and has automatic, execution-based evaluation.

Execution-based evaluation for coding. Evaluation for NL-to-code generation models has recently shifted away from surface form similarity metrics (BLEU [240, 5], ROUGE [176], Exact Match) towards execution oriented ratings (unit tests [15, 46, 117, 146, 110], output matching [70, 117, 378]). The rigidity of surface form analysis overlooks code syntax features, ignores execution effect, or over-penalizes alternative solutions [382], On the contrary, execution-based assessment is a more thorough and comprehensive score of code functionality [110] and is a more natural fit for open-domain program usage that does not constrain code generation to a subset of the language space [327]. However, for newer benchmarks and datasets that put forth task definitions incorporating execution-based evaluation (APPS [110], ExeDS [117], ODEX [327]), the fundamental code generation task (Context + Code $\rightarrow$ Execution $\rightarrow$ Score) is still devoid of interaction. InterCode combines execution-based evaluation with flexible task construction, enabling more diverse problem-solving paradigms within a unified coding task formulation. InterCode's use of virtual containers as execution sandboxes protect against harmful actions and allow for advanced evaluation criteria beyond the aforementioned ones.

Methods for interactive or execution-based coding. The value of generative code models and interactive problem solving has motivated a recent proliferation of work to augment reasoning capabilities' of existing language models [360, 276, 316, 358, 372, 47] or propose new modeling techniques to tackle coding as a sequential decision making and reasoning tasks [35, 48, 75, 170, 42, 158], of which evaluation is unit test based. Approaches that leverage execution typically use re-ranking [374, 226, 362, 369] or majority vote [48, 170, 273] to decide on a final prediction. Additional work also explores incorporating human-in-the-loop [40, 145], compilers [321], and text [322, 371] feedback. A common thread among these contributions is that 1) the task setting can only provide the investigated form of feedback and 2) sought-after capabilities are exemplified by strong performance on favorably curated tasks and datasets, rendering comparisons across benchmarks tedious. InterCode has the potential to standardize the evaluation of these methods because 1) the interactive coding task is a conglomeration of many interesting interaction, reasoning, and decision-making challenges and 2)

InterCode's task construction makes it possible to incorporate a wide variety of sources of feedback.

3.3 The InterCode Benchmark

3.3.1 Formulation

The InterCode benchmark formalizes interactive coding with execution feedback as a partially observable Markov decision process (POMDP) $(\mathcal{U}, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R})$ with instruction space $\mathcal{U}$, state space $\mathcal{S}$, action space $\mathcal{A}$, observation space $\mathcal{O}$, transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, and reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$. Given a coding instruction $u \in \mathcal{U}$ in natural language, an agent issues code or a special `submit` keyword as an action $a_t \in \mathcal{A}$. An action is *admissible* [356] if it can be parsed and executed in the compiler/interpreter environment, and an admissible action incurs a change in the latent state space $s_{t+1} \in \mathcal{S}$, and an execution feedback as observation $o_{t+1} \in \mathcal{O}$. The interaction loop repeats until the `submit` action is issued, wherein the task episode ends and a reward $r = \mathcal{R}(s_T, \texttt{submit}) \in [0, 1]$ is computed, with 1 representing task completion. We use the **Success Rate (SR)** metric, defined as the proportion of task episodes where $r = 1$. We also define the **Error %** metric, which is the percentage of *non* admissible actions across task episodes.

3.3.2 Construction pipeline

At a high level, InterCode decomposes the construction of an interactive coding task into three **modular** parts: (1) environment construction, (2) data collection, and (3) reward design. This workflow allows for the safe execution of transition functions, flexible reward design, and convenient adaptation of existing instructions to an interactive setting.

Docker-based environments. InterCode uses Docker [202] virtual containers as a general-purpose execution sandbox. Given a `Dockerfile` that defines a system and execution entrypoint, InterCode creates a corresponding, stateful virtual container that hosts the desired state space and transition function. We choose Docker as the basis of InterCode's environment construction for its safe execution in virtual containers, reproducibility of a `Dockerfile` across any Docker-equipped machine, and excellent coverage of application code, libraries, and dependencies offered by the `Dockerfile` DSL.

Data collection. InterCode requires that a dataset has at minimum two fields: `query`, a natural language instruction $u \in \mathcal{U}$, and `gold`, an answer or code block that is a procedure for generating the correct answer. We define these conditions to make it easy to adapt existing text-to-code datasets to an interactive setting while also leaving plenty of bandwidth for constructing new tasks and datasets.

Reward design. Across a single task episode, the action, observation, and state modification (if any) per interaction loop are implicitly logged by InterCode. InterCode's default reward function determines task completion via an exact match of the agent's execution output (observation and state modifications) against the gold command, where 1 is awarded only if all components match. Since Exact Match is usually too stringent of an evaluation criteria, InterCode exposes a reward function endpoint that has access to both the interaction history and the execution container, allowing for custom reward function definitions that can incorporate multiple signals.

3.3.3 Implementations

Following the procedure discussed in Section 3.3.2, we create two separate InterCode based environments where Bash and SQL are the action spaces respectively. Table 3.1 summarizes them.

	Environment	Dataset	Reward Function
Bash	Ubuntu Terminal	NL2Bash [178] (200)	Latest Output+ File Diff
SQL	MySQL Database	Spider 1.0 [364] (1034)	Latest Output
Python	Python Interpreter	MBPP [15] (117)	Submitted Function

Table 3.1: Rundown of the three environments developed using the InterCode framework. The numbers in parentheses refer to the number of task instances adopted from each dataset. Each environment is defined in under 200 lines of code total.

InterCode-Bash. We define a `bash` shell within an Ubuntu Operating System as the task setting. To evaluate an agent's ability to adapt generations to different situations, we architect four distinct file systems that can be swapped into the Bash environment by changing a single line in the `Dockerfile`.

We bootstrap the NL2Bash [178] dataset (which lacks specificity in queries and grounding to any underlying file system, preventing it from being used directly for interactive evaluations) to create an interactive coding task where an agent completes an instruction via `bash` actions. Transferring NL2Bash to the interactive task setting requires simple transformations to ground instructions and `gold` code blocks in the file system. First, we consider a subset of 1000 commands with each having ≥ 4 utilities. We then filter out commands that are non-UNIX, non-Linux, or use utilities we currently do not support (eg. "ssh", "sudo", time, and GUI-dependent utilities). Finally, we enhance under-specified commands with specific file names/directory names/paths and update deprecated utilities/flags. The resulting 200 commands are grouped into 4 disjoint sets, 3 of which were grounded to custom-designed file systems, while one set is file-system agnostic. This categorization allows for a comprehensive evaluation of different command-grounding scenarios.

The InterCode-Bash dataset instructions typically make one or both of the following two types of requests. It either 1. Requests information that can be answered via execution output (i.e. "How many files...", "What is the size of...", "Where is `<file>` stored?") or 2. Requests a change to the location/configuration/content of a file or folder (i.e. "Move `dir1` folder...", "Set permissions of...", "Append a line to..."). Therefore, we define a custom reward function that evaluates an agent's performance against file system modifications and the latest execution output. Execution output is graded with a simple lexical similarity function. File system assessment is done in two parts. First, a comparison of the agent's and `gold` command's list of file system changes (list of `[path, modification type` $\in$ `[added, changed, deleted]]` entries) reveals any extraneous or missing changes. Second, `md5sum` hashes of each commonly edited file path are compared to determine if an added or changed file was altered correctly. A max score of 1 is achieved only if the correct file paths are changed, the changes are correct, and the latest execution output matches the gold command output exactly.

InterCode-SQL. We write a `Dockerfile` that defines a SQL interpreter within a MySQL database as the task setting. To create the databases and tables necessary for the task dataset, we write type resolution scripts and perform database conversions using the `sqlite3mysql` [307] Python library to adapt the Spider [364] database and table schema to a MySQL format. We then consolidate all setup code into a single, unified MySQL `.sql` dump that contains the complete set of schemas for all tables across 20 different databases. On container start-up, this file is invoked automatically, creating and populating databases with tables and tables with records.

The Spider [364] dataset is a large-scale cross-domain dataset originally meant for evaluating SQL query generations from natural language questions. We adapt the development set, which contains 1034 task instances, and remove all extraneous columns aside from the natural language questions and gold SQL command. The `instruction` and `gold` values do not require any additional pre-processing to be compatible with the MySQL task environment.

Finally, we employ Intersection over Union (IoU), or more formally the Jaccard Index, to quantify the correctness of the latest execution output generated by the agent against the gold output, where both outputs are a list of records. A non-tabular execution output receives a reward of 0 by default. Among the items that lie in the intersection of the agent and gold execution outputs, we also apply a penalty if the records are in the incorrect order. To quantify how sorted the agent output is relative to the gold output, we lean on Kendall's τ and adjust the output range to $[0, 1]$. The IoU score is then directly scaled by this coefficient. All in all, only a correctly ordered list with the exact set of records found in the gold output receives a score of 1.

InterCode-Python. In this setting, we define a Python interpreter running within an Ubuntu operating System as the task setting. The Dockerfile can be configured to run any Python version. The interpreter is not initialized with any dependencies, but PyPI packages can be installed and used by the agent.

We use the MBPP [15] dataset which presents the code completion task of synthesizing Python code from a method header and docstring. Evaluation of correctness is performed with an associated set of unit tests given by MBPP. The MBPP dataset is straightforward to adapt to the interactive setting, requiring no modifications to the query or evaluation components. Finally, we directly inherit MBPP's evaluation procedure of proportion of unit tests passed. With InterCode, it is easy to use existing datasets to evaluate how well models can use different programming languages as actions.

Validations. To verify the functionality of action execution in the task environment and the correctness of custom reward functions, we write testing scripts for both Bash and SQL that pass the gold command in as a dummy agent's action to ensure that the command is admissible and executes without error, and to verify that the reward received by the command is 1. To confirm that InterCode's dataset specification is enforced across multiple accepted file formats, we define a custom InterCode data loader class which is then rigorously unit tested.

3.4 Methods

We perform preliminary experiments to gauge the proficiency and behavior of current large language models on interactive coding tasks with Bash and SQL. To observe and elicit relevant reasoning skills, we draw on several existing prompting strategies that have been put forth to augment language models' reasoning and problem-solving skills. We apply these prompting strategies to models across the following three families: OpenAI (`text-davinci-003`, `gpt-3.5-turbo`, `gpt-4`), PaLM-2 (`text-bison-001`, `chat-bison-001`) [13], and Open Source (Vicuna-13B [50], StarChat-16B [167]).

Figure 3.2 visualizes the four adjusted prompting strategies we evaluate on InterCode.

Single Turn is a zero-shot attempt. A model is given a simple description of the task setting and asked to generate code in a specific programming language that would address the query. The first generation in response to the user's question is then evaluated in the InterCode environment.

"Try Again" is an iterative feedback set up. In the initial message, the agent is informed of the task setting and its interactive nature; an agent has multiple turns to interact with the system, wherein each turn, upon generating an action, the execution output of the action is fed back as an observation. This continues until a reward of 1 (task completion) is achieved or the number of turns (n) is exhausted. The agent's position in this approach is meant to mirror human software development as closely as possible. The goal of this method is to probe language models' raw interactive coding abilities in addition to illustrating the benefits and different challenges that arise in interactive coding tasks.

ReAct and Plan & Solve. We write prompts and design workflows that follow the text and task configurations described in ReAct [360] (which will be detailed in Chapter 4) and Plan & Solve [316] as faithfully as possible. For these two approaches, the termination of a task episode is conditioned upon the agent's own judgment, as

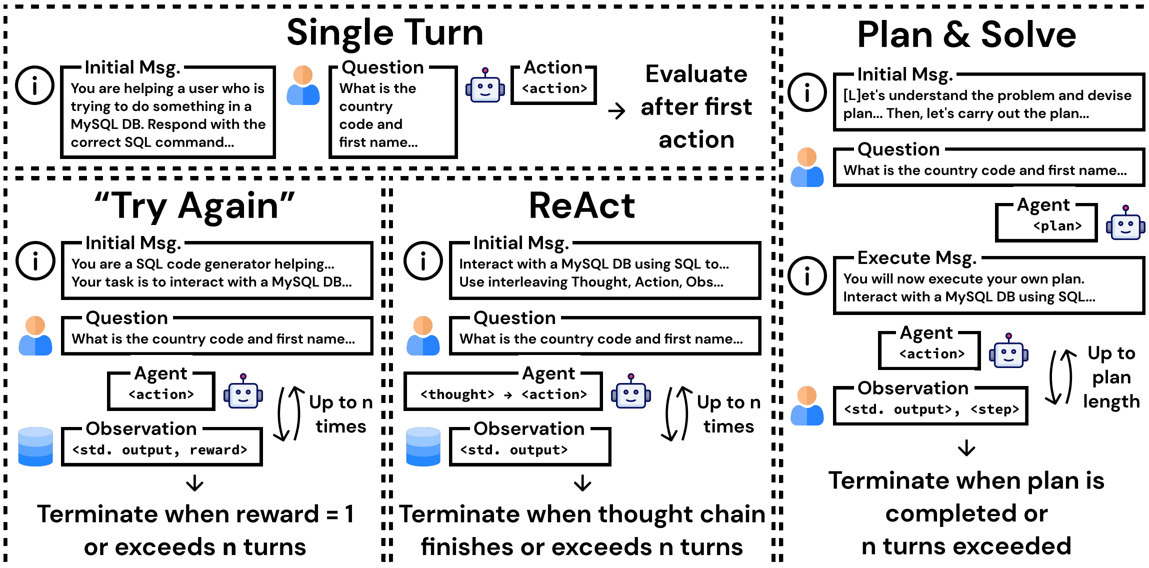


Figure 3.2: Overview of Prompting Strategies adjusted for evaluation on InterCode. The "Try Again" termination constraint is conditioned on reward = 1, while ReAct [360] and Plan & Solve [316] are determined by the agent itself. This is because the purpose of the "Try Again" method is to explore how capable agents are at error correction from feedback, while the other two are more concerned with the overall success of general problem-solving strategies.

InterCode-SQL	Single Turn					Try Again ($n = 10$)				
Model / Hardness	Easy	Med	Hard	Extra	All	Easy	Med	Hard	Extra	All
text-davinci-003	20.6	4.9	1.7	0.0	7.4	32.4	14.6	5.2	4.2	15.6
gpt-3.5-turbo	22.6	8.3	**5.7**	**3.6**	10.5	72.5	44.3	43.7	21.1	47.3
gpt-4	19.8	7.2	4.6	3.0	9.1	**87.5**	**76.7**	**66.7**	**52.4**	**73.7**
text-bison-001	**23.8**	**10.9**	**5.7**	0.6	**11.5**	27.0	12.3	5.7	0.6	12.9
chat-bison-001	18.5	6.5	4.0	0.0	7.9	22.2	7.8	6.9	0.0	9.9
Vicuna-13B	8.1	1.3	0.6	0.0	2.6	18.9	3.4	1.7	0.0	6.3
StarChat-16B	21.8	7.4	2.9	0.0	8.9	22.3	8.5	2.9	1.2	9.7

Table 3.2: Success Rate for single vs. multi turn evaluation on InterCode-SQL. Query difficulty is adopted from Spider [364]. Best metrics are in **bold**.

our goal with these methods is to gauge the transferability to and efficacy of existing reasoning frameworks with respect to the interactive coding task.

InterCode-Bash	Single Turn					Try Again ($n = 10$)				
Model / File System	1	2	3	4	All	1	2	3	4	All
text-davinci-003	10.0	32.1	28.8	33.3	24.6	30.0	**52.8**	32.2	44.4	38.7
gpt-3.5-turbo	**30.0**	**39.6**	33.3	37.0	**34.5**	**45.0**	49.1	45.0	48.1	46.5
gpt-4	25.0	37.7	**36.7**	**40.7**	34.0	41.7	47.2	**51.7**	**59.2**	**48.5**
text-bison-001	15.0	22.6	11.7	22.2	17.0	23.3	28.3	16.7	22.2	22.5
chat-bison-001	12.1	22.5	16.7	22.2	17.7	13.8	24.5	18.3	22.2	19.2
Vicuna-13B	10.0	24.5	18.3	7.4	16.0	15.0	35.8	25.0	22.2	24.5
StarChat-16B	15.5	22.6	13.3	22.2	17.7	17.2	30.2	21.7	29.6	23.7

Table 3.3: Success Rate across file systems for single vs. multi-turn evaluation on InterCode-Bash. To evaluate models' ability to interact with different task settings, we evaluate disjoint sets of Bash instructions across four different file systems. Best metrics are in **bold**.

3.5 Experiments

3.5.1 Base models comparison

Task performances. We first compare the success rate of models in the Single Turn and Try Again settings for both the InterCode-Bash and SQL datasets. From Table 3.2 and Table 3.3, we observe that performance across different levels of task difficulty (SQL) and different file systems (Bash) is superior in the interactive setting for all models, with a notable multi-fold increase for GPT-4 (9.1% $\rightarrow$ 73.7%) on the InterCode-SQL task.

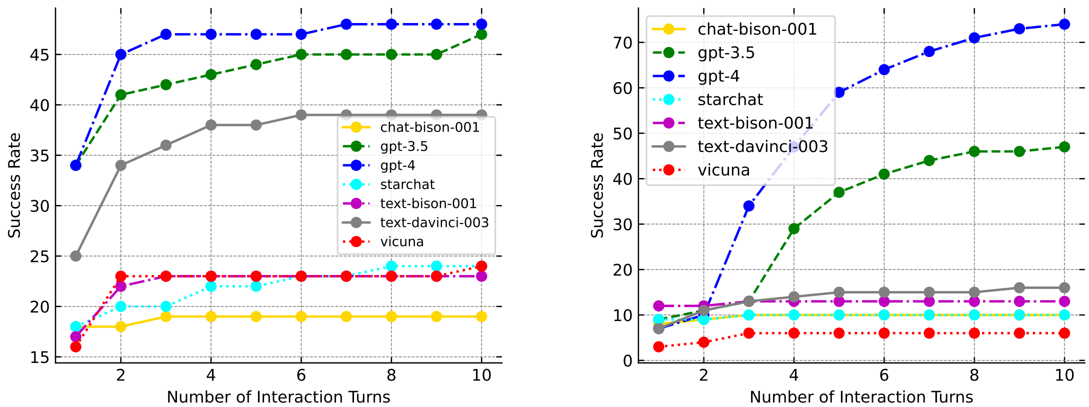
Analysis of interactions. Manual inspection of trajectory logs indicates that models actively exercise later turns for discovering relevant context, correcting errors via execution feedback as observations, and solving problems via iteratively constructing and editing actions as affirmed by Figure 3.3. In addition, models also demonstrate a level of planning and modular problem solving; for instructions with `gold` commands that chain multiple commands together (i.e. with |, >, or `;` in `bash`) or consist of multiple sub-problems (i.e. subqueries in `SQL`), models will use observations from solving smaller sub-problems in earlier turns to compose the higher-order action.

Failure cases. With that said, both Figure 3.3 exhibits a plateauing in Success Rate and and Error %. This suggests that as the amount of context and feedback builds up, models are less capable of discerning relevant past history toward future actions. In late-turn scenarios, task episode trajectories often reveal repetition of earlier actions, a failure to effectively use recent observations towards deciding an appropriate next action, or an inability to recognize that a current problem-solving chain of thought is inconclusive or futile. This is particularly evident for `hard` and `extra` level InterCode-SQL task instructions that require context spanning across several tables and actions that incorporate multiple clauses. We note that even when the full schema of all tables and their descriptions are offered in addition to the original instructions, models still benefit greatly from using interaction to experiment with different `JOIN` and filtering operators across multiple turns. A larger context window size, retrieval of useful memory, and more adaptive reasoning paradigms are just a handful of potential solutions to overcoming such challenges.

3.5.2 Prompting strategy comparison

Initiating language agents with prompting strategies that encourage different forms of reasoning toward problem-solving improves performance on the interactive coding task to varying degrees. Table 3.4 presents side-by-side comparisons of the success rate, number of turns, and error rate per strategy. Compared to Try Again, which lacks specific guidance on leveraging multiple turns, more explicit reasoning frameworks such as ReAct and Plan & Solve policies generally achieve higher success rates (SQL: 47.3% $\rightarrow$ 58.7%) with fewer turns and a higher rate of admissible commands.

Different tasks present different learning challenges. An important skill to solving the InterCode-SQL task is the ability to discover context and construct actions conditionally based on information revealed in prior observations. Given that InterCode-SQL task instructions are phrased most commonly as questions, adapting



(a) Success rate vs. turns for InterCode-Bash (b) Success rate vs. turns for InterCode-SQL

Figure 3.3: Growth in Success Rate with increase in number of interaction turns across models configured with Try Again prompting strategy for InterCode-Bash and SQL tasks.

	Try Again ($n = 10$)			ReAct ($n = 10$)			Plan & Solve		
	SR	Turns	Error %	SR	Turns	Error %	SR	Turns	Error %
SQL	47.3	7.25	46.4	**58.7**	5.30	**6.94**	49.1	**4.29**	16.2
Bash	**46.5**	6.15	24.9	20.5	**4.40**	**20.4**	28.0	6.65	53.3

Table 3.4: Comparison of different prompting strategies across the entire InterCode-SQL and InterCode-Bash datasets using `gpt-3.5-turbo` as the base model. *Turns* refers to the average number of turns taken for a single task episode. For Try Again and ReAct, the max number of turns $n = 10$. The highest Success Rate, fewest Turns, and lowest Error % are highlighted per dataset since they reflect more accuracy and efficient task solving. Best metrics are in **bold**.

to the task setting and new information discovered along the way puts more emphasis on error correction and context discovery. On the other hand, the more declarative and multi-step nature of the InterCode-Bash task instructions is more aptly solved by planning and modular task completion. These distinctions manifest in the Plan & Solve strategy's performance gap between the InterCode-SQL and InterCode-Bash tasks; while Plan & Solve encourages a model to decompose problems into more manageable steps, the strategy is less favorable towards adjusting on the fly in response to execution feedback.

More adaptive reasoning is favorable. Compared to "imperative" reasoning paradigms such as Plan & Solve which prescribe a relatively rigid procedure, more flexible frameworks like ReAct, which do not enforce any particular logical formula or roadmap, are more conducive to eliciting a broader set of reasoning capabilities. However, while ReAct's performance is generally superior to Plan & Solve, tasks solved by *both* strategies with `gpt-3.5-turbo` make up 57% (407/708) and 27.6% (21/76) of the union of all successfully solved InterCode-SQL and InterCode-Bash tasks respectively. This discrepancy highlights a trade-off between the guidance and structural constraints that are inherent to prompting strategies; schemes that draw out specific reasoning patterns often overlook other equally useful capabilities. InterCode's interactive coding task can serve as a strong litmus test toward more adaptable, variegated model reasoning.

InterCode's task formulation, modular design, flexible task construction, and use of virtual containers enable task designers to manifest new, complex, code-driven tasks, where completion is much more attainable through interaction. We draw inspiration from Capture the Flag (CTF) [58], a competitive cybersecurity game that requires expertise in coding, cryptography (i.e. binary exploitation, forensics), reverse engineering, and recognizing security vulnerabilities to accomplish the primary objective of discovering encrypted "flags" concealed within code snippets or file systems.

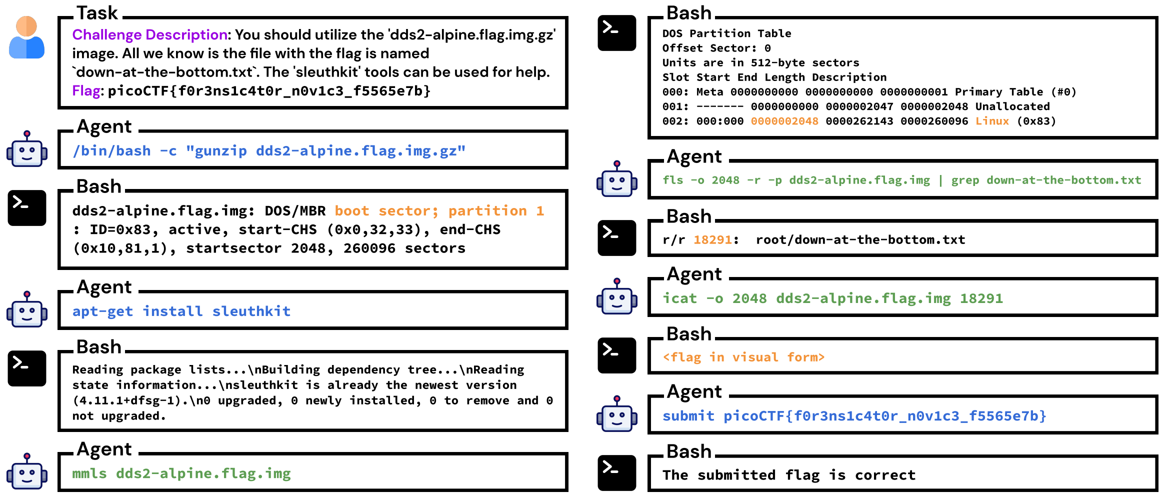


Figure 3.4: GPT-4's interaction trajectory for a binary exploitation CTF task. This requires proficiency in Bash and Python, among additional knowledge and reasoning. Orange text and arrows highlight the feedback that the model attends to in generating the next action. In last step, agent submits flag.

Compared to InterCode-Bash & -SQL, CTF is much more complicated, requiring an agent to exercise knowledge of multiple coding languages, modularize a higher-order objective into sub-problems, construct multi-step plans towards solving each problem, and adjust strategy when a plan fails to yield any useful insights.

We establish InterCode-CTF, a new dataset consisting of 100 CTF objectives from picoCTF [309]. Following the interactive coding task formulation, each task instance in InterCode-CTF is given as a `<instruction, assets, hidden flag>` tuple. We first construct a Bourne Shell within an Ubuntu OS as the task environment. Here, InterCode's use of virtual containers is crucial, as necessary actions can be irreversibly damaging on real systems (i.e. `rm -rf`, `sudo` access). Per task instance, the associated assets (e.g., images, executables, code), necessary for task completion, are copied into the OS file system. Given this setting, a task worker must understand the given material and investigate the assets to develop potential solutions. Executing a successful approach must be done across multiple steps with various conditionals, where the execution feedback of a prior step could have a significant effect on the next step. Figure 3.4 spotlights the diverse skills needed for CTF.

3.6 Towards More Challenging and Practical Code Interaction

InterCode opens up the general paradigm of solving code-based problems in interactive settings with execution feedback. Under this paradigm, we can study more challenging and practical problems beyond traditional coding datasets. In particular, we can benchmark code interaction based on real-world problems for two representative groups that code, competitive programmers and software engineers.

3.6.1 USACO: Towards Olympiad-level programming

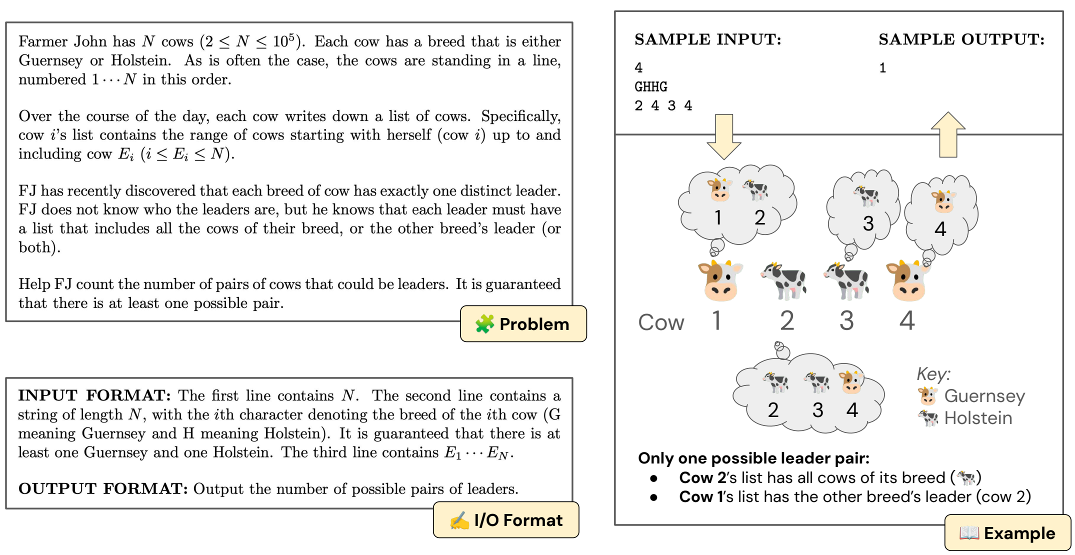


Figure 3.5: Example USACO problem description, formatting instructions, and illustration (problem id: `1275_bronze_leaders`). Solving this problem requires a combination of *grounded* reasoning about the concept of leaders, *creative* thinking to precisely count different cases of leader pairs, and *algorithmic* reasoning to perform these ad hoc operations in linear time.

Computing Olympiads contain some of the most challenging problems for humans, requiring complex algorithmic reasoning, puzzle solving, in addition to generating efficient code. However, it has been understudied as a domain to evaluate language models (LMs). We introduce the USACO benchmark with 307 problems from the USA Computing Olympiad, along with high-quality unit tests, reference code, and official analyses for each problem. These resources enable us to construct and test a range of LM inference methods for competitive programming for the first time. We find GPT-4 only achieves a 8.7% pass@1 accuracy with zero-shot chain-of-thought prompting, and our best inference method improves it to 20.2% using a combination of self-reflection and retrieval over episodic knowledge. However, this is far from solving the benchmark, and new models or techniques for interactive coding are clearly needed.

To better understand the remaining challenges, we design a novel human-in-the-loop study and surprisingly find that a small number of targeted hints enable GPT-4 to solve 13 out of 15 problems previously unsolvable by any model and method. Our benchmark, baseline methods, quantitative results, and qualitative analysis serve as an initial step toward LMs with grounded, creative, and algorithmic reasoning.

More details can be seen in [274].

3.6.2 SWE-bench: Towards solving real-world GitHub issues

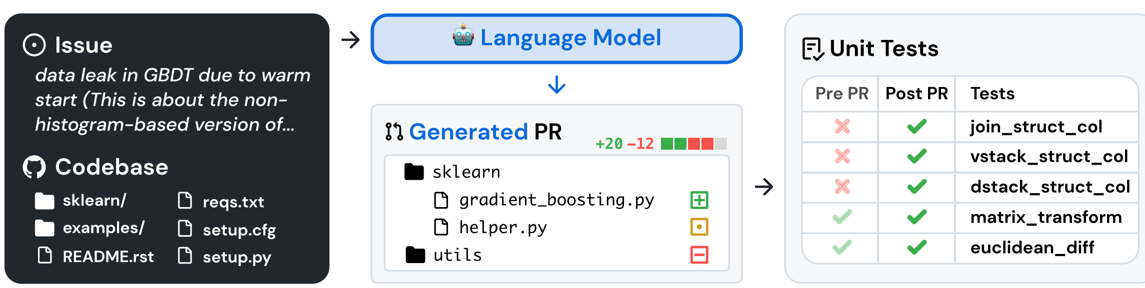


Figure 3.6: SWE-bench sources task instances from real-world Python repositories by connecting GitHub issues to merged pull request solutions that resolve related tests. Provided with the issue text and a codebase snapshot, models generate a patch that is evaluated against real tests.

Real-world software engineering can be a rich, sustainable, and challenging testbed for evaluating the next generation of language models for code interaction, and offer complementary challenges to Olympiad programming. We therefore introduce SWE-bench, an evaluation framework including 2294 software engineering problems drawn from real GitHub issues and corresponding pull requests across 12 popular Python repositories. Given a codebase along with a description of an issue to be resolved, a language model is tasked with editing the codebase to address the issue. Resolving issues in SWE-bench frequently requires understanding and coordinating changes across multiple functions, classes, and even files simultaneously, calling for models to interact with execution environments, process extremely long contexts and perform complex reasoning that goes far beyond traditional code generation. Claude 2 and GPT-4 solve a mere 4.8% and 1.7% of instances respectively, even when provided with an oracle retriever, clearly calling for new methodology. More details can be seen in [129].

3.6.3 DevBench: Towards comprehensive software development

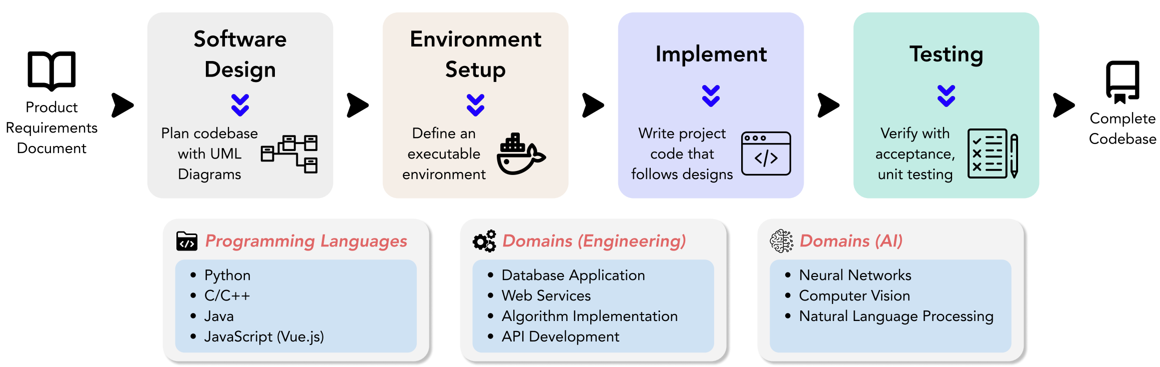


Figure 3.7: DevBench features multiple stages of software development, including software design, environment setup, implementation, and testing (both acceptance and unit testing).

Single-file code generation or repository issue debugging do not measure the full spectrum of challenges raised by real-world programming activities. To this end, we propose DevBench, a comprehensive benchmark that evaluates LLMs across various stages of the software development lifecycle, including software design, environment setup, implementation, acceptance testing, and unit testing. DevBench features a wide range of programming languages and domains, high-quality data collection, and carefully designed and verified metrics for each task. Empirical studies show that current LLMs, including GPT-4-Turbo, fail to solve the challenges presented within DevBench. Analyses reveal that models struggle with understanding the complex structures in the repository, managing the compilation process, and grasping advanced programming concepts. More details can be seen in [164].

3.7 Discussion

We have developed InterCode, a novel lightweight framework that facilitates interaction between language models and the underlying environment, enabling them to mimic the human approach to language-to-code generation. Our framework has shown promising results when applied to state-of-the-art models using different prompting styles. It effectively leverages the capabilities of LMs to break down complex tasks and recover from errors within a secure and isolated environment. The ability to seamlessly convert existing datasets into the interactive format using `InterCodeEnv` API, and furthermore, the Bash and SQL environments, empowers task designers to construct new tasks to unlock the plethora of challenges that await in the space of interactive coding.

Leveraging the paradigm of interactive coding, we have also proposed several benchmarks (USACO, SWE-bench, DevBench) with more challenging and practical coding problems than traditional coding datasets. LMs clearly cannot solve these in a sequence-to-sequence setup, which motivates the next part of the thesis that constructs language agents to interactively reason and act.

Part II

Methods

Chapter 4

ReAct: Building Agents that Reason to Act

4.1 Introduction

In Part I of the thesis, we have introduced a set of digital automation problems for autonomous agents, such as web browsing, interactive coding, and software engineering. In contrast to traditional agent benchmarks such as video games or robotics simulations, they present direct and tremendous practical values, and the key challenge of decision-making in open-ended and complex real-world environments. We have shown that traditional imitation or reinforcement learning approaches, or standard LLM-based approaches, do not work well on digital automation tasks like WebShop or SWE-bench. In Part II of the thesis, we establish the methodological foundation of a new type of AI agents that we term "language agents", which perform language reasoning to act.

Why reasoning to act? A unique feature of human intelligence is the ability to seamlessly combine task-oriented actions with verbal reasoning (or inner speech, 10), which has been theorized to play an important role in human cognition for enabling self-regulation or strategization [312, 192, 81] and maintaining a working memory [16].

Consider the example of cooking up a dish in the kitchen. Between any two specific actions, we may reason in language in order to track progress ("now that everything is cut, I should heat up the pot of water"), to handle exceptions or adjust the plan according to the situation ("I don't have salt, so let me use soy sauce and pepper instead"), and to realize when external information is needed ("how do I prepare dough? Let me search on the Internet"). We may also act (open a cookbook to read the recipe, open the fridge, check ingredients) to support the reasoning and to answer questions ("What dish can I make right now?"). This tight synergy between "acting" and "reasoning" allows humans to learn new tasks quickly and perform robust decision making or reasoning, even under previously unseen circumstances or facing information uncertainties.

Recent results have hinted at the possibility of combining verbal reasoning with interactive decision making in autonomous systems. On one hand, properly prompted large language models (LLMs) have demonstrated emergent capabilities to carry out several steps of reasoning traces to derive answers from questions in arithmetic, commonsense, and symbolic reasoning tasks [332]. However, this "chain-of-thought" reasoning is a static black box, in that the model uses its own internal representations to generate thoughts and is not grounded in the external world, which limits its ability to reason reactively or update its knowledge. This can lead to issues like fact hallucination and error propagation over the reasoning process (Figure 4.1 (1b)). On the other hand, recent work has explored the use of pre-trained language models for planning and acting in interactive environments [7, 209, 356, 119], with a focus on predicting actions via language priors. These approaches usually convert multi-modal observations into text, use a language model to generate domain-specific actions or plans, and then use a controller to choose or execute them. However, they do not employ language models to reason abstractly about high-level goals or maintain a working memory to support acting, barring [120] who perform a limited form of

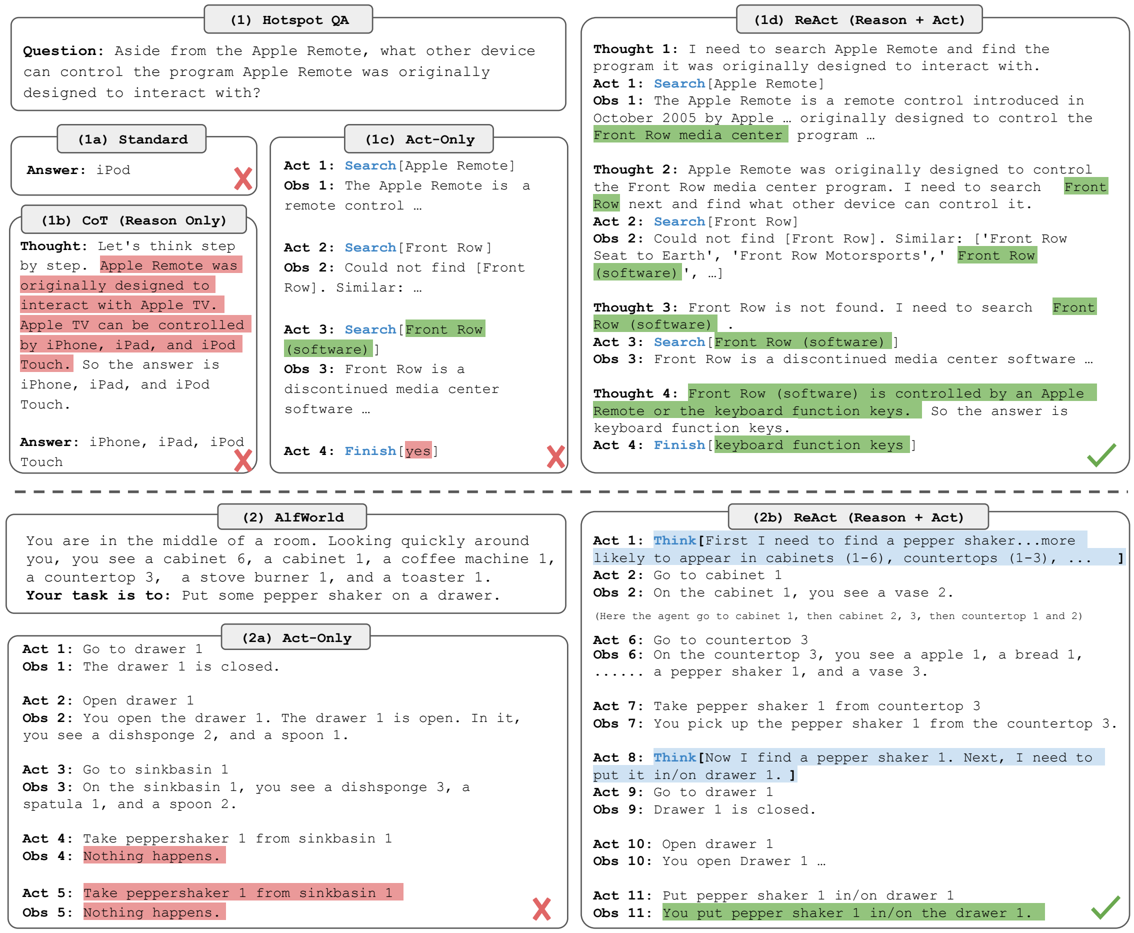


Figure 4.1: (1) Comparison of 4 prompting methods, (a) Standard, (b) Chain-of-thought (CoT, Reason Only), (c) Act-only, and (d) ReAct (Reason+Act), solving a HotpotQA [351] question. (2) Comparison of (a) Act-only and (b) ReAct prompting to solve an AlfWorld [278] game. In both domains, we omit in-context examples in the prompt, and only show task solving trajectories generated by the model (Act, Thought) and the environment (Obs).

verbal reasoning to reiterate spatial facts about the current state. Beyond such simple embodied tasks to interact with a few blocks, there have not been studies on how reasoning and acting can be combined in a synergistic manner for general task solving, and if such a combination can bring systematic benefits compared to reasoning or acting alone.

In this work, we present ReAct, a general paradigm to combine reasoning and acting with language models for solving diverse language reasoning and decision making tasks (Figure 4.1). ReAct prompts LLMs to generate both verbal reasoning traces and actions pertaining to a task in an interleaved manner, which allows the model to perform dynamic reasoning to create, maintain, and adjust high-level plans for acting (reason to act), while also interact with the external environments (e.g. Wikipedia) to incorporate additional information into reasoning (act to reason).

We conduct empirical evaluations of ReAct and state-of-the-art baselines on four diverse benchmarks: question answering (HotPotQA, 351), fact verification (Fever, 304), text-based game (ALFWorld, 278), and webpage navigation (WebShop, 352). For HotPotQA and Fever, with access to a Wikipedia API that the model can interact with, ReAct outperforms vanilla action generation models while being competitive with chain-of-thought reasoning (CoT) [332]. The best approach overall is a combination of ReAct and CoT that allows for the use of both internal knowledge and externally obtained information during reasoning. On ALFWorld and WebShop, two or even one-shot ReAct prompting is able to outperform imitation or reinforcement learning methods trained with $10^3 \sim 10^5$ task instances, with an absolute improvement of 34% and 10% in success rates respectively. We also demonstrate the importance of sparse, versatile reasoning in decision making by showing consistent advantages over controlled baselines with actions only. Besides general applicability and performance boost, the combination of reasoning and acting also contributes to model interpretability, trustworthiness, and diagnosability across all domains, as humans can readily distinguish information from model's internal knowledge versus external environments, as well as inspect reasoning traces to understand the decision basis of model actions.

To summarize, our key contributions are the following: (1) we introduce ReAct, a novel prompt-based paradigm to synergize reasoning and acting in language models for general task solving; (2) we perform extensive experiments across diverse benchmarks to showcase the advantage of ReAct in a few-shot learning setup over prior approaches that perform either reasoning or action generation in isolation; (3) we present systematic ablations and analysis to understand the importance of acting in reasoning tasks, and reasoning in interactive tasks; (4) we analyze the limitations of ReAct under the prompting setup (i.e. limited support of reasoning and acting behaviors), and perform initial finetuning experiments showing the potential of ReAct to improve with additional training data. Scaling up ReAct to train and operate on more tasks and combining it with complementary paradigms like reinforcement learning could further unlock the potential of large language models.

4.2 Related Work

Language model for reasoning. Perhaps the most well-known work of using LLMs for reasoning is Chain-of-Thought (CoT) [332], which reveals the ability of LLMs to formulate their own "thinking procedure" for problem solving. Several follow-up works have since been performed, including least-to-most prompting for solving complicated tasks [380], zero-shot-CoT [142], and reasoning with self-consistency [323]. Recently, [197] systematically studied the formulation and structure of CoT, and observed that the presence of symbols, patterns and texts is crucial to the effectiveness of CoT. Other work has also been extended to more sophisticated reasoning architecture beyond simple prompting. For example Selection-Inference [60] divides the reasoning process into two steps of "selection" and "inference". STaR [367] bootstraps the reasoning process by finetuning the model on correct rationales generated by the model itself. Faithful reasoning [59] decomposes multi-step reasoning into three steps, each performed by a dedicated LM respectively. Similar approaches like Scratchpad [234], which finetunes a LM on intermediate computation steps, also demonstrate improvement on multi-step computation problems. In contrast to these methods, ReAct performs more than just isolated, fixed reasoning, and integrates model actions and their corresponding observations into a coherent stream of inputs for the model to reason more accurately and tackle tasks beyond reasoning (e.g. interactive decision making).

Language model for decision making. The strong capability of LLMs has enabled them to perform tasks beyond language generation, and it is becoming more popular to take advantage of LLMs as a policy model for decision making, especially in interactive environments. WebGPT [209] uses an LM to interact with web browsers, navigate through web pages, and infer answers to complicated questions from ELI5 [78]. In comparison to ReAct, WebGPT does not explicitly model the thinking and reasoning procedure, instead rely on expensive human feedback for reinforcement learning. In conversation modeling, chatbots like BlenderBot [280] and Sparrow [90] and task-oriented dialogue systems like SimpleTOD [114] also train LMs to make decision about API calls. Unlike ReAct, they do not explicitly consider the reasoning procedure either, and also relies on expensive datasets and human feedback collections for policy learning. In contrast, ReAct learns a policy in a much cheaper way, since the decision making process only requires language description of the reasoning procedure.[1]

LLMS have also been increasingly employed in interactive and embodied environments for planning and decision making. Perhaps most relevant to ReAct in this respect are SayCan [7] and Inner Monologue [120], which use LLMs for robotic action planning and decision making. In SayCan, LLMs were prompted to directly predict possible actions a robot can take, which is then reranked by an affordance model grounded on the visual environments for final prediction. Inner Monologue made further improvements by adding the eponymous "inner monologue", which is implemented as injected feedback from the environment. To our knowledge, Inner Monologue is the first work that demonstrates such a closed-loop system, which ReAct builds on. However, we argue that Inner Monologue does not truly comprise of inner

[1] Human feedback can also be incorporated in a complementary manner but we leave it for future work.

thoughts — this is elaborated in Section 4.5. We also note that leveraging language as semantically-rich inputs in the process of interactive decision making has been shown to be successful under other settings [1, 136, 119, 168]. It is becoming more evident that with the help of LLMs, language as a fundamental cognitive mechanism will play a critical role in interaction and decision making. What is more, progress in LLMs has also inspired the development of versatile and generalist agents like [260].

4.3 ReAct: Synergizing Reasoning and Acting

Consider a general setup of an agent interacting with an environment for task solving. At time step t, an agent receives an observation $o_t \in \mathcal{O}$ from the environment and takes an action $a_t \in \mathcal{A}$ following some policy $\pi(a_t|c_t)$, where $c_t = (o_1, a_1, \cdots, o_{t-1}, a_{t-1}, o_t)$ is the *context* to the agent. Learning a policy is challenging when the mapping $c_t \mapsto a_t$ is highly implicit and requires extensive computation. For example, the agent shown in Figure 4.1(1c) is unable to generate the correct final action (Act 4) to finish the QA task as it requires complex reasoning over the trajectory context (Question, Act 1-3, Obs 1-3). Similarly, the agent shown in Figure 4.1(2a) fails to comprehend from the context that sinkbasin 1 does not contain peppershaker 1, thus keep producing hallucinating actions.

The idea of ReAct is simple: we augment the agent's action space to $\hat{\mathcal{A}} = \mathcal{A} \cup \mathcal{L}$, where $\mathcal{L}$ is the space of language. An action $\hat{a}_t \in \mathcal{L}$ in the language space, which we will refer to as a *thought* or a *reasoning trace*, does not affect the external environment, thus leading to no observation feedback. Instead, a thought $\hat{a}_t$ aims to compose useful information by reasoning over the current context c_t, and update the context $c_{t+1} = (c_t, \hat{a}_t)$ to support future reasoning or acting. As shown in Figure 4.1, there could be various types of useful thoughts, e.g. decomposing task goals and create action plans (2b, Act 1; 1d, Thought 1), injecting commonsense knowledge relevant to task solving (2b, Act 1), extracting important parts from observations (1d, Thought2, 4), track progress and transit action plans (2b, Act 8), handle exceptions and adjust action plans (1d, Thought 3), and so on.

However, as the language space $\mathcal{L}$ is unlimited, learning in this augmented action space is difficult and requires strong language priors. Here, we mainly focus on the setup where a frozen large language model, PaLM-540B [52], is prompted with few-shot in-context examples to generate both domain-specific actions and free-form language thoughts for task solving (Figure 4.1 (1d), (2b)). Each in-context example is a human trajectory of actions, thoughts, and environment observations to solve a task instance. For the tasks where reasoning is of primary importance (Figure 4.1(1)), we alternate the generation of thoughts and actions so that the task-solving trajectory consists of multiple thought-action-observation steps. In contrast, for decision making tasks that potentially involve a large number of actions (Figure 4.1(2)), thoughts only need to appear sparsely in the most relevant positions of a trajectory, so we let the language model decide the asynchronous occurrence of thoughts and actions for itself.

Since decision making and reasoning capabilities are integrated into a large language model, ReAct enjoys several unique features: **A) Intuitive and easy to design**: Designing ReAct prompts is straightforward as human annotators just type down their thoughts in language on top of their actions taken. No ad-hoc format choice, thought design, or example selection is used here. **B) General and flexible**: Due to the flexible thought space and thought-action occurrence format, ReAct works for diverse tasks with distinct action spaces and reasoning needs, including but not limited to QA, fact verification, text game, and web navigation. **C) Performant and robust**: ReAct shows strong generalization to new task instances while learning solely from one to six in-context examples, consistently outperforming baselines with only reasoning or acting across different domains. We also show in Section 4.4 additional benefits when finetuning is enabled, and in Section 4.5 how ReAct performance is robust to prompt selections. **D) Human aligned and controllable**: ReAct promises an interpretable sequential decision making and reasoning process where humans can easily inspect reasoning and factual correctness.

4.4 Experiments: Knowledge-Intensive Reasoning

We begin with knowledge-intensive reasoning tasks like multi-hop question answering and fact verification. As shown in Figure 4.1(1d), by interacting with a Wikipedia API, ReAct is able to retrieve information to support reasoning, while also use reasoning to target what to retrieve next, demonstrating a synergy of reasoning and acting.

4.4.1 Setup

Domains. We consider two datasets challenging knowledge retrieval and reasoning: (1) HotPotQA [351], a multi-hop question answering benchmark that requires reasoning over two or more Wikipedia passages, and (2) FEVER [304], a fact verification benchmark where each claim is annotated SUPPORTS, REFUTES, or NOT ENOUGH INFO, based on if there exists a Wikipedia passage to verify the claim. In this work, we operate in a question-only setup for both tasks, where models only receive the question/claim as input without access to support paragraphs, and have to rely on their internal knowledge or retrieve knowledge via interacting with an external environment to support reasoning.

Action Space. We design a simple Wikipedia web API with three types of actions to support interactive information retrieval: (1) `search`[`entity`], which returns the first 5 sentences from the corresponding `entity` wiki page if it exists, or else suggests top-5 similar entities from the Wikipedia search engine, (2) `lookup`[`string`], which would return the next sentence in the page containing `string`, simulating Ctrl+F functionality on the browser. (3) `finish`[`answer`], which would finish the current task with `answer`. We note that this action space mostly can only retrieve a small part of a passage based on exact passage name, which is significantly weaker than state-of-the-art lexical or neural retrievers. The purpose is to simulate how humans would interact with Wikipedia, and force models to retrieve via explicit reasoning in language.

4.4.2 Methods

ReAct Prompting. For HotpotQA and Fever, we randomly select 6 and 3 cases[2] from the training set and manually compose ReAct-format trajectories to use as few-shot exemplars in the prompts. Similar to Figure 4.1(d), each trajectory consists of multiple thought-action-observation steps (i.e. dense thought), where free-form thoughts are used for various purposes. Specifically, we use a combination of thoughts that decompose questions ("I need to search x, find y, then find z"), extract information from Wikipedia observations ("x was started in 1844", "The paragraph does not tell x"), perform commonsense ("x is not y, so z must instead be...") or arithmetic reasoning ("1844 ¡ 1989"), guide search reformulation ("maybe I can search/look up x instead"), and synthesize the final answer ("...so the answer is x").

Baselines. We systematically ablate ReAct trajectories to build prompts for multiple baselines (with formats as Figure 4.1(1a-1c)): (a) **Standard prompting** (Standard), which removes all thoughts, actions, observations in ReAct trajectories. (b) **Chain-of-thought prompting** (CoT) [332], which removes actions and observations and serve as a reasoning-only baseline. We also build a self-consistency baseline (CoT-SC) [323, 324] by sampling 21 CoT trajectories with decoding temperature 0.7 during inference and adopting the majority answer, which is found to consistently boost performance over

[2]We find more examples do not improve performance.

CoT. (c) **Acting-only prompt** (Act), which removes thoughts in ReAct trajectories, loosely resembling how WebGPT [209] interacts with the Internet to answer questions, though it operates on a different task and action space, and uses imitation and reinforcement learning instead of prompting.

Combining Internal and External Knowledge. As will be detail in Section 4.4.3, we observe that the problem solving process demonstrated by ReAct is more factual and grounded, whereas CoT is more accurate in formulating reasoning structure but can easily suffer from hallucinated facts or thoughts. We therefore propose to incorporate ReAct and CoT-SC, and let the model decide when to switch to the other method based on the following heuristics: A) **ReAct** $\rightarrow$ **CoT-SC**: when ReAct fails to return an answer within given steps, back off to CoT-SC. We set 7 and 5 steps for HotpotQA and FEVER respectively as we find more steps will not improve ReAct performance[3]. B) **CoT-SC** $\rightarrow$ **ReAct**: when the majority answer among n CoT-SC samples occurs less than $n/2$ times (i.e. internal knowledge might not support the task confidently), back off to ReAct.

Finetuning. Due to the challenge of manually annotating reasoning traces and actions at scale, we consider a bootstraping approach similar to [367], using 3,000 trajectories with correct answers generated by ReAct (also for other baselines) to finetune smaller language models (PaLM-8/62B) to decode trajectories (all thoughts, actions, observations) conditioned on input questions/claims.

4.4.3 Results and observations

ReAct outperforms Act consistently. Table 4.1 shows HotpotQA and Fever results using PaLM-540B as the base model with different prompting methods. We

[3]Of all trajectories with correct final answers, those with 7 steps on HotpotQA and 5 steps on FEVER only take up 0.84% and 1.33% respectively.

Prompting method[a]	**HotpotQA** (EM)	**Fever** (Acc)
Standard	28.7	57.1
CoT [332]	29.4	56.3
CoT-SC [323]	33.4	60.4
Act	25.7	58.9
ReAct	27.4	60.9
CoT-SC → ReAct	34.2	**64.6**
ReAct→ CoT-SC	**35.1**	62.0
Supervised SoTA [387, 162]	67.5	89.5

Table 4.1: PaLM-540B Results on HotpotQA and Fever.

[a]HotpotQA EM is 27.1, 28.9, 33.8 for Standard, CoT, CoT-SC in [324].

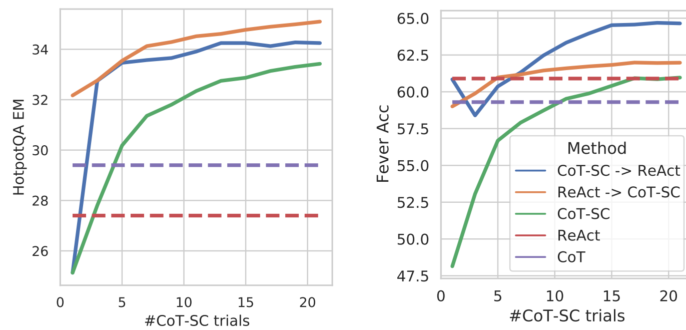


Figure 4.2: PaLM-540B prompting results with respect to number of CoT-SC samples used.

	Type	Definition	ReAct	CoT
Success	True positive	Correct reasoning trace and facts	94%	86%
	False positive	Hallucinated reasoning trace or facts	6%	14%
Failure	Reasoning error	Wrong reasoning trace or repetitive failure	47%	16%
	Search result error	Search return empty or does not contain useful information	23%	-
	Hallucination	Hallucinated reasoning trace or facts	0%	56%
	Label ambiguity	Right prediction but did not match the label precisely	29%	28%

Table 4.2: Types of success and failure modes of ReAct and CoT on HotpotQA, as well as their percentages in randomly selected examples studied by human.

note that ReAct is better than Act on both tasks, demonstrating the value of reasoning to guide acting, especially for synthesizing the final answer, as shown in Figure 4.1 (1c-d). Fine-tuning results 4.3 also confirm the benefit of reasoning traces for more informed acting.

ReAct vs. CoT. On the other hand, ReAct outperforms CoT on Fever (60.9 vs. 56.3) and slightly lags behind CoT on HotpotQA (27.4 vs. 29.4). Fever claims for SUPPORTS/REFUTES might only differ by a slight amount, so acting to retrieve accurate and up-to-date knowledge is vital. To better understand the behavioral difference between ReAct and CoT on HotpotQA, we randomly sampled 50 trajectories with correct and incorrect answers (judged by EM) from ReAct and CoT respectively (thus 200 examples in total), and manually labeled their success and failure modes in Table 4.2. Some key observations are as follows:

A) **Hallucination is a serious problem for CoT**, resulting in much higher false positive rate than ReAct (14% vs. 6%) in success mode, and make up its major failure mode (56%). In contrast, the problem solving trajectory of ReActis more grounded, fact-driven, and trustworthy, thanks to the access of an external knowledge base.

B) **While interleaving reasoning, action and observation steps improves ReAct's groundedness and trustworthiness, such a structural constraint also reduces its flexibility in formulating reasoning steps**, leading to more reasoning error rate than CoT. we note that there is one frequent error pattern specific to ReAct, in which the model repetitively generates the previous thoughts and actions, and we categorize it as part of "reasoning error" as the model fails to reason about what the proper next action to take and jump out of the loop[4].

C) **For ReAct, successfully retrieving informative knowledge via search is critical.** Non-informative search, which counts for 23% of the error cases, derails the model reasoning and gives it a hard time to recover and reformulate thoughts. This is perhaps an expected trade-off between factuality and flexibility, which motivates our proposed strategies of combining two methods.

ReAct + CoT-SC perform best for prompting LLMs. Also shown in Table 4.1, the best prompting method on HotpotQA and Fever are ReAct → CoT-SC and CoT-SC → ReAct respectively. Furthermore, Figure 4.2 shows how different methods perform with respect to the number of CoT-SC samples used. While two ReAct + CoT-SC methods are advantageous at one task each, they both significantly and consistently outperform CoT-SC across different number of samples, reaching CoT-SC performance with 21 samples using merely 3-5 samples. These results indicate the

[4]We suspect that this could be due to the sub-optimal greedy decoding procedure, and future work using better decoding (e.g. beam search) might help address this issue.

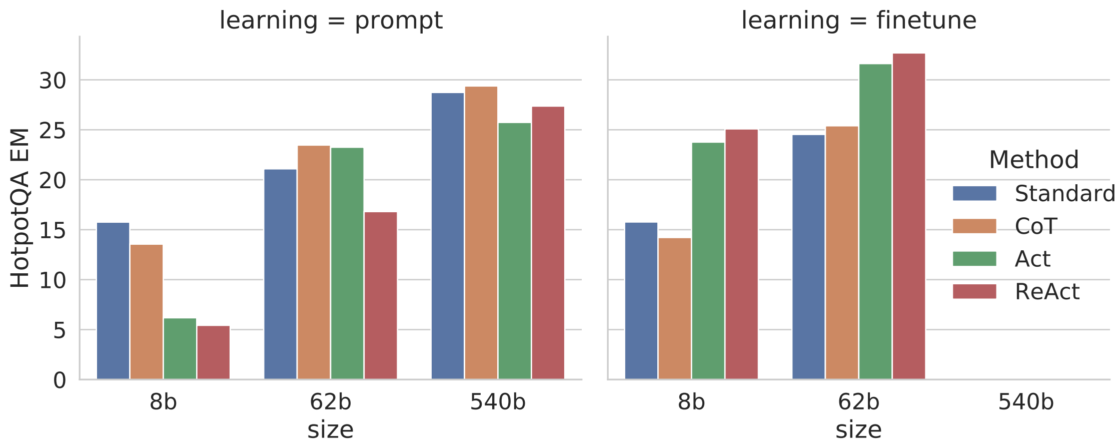


Figure 4.3: Scaling results for prompting and finetuning on HotPotQA with ReAct (ours) and baselines.

value of properly combining model internal knowledge and external knowledge for reasoning tasks.

ReAct performs best for fine-tuning. Figure 4.3 shows the scaling effect of prompting/finetuning four methods (Standard, CoT, Act, ReAct) on HotpotQA. With PaLM-8/62B, prompting ReAct performs worst among four methods due to the difficulty to learn both reasoning and acting from in-context examples. However, when finetuned with just 3,000 examples, ReAct becomes the best method among the four, with PaLM-8B finetuned ReAct outperforming all PaLM-62B prompting methods, and PaLM-62B finetuned ReAct outperforming all 540B prompting methods. In contrast, finetuning Standard or CoT is significantly worse than finetuning ReAct or Act for both PaLM-8/62B, as the former essentially teaches models to memorize (potentially halluincated) knowledge facts, and the latter teaches models how to (reason and) act to access information from Wikipedia, a more generalizable skill for knowledge reasoning. As all prompting methods are still significantly far from domain-specific state-of-the-art approaches (Table 4.1), we believe finetuning with more human-written data might be a better way to unleash the power of ReAct.

4.5 Experiments: Sequential Decision Making

We also test ReAct on two language-based interactive decision-making tasks, ALFWorld and WebShop, both of which feature complex environments that require agents to act over long horizons with sparse rewards, warranting the need for reasoning to act and explore effectively.

ALFWorld. ALFWorld [278] (Figure 4.1(2)) is a synthetic text-based game designed to align with the embodied ALFRED benchmark [277]. It includes 6 types of tasks in which an agent needs to achieve a high-level goal (e.g. examine paper under desklamp) by navigating and interacting with a simulated household via text actions (e.g. go to coffeetable 1, take paper 2, use desklamp 1). A task instance can have more than 50 locations and take an expert policy more than 50 steps to solve, thus challenging an agent to plan and track subgoals, as well as explore systematically (e.g. check all desks one by one for desklamp). In particular, one challenge built into ALFWorld is the need to determine likely locations for common household items (e.g. desklamps will likely be on desks, shelfs, or dressers), making this environment a good fit for LLMs to exploit their pretrained commonsense knowledge. To prompt ReAct, we randomly annotate three trajectories from the training set for each task type, where each trajectory includes sparse thoughts that (1) decompose the goal, (2) track subgoal completion, (3) determine the next subgoal, and (4) reason via commonsense where to find an object and what to do with it.

Following [278], we evaluate on 134 unseen evaluation games in a task-specific setup. For robustness, we construct 6 prompts for each task type through each permutation of 2 annotated trajectories from the 3 we annotate. Act prompts are constructed using the same trajectories, but without thoughts — since task instances are randomly chosen from the training set, it favors neither ReAct nor Act and provides a fair and controlled comparison to test the importance of sparse thoughts. For baselines, we use BUTLER [278], an imitation learning agent trained on 10^5 expert trajectories for each task type[5].

WebShop. Can ReAct also interact with noisy real-world language environments for practical applications? We investigate WebShop [352], a recently proposed online shopping website environment with 1.18M real-world products and 12k human instructions (detailed in Chapter 2). Unlike ALFWorld, Webshop contains a high variety of structured and unstructured texts (e.g. product titles, descriptions, and options crawled from Amazon), and requires an agent to purchase a product based on a user instruction (e.g. "I am looking for a nightstand with drawers. It should have a nickel finish, and priced lower than $140") through web interactions (e.g. search "nightstand drawers", choose buttons such as "color: modern-nickel-white" or "back to search"). This task is evaluated by average score (percentage of desired attributes covered by the chosen product averaged across all episodes) and success rate (percentage of episodes where the chosen product satisfies all requirements) on 500 test instructions. We formulate Act prompts with actions to search, choose product, choose options, and buy, with ReAct prompts additionally reasoning to determine what to explore, when to buy, and what products options are relevant to the instruction. We compare to an imitation learning (IL) method trained with 1,012 human annotated trajectories, and a imitation + reinforcement learning (IL + RL) method additionally trained with 10,587 training instructions.

Results. ReAct outperforms Act on both ALFWorld (Table 4.3) and Webshop (Table 4.4). On ALFWorld, the best ReAct trial achieves an average success rate of 71%, significantly outperforming the best Act (45%) and BUTLER (37%) trials. In fact, even the worse ReAct trial (48%) beats the best trial of both methods. Moreover,

[5] [204] finetuned a GPT-2 model on 3553 task instances and achieved a much improved performance than BUTLER, but it is trained on all task types, thus not included as a baseline.

Method	Pick	Clean	Heat	Cool	Look	Pick 2	All
Act (best of 6)	88	42	74	67	72	**41**	45
ReAct (avg)	65	39	83	76	55	24	57
ReAct (best of 6)	**92**	58	**96**	86	**78**	**41**	**71**
ReAct-IM (avg)	55	59	60	55	23	24	48
ReAct-IM (best of 6)	62	**68**	87	57	39	33	53
BUTLER$_g$ (best of 8)	33	26	70	76	17	12	22
BUTLER (best of 8)	46	39	74	**100**	22	24	37

Table 4.3: AlfWorld task-specific success rates (%). BUTLER and BUTLER$_g$ results are from Table 4 of [278]. All methods use greedy decoding, except that BUTLER uses beam search.

Method	Score	SR
Act	62.3	30.1
ReAct	**66.6**	**40.0**
IL	59.9	29.1
IL+RL	62.4	28.7
Human Expert	82.1	59.6

Table 4.4: Score and success rate (SR) on Webshop. IL/IL+RL taken from [352].

the advantage of ReAct over Act is consistent across six controlled trials, with relative performance gain ranging from 33% to 90% and averaging 62%. Qualitatively, we saw that, without any thoughts at all, Act fails to correctly decompose goals into smaller subgoals, or loses track of the current state of the environment.

On Webshop, one-shot Act prompting already performs on par with IL and IL+RL methods. With additional sparse reasoning, ReAct achieves significantly better performance, with an absolute 10% improvement over the previous best success rate. By checking examples, we find that ReAct is more likely to identify instruction-relevant products and options by reasoning to bridge the gap between noisy observations and actions (e.g. "For 'space-saving ottoman bench for living room', the item has options '39x18x18inch' and 'blue' and seems good to buy."). However, existing methods are still far from the performance of expert humans (Table 4.4), who perform significantly more product explorations and query re-formulations that are still challenging for prompting-based methods.

On the value of internal reasoning vs. external feedback. To our knowledge, ReAct is the first demonstration of combined reasoning and action using an LLM applied to an interactive environment within a closed-loop system. Perhaps the closest prior work is Inner Monologue (IM), from [120], in which actions from an embodied agent are motivated by an eponymous "inner monologue". However, IM's "inner monologue" is limited to observations of the environment state and what needs to be completed by the agent for the goal to be satisfied. In contrast, the reasoning traces in ReAct for decision making is flexible and sparse, allowing diverse reasoning types (see Section 4.3) to be induced for different tasks.

To demonstrate the differences between ReAct and IM, and to highlight the importance of internal reasoning vs. simple reactions to external feedback, we ran an ablation experiment using a thought pattern composed of IM-like dense external feedback. As can be seen in Table 4.3, ReAct substantially outperforms IM-style prompting (ReAct-IM) (71 vs. 53 overall success rate), with consistent advantages on five out of six tasks. Qualitatively, we observed that ReAct-IM often made mistakes in identifying when subgoals were finished, or what the next subgoal should be, due to a lack of high-level goal decomposition. Additionally, many ReAct-IM trajectories struggled to determine where an item would likely be within the ALFWorld environment, due to a lack of commonsense reasoning. Both shortcomings can be addressed in the ReAct paradigm.

4.6 Incorporating Reasoning and Acting with Learning

ReAct is a general paradigm to use language models, and the ReAct format can be used to both prompt and fine-tune LMs. However, the prompting paradigm has fundamental limitations, as the LM only has limited learning from in-context examples and cannot improve through task experience. In Section 4.4, we have seen initial evidence that ReAct fine-tuning can further improve the task performance. Here, we briefly describe two follow-up directions that improve ReAct agents by either updating its language prompt, or underlying model weights. These two directions will be further conceptually characterized and unified as "learning" actions that update the long-term memory of agents in Chapter 6.

4.6.1 Reflexion: Reasoning to Learn

Reflexion is a novel framework to reinforce language agents not by updating weights, but instead through linguistic feedback. Concretely, Reflexion agents verbally reflect on task feedback signals, then maintain their own reflective text in an episodic memory buffer to induce better decision-making in subsequent trials. Reflexion is flexible enough to incorporate various types (scalar values or free-form language) and sources (external or internally simulated) of feedback signals, and obtains significant improvements over a baseline agent across diverse tasks (sequential decision-making, coding, language reasoning). For example, Reflexion achieves a 91% pass@1 accuracy on the HumanEval coding benchmark, surpassing the previous state-of-the-art GPT-4 that achieves 80%. We also conduct ablation and analysis studies using different feedback signals, feedback incorporation methods, and agent types, and provide insights into how they affect performance. Detailed can be seen in [276].

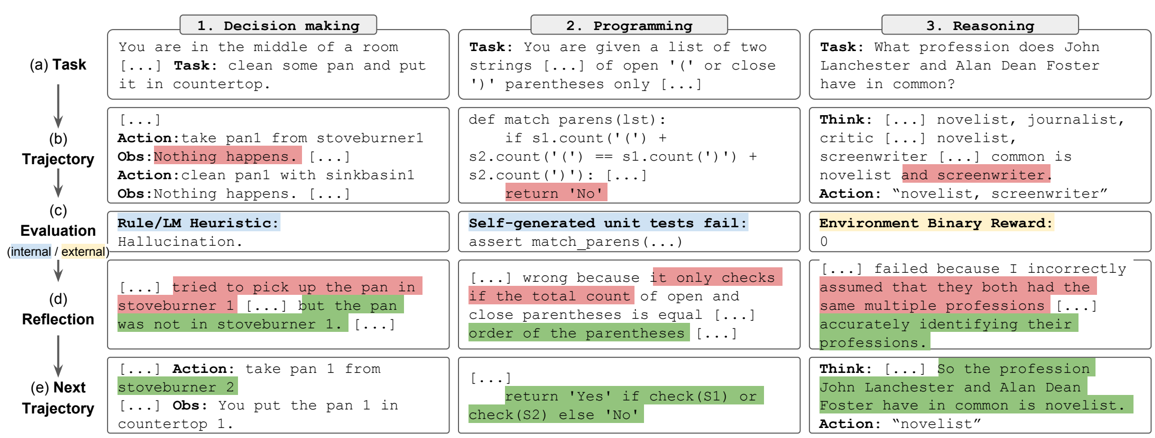


Figure 4.4: Reflexion works on decision-making (ALFWorld [278]), programming (HumanEval [46]), and reasoning (HotpotQA [351]) tasks. Compared to traditional reinforcement learning via back-propagation of scalar feedback, Reflexion can be seen as "verbal reinforcement learning" via reflective reasoning of more general and flexible language feedback.

4.6.2 FireAct: Fine-tuning to Learn

Extending upon the initial ReAct fine-tuning experiments in Section 4.4, FireAct (Fine-tune ReAct) delves into the exploration and argumentation that fine-tuning LMs with agentic data not only equips various base LMs with improved agent functionalities but also serves as a valuable supplement to instruction datasets. It is a novel method to fine-tune LMs as agents by incorporating diverse reasoning and agentic trajectories derived from multiple tasks and prompting techniques. The outcomes demonstrate that moving from traditional instruction tuning to agent tuning allows LMs ranging from Llama2-7B to GPT-3.5 to further enhance their problem-solving capabilities on various tasks including questions answering, math word problem solving and code generation by autonomously choosing to utilize their own knowledge for reasoning or employing external tools with agent actions.

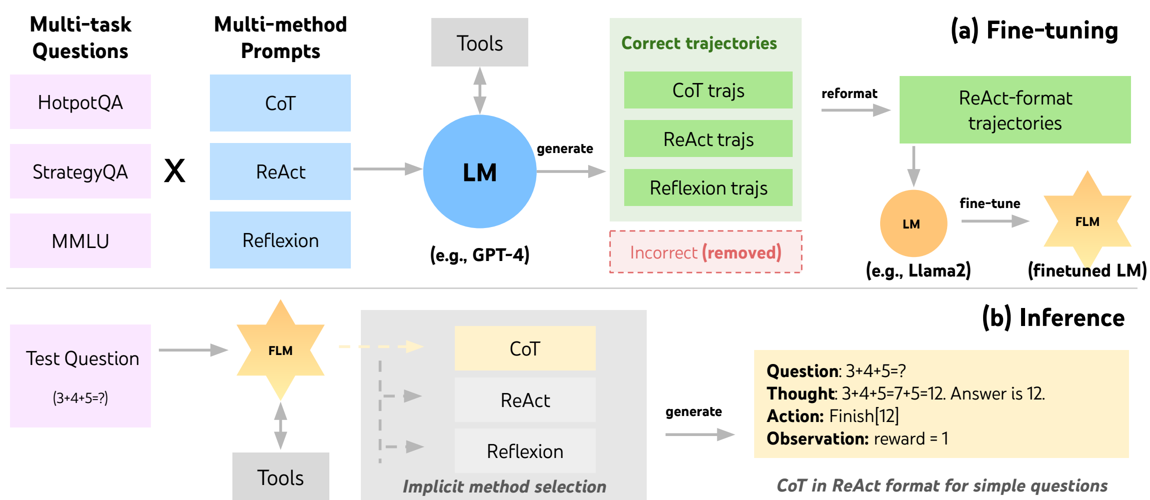


Figure 4.5: Illustration of FireAct. (a) **During fine-tuning**, a large LM (e.g., GPT-4) generates task-solving trajectories based on questions from different datasets and prompts from different methods. The successful trajectories are then converted into the ReAct format to fine-tune a smaller LM. (b) **During inference**, the fine-tuned LM could operate without few-shot prompting, and could implicitly select an prompting method to complete a ReAct trajectory with flexible lengths, adapting to different question complexities. For example, a simple question could be solved using only one thought-action-observation round, without using tools.

4.7 Discussion

ReAct is the first method to generally apply LLMs as agents, and has become the foundation of various language agents deployed to various domains, such as art [294], healthcare [124], robotics [111], education [33], disaster control [56], fact checking [255], networks [99], and autonomous driving [83]. By abstracting actuating, information retrieval, code execution, robotic control, utterance with humans, and various tool usages as external acting, and verbal planning and re-planning, self-reflection, task progress tracking, commonsense deduction, and various belief updates as internal reasoning, and by treating reasoning as augmented actions for agents, ReAct is able to provide a simple and general paradigm for language agents, where reasoning and acting complement each other.

In the next chapter, we will see the other benefit of viewing reasoning as actions — we can readily apply action planning techniques such as tree search to improve reasoning.

Chapter 5

Tree of Thoughts: Building Agents that Reason to Plan

5.1 Introduction

Originally designed to generate text, scaled-up versions of language models (LMs) such as GPT [257, 259, 30, 236] and PaLM [52], have been increasingly capable of performing an ever wider range of tasks requiring mathematical, symbolic, commonsense, and knowledge reasoning. In the last chapter, we have witness that ReAct enabled LMs to even perform general agentic tasks to interact with the world. Perhaps more surprising than all the progress is the fact that underlying all this progress is still the original autoregressive mechanism for generating text, which makes token-level decisions one by one and in a left-to-right fashion. Is such a simple mechanism sufficient for a LM to be built toward a general problem solver? If not, what problems would challenge the current paradigm, and what should be alternative mechanisms?

The literature on human cognition provides some clues to answer these questions. Research on "dual process" models suggests that people have two modes in which they engage with decisions – a fast, automatic, unconscious mode ("System 1") and a slow, deliberate, conscious mode ("System 2") [285, 286, 135, 134]. These two modes have previously been connected to a variety of mathematical models used in machine learning. For example, research on reinforcement learning in humans and other animals has explored the circumstances under which they engage in associative "model free" learning or more deliberative "model based" planning [64]. The simple associative token-level choices of LMs are also reminiscent of "System 1", and thus might benefit from augmentation by a more deliberate "System 2" planning process that (1) maintains and explores diverse alternatives for current choices instead of just picking one, and (2) evaluates its current status and actively looks ahead or backtracks to make more global decisions.

To design such a planning process, we return to the origins of artificial intelligence (and cognitive science), drawing inspiration from the planning processes explored by Newell, Shaw, and Simon starting in the 1950s [218, 219]. Newell and colleagues characterized problem solving [218] as search through a combinatorial problem space, represented as a tree. We thus propose the Tree of Thoughts (ToT) framework for general problem solving with language models. As Figure 5.1 illustrates, while existing methods (detailed below) sample continuous language sequences for problem solving, ToT actively maintains a tree of thoughts, where each *thought* is a coherent language sequence that serves as an intermediate step toward problem solving (Table 5.1). Such a high-level semantic unit allows the LM to self-evaluate the progress different intermediate thoughts make towards solving the problem through a deliberate reasoning process that is also instantiated in language (Figures 5.2,5.4,5.6). This implementation of search heuristics via LM self-evaluation and deliberation is novel, as previous search heuristics are either programmed or learned. Finally, we combine this language-based capability to generate and evaluate diverse thoughts with search algorithms, such as breadth-first search (BFS) or depth-first search (DFS), which allow systematic exploration of the tree of thoughts with lookahead and backtracking.

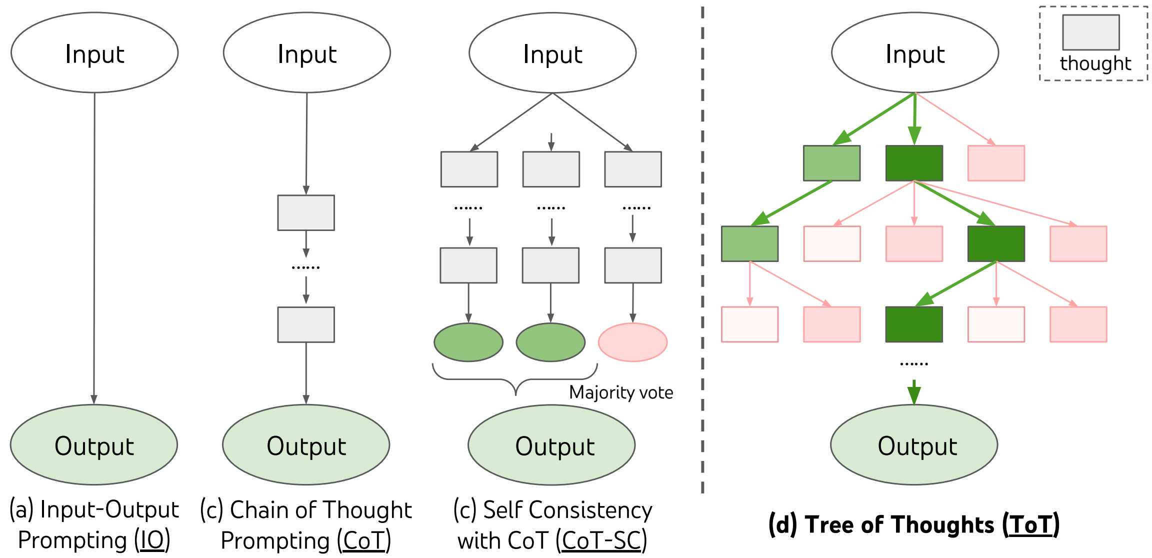


Figure 5.1: Schematic illustrating various approaches to problem solving with LLMs. Each rectangle box represents a *thought*, which is a coherent language sequence that serves as an intermediate step toward problem solving. See concrete examples of how thoughts are generated, evaluated, and searched in Figures 5.2,5.4,5.6.

Empirically, we propose three new problems that challenge existing LM inference methods even with the state-of-the-art language model, GPT-4 [236]: Game of 24, Creative Writing, and Crosswords (Table 5.1). These tasks require deductive, mathematical, commonsense, lexical reasoning abilities, and a way to incorporate systematic planning or search. We show ToT obtains superior results on all three tasks by being general and flexible enough to support different levels of thoughts, different ways to generate and evaluate thoughts, and different search algorithms that adapt to the nature of different problems. We also analyze how such choices affect model performances via systematic ablations and discuss future directions to better train language models and use them towards language agents.

5.2 Related Work

Planning and decision making. Smart planning and decision making are critical to achieving predefined goals. As they are trained on vast amount of world knowledge and human examples, LMs are known to have already absorbed rich commonsense that makes it possible to propose reasonable plans conditioned on problem setting and environmental states [119, 372, 328, 120, 316, 360, 350]. Our proposed ToT approach extends existing planning formulations by considering multiple potentially feasible plans simultaneously at each problem-solving step, and proceeding with the most promising ones. The integration between thought sampling and value feedback organically integrates planning and decision-making mechanisms, enabling effective search inside a solution tree. On the other hand, traditional decision-making procedures usually require training dedicated reward and policy models as in reinforcement learning (for example CHAI [311]), whereas we use the LM itself to provide the value estimates for decision making. RAP [102] is a concurrent work that treats language model reasoning as planning with its internal world model, and proposes a MCTS-based method similar to ToT. However, its tasks are simpler than ours, and its framework lacks the modularity to incorporate different tree search algorithms.

Self-reflection. Using LLMs to assess the viability of their own predictions is becoming an increasingly important procedure in problem solving. [276, 196, 247] introduced the "self-reflection" mechanism, in which LMs provide feedback to their generation candidates. [47] improves LMs code generation accuracy by injecting feedback messages generated by the LM itself based on its code execution results. Similarly, [138] also introduces "critic" or review steps over the actions and states, deciding the next action to take in solving computer operation tasks. Another recent work very relevant to ours is "self-eval guided decoding" [345]. Similar to our method, self-eval decoding also follows a tree-search procedure with leaves sampled from stochastic beam search decoding, which are then evaluated by LLM itself with carefully prepared self-eval prompts. Their approach however, uses the PAL formulation [87] which represents thoughts as codes, which makes it difficult to tackle challenging tasks like creative writing which we consider in this chapter. Our Tree-of-Thought formulation is thus more versatile and handles challenging tasks on which GPT-4 only achieves very low accuracy with standard prompts.

Program-guided LLM generation. Our proposal is also related to recent advancements that organize LM's behavior with systematic procedures [133, 386, 59, 380] or symbolic program guidance. For example, [270] embeds LMs in an algorithmic search procedure to help solve problems like question answering step-by-step, in which the search trees are expanded by relevant paragraphs that might provide answers. This approach however differs from ours in that trees are expanded by sampling external paragraphs instead of the LM's own thoughts, and there is no reflection or voting steps. Another approach, LLM+P [180], goes one step further and delegates the actual planning process to a classical planner.

Classical search methods. Last but not least, our approach can be treated as a modern rendition of classical search methods for problem solving. For example it can be considered as a heuristic search algorithm like A*, in which the heuristic at each search node is provided by the LM's self-assessment. From this perspective, our method is also related to NeuroLogic A*esque decoding [191], which is inspired by A* search but introduces look-ahead heuristics that are efficient for LMs to improve the beam-search or top-k sampling decoding. This method however is constrained to sentence generation tasks, whereas our framework are designed for complex, multi-step problem solving guarded by value feedback.

5.3 Background

We first formalize some existing methods that use large language models for problem-solving, which our approach is inspired by and later compared with. We use p_θ to denote a pre-trained LM with parameters θ, and **lowercase letters** $x, y, z, s, \cdots$ **to denote a language sequence**, i.e. $x = (x[1], \cdots, x[n])$ where each $x[i]$ is a token, so that $p_\theta(x) = \prod_{i=1}^{n} p_\theta(x[i]|x[1...i])$. We use uppercase letters $S, \cdots$ to denote a collection of language sequences.

Input-output (IO) prompting is the most common way to turn a problem input x into output y with LM: $y \sim p_\theta(y|\texttt{prompt}_{IO}(x))$, where $\texttt{prompt}_{IO}(x)$ wraps input x with task instructions and/or few-shot input-output examples. For simplicity, let us denote $p_\theta^{\text{prompt}}(\texttt{output} \mid \texttt{input}) = p_\theta(\texttt{output} \mid \texttt{prompt}(\texttt{input}))$, so that IO prompting can be formulated as $y \sim p_\theta^{IO}(y|x)$.

Chain-of-thought (CoT) prompting [332] was proposed to address cases where the mapping of input x to output y is non-trivial (e.g. when x is a math question and y is the final numerical answer). The key idea is to introduce a chain of *thoughts* $z_1, \cdots, z_n$ to bridge x and y, where each z_i is a coherent language sequence that serves as a meaningful intermediate step toward problem solving (e.g. z_i could be an intermediate equation for math QA). To solve problems with CoT, each thought $z_i \sim p_\theta^{CoT}(z_i \mid x, z_{1\cdots i-1})$ is sampled sequentially, then the output $y \sim p_\theta^{CoT}(y|x, z_{1\cdots n})$. In practice, $[z_{1\cdots n}, y] \sim p_\theta^{CoT}(z_{1\cdots n}, y|x)$ is sampled as a continuous language sequence, and the **decomposition** of thoughts (e.g. is each z_i a phrase, a sentence, or a paragraph) is left ambiguous.

Self-consistency with CoT (CoT-SC) [323] is an ensemble approach that samples k i.i.d. chains of thought: $[z_{1\cdots n}^{(i)}, y^{(i)}] \sim p_\theta^{CoT}(z_{1\cdots n}, y|x)$ $(i = 1 \cdots k)$, then returns the most frequent output: $\arg\max_y \#\{i \mid y^{(i)} = y\}$. CoT-SC improves upon CoT, because there are generally different thought processes for the same problem (e.g. different ways to prove the same theorem), and the output decision can be more faithful by exploring a richer set of thoughts. However, within each chain there is no local exploration of different thought steps, and the "most frequent" heuristic only applies when the output space is limited (e.g. multi-choice QA).

5.4 Tree of Thoughts

> *A genuine problem-solving process involves the repeated use of available information to initiate exploration, which discloses, in turn, more information until a way to attain the solution is finally discovered.*
>
> —— *[218]*

Research on human problem-solving suggests that people search through a combinatorial problem-space – a tree where the nodes represent partial solutions, and the branches correspond to operators that modify them [218, 219]. Which branch to take is determined by heuristics that help to navigate the problem-space and guide the problem-solver towards a solution. This perspective highlights two key shortcomings of existing approaches that use LMs to solve general problems: 1) Locally, they do not explore *different* continuations within a thought process – the branches of the tree. 2) Globally, they do not incorporate any type of planning, lookahead, or backtracking to help evaluate these different options – the kind of heuristic-guided search that seems characteristic of human problem-solving.

To address these shortcomings, we introduce *Tree of Thoughts (ToT)*, a paradigm that allows LMs to explore multiple reasoning paths over thoughts (Figure 5.1(c)). ToT frames any problem as a search over a tree, where each node is a **state** $s = [x, z_{1\cdots i}]$ representing a partial solution with the input and the sequence of thoughts so far. A specific instantiation of ToT involves answering four questions: 1. How to **decompose** the intermediate process into thought steps; 2. How to **generate** potential thoughts from each state; 3. How to heuristically **evaluate** states; 4. What **search** algorithm to use.

1. Thought decomposition While CoT samples thoughts coherently without explicit decomposition, ToT leverages problem properties to design and decompose intermediate thought steps. As Table 5.1 shows, depending on different problems, a thought could be a couple of words (Crosswords), a line of equation (Game of 24), or a whole paragraph of writing plan (Creative Writing). In general, a thought should be "small" enough so that LMs can generate promising and diverse samples (e.g. generating a whole book is usually too "big" to be coherent), yet "big" enough so that LMs can evaluate its prospect toward problem solving (e.g. generating one token is usually too "small" to evaluate).

2. Thought generator $G(p_\theta, s, k)$ Given a tree state $s = [x, z_{1\cdots i}]$, we consider two strategies to generate k candidates for the next thought step:

(a) **Sample** i.i.d. thoughts from a CoT prompt (Creative Writing, Figure 5.4): $z^{(j)} \sim p_\theta^{CoT}(z_{i+1}|s) = p_\theta^{CoT}(z_{i+1}|x, z_{1\cdots i})$ $(j = 1 \cdots k)$. This works better when the thought space is rich (e.g. each thought is a paragraph), and i.i.d. samples lead to diversity;

(b) **Propose** thoughts sequentially using a "propose prompt" (Game of 24, Figure 5.2; Crosswords, Figure 5.6): $[z^{(1)}, \cdots, z^{(k)}] \sim p_\theta^{propose}(z_{i+1}^{(1\cdots k)} \mid s)$. This works better when the thought space is more constrained (e.g. each thought is just a word or a line), so proposing different thoughts in the same context avoids duplication.

3. State evaluator $V(p_\theta, S)$ Given a frontier of different states, the state evaluator evaluates the progress they make towards solving the problem, serving as a *heuristic* for the search algorithm to determine which states to keep exploring and in which order. While heuristics are a standard approach to solving search problems, they are typically either programmed (e.g. DeepBlue [38]) or learned (e.g. AlphaGo [281]). We propose a third alternative, by using the LM to deliberately reason about states. When applicable, such a deliberate heuristic can be more flexible than programmed rules, and more sample-efficient than learned models. Similar to the thought generator, we consider two strategies to evaluate states either independently or together:

(a) **Value** each state independently: $V(p_\theta, S)(s) \sim p_\theta^{value}(v|s)\ \forall s \in S$, where a value prompt reasons about the state s to generate a scalar value v (e.g. 1-10) or a classification (e.g. sure/likely/impossible) that could be heuristically turned into a value. The basis of such evaluative reasoning can vary across problems and thought steps. In this work, we explore evaluation via few *lookahead* simulations (e.g. quickly confirm that 5, 5, 14 can reach 24 via 5 + 5 + 14, or "hot_l" can mean "inn" via filling "e" in "_") plus commonsense (e.g. 1 2 3 are too small to reach 24, or no word can start with "tzxc"). While the former might promote "good" states, the latter could help eliminate "bad" states. Such valuations do not need to be perfect, and only need to be approximately helpful for decision making.

(b) **Vote** across states: $V(p_\theta, S)(s) = \mathbb{1}[s = s^*]$, where a "good" state $s^* \sim p_\theta^{vote}(s^*|S)$ is voted out based on deliberately comparing different states in S in a vote prompt. When problem success is harder to directly value (e.g. passage coherency), it is natural to to instead compare different partial solutions and vote for the most promising one. This is similar in spirit to a "step-wise" self-consistency strategy, i.e. cast "which state to explore" as a multi-choice QA, and use LM samples to vote for it.

For both strategies, we could prompt the LM multiple times to aggregate the value or vote results to trade time/resource/cost for more faithful/robust heuristics.

4. Search algorithm Finally, within the ToT framework, one can plug and play different search algorithms depending on the tree structure. We explore two relatively simple search algorithms and leave more advanced ones (e.g. A* [103], MCTS [31]) for future work:

Algorithm 1 ToT-BFS($x, p_\theta, G, k, V, T, b$)

Require: Input x, LM p_θ, thought generator $G()$ & size limit k, states evaluator $V()$, step limit T, breadth limit b.

$S_0 \leftarrow \{x\}$
for $t = 1, \cdots, T$ **do**
  $S'_t \leftarrow \{[s, z] \mid s \in S_{t-1}, z_t \in \mathrm{G}(p_\theta, s, k)\}$
  $V_t \leftarrow V(p_\theta, S'_t)$
  $S_t \leftarrow \arg\max_{S \subset S'_t, |S|=b} \sum_{s \in S} V_t(s)$
end for
return $G(p_\theta, \arg\max_{s \in S_T} V_T(s), 1)$

Algorithm 2 ToT-DFS($s, t, p_\theta, G, k, V, T, v_{\text{th}}$)

Require: Current state s, step t, LM p_θ, thought generator $G()$ and size limit k, states evaluator $V()$, step limit T, threshold v_{th}

if $t > T$ **then** record output $G(p_\theta, s, 1)$
end if
for $s' \in G(p_\theta, s, k)$ **do** ▷ sorted candidates
  if $V(p_\theta, \{s'\})(s) > v_{\text{thres}}$ **then** ▷ pruning
    DFS($s', t + 1$)
  end if
end for

(a) **Breadth-first search (BFS)** (Algorithm 1) maintains a set of the b most promising states per step. This is used for Game of 24 and Creative Writing where the tree depth is limit ($T \leq 3$), and initial thought steps can be evaluated and pruned to a small set ($b \leq 5$).

(b) **Depth-first search (DFS)** (Algorithm 2) explores the most promising state first, until the final output is reached ($t > T$), or the state evaluator deems it impossible to solve the problem from the current s ($V(p_\theta, \{s\})(s) \leq v_{th}$ for a value threshold v_{th}). In the latter case, the subtree from s is *pruned* to trade exploration for exploitation. In both cases, DFS *backtracks* to the parent state of s to continue exploration.

Conceptually, ToT has several benefits as a method for general problem-solving with LMs: (1) *Generality.* IO, CoT, CoT-SC, and self-refinement can be seen as special cases of ToT (i.e. trees of limited depth and breadth; Figure 5.1). (2) *Modularity.* The base LM, as well as the thought decomposition, generation, evaluation, and search procedures can all be varied independently. (3) *Adaptability.* Different problem properties, LM capabilities, and resource constraints can be accommodated. (4) *Convenience.* No extra training is needed, just a pre-trained LM is sufficient. The next section will show how these conceptual benefits translate to strong empirical performance in different problems.

5.5 Experiments

	Game of 24	**Creative Writing**	**5x5 Crosswords**
Input	4 numbers (4 9 10 13)	4 random sentences	10 clues (h1. presented;..)
Output	An equation to reach 24 (13-9)*(10-4)=24	A passage of 4 paragraphs ending in the 4 sentences	5x5 letters: SHOWN; WIRRA; AVAIL; ...
Thoughts	3 intermediate equations (13-9=4 (left 4,4,10); 10-4=6 (left 4,6); 4*6=24)	A short writing plan (1. Introduce a book that connects...)	Words to fill in for clues: (h1. shown; v5. naled; ...)
#ToT steps	3	1	5-10 (variable)

Table 5.1: Task overview. Input, output, thought examples are in blue.

We propose three tasks that are hard even when sampling from the state-of-the-art language model, GPT-4 [236], using standard IO prompting or chain-of-thought (CoT) prompting. We show how deliberate search in trees of thoughts (ToT) produces better results, and more importantly, interesting and promising new ways to use language models to solve problems requiring search or planning. Unless otherwise stated, we perform experiments using a Chat Completion mode GPT-4[1] with a sampling temperature of 0.7.

[1]Experiments were done between May 5-16, 2023.

5.5.1 Game of 24

Game of 24 is a mathematical reasoning challenge, where the goal is to use 4 numbers and basic arithmetic operations (+-*/) to obtain 24. For example, given input "4 9 10 13", a solution output could be "(10 - 4) * (13 - 9) = 24".

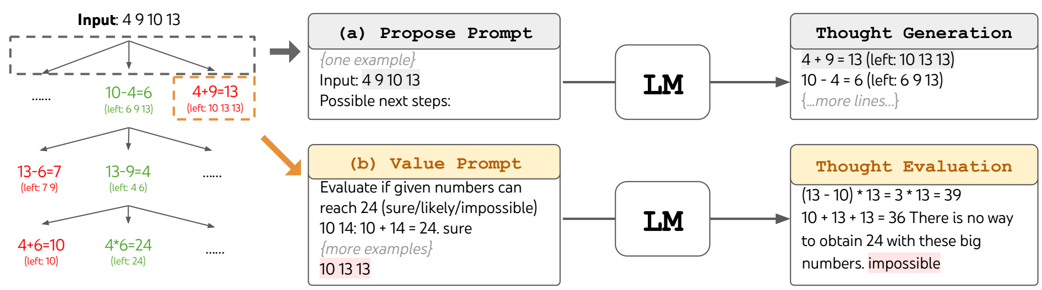


Figure 5.2: ToT in a game of 24. The LM is prompted for (a) thought generation and (b) valuation.

Task Setup We scrape data from 4nums.com, which has 1,362 games that are sorted from easy to hard by human solving time, and use a subset of relatively hard games indexed 901-1,000 for testing. For each task, we consider the output as success if it is a valid equation that equals 24 and uses the input numbers each exactly once. We report the success rate across 100 games as the metric.

Baselines We use a standard input-output (IO) prompt with 5 in-context examples. For chain-of-thought (CoT) prompting, we augment each input-output pair with 3 intermediate equations, each operating on two remaining numbers. For example, given input "4 9 10 13", the thoughts could be "13 - 9 = 4 (left: 4 4 10); 10 - 4 = 6 (left: 4 6); 4 * 6 = 24 (left: 24)". For each game, we sample IO and CoT prompting for 100 times for average performance. We also consider a CoT self-consistency baseline, which takes the majority output from 100 CoT samples, and an iterative-refine approach on top of an IO sample for at most 10 iterations. At each iteration, the LM is conditioned on all previous history to "reflect on your mistakes and generate a refined answer" if

Method	Success
IO prompt	7.3%
CoT prompt	4.0%
CoT-SC (k=100)	9.0%
ToT (ours) (b=1)	45%
ToT (ours) (b=5)	**74%**
IO + Refine (k=10)	27%
IO (best of 100)	33%
CoT (best of 100)	49%

Table 5.2: Game of 24 Results.

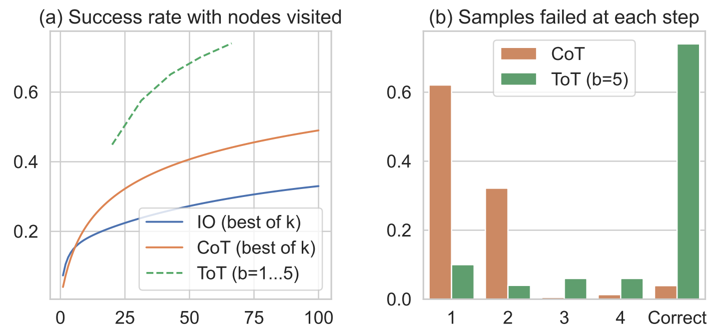


Figure 5.3: Game of 24 (a) scale analysis & (b) error analysis.

the output is incorrect. Note that it uses groundtruth feedback signals about equation correctness.

ToT Setup To frame Game of 24 into ToT, it is natural to decompose the thoughts into 3 steps, each an intermediate equation. As shown in Figure 5.2(a), at each tree node, we exact the remaining numbers and prompt the LM to propose some possible next steps. The same "propose prompt" is used for all 3 thought steps, though it only has one example with 4 input numbers. We perform a breadth-first search (BFS) in ToT, where at each step we keep the best $b = 5$ candidates. To perform deliberate BFS in ToT, as shown in Figure 5.2(b), we prompt LM to evaluate each thought candidate as "sure/maybe/impossible" with regard to reaching 24. The aim is to promote correct partial solutions that can be verdicted within few lookahead trials, and eliminate impossible partial solutions based on "too big/small" commonsense, and keep the rest "maybe". We sample values 3 times for each thought.

Results As shown in Table 5.2, IO, CoT, and CoT-SC prompting methods perform badly on the task, achieving only 7.3%, 4.0%, and 9.0% success rates. In contrast, ToT with a breadth of $b = 1$ already achieves a success rate of 45%, while $b = 5$ achieves 74%. We also consider an oracle setup for IO/CoT, by calculating the success rate using best of k samples ($1 \leq k \leq 100$). To compare IO/CoT (best of k) with ToT, we consider calculating the tree nodes visited per task in ToT across $b = 1 \cdots 5$, and map the 5 success rates in Figure 5.3(a), treating IO/CoT (best of k) as visiting k nodes in a bandit. Not surprisingly, CoT scales better than IO, and best of 100 CoT samples achieve a success rate of 49%, but still much worse than exploring more nodes in ToT ($b > 1$).

Error analysis Figure 5.3(b) breaks down at which step CoT and ToT samples fail the task, i.e. the thought (in CoT) or all b thoughts (in ToT) are invalid or impossible to reach 24. Notably, around 60% of CoT samples already failed the task after generating the first step, or equivalently, the first three words (e.g. "$4 + 9$"). This highlights the issues with direct left-to-right decoding.

5.5.2 Creative writing

Next, we invent a creative writing task where the input is 4 random sentences and the output should be a coherent passage with 4 paragraphs that end in the 4 input sentences respectively. Such a task is open-ended and exploratory, and challenges creative thinking as well as high-level planning.

Task setup We sample random sentences from randomwordgenerator.com to form 100 inputs, and there is no groundtruth passage for each input constraint. As we find that GPT-4 can follow the input constraints most of the time, we focus on evaluating passage coherency in two ways: using a GPT-4 zero-shot prompt to provide a 1-10 scalar score, or using human judgments to compare pairs of outputs from different methods. For the former, we sample 5 scores and average them for each task output, and we find these 5 scores usually consistent, with a standard deviation of around 0.56 on average across outputs. For the latter, we employ a subset of the authors in a blind study to compare the coherency of CoT vs. ToT generated passage pairs, where the order of passages is random flipped over 100 inputs.

Baselines Given the creative nature of the task, both IO and CoT prompts are zero-shot. While the former prompts the LM to directly generate a coherent passage given input constraints, the latter prompts the LM to first make a brief plan then write the passage, i.e. the plan serves as the intermediate thought step. We generate 10 IO and CoT samples per task. We also consider an iterative-refine ($k \leq 5$) method on top of a random IO sample for each task, where the LM is conditioned on input constraints and the last generated passage to decide if the passage is already "perfectly coherent", and if not generate a refined one.

ToT setup We build a ToT with depth 2 (and only 1 intermediate thought step) — the LM first generates $k = 5$ plans and votes for the best one (Figure 5.4), then similarly generate $k = 5$ passages based on the best plan then vote for the best one. Here the breadth limit $b = 1$, as only one choice is kept per step. A simple zero-shot vote prompt ("analyze choices below, then conclude which is most promising for the instruction") is used to sample 5 votes at both steps.

Results Figure 5.5(a) shows average GPT-4 scores across 100 tasks, where ToT (7.56) is deemed to generate more coherent passages than IO (6.19) and CoT (6.93) on average. While such an automatic metric might be noisy, Figure 5.5(b) confirms the finding by showing that humans prefer ToT over CoT in 41 out of 100 passage pairs, while only prefer CoT over ToT in 21 (other 38 pairs are found "similarly coherent"). Lastly, iterative-refine is more effective on this natural language task, where it improves IO coherency score from 6.19 to 7.67, and ToT coherency score from 7.56 to 7.91. We believe it could be thought of as a third approach to thought

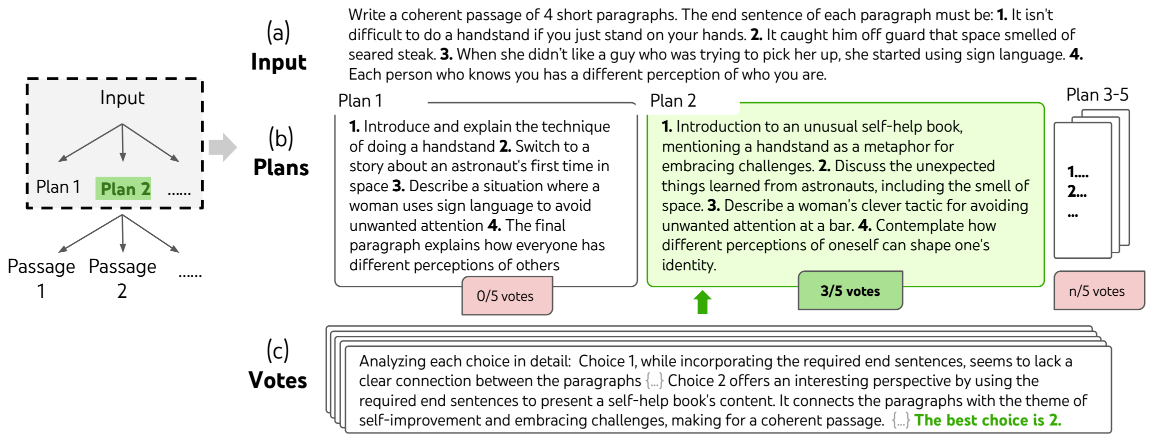


Figure 5.4: A step of deliberate search in a randomly picked Creative Writing task. Given the input, the LM samples 5 different plans, then votes 5 times to decide which plan is best. The majority choice is used to consequently write the output passage with the same sample-vote procedure.

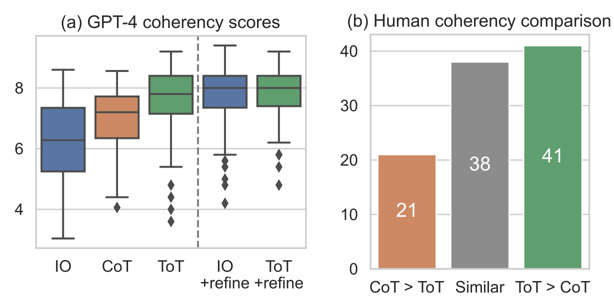


Figure 5.5: Creative Writing results.

Method	Success Rate (%)		
	Letter	**Word**	**Game**
IO	38.7	14	0
CoT	40.6	15.6	1
ToT (ours)	**78**	**60**	**20**
+best state	82.4	67.5	35
-prune	65.4	41.5	5
-backtrack	54.6	20	5

Table 5.3: Mini Crosswords results.

generation in the ToT framework, where new thoughts can arise from refining old thoughts instead of i.i.d. or sequentially generated.

5.5.3 Mini crosswords

In Game of 24 and Creative Writing, ToT is relatively shallow — at most 3 thought steps are needed to reach the final output. Here we explore 5×5 mini crosswords as a harder search problem involving natural language. Again, the goal is not just to solve the task, as more general crosswords can be readily solved with specialized NLP

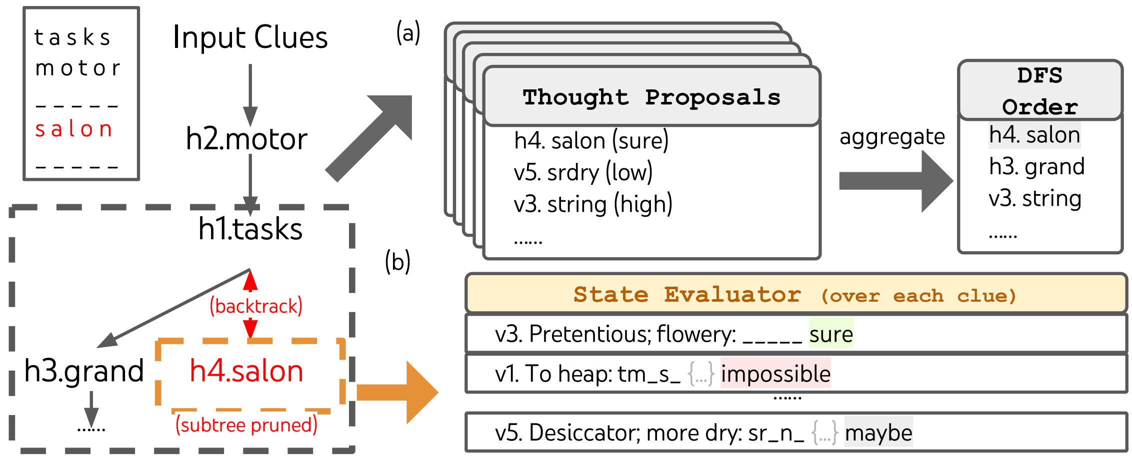


Figure 5.6: In Mini Crosswords, (a) how thoughts are proposed and aggregated in a priority queue for depth-first search (DFS), and (b) how a state is evaluated based on the possibility of filling in each remaining word clue, and pruned if any remaining clue is deemed not possible to fill by the LM. Then DFS backtracks to the parent state and explore the next promising thought for clue.

pipelines [313] that leverages large-scale retrieval instead of LM. Rather, we aim to explore the limit of LM as a general problem solver that explores its own thoughts and guides its own exploration with deliberate reasoning as heuristics.

Task setup We scrape data from GooBix, which contains 156 games of 5×5 mini crosswords. As we observe adjacent games contain similar clues, we use 20 games with indices $1, 6, \cdots, 91, 96$ for testing, and games $136, 141, 146, 151, 156$ for prompting. For each task, the input describes the 5 horizontal clues and 5 vertical clues, and the output should be a board of $5 \times 5 = 25$ letters to solve the crosswords. For evaluation, we consider three levels of success: the portion of correct letters (25 per game), words (10 per game), and games.

Baselines We provide 5 example input-output pairs in the IO prompt, and in the CoT prompt additionally include intermediate words in the order h1..5 then v1..5. We run each prompt for 10 samples and average the results.

ToT setup We leverage a depth-first search (Algorithm 2) that keeps exploring the most promising subsequent word clue until the state is no longer promising, then backtrack to the parent state to explore alternative thoughts. To make search tractable, subsequent thoughts are constrained not to change any filled words or letters, so that the ToT has at most 10 intermediate steps. For thought generation, at each state we translate all existing thoughts (e.g. "h2.motor; h1.tasks" for the state in Figure 5.6(a)) into letter constraints for remaining clues (e.g. "v1.To heap: tm___;...") and prompt a proposal prompt 5 times to come up with candidates for where and what to fill in the next word. Importantly, we also prompt the LM to give a confidence level for different thoughts, and aggregate these across proposals to obtain a sorted list of next thoughts to explore (Figure 5.6(a)). For state evaluations, we similarly translate each state into letter constraints for remaining clues, then evaluate for each clue if it is possible to fill given the constraints. If any remaining clue is deemed "impossible" to fill in (e.g. "v1. To heap: tm_s_"), then the exploration of the state's subtree is pruned and DFS backtracks to its parent to explore the next promising thought. We limit DFS search steps to 100, and simply render the deepest explored state (the first explored one if multiple) into the final output.

Results As shown in Table 5.3, IO and CoT prompting methods perform poorly with a word-level success rate less than 16%, while ToT significantly improves all metrics, achieving a word-level success rate of 60% and solving 4 out of 20 games. Such an improvement is not surprising, given IO and CoT lack mechanisms to try different clues, make changes to decisions, or backtrack.

Oracle and ablation studies When outputting from the oracle best DFS state (instead of the heuristically determined best state) per task, ToT performance is even higher and actually solves 7/20 games (Table 5.3, "+best state"), indicating our simple output heuristics can be readily improved. Interestingly, sometimes when the crosswords game is actually solved, the state evaluator might still deem some words as "impossible" and prune — possibly because 5×5 crosswords by design have some rare or obselete words that GPT-4 cannot recognize[2]. Given the state evaluation as a pruning heuristic is imperfect, we also explore ablating the pruning, and find the performance generally worse (Table 5.3, "-prune"). However, it could actually find the correct solution for 4/20 games (though only outputting 1 via heuristic), 3 of which are games ToT+pruning cannot solve within 100 steps. Thus, better heuristics for DFS pruning are critical for problem solving in this case. Lastly, we confirm the importance of backtracking by running an ablation that keeps filling the most promising clue for at most 20 steps, allowing overwrites. This is similar to a "greedy" BFS search with breadth limit of $b = 1$, and performs poorly with a word level success of only 20% (Table 5.3, "-backtrack").

5.6 Discussion

The associative "System 1" of LMs can be beneficially augmented by a "System 2" based on searching a tree of possible paths to the solution to a problem. The Tree of Thoughts framework provides a way to translate classical insights about problem-solving into actionable methods for contemporary LMs. At the same time, LMs address a weakness of these classical methods, providing a way to solve complex problems that are not easily formalized, such as creative writing. We see this intersection of LMs with classical approaches to AI as an exciting direction, which leads to the next chapter where we go beyond leveraging one classical algorithm (tree search) to improve one capability (deliberate reasoning and planning) of language agents, but to leverage the classical research field of cognitive architectures to systematize and consolidate the whole research field of language agents.

[2]For example, "agend" is an obsolete form of "agendum", but GPT-4 deems it a typo for "agenda". External retrieval or web interaction could augment LM for problem solving under knowledge uncertainty.

Part III

Framework

Chapter 6

CoALA: Cognitive Architectures for Language Agents

6.1 Introduction

Language agents, as shown in all previous chapters, are an emerging class of artificial intelligence (AI) systems that use large language models (LLMs) to interact with the world. They apply the latest advances in LLMs to the existing field of agent design [266]. Intriguingly, this synthesis offers benefits for both fields. On one hand, LLMs possess limited knowledge and reasoning capabilities. Language agents mitigate these issues by connecting LLMs to internal memory and environments, grounding them to existing knowledge or external observations. On the other hand, traditional agents often require handcrafted rules [336] or reinforcement learning [295], making generalization to new environments challenging [153]. Language agents leverage commonsense priors present in LLMs to adapt to novel tasks, reducing the dependence on human annotation or trial-and-error learning.

While the earliest agents used LLMs to directly select or generate actions [Figure 6.1B; 7, 119], more recent agents additionally use them to reason [360], plan [102, 358], and manage long-term memory [242, 314] to improve decision-making. This latest generation of *cognitive* language agents use remarkably sophisticated internal processes (Figure 6.1C). Today, however, individual works use custom terminology to describe these processes (such as 'tool use', 'grounding', 'actions'), making it difficult to compare different agents, understand how they are evolving over time, or build new agents with clean and consistent abstractions.

In order to establish a conceptual framework organizing these efforts, we draw parallels with two ideas from the history of computing and artificial intelligence (AI): *production systems* and *cognitive architectures*. Production systems generate a set of outcomes by iteratively applying rules [219]. They originated as string manipulation systems – an analog of the problem that LLMs solve – and were subsequently adopted by the AI community to define systems capable of complex, hierarchically structured behaviors [217]. To do so, they were incorporated into cognitive architectures that specified control flow for selecting, applying, and even generating new productions [150, 149, 144]. We suggest a meaningful analogy between production systems and LLMs: just as productions indicate possible ways to modify strings, LLMs define a distribution over changes or additions to text. This further suggests that controls from cognitive architectures used with production systems might be equally applicable to transform LLMs into language agents.

Thus, we propose **Co**gnitive **A**rchitectures for **L**anguage **A**gents (CoALA), a conceptual framework to characterize and design general purpose language agents. CoALA organizes agents along three key dimensions: their *information storage* (divided into working and long-term memories); their *action space* (divided into internal and external actions); and their *decision-making procedure* (which is structured as an interactive loop with planning and execution). Through these three concepts (memory, action, and decision-making), we show CoALA can neatly express a large body of existing agents and identify underexplored directions to develop new ones.

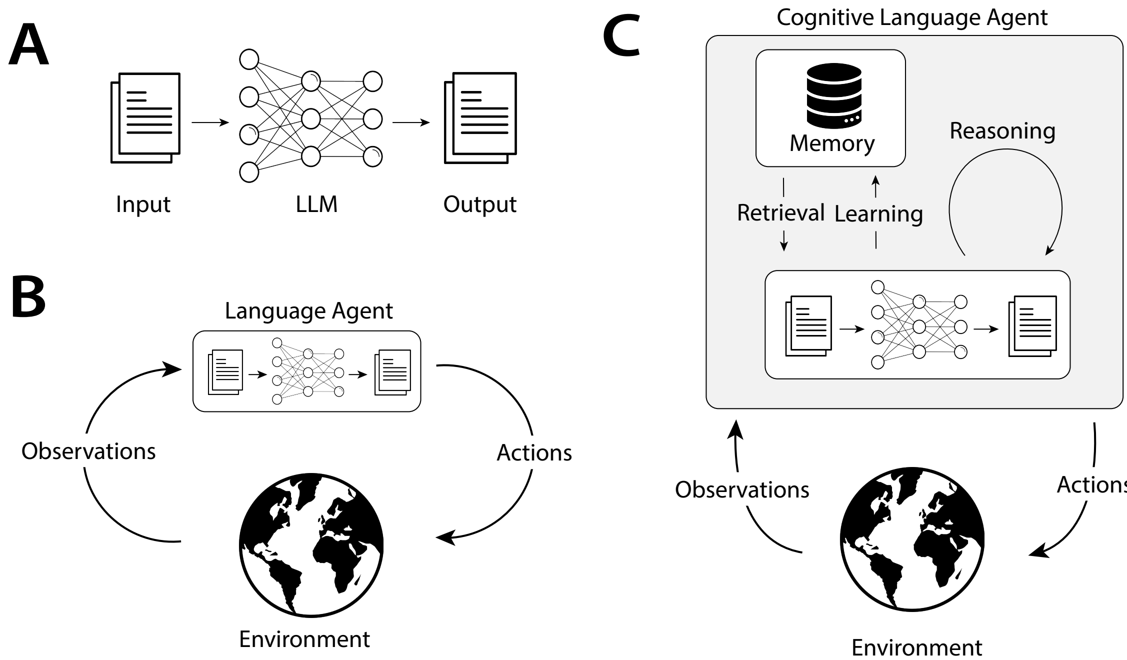


Figure 6.1: Different uses of large language models (LLMs). **A**: In natural language processing (NLP), an LLM takes text as input and outputs text. **B**: *Language agents* [7, 120] place the LLM in a direct feedback loop with the external environment by transforming observations into text and using the LLM to choose actions. **C**: *Cognitive* language agents [360, 276, 314] additionally use the LLM to manage the agent's internal state via processes such as learning and reasoning. In this work, we propose a blueprint to structure such agents.

Notably, while several recent papers propose conceptual architectures for general intelligence [159, 201] or empirically survey language models and agents [203, 333, 315], this chapter combines elements of both: we propose a theoretical framework *and* use it to organize diverse empirical work. This grounds our theory to existing practices and allows us to identify both short-term and long-term directions for future work.

The plan for the rest of the chapter is as follows. We first introduce production systems and cognitive architectures (Section 6.2) and show how these recent developments in LLMs and language agents recapitulate these historical ideas (Section 6.3). Motivated by these parallels, Section 6.4 introduces the CoALA framework and uses it to survey existing language agents. Section 6.5 provides a deeper case study of several prominent agents. Section 6.6 suggests actionable steps to construct future language agents, while Section 6.7 highlights open questions in the broader arc of cognitive science and AI. Finally, Section 3.7 concludes. Readers interested in applied agent design may prioritize Sections 4-6.

6.2 Background: From Strings to Symbolic AGI

We first introduce production systems and cognitive architectures, providing a historical perspective on cognitive science and artificial intelligence: beginning with theories of logic and computation [249], and ending with attempts to build symbolic artificial general intelligence [217]. We then briefly introduce language models and language agents. Section 6.3 will connect these ideas, drawing parallels between production systems and language models.

6.2.1 Production systems for string manipulation

In the first half of the twentieth century, a significant line of intellectual work led to the reduction of mathematics [335] and computation [54, 306] to symbolic manipulation. Production systems are one such formalism. Intuitively, production systems consist of a set of rules, each specifying a precondition and an action. When the precondition is met, the action can be taken. The idea originates in efforts to characterize the limits of computation. [249] proposed thinking about arbitrary logical systems in these terms, where formulas are expressed as strings and the conclusions they license are identified by production rules (as one string "produces" another). This formulation was subsequently shown to be equivalent to a simpler string rewriting system. In such a system, we specify rules of the form

$$X\,Y\,Z \to X\,W\,Z$$

indicating that the string XYZ can be rewritten to the string XWZ. String rewriting plays a significant role in the theory of formal languages, in the form of Chomsky's phrase structure grammar [51].

6.2.2 Control flow: From strings to algorithms

By itself, a production system simply characterizes the set of strings that can be generated from a starting point. However, they can be used to specify algorithms if we impose *control flow* to determine which productions are executed. For example, Markov algorithms are production systems with a priority ordering [198]. The following algorithm implements division-with-remainder by converting a number written as strokes $|$ into the form $Q * R$, where Q is the quotient of division by 5 and R is the remainder:

$$\begin{aligned} *||||| &\rightarrow |* \\ * &\overset{\bullet}{\rightarrow} * \\ &\rightarrow * \end{aligned}$$

where the priority order runs from top to bottom, productions are applied to the first substring matching their preconditions when moving from left to right (including the empty substring, in the last production), and $\overset{\bullet}{\rightarrow}$ indicates the algorithm halts after executing the rule. The first rule effectively "subtracts" five if possible; the second handles the termination condition when no more subtraction is possible; and the third handles the empty substring input case. For example, given the input 11, this would yield the sequence of productions $*||||||||||| \rightarrow |*|||||| \rightarrow ||*| \overset{\bullet}{\rightarrow} ||*|$ which is interpreted as 2 remainder 1. Simple productions can result in complex behavior – Markov algorithms can be shown to be Turing complete.

6.2.3 Cognitive architectures: From algorithms to agents

Production systems were popularized in the AI community by Allen Newell, who was looking for a formalism to capture human problem solving [214, 219]. Productions were generalized beyond string rewriting to logical operations: *preconditions* that could be checked against the agent's goals and world state, and *actions* that should be taken if the preconditions were satisfied. In their landmark book *Human Problem Solving* [219], Allen Newell and Herbert Simon gave the example of a simple production system implementing a thermostat agent:

$$
\begin{aligned}
(\text{temperature} > 70^\circ) \wedge (\text{temperature} < 72^\circ) &\rightarrow \text{stop} \\
\text{temperature} < 32^\circ &\rightarrow \text{call for repairs; turn on electric heater} \\
(\text{temperature} < 70^\circ) \wedge (\text{furnace off}) &\rightarrow \text{turn on furnace} \\
(\text{temperature} > 72^\circ) \wedge (\text{furnace on}) &\rightarrow \text{turn off furnace}
\end{aligned}
$$

Following this work, production systems were adopted by the AI community. The resulting agents contained large production systems connected to external sensors, actuators, and knowledge bases – requiring correspondingly sophisticated control flow. AI researchers defined "cognitive architectures" that mimicked human cognition – explicitly instantiating processes such as perception, memory, and planning [2] to achieve flexible, rational, real-time behaviors [293, 215, 216, 11]. This led to applications from psychological modeling to robotics, with hundreds of architectures and thousands of publications (see [144] for a recent survey).

A canonical example is the Soar architecture (Fig. 6.2A). Soar stores productions in long-term memory and executes them based on how well their preconditions match working memory (Fig. 6.2B). These productions specify actions that modify the contents of working and long-term memory. We next provide a brief overview of Soar and refer readers to [149, 148] for deeper introductions.

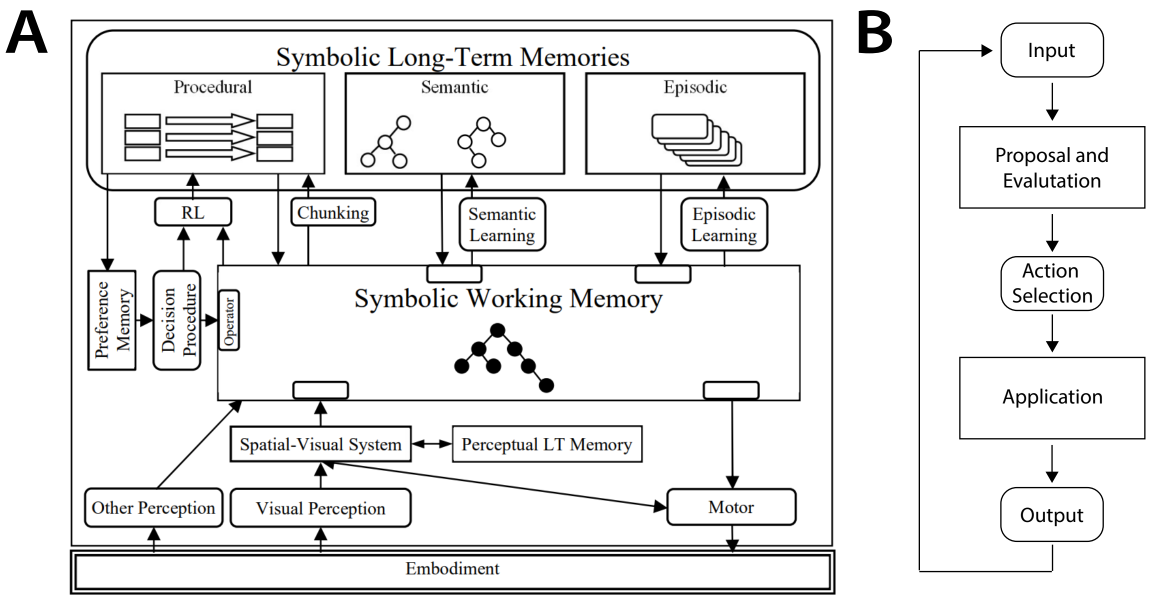


Figure 6.2: Cognitive architectures augment a production system with sensory groundings, long-term memory, and a decision procedure for selecting actions. **A**: The Soar architecture, reproduced with permission from [149]. **B**: Soar's decision procedure uses productions to select and implement actions. These actions may be *internal* (such as modifying the agent's memory) or *external* (such as a motor command).

Memory Building on psychological theories, Soar uses several types of memory to track the agent's state [14]. *Working memory* [17] reflects the agent's current circumstances: it stores the agent's recent perceptual input, goals, and results from intermediate, internal reasoning. *Long term memory* is divided into three distinct types. *Procedural* memory stores the production system itself: the set of rules that can be applied to working memory to determine the agent's behavior. *Semantic* memory stores facts about the world [179], while *episodic* memory stores sequences of the agent's past behaviors [233].

Grounding Soar can be instantiated in simulations [298, 132] or real-world robotic systems [152]. In embodied contexts, a variety of sensors stream perceptual input into working memory, where it is available for decision-making. Soar agents can also be equipped with actuators, allowing for physical actions and interactive learning via language [208, 207, 139].

Decision making Soar implements a decision loop that evaluates productions and applies the one that matches best (Fig. 6.2B). Productions are stored in long-term procedural memory. During each decision cycle, their preconditions are checked against the agent's working memory. In the *proposal and evaluation* phase, a set of productions is used to generate and rank a candidate set of possible actions.[1] The best action is then chosen.[2] Another set of productions is then used to implement the action – for example, modifying the contents of working memory or issuing a motor command.

Learning Soar supports multiple modes of learning. First, new information can be stored directly in long-term memory: facts can be written to semantic memory, while experiences can be written to episodic memory [66]. This information can later be retrieved back into working memory when needed for decision-making. Second, behaviors can be modified. Reinforcement learning [295] can be used to up-weight productions that have yielded good outcomes, allowing the agent to learn from experience [213]. Most remarkably, Soar is also capable of writing new productions into its procedural memory [151] – effectively updating its source code.

Cognitive architectures were used broadly across psychology and computer science, with applications including robotics [152], military simulations [132, 298], and intelligent tutoring [141]. Yet they have become less popular in the AI community over the last few decades. This decrease in popularity reflects two of the challenges involved in such systems: they are limited to domains that can be described by logical predicates and require many pre-specified rules to function.

Intriguingly, LLMs appear well-posed to meet these challenges. First, they operate over arbitrary text, making them more flexible than logic-based systems. Second, rather than requiring the user to specify productions, they learn a distribution over

[1] In more detail, Soar divides productions into two types: "operators," which we refer to as actions, and "rules" which are used to propose, evaluate, and execute operators.

[2] If no actions are valid, or multiple actions tie, then an *impasse* occurs. Soar creates a subgoal to resolve the impasse, resulting in hierarchical task decomposition. We refer the reader to [149] for a more detailed discussion.

productions via pre-training on an internet corpus. Recognizing this, researchers have begun to use LLMs within cognitive architectures, leveraging their implicit world knowledge [340] to augment traditional symbolic approaches [140, 262]. Here, we instead import principles from cognitive architecture to guide the design of LLM-based agents.

6.3 Connections between Language Models and Production Systems

Based on their common origins in processing strings, there is a natural analogy between production systems and language models. We develop this analogy, then show that prompting methods recapitulate the algorithms and agents based on production systems. The correspondence between production systems and language models motivates our use of cognitive architectures to build language agents, which we introduce in Section 6.4.

6.3.1 Language models as probabilistic production systems

In their original instantiation, production systems specified the set of strings that could be generated from a starting point, breaking this process down into a series of string rewriting operations. Language models also define a possible set of expansions or modifications of a string – the prompt provided to the model.[3]

For example, we can formulate the problem of completing a piece of text as a production. If X is the prompt and Y the continuation, then we can write this as the production $X \to X\ Y$.[4] We might want to allow multiple possible continuations, in

[3] In this work, we focus on autoregressive LLMs which are typically used for language agents. However, bidirectional LLMs such as BERT [67] can be seen in a similar light: they define a distribution over *in-filling* productions.

[4] Alternatively, we can treat the prompt as input and take the output of the LLM as the next state, represented by the production $X \to Y$ – a more literal form of rewriting.

which case we have $X \to X\ Y_i$ for some set of Y_i. LLMs assign a *probability* to each of these completions. Viewed from this perspective, the LLM defines a probability distribution over *which productions to select* when presented with input X, yielding a distribution $P(Y_i|X)$ over possible completions [69]. LLMs can thus be viewed as probabilistic production systems that sample a possible completion each time they are called, e.g., $X \rightsquigarrow X\,Y$.

This probabilistic form offers both advantages and disadvantages compared to traditional production systems. The primary disadvantage of LLMs is their inherent opaqueness: while production systems are defined by discrete and human-legible rules, LLMs consist of billions of uninterpretable parameters. This opaqueness – coupled with inherent randomness from their probabilistic formulation – makes it challenging to analyze or control their behaviors [262, 310]. Nonetheless, their scale and pre-training provide massive advantages over traditional production systems. LLMs pre-trained on large-scale internet data learn a remarkably effective prior over string completions, allowing them to solve a wide range of tasks out of the box [119].

6.3.2 Prompt engineering as control flow

The weights of an LLM define a prioritization over output strings (completions), conditioned by the input string (the prompt). The resulting distribution can be interpreted as a task-specific prioritization of productions – in other words, a simple control flow. Tasks such as question answering can be formulated directly as an input string (the question), yielding conditional distributions over completions (possible answers).

Early work on few-shot learning [30] and prompt engineering [332, 142, 346] found that the LLM could be further biased towards high-quality productions by pre-processing the input string. These simple manipulations – typically concatenating additional text to the input – can themselves be seen as productions, meaning that

Prompting Method	Production Sequence
Zero-shot	$Q \overset{\text{LLM}}{\rightsquigarrow} Q\ A$
Few-shot	$Q \longrightarrow Q_1\ A_1\ Q_2\ A_2\ Q \overset{\text{LLM}}{\rightsquigarrow} Q_1\ A_1\ Q_2\ A_2\ Q\ A$
Retrieval Augmented Generation	$Q \xrightarrow{\text{Wiki}} Q\,O \overset{\text{LLM}}{\rightsquigarrow} Q\,O\,A$
Socratic Models	$Q \overset{\text{VLM}}{\rightsquigarrow} Q\,O \overset{\text{LLM}}{\rightsquigarrow} Q\,O\,A$
Self-Critique	$Q \overset{\text{LLM}}{\rightsquigarrow} Q\,A \overset{\text{LLM}}{\rightsquigarrow} Q\,A\,C \overset{\text{LLM}}{\rightsquigarrow} Q\,A\,C\,A$

Table 6.1: Conceptual diagram illustrating how prompting methods manipulate the input string before generating completions. Q = question, A = answer, O = observation, C = critique, and $\rightsquigarrow$ denotes sampling from a stochastic production. These pre-processing manipulations – which can employ other models such as vision-language models (VLMs), or even the LLM itself – can be seen as productions. Prompting methods thus define a *sequence* of productions.

these methods define a sequence of productions (Table 6.1). Later work extended these approaches to dynamic, context-sensitive prompts: for example, selecting few-shot examples that are maximally relevant to the input [185] or populating a template with external observations from video [368] or databases [162]. For a survey of such prompting techniques, see [186].

Subsequent work used the LLM itself as a pre-processing step, eliciting targeted reasoning to foreground a particular aspect of the problem [18, 130, 84, 196, 268, 138, 140] or generate intermediate reasoning steps [296, 61, 358] before returning an answer. *Chaining* multiple calls to an LLM [342, 343, 69] allows for increasingly complicated algorithms (Fig. 6.3).

6.3.3 Towards cognitive language agents

Language agents move beyond pre-defined prompt chains and instead place the LLM in a feedback loop with the external environment (Fig. 6.1B). These approaches first transform multimodal input into text and pass it to the LLM. The LLM's output is then

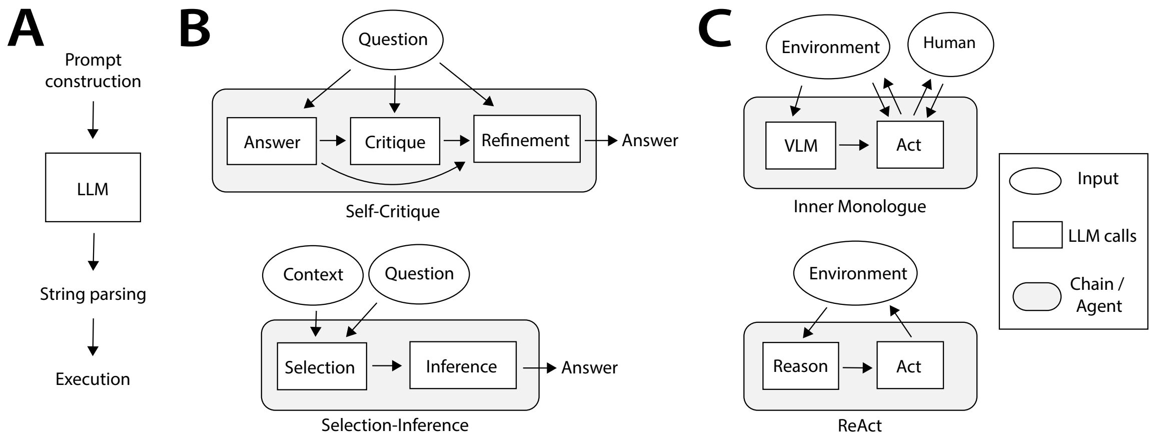


Figure 6.3: From language models to language agents. **A**: Basic structure of an LLM call. Prompt construction selects a template and populates it with variables from working memory. After calling the LLM, the string output is parsed into an action space and executed. An LLM call may result in one or more actions – for example, returning an answer, calling a function, or issuing motor commands. **B**: *Prompt chaining* techniques such as Self-Critique [323] or Selection-Inference [61] use a pre-defined sequence of LLM calls to generate an output. **C**: *Language agents* such as Inner Monologue [120] and ReAct [360] instead use an interactive feedback loop with the external environment. Vision-language models (VLMs) can be used to translate perceptual data into text for the LLM to process.

parsed and used to determine an external action (Fig. 6.3C). Early agents interfaced the LLM directly with the external environment, using it to produce high-level instructions based on the agent's state [7, 120, 63]. Later work developed more sophisticated language agents that use the LLM to perform intermediate reasoning before selecting an action [360]. The most recent agents incorporate sophisticated learning strategies such as reflecting on episodic memory to generate new semantic inferences [276] or modifying their program code to generate procedural knowledge [314], using their previous experience to adapt their future behaviors.

These *cognitive* language agents employ nontrivial LLM-based reasoning and learning (Fig. 6.1C). Just as cognitive architectures were used to structure production systems' interactions with agents' internal state and external environments, we suggest that they can help design LLM-based cognitive agents. In the remainder of the chapter,

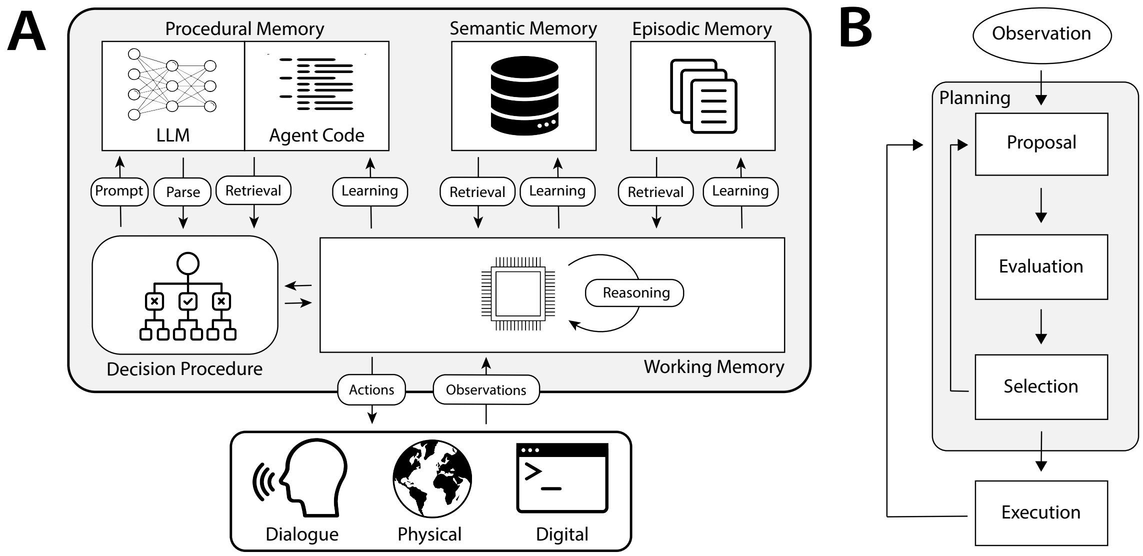


Figure 6.4: Cognitive architectures for language agents (CoALA). **A**: CoALA defines a set of interacting modules and processes. The **decision procedure** executes the agent's source code. This source code consists of procedures to interact with the LLM (prompt templates and parsers), internal memories (retrieval and learning), and the external environment (grounding). **B**: Temporally, the agent's decision procedure executes a **decision cycle** in a loop with the external environment. During each cycle, the agent uses **retrieval** and **reasoning** to plan by proposing and evaluating candidate **learning** or **grounding** actions. The best action is then selected and executed. An observation may be made, and the cycle begins again.

we use this perspective to organize existing approaches and highlight promising extensions.

6.4 Cognitive Architectures for Language Agents

We present Cognitive Architectures for Language Agents (CoALA) as a framework to organize existing language agents and guide the development of new ones. CoALA positions the LLM as the core component of a larger cognitive architecture (Figure 6.4). Under CoALA, a language agent stores information in **memory** modules (Section 6.4.1), and acts in an action space structured into external and internal parts (Figure 6.5):

- **External actions** interact with external environments (e.g., control a robot, communicate with a human, navigate a website) through **grounding** (Section 6.4.2).

- **Internal actions** interact with internal memories. Depending on which memory gets accessed and whether the access is read or write, internal actions can be further decomposed into three kinds: **retrieval** (read from long-term memory; Section 6.4.3), **reasoning** (update the short-term working memory with LLM; Section 6.4.4), and **learning** (write to long-term memory; Section 6.4.5).

Language agents choose actions via **decision-making**, which follows a repeated cycle (Section 6.4.6, Figure 6.4B). In each cycle, the agent can use reasoning and retrieval actions to plan. This planning subprocess selects a grounding or learning action, which is executed to affect the outside world or the agent's long-term memory. CoALA's decision cycle is analogous to a program's "main" *procedure* (a *method* without return values, as opposed to *functions*) that runs in loops continuously, accepting new perceptual input and calling various action *procedures* in response.

CoALA (Figure 6.4) is inspired by the decades of research in cognitive architectures (Section 6.2.3), leveraging key concepts such as memory, grounding, learning, and decision-making. Yet the incorporation of an LLM leads to the addition of "reasoning" actions, which can flexibly produce new knowledge and heuristics for various purposes – replacing hand-written rules in traditional cognitive architectures. It also makes text the *de facto* internal representation, streamlining agents' memory modules. Finally, recent advances in vision-language models [VLMs; 9] can simplify grounding by providing a straightforward translation of perceptual data into text [368].

The rest of this section details key concepts in CoALA: memory, actions (grounding, reasoning, retrieval, and learning), and decision-making. For each concept, we use existing language agents (or relevant NLP/RL methods) as examples – or note gaps in the literature for future directions.

6.4.1 Memory

Language models are *stateless*: they do not persist information across calls. In contrast, language agents may store and maintain information internally for multi-step interaction with the world. Under the CoALA framework, language agents explicitly organize information (mainly textural, but other modalities also allowed) into multiple memory modules, each containing a different form of information. These include short-term working memory and several long-term memories: episodic, semantic, and procedural.

Working memory Working memory maintains active and readily available information as symbolic variables for the current decision cycle (Section 6.4.6). This includes perceptual inputs, active knowledge (generated by reasoning or retrieved from long-term memory), and other core information carried over from the previous decision cycle (e.g., agent's active goals). Previous methods encourage the LLM to generate intermediate reasoning [332, 234], using the LLM's own context as a form of working memory. CoALA's notion of working memory is more general: it is a data structure that persists across LLM calls. On each LLM call, the LLM input is synthesized from a subset of working memory (e.g., a prompt template and relevant variables). The LLM output is then parsed back into other variables (e.g., an action name and arguments) which are stored back in working memory and used to execute the corresponding action (Figure 6.3A). Besides the LLM, the working memory also interacts with long-term memories and grounding interfaces. It thus serves as the central hub connecting different components of a language agent.

Episodic memory Episodic memory stores experience from earlier decision cycles. This can consist of training input-output pairs [264], history event flows [334, 242], game trajectories from previous episodes [356, 308], or other representations of the

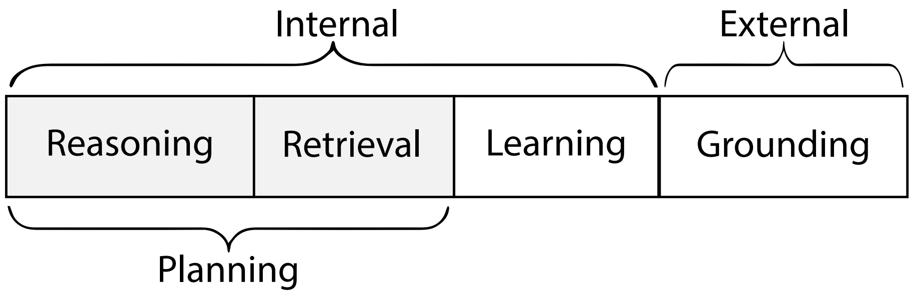


Figure 6.5: Agents' action spaces can be divided into **internal** memory accesses and **external** interactions with the world. **Reasoning** and **retrieval** actions are used to support planning.

agent's experiences. During the planning stage of a decision cycle, these episodes may be retrieved into working memory to support reasoning. An agent can also write new experiences from working to episodic memory as a form of learning (Section 6.4.5).

Semantic memory Semantic memory stores an agent's knowledge about the world and itself. Traditional NLP or RL approaches that leverage retrieval for reasoning or decision-making initialize semantic memory from an external database for knowledge support. For example, retrieval-augmented methods in NLP [162, 23, 44] can be viewed as retrieving from a semantic memory of unstructured text (e.g., Wikipedia). In RL, "reading to learn" approaches [24, 210, 101, 377] leverage game manuals and facts as a semantic memory to affect the policy. While these examples essentially employ a fixed, read-only semantic memory, language agents may also write new knowledge obtained from LLM reasoning into semantic memory as a form of learning (Section 6.4.5) to incrementally build up world knowledge from experience.

Procedural memory Language agents contain two forms of procedural memory: *implicit* knowledge stored in the LLM weights, and *explicit* knowledge written in the agent's code. The agent's code can be further divided into two types: procedures that implement actions (reasoning, retrieval, grounding, and learning procedures), and procedures that implement decision-making itself (Section 6.4.6). During a decision cycle, the LLM can be accessed via reasoning actions, and various code-based procedures can be retrieved and executed. Unlike episodic or semantic memory that may be initially empty or even absent, procedural memory must be initialized by the designer with proper code to bootstrap the agent. Finally, while learning new actions by writing to procedural memory is possible (Section 6.4.5), it is significantly riskier than writing to episodic or semantic memory, as it can easily introduce bugs or allow an agent to subvert its designers' intentions.

6.4.2 Grounding actions

Grounding procedures execute external actions and process environmental feedback into working memory as text. This effectively simplifies the agent's interaction with the outside world as a "text game" with textual observations and actions. We categorize three kinds of external environments:

Physical environments Physical embodiment is the oldest instantiation envisioned for AI agents [229]. It involves processing perceptual inputs (visual, audio, tactile) into textual observations (e.g., via pre-trained captioning models), and affecting the physical environments via robotic planners that take language-based commands. Recent advances in LLMs have led to numerous robotic projects [7, 171, 284, 239, 261] that leverage LLMs as a "brain" for robots to generate actions or plans in the physical world. For perceptual input, vision-language models are typically used to convert images to text [9, 290] providing additional context for the LLM [72, 118, 29, 28].

Dialogue with humans or other agents Classic linguistic interactions allow the agent to accept instructions [338, 302, 45, 20] or learn from people [223, 289, 291, 319]. Agents capable of *generating* language may ask for help [261, 222, 221, 220] or clarification [21, 267, 238, 303, 212] – or entertain or emotionally help people [375, 381, 245, 104, 194]. Recent work also investigates interaction among multiple language agents for social simulation [242, 131, 85], debate [39, 173, 73], improved safety [125], or collabrative task solving [253, 341, 112, 71].

Digital environments This includes interacting with games [106, 57, 278, 318, 187], APIs [269, 360, 241, 300], and websites [275, 209, 352, 384, 95, 65] as well as general code execution [349, 158, 226]. Such digital grounding is cheaper and faster than physical or human interaction. It is thus a convenient testbed for language agents and has been studied with increasing intensity in recent years. In particular, for NLP tasks that require augmentation of external knowledge or computation, stateless digital APIs (e.g., search, calculator, translator) are often packaged as "**tools**" [241, 269, 347, 300, 254], which can be viewed as special "single-use" digital environments.

6.4.3 Retrieval actions

In CoALA, a retrieval procedure [166, 92] reads information from long-term memories into working memory. Depending on the information and memory type, it could be implemented in various ways, e.g., rule-based, sparse, or dense retrieval. For example, Voyager [314] loads code-based skills from a skill library via dense retrieval to interact with the Minecraft world – effectively retrieving grounding procedures from a procedural memory. Generative Agents [242] retrieves relevant events from episodic memory via a combination of recency (rule-based), importance (reasoning-based), and relevance (embedding-based) scores. DocPrompting [383] proposes to leverage library documents to assist code generation, which can be seen as retrieving knowledge from semantic memory. While retrieval plays a key role in human decision-making [379, 376], adaptive and context-specific recall remains understudied in language agents. In Section 6.6, we suggest a principled integration of decision-making and retrieval as an important future direction.

6.4.4 Reasoning actions

Reasoning allows language agents to process the contents of working memory to generate new information. Unlike retrieval (which reads from long-term memory into working memory), reasoning reads from *and* writes to working memory. This allows the agent to summarize and distill insights about the most recent observation [360, 248], the most recent trajectory [276], or information retrieved from long-term memory [242]. Reasoning can be used to support learning (by writing the results into long-term memory) or decision-making (by using the results as additional context for subsequent LLM calls).

6.4.5 Learning actions

Learning occurs by writing information to long-term memory, which includes a spectrum of diverse procedures.

Updating episodic memory with experience It is common practice for RL agents to store episodic trajectories to update a parametric policy [22, 250] or establish a non-parametric policy [74, 308]. For language agents, added experiences in episodic memory may be retrieved later as examples and bases for reasoning or decision-making [334, 264, 242].

Updating semantic memory with knowledge Recent work [276, 242] has applied LLMs to reason about raw experiences and store the resulting inferences in semantic memory. For example, Reflexion [276] uses an LLM to reflect on failed episodes and stores the results (e.g., "there is no dishwasher in kitchen") as semantic knowledge to be attached to LLM context for solving later episodes. Finally, work in robotics [43] uses vision-language models to build a semantic map of the environment, which can later be queried to execute instructions.

Updating LLM parameters (procedural memory) The LLM weights represent implicit procedural knowledge. These can be adjusted to an agent's domain by fine-tuning during the agent's lifetime. Such fine-tuning can be accomplished via supervised [183, 373] or imitation learning [123], reinforcement learning (RL) from environment feedback [295], human feedback [RLHF; 53, 237, 209], or AI feedback [18, 188]. Classic LLM self-improvement methods [116, 367] use an external measure such as consistency [323] to select generations to fine-tune on. In reinforcement learning settings, this can be extended to use environmental feedback instead: for example, XTX [308] periodically fine-tunes a small language model on high-scoring trajectories stored in episodic memory, which serves as a robust "exploitation" policy to reach exploration frontiers in the face of stochasity. Fine-tuning the agent's LLM is a costly form of learning; thus, present studies specify learning schedules. However, as training becomes more efficient – or if agents utilize smaller subtask-specific LLMs – it may be possible to allow language agents to autonomously determine when and how to fine-tune their LLMs.

Updating agent code (procedural memory) CoALA allows agents to update their source code, thus modifying the implementation of various procedures. These can be broken down as follows:

- **Updating reasoning** [e.g., prompt templates; 88, 385]. For example, APE [385] infers prompt instructions from input-output examples, then uses these instructions as part of the LLM prompt to assist task solving. Such a prompt update can be seen as a form of learning to reason.

- **Updating grounding** [e.g., code-based skills; 171, 76, 314]. For example, Voyager [314] maintains a curriculum library. Notably, current methods are limited to creating new code skills to interact with external environments.

- **Updating retrieval**. To our knowledge, these learning options are not studied in recent language agents. Retrieval is usually considered a basic action designed with some fixed implementation (e.g., BM25 or dense retrieval), but research in query/document expansion [232, 317, 299] or retrieval distillion [126] may be helpful for language agents to learn better retrieval procedures.

- **Updating learning or decision-making**. Finally, it is theoretically possible for CoALA agents to learn new procedures for learning or decision-making, thus providing significant adaptability. In general, however, updates to these procedures are risky both for the agent's functionality and alignment. At present, we are not aware of any language agents that implement this form of learning; we discuss such possibilities more in Section 6.6.

While RL agents usually fix one way of learning (e.g., Q-learning, PPO, or A3C) and learn by updating model parameters, language agents can select from a diversity of learning procedures. This allows them to learn rapidly by storing task-relevant language (cheaper and quicker than parameter updates), and leverage multiple forms of learning to compound their self-improvement (e.g., Generative Agents discussed in Section 6.5).

Finally, while our discussion has mostly focused on **adding** to memory, **modifying** and **deleting** (a case of "unlearning") are understudied in recent language agents. We address these areas more in Section 6.6.

6.4.6 Decision making

With various actions (grounding, learning, reasoning, retrieval) in the action space, how should a language agent choose which action to apply? This is handled by the decision-making procedure, which is effectively the top-level or "main" agent program. CoALA structures this top-level program into decision cycles (Figure 6.4B) which yield an external *grounding* action (Section 6.4.2) or internal *learning* action (Section 6.4.5). In each cycle, program code defines a sequence of reasoning and retrieval actions to propose and evaluate alternatives (**planning stage**), then executes the selected action (**execution stage**) – then the cycle loops again.

Planning stage During planning, reasoning and retrieval can be flexibly applied to propose, evaluate, and select actions, and these sub-stages could interleave or iterate to build up multi-step simulations [297] before taking an external action [358, 102]. It also enables agents to iteratively improve candidate solutions – for example, by using the LLM to simulate them, identifying defects, and proposing modifications that address those defects [140, 276].

- **Proposal**. The proposal sub-stage generates one or more action candidates. The usual approach is to use **reasoning** (and optionally retrieval) to sample one [120] or more [46, 323] external grounding actions from the LLM. For simple domains with limited actions, the proposal stage might simply include all actions (e.g., SayCan in Section 6.5). More sophisticated agents use if-else or while-if code structures [314, 242]; while agents deployed in well-defined domains may utilize structured simulators [105] to generate plausible rollouts [181, 62].

- **Evaluation**. If multiple actions are proposed, the evaluation sub-stage assigns a value to each. This may use heuristic rules, LLM (perplexity) values [7], learned values [356], LLM reasoning [358, 102], or some combination. Particularly, LLM reasoning can help evaluate actions by internally simulating their grounding feedback from the external world [102, 349].

- **Selection**. Given a set of actions and their values, the selection step either selects one to execute or rejects them and loops back to the proposal step. Depending on the form of action values, selection may occur via argmax, softmax, or an alternative such as majority vote [323].

Execution stage The selected action is applied by executing the relevant procedures from the agent's source code. Depending on the agent implementation, this might be an external *grounding* action (e.g., an API call; Section 6.4.2) or an internal *learning* action (e.g., a write to episodic memory; Section 6.4.5). An observation can be made from the environment, providing feedback from the agent's action, and the cycle loops again.

Empirically, many early language agents simply use LLMs to propose an action [269], a sequence of actions [119], or evaluate a fixed set of actions [7] without intermediate reasoning or retrieval. Followup work [360, 276, 348, 175, 314, 242] has exploited intermediate reasoning and retrieval to analyze the situation, make and maintain action plans, refine the previous action given the environmental feedback, and leveraged a more complex procedure to propose a single action. Most recently, research has started to investigate more complex decision-making employing iterative proposal and evaluation to consider multiple actions. These procedures are modeled after classical planning algorithms: for example, Tree of Thoughts [358] and RAP [102] use LLMs to implement BFS/DFS and Monte Carlo Tree Search [MCTS; 32] respectively. LLMs are used to generate proposals (i.e., to simulate rollouts conditioned on an action) and evaluate them (i.e., to value the outcome of the proposed action).

6.5 Case Studies

With variations and ablations of the memory modules, action space, and decision-making procedures, CoALA can express a wide spectrum of language agents. Table 6.2 lists some popular recent methods across diverse domains — from Minecraft to robotics, from pure reasoning to social simulacra. CoALA helps characterize their internal mechanisms and reveal their similarities and differences in a simple and structured way.

	Long-term Memory[5]	External Grounding	Internal Actions	Decision Making
SayCan [7]	-	physical	-	evaluate
ReAct [360]	-	digital	reason	propose
Voyager [314]	procedural	digital	reason/retrieve/learn	propose
Generative Agents [242]	episodic/semantic	digital/agent	reason/retrieve/learn	propose
Tree of Thoughts [358]	-	digital[6]	reason	propose, evaluate, select

Table 6.2: Some recent language agents cast into the CoALA framework.

SayCan [7] grounds a language model to robotic interactions in a kitchen to satisfy user commands (e.g., "I just worked out, can you bring me a drink and a snack to recover?"). Its long-term memory is procedural only (an LLM and a learned value function). The action space is external only – a fixed set of 551 grounding skills (e.g., "find the apple", "go to the table"), with no internal actions of reasoning, retrieval, or learning. During decision-making, SayCan evaluates each action using a combination of LLM and learned values, which balance a skill's usefulness and groundedness. SayCan therefore employs the LLM (in conjunction with the learned value function) as a single-step planner.

ReAct [360] is a language agent grounded to various digital environments (e.g., Wikipedia API, text game, website). Like SayCan, it lacks semantic or episodic memory and therefore has no retrieval or learning actions. Its action space consists of (internal) reasoning and (external) grounding. Its decision cycle is fixed to use a single reasoning action to analyze the situation and (re)make action plans, then generates a grounding action without evaluation or selection stages. ReAct can be considered the simplest language agent that leverages both internal and external actions, and is the initial work that demonstrates their synergizing effects: reasoning helps guide acting, while acting provides environmental feedback to support reasoning.

Voyager [314] is a language agent grounded to the Minecraft API. Unlike SayCan, which grounds to perception via the learned value function, Voyager's grounding

[5]All agents contain some procedural memory (agent code and LLM weights), so here we only list writable procedural memory.

[6]Special digital grounding with the only external action being submitting a final answer.

is text-only. It has a long-term procedural memory that stores a library of code-based grounding procedures a.k.a. skills (e.g., "combatZombie", "craftStoneSword"). This library is hierarchical: complex skills can use simpler skills as sub-procedures (e.g., "combatZombie" may call "craftStoneSword" if no sword is in inventory). Most impressively, its action space has all four kinds of actions: grounding, reasoning, retrieval, and learning (by adding new grounding procedures). During a decision cycle, Voyager first reasons to propose a new task objective if it is missing in the working memory, then reasons to propose a code-based grounding procedure to solve the task. In the next decision cycle, Voyager reasons over the environmental feedback to determine task completion. If successful, Voyager selects a learning action adding the grounding procedure to procedural memory; otherwise, it uses reasoning to refine the code and re-executes it. The importance of long-term memory and procedural learning is empirically verified by comparing to baselines like ReAct and AutoGPT and ablations without the procedural memory. Voyager is shown to better explore areas, master the tech tree, and zero-shot generalize to unseen tasks.

Generative Agents [242] are language agents grounded to a sandbox game affording interaction with the environment and other agents. Its action space also has all four kinds of actions: grounding, reasoning, retrieval, and learning. Each agent has a long-term episodic memory that stores events in a list. These agents use retrieval and reasoning to generate reflections on their episodic memory (e.g., "I like to ski now.") which are then written to long-term semantic memory. During decision-making, it retrieves relevant reflections from semantic memory, then reasons to make a high-level plan of the day. While executing the plan, the agent receives a stream of grounding observations; it can reason over these to maintain or adjust the plan.

Tree of Thoughts (ToT) [358] can be seen as a special kind of language agent with only one external action: submitting a final solution to a reasoning problem (game of 24, creative writing, crosswords puzzle). It has no long-term memory, and only reasoning in its internal action space, but differs from all previous agents in its deliberate decision-making. During planning, ToT iteratively proposes, evaluates, and selects "thoughts" (reasoning actions) based on LLM reasoning, and maintains them via a tree search algorithm to enable global exploration as well as local backtrack and foresight.

6.6 Actionable Insights

Compared to some recent empirical surveys around language agents [203, 333, 315], CoALA offers a theoretical framework grounded in the well-established research of cognitive architectures. This leads to a unique and complementary set of actionable insights.

Modular agents: thinking beyond monoliths Perhaps our most important suggestion is that *agents should be structured and modular*. Practically, just as standardized software is used across robotics platforms [256, 195], a framework for language agents would consolidate technical investment and improve compatibility.

- **In academic research**, standardized terms allow conceptual comparisons across works (Table 6.2), and open-source implementations would further facilitate modular plug-and-play and re-use. For example, the theoretical framework of Markov Decision Processes [252] provides a standardized set of concepts and terminology (e.g., state, action, reward, transition) for reinforcement learning [295]. Correspondingly, empirical frameworks like OpenAI Gym [27] provided standardized abstractions (e.g., `obs, reward, done, info = env.step(action)`) that facilitate empirical RL work. Thus, it would be timely and impactful to also implement useful abstractions (e.g., `Memory`, `Action`, `Agent` classes) for language agents, and cast simpler agents into such an empirical CoALA framework as examples for building more complex agents.

- **In industry applications**, maintaining a single company-wide "language agent library" would reduce technical debt [271, 193] by facilitating testing and component re-use across individual agent deployments. It could also standardize the customer experience: rather than interacting with a hodgepodge of language agents developed by individual teams, end users would experience a context-specific instantiation of the same base agent.

- **LLMs vs. code in agent design**. CoALA agents possess two forms of procedural memory: agent code (deterministic rules) and LLM parameters (a large, stochastic production system). Agent code is interpretable and extensible, but often brittle in face of stochasticity and limited to address situations the designer anticipates. In contrast, LLM parameters are hard to interpret, but offer significant zero-shot flexibility in new contexts [119]. CoALA thus suggests using code sparingly to implement generic algorithms that complement LLM limitations, e.g., implementing tree search to mitigate myopia induced by autoregressive generation [358, 102].

Agent design: thinking beyond simple reasoning CoALA defines agents over three distinct concepts: (i) internal memory, (ii) a set of possible internal and external actions, and (iii) a decision making procedure over those actions. Using CoALA to develop an application-specific agent consists of specifying implementations for each of these components in turn. We assume that the agent's environment and external action space are given, and show how CoALA can be used to determine an appropriate high-level architecture. For example, we can imagine designing a personalized retail assistant [352] that helps users find relevant items based on their queries and purchasing history. In this case, the external actions would consist of dialogue or returning search results to the user.

- **Determine what memory modules are necessary.** In our retail assistant example, it would be helpful for the agent to have semantic memory containing the set of items for sale, as well as episodic memory about each customer's previous purchases and interactions. It will need procedural memory defining functions to query these datastores, as well as working memory to track the dialogue state.

- **Define the agent's internal action space.** This consists primarily of defining read and write access to each of the agent's memory modules. In our example, the agent should have read and write access to episodic memory (so it can store new interactions with customers), but read-only access to semantic and procedural memory (since it should not update the inventory or its own code).

- **Define the decision-making procedure.** This step specifies how reasoning and retrieval actions are taken in order to choose an external or learning action. In general, this requires a tradeoff between performance and generalization: more complex procedures can better fit to a particular problem (e.g., Voyager [314] for Minecraft) while simpler ones are more domain-agnostic and generalizable (e.g., ReAct [360]). For our retail assistant, we may want to encourage retrieval of episodic memory of interactions with a user to provide a prior over their search intent, as well as an explicit evaluation step reasoning about whether a particular set of search results will satisfy that intent. We can simplify the decision procedure by deferring learning to the end of the interaction [276, 242], summarizing the episode prior to storing it in episodic memory.

Structured reasoning: thinking beyond prompt engineering Early work on prompt engineering manipulated the LLM's input and output via low-level string operations. CoALA suggests a more structured reasoning procedure to update working memory variables.

- **Prompting frameworks** like LangChain [155] and LlamaIndex [189] can be used to define higher-level sequences of reasoning steps, reducing the burden of reasoning per LLM call and the low-level prompt crafting efforts. **Structural output parsing solutions** such as Guidance [94] and OpenAI function calling [235] can help update working memory variables. Defining and building good working memory modules will also be an important direction of future research. Such modules may be especially important for industry solutions where LLM reasoning needs to seamlessly integrate with large-scale code infrastructure.

- **Reasoning usecases in agents can inform and reshape LLM training** in terms of the types (e.g., reasoning for self-evaluation, reflection, action generation, etc.) and formats (e.g., CoT [332], ReAct [360], Reflexion [276]) of training instances. By default, existing LLMs are trained and optimized for NLP tasks, but agent applications have explored new modes of LLM reasoning (e.g., self-evaluation) that have proven broadly useful. LLMs trained or finetuned towards these capabilities will more likely be the backbones of future agents.

Long-term memory: thinking beyond retrieval augmentation . While traditional retrieval-augmented language models [98, 162, 23] only read from human-written corpora, memory-augmented language agents can both read and write self-generated content autonomously. This opens up numerous possibilities for efficient lifelong learning.

- **Combining existing human knowledge with new experience and skills** can help agents bootstrap to learn efficiently. For example, a code-writing agent could be endowed with semantic programming knowledge in the form of manuals or textbooks. It could then generate its own episodic knowledge from experience; reflect on these experiences to generate new semantic knowledge; and gradually create procedural knowledge in the form of a code library storing useful methods.

- **Integrating retrieval and reasoning** can help to better ground planning. Recent computational psychological models implicate an integrated process of memory recall and decision-making [379, 376] – suggesting that adaptive mechanisms interleaving memory search and forward simulation will allow agents to make the most of their knowledge.

Learning: thinking beyond in-context learning or finetuning CoALA's definition of "learning" encompasses these methods, but extends further to storing new experience or knowledge, or writing new agent code (Section 6.4.5). Important future directions include:

- **Meta-learning by modifying agent code** would allow agents to learn more effectively. For example, learning better retrieval procedures could enable agents to make better use of their experience. Recent expansion-based techniques [232, 317, 299] could allow agents to reason about when certain knowledge would be useful, and store this as metadata to facilitate later recall. These forms of meta-learning would enable agents to go beyond human-written code, yet are understudied due to their difficulty and risk.

- **New forms of learning (and unlearning)** could include fine-tuning smaller models for specific reasoning sub-tasks [367, 116, 7], deleting unneeded memory items for "unlearning" [225], and studying the interaction effects between multiple forms of learning [308, 242, 344, 137].

Action space: thinking beyond external tools or actions Although "action space" is a standard term in reinforcement learning, it has been used sparingly with language agents. CoALA argues for defining a clear and task-suitable action space with both internal (reasoning, retrieval, learning) and external (grounding) actions, which will help systematize and inform the agent design.

- **Size of the action space.** More capable agents (e.g., Voyager, Generative Agents) have larger action spaces – which in turn means they face a more complex decision-making problem. As a result, these agents rely on more customized or hand-crafted decision procedures. The tradeoff of the action space vs. decision-making complexities is a basic problem to be considered before agent development, and taking the minimal action space necessary to solve a given task might be preferred.

- **Safety of the action space.** Some parts of the action space are inherently riskier. "Learning" actions (especially procedural deletion and modification) could cause internal harm, while "grounding" actions (e.g., "rm" in bash terminal, harmful speech in human dialog, holding a knife in physical environments) could cause external harm. Today, safety measures are typically task-specific heuristics (e.g., remove "os" operations in Python [46], filter keywords in dialog [52, 72], limit robots to controlled environments [7]). However, as agents are grounded to more complex environments with richer internal mechanisms, it may be necessary to specify and ablate the agent's action space for worst-case scenario prediction and prevention [353].

Decision making: thinking beyond action generation We believe one of the most exciting future directions for language agents is decision-making: as detailed in Section 6.4.6, most works are still confined to proposing (or directly generating) a single action. Present agents have just scratched the surface of more deliberate, propose-evaluate-select decision-making procedures.

- **Mixing language-based reasoning and code-based planning** may offer the best of both worlds. Existing approaches either plan directly in natural language [120, 7] or use LLMs to translate from natural language to structured world models [339, 181, 370, 163, 93, 283, 282]. Future work could integrate these: just as Soar incorporates a simulator for physical reasoning [149], agents may write and execute simulation code on the fly to evaluate the consequences of plans. See Section 6.7 for more discussion.

- **Extending deliberative reasoning to real-world settings**. Initial works have implemented classical planning and tree search [358, 102, 181, 62], using toy tasks such as game of 24 or block building. Extending these schemes to more complicated tasks with grounding [254] and long-term memory is an exciting direction.

- **Metareasoning to improve efficiency.** LLM calls are both slow and computationally intensive. Using LLMs for decision-making entails a balance between their computational cost and the utility of the resulting improved plan. Most LLM reasoning methods fix a search budget by specifying a depth of reasoning [358], but humans appear to adaptively allocate computation [265, 174, 37, 89]. Future work should develop mechanisms to estimate the utility of planning [147] and modify the decision procedure accordingly, either via amortization [fine-tuning the LLM based on the results of previous actions, e.g. 224, 100], routing among several decision sub-procedures (e.g., ReAct [360] investigated backing off to CoT [332] and vice versa), or updates to the decision-making procedure.

- **Calibration and alignment.** More complex decision-making is currently bottlenecked by issues such as over-confidence and miscalibration [128, 25, 49], misalignment with human values or bias [172, 79], hallucinations in self-evaluation [276], and lack of human-in-the-loop mechanisms in face of uncertainties [222, 261]. Solving these issues will significantly improve LLMs' utilities as agent backbones.

6.7 Discussion

In addition to the practical insights presented above, CoALA raises a number of open conceptual questions. We briefly highlight the most interesting as important directions for future research and debate.

LLMs vs VLMs: should reasoning be language-only or multimodal? Most language agents use language-only models for decision-making [360, 314, 358], employing a separate captioning model to convert environment observations to text when necessary [7, 368]. However, the latest generation of language models are multimodal, allowing interleaved image and text input [236, 9, 301, 165]. Language agents built on such multimodal models natively reason over both image and text input [19, 77, 184, 113, 72], allowing them to ingest perceptual data and directly produce actions. This bypasses the lossy image-to-text conversion; however, it also tightly couples the reasoning and planning process to the model's input modalities.

At a high level, the two approaches can be seen as different tokenization schemes to convert non-linguistic modalities into the core reasoning model's language domain. The modular approach uses a separate image-to-text model to convert perceptual data into language [7, 368], while the integrated approach projects images directly into the language model's representation space [19, 77, 184]. Integrated, multimodal reasoning may allow for more human-like behaviors: a VLM-based agent could "see" a webpage, whereas a LLM-based agent would more likely be given raw HTML. However, coupling the agent's perception and reasoning systems makes the agent more domain-specific and difficult to update. In either case, the basic architectural principles described by CoALA — internal memories, a structured action space, and generalized decision-making — can be used to guide agent design.

Internal vs. external: what is the boundary between an agent and its environment? While humans or robots are clearly distinct from their embodied environment, digital language agents have less clear boundaries. For example, is a Wikipedia database an internal semantic memory or an external digital environment [360]? If an agent iteratively executes and improves code before submitting an answer [276, 349], is the code execution internal or external? If a method consists of proposal and evaluation prompts [358], should it be considered a single agent or two collaborating simpler agents (proposer and evaluator)?

We suggest the boundary question can be answered in terms of *controllability* and *coupling*. For example, Wikipedia is not *controllable*: it is an external environment that may be unexpectedly modified by other users. However, an offline version that only the agent may write to *is* controllable, and thus can be considered an internal memory. Similarly, code execution on a internal virtual environment should be considered an internal reasoning action, whereas code execution on an external machine (which may possess security vulnerabilities) should be considered an external grounding action. Lastly, if aspects of the agent – such as proposal and evaluation prompts – are designed for and dependent on each other, then they are *tightly coupled* and best conceptualized as components in an individual agent. In contrast, if the steps are independently useful, a multi-agent perspective may be more appropriate. While these dilemmas are primarily conceptual, such understanding can support agent design and help the field align on shared terminology. Practioners may also just choose their preferred framing, as long as it is consistent and useful for their own work.

Physical vs. digital: what differences beget attention? While animals only live once in the physical world, digital environments (e.g., the Internet) often allow sequential (via resets) and parallel trials. This means digital agents can more boldly explore (e.g., open a million webpages) and self-clone for parallel task solving (e.g., a million web agents try different web paths), which may result in decision-making procedures different from current ones inspired by human cognition [91].

Learning vs. acting: how should agents continuously and autonomously learn? In the CoALA framework, learning is a result action of a decision-making cycle just like grounding: the agent deliberately chooses to commit information to long-term memory. This is in contrast to most agents, which simply fix a learning schedule and only use decison making for external actions. Biological agents, however, do not have this luxury: they must balance learning against external actions in their lifetime, choosing when and what to learn [199]. More flexible language agents [314, 242] would follow a similar design and treat learning on par with external actions. Learning could be proposed as a possible action during regular decision-making, allowing the agent to "defer" it until the appropriate time.

GPT-4 vs GPT-N: how would agent design change with more powerful LLMs? Agent design is a moving target as new LLM capabilities emerge with scale [331]. For example, earlier language models such as GPT-2 [258] would not support LLM agents — indeed, work at that time needed to combine GPT-2 with reinforcement learning for action generation [356]; GPT-3 [30] unlocked flexible few-shot and zero-shot reasoning for NLP tasks; while only GPT-4 [236] starts to afford more reliable self-evaluation [268, 276, 358] and self-refinement [196, 47]. Will future LLMs further reduce the need for coded rules and extra-learned models? Will this necessitate changes to the CoALA framework? As a thought experiment, imagine GPT-N could "simulate" memory, grounding, learning, and decision-making in context: list all the possible actions, simulate and evaluate each one, and maintain its entire long-term memory explicitly in a very long context. Or even more boldly: perhaps GPT-N+1 succeeds at generating the next action by simulating these implicitly in neurons, without any intermediate reasoning in context. While these extreme cases seem unlikely in the immediate future, incremental improvements may alter the importance of different CoALA components. For example, a longer context window could reduce the importance of long-term memory, while more powerful reasoning for internal evaluation and simulation could allow longer-horizon planning. In general, LLMs are not subject to biological limitations [91], and their emergent properties have been difficult to predict. Nonetheless, CoALA – and cognitive science more generally – may still help organize tasks where language agents succeed or fail, and suggest code-based procedures to complement a given LLM on a given task. Even in the most extreme case, where GPT implements all of CoALA's mechanisms in neurons, it may be helpful to leverage CoALA as a conceptual guide to discover and interpret those implicit circuits. Of course, as discussed in Section 6.6, agent usecases will also help discover, define and shape LLM capabilities. Similar to how chips and computer architectures have co-evolved, language model and agent design should also develop a reciprocal path forward.

Chapter 7

Conclusion

In this thesis, we establish the study of language agents as an independent, holistic, and interdisciplinary research subject, which applies language models for autonomous agents, enables exciting real-world applications such as automating various computer and web tasks, and synthesizes fundamental insights in modern machine learning, natural language processing, and cognitive science.

7.1 Ongoing and Future Work

Following up on actionable insights proposed in Section 6.6, here are some more concrete future directions that I am excited to work on.

Training LLMs for agents. Most open-source LLMs perform poorly on agent tasks as they were not trained to act, and proprietary models like GPT-4 are expensive to use and lack transparency. My work has shown training LLMs how to reason and use tools leads to a stronger generalization than either alone (Chapter 4). I am excited to work with NLP and systems researchers to develop more effective and efficient open-source LLMs for agents, and establish a reciprocating cycle where better LLMs enable exploration of agent design, and strong agents in turn provide training data to shape LLMs. I also want to work with CV and RL researchers to build agent backbones in multimodal and embodied setups, like a general-purpose computer agent reading screen pixels and using the mouse and keyboard.

Robust and safe deployment. Language agents indicate great opportunities for task automation, personal freedom, and social progress, but also enhanced potential harms like deleting files or attacking servers. I believe it takes concrete and multidisciplinary efforts to better understand and control these emerging systems, such as statistical and mathematical characterization [355] of their capabilities and robustness, defining threat models and finding defenses, and engaging ethics, law, and policy experts in capturing and shaping their societal impact [353]. Across these efforts, it is important to have a holistic view of not just LLMs, but how they are and will be used to interact with the world. CoALA [357] could help organize and guide these efforts, e.g., we could analyze and control risks by defining the action space of language agents (Chapter 6). Another important direction is automated coding [349, 129] (Chapter 3), as agent-generated code can act in more interpretable and reliable ways than agents.

Knowledge and scientific discovery. So far, the success of LLMs and language agents relies mostly on imitating patterns of how humans write and act, thus happening mostly on tasks that humans have already explored and summarized knowledge about. But to go beyond imitation, we need to equip language agents with intrinsic rewards like curiosity [354], means to planning [358] (Chapter 5) and reinforcement learning [276] using such intrinsic rewards, and a long-term memory [357] to maintain experience, knowledge, and skills. I envision agents that navigate gigantic networks of knowledge (e.g., via ArXiv APIs) to answer self-asked questions, and learn by checking follow-up research via citations [299], interacting with humans [294], or coding [349] (Chapter 3), similar to how PhD students expand human knowledge.

Understanding and helping humans. My work has been inspired by human cognition [358, 357] to build autonomous agents that solve hard tasks with minimal guidance. But to deploy language agents in our society, they will need to infer human intention, invoke and incorporate human feedback, and collaborate with humans or other agents. I hope to engage insights from pragmatics, game theory, social cognition, and HCI to understand how humans perceive language agents [294], and how agents could in turn better model and interact with humans. Particularly, I want to develop a tutor agent with a long-term memory [357] of agent-student interaction histories and the student profile to personalize education. Beyond teaching existing knowledge, I also envision agents communicate their discovered concepts (e.g., Move 37 of AlphaGo) to humans by linking their "emergent languages" to human ones [359]. These will help ensure that AI complements and augments human abilities, rather than surpassing or replacing them.

Bibliography

[1] Josh Abramson, Arun Ahuja, Iain Barr, Arthur Brussee, Federico Carnevale, Mary Cassin, Rachita Chhaparia, Stephen Clark, Bogdan Damoc, Andrew Dudzik, Petko Georgiev, Aurelia Guy, Tim Harley, Felix Hill, Alden Hung, Zachary Kenton, Jessica Landon, Timothy Lillicrap, Kory Mathewson, Soňa Mokrá, Alistair Muldal, Adam Santoro, Nikolay Savinov, Vikrant Varma, Greg Wayne, Duncan Williams, Nathaniel Wong, Chen Yan, and Rui Zhu. Imitating interactive intelligence, 2020.

[2] Sam Adams, Itmar Arel, Joscha Bach, Robert Coop, Rod Furlan, Ben Goertzel, J Storrs Hall, Alexei Samsonovich, Matthias Scheutz, Matthew Schlesinger, et al. Mapping the landscape of human-level artificial general intelligence. *AI magazine*, 33(1):25–42, 2012.

[3] Leonard Adolphs, Benjamin Boerschinger, Christian Buck, Michelle Chen Huebscher, Massimiliano Ciaramita, Lasse Espeholt, Thomas Hofmann, and Yannic Kilcher. Boosting Search Engines with Interactive Agents. *arXiv preprint arXiv:2109.00527*, 2021.

[4] Mayank Agarwal, Tathagata Chakraborti, Quchen Fu, David Gros, Xi Victoria Lin, Jaron Maene, Kartik Talamadupula, Zhongwei Teng, and Jules White. Neurips 2020 nlc2cmd competition: Translating natural language to bash commands. In Hugo Jair Escalante and Katja Hofmann, editors, *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*, volume 133 of *Proceedings of Machine Learning Research*, pages 302–324. PMLR, 06–12 Dec 2021.

[5] Rajas Agashe, Srini Iyer, and Luke Zettlemoyer. Juice: A large scale distantly supervised dataset for open domain context-based code generation. *ArXiv*, abs/1910.02216, 2019.

[6] Armen Aghajanyan, Dmytro Okhonko, Mike Lewis, Mandar Joshi, Hu Xu, Gargi Ghosh, and Luke Zettlemoyer. Htlm: Hyper-text pre-training and prompting of language models. *ArXiv*, abs/2107.06955, 2021.

[7] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as I can, not as I say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[8] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[9] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in Neural Information Processing Systems*, 35:23716–23736, 2022.

[10] Ben Alderson-Day and Charles Fernyhough. Inner speech: development, cognitive functions, phenomenology, and neurobiology. *Psychological bulletin*, 141(5):931, 2015.

[11] John R Anderson and Christian Lebiere. The Newell test for a theory of cognition. *Behavioral and Brain Sciences*, 26(5):587–601, 2003.

[12] Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, et al. Task-oriented dialogue as dataflow synthesis. *Transactions of the Association for Computational Linguistics*, 8:556–571, 2020.

[13] 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, and et al. Palm 2 technical report, 2023.

[14] Richard C Atkinson and Richard M Shiffrin. Human memory: A proposed system and its control processes. In *Psychology of Learning and Motivation*, volume 2, pages 89–195. Elsevier, 1968.

[15] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021.

[16] Alan Baddeley. Working memory. *Science*, 255(5044):556–559, 1992.

[17] Alan D Baddeley and Graham Hitch. Working memory. In *Psychology of Learning and Motivation*, volume 8, pages 47–89. Elsevier, 1974.

[18] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*, 2022.

[19] Rohan Bavishi, Erich Elsen, Curtis Hawthorne, Maxwell Nye, Augustus Odena, Arushi Somani, and Sağnak Taşırlar. Introducing our multimodal models, 2023.

[20] Yonatan Bisk, Daniel Marcu, and William Wong. Towards a dataset for human computer communication via grounded language acquisition. In *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[21] Erdem Biyik and Malayandi Palan. Asking easy questions: A user-friendly approach to active reward learning. In *Proceedings of the 3rd Conference on Robot Learning*, 2019.

[22] Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016.

[23] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International Conference on Machine Learning*, pages 2206–2240, 2022.

[24] SRK Branavan, David Silver, and Regina Barzilay. Learning to win by reading manuals in a Monte-Carlo framework. *Journal of Artificial Intelligence Research*, 43:661–704, 2012.

[25] Mark Braverman, Xinyi Chen, Sham Kakade, Karthik Narasimhan, Cyril Zhang, and Yi Zhang. Calibration, entropy rates, and memory in language models. In *International Conference on Machine Learning*, pages 1089–1099, 2020.

[26] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[27] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[28] 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 arXiv:2307.15818*, 2023.

[29] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. RT-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

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

[31] Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4:1–43, 2012.

[32] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[33] Toktam B.Tabrizi, Ozgur Gocer, Arash Sadrieh, and Anastasia Globa. Leveraging ai to instruct architecture students on circular design techniques and life cycle assessment. *9th International Conference on Higher Education Advances (HEAd'23)*, 2023.

[34] Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Inigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. Multiwoz–a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. *arXiv preprint arXiv:1810.00278*, 2018.

[35] Rudy Bunel, Matthew Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. Leveraging grammar and reinforcement learning for neural program synthesis, 2018.

[36] Andrea Burns, Deniz Arsan, Sanjna Agrawal, Ranjitha Kumar, Kate Saenko, and Bryan A Plummer. Interactive Mobile App Navigation with Uncertain or Under-specified Natural Language Commands. *arXiv preprint arXiv:2202.02312*, 2022.

[37] Frederick Callaway, Bas van Opheusden, Sayan Gul, Priyam Das, Paul M Krueger, Thomas L Griffiths, and Falk Lieder. Rational use of cognitive resources in human planning. *Nature Human Behaviour*, 6(8):1112–1125, 2022.

[38] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

[39] Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. Chateval: Towards better llm-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201*, 2023.

[40] Angelica Chen, Jérémy Scheurer, Tomasz Korbak, Jon Ander Campos, Jun Shern Chan, Samuel R. Bowman, Kyunghyun Cho, and Ethan Perez. Improving code generation by training with natural language feedback, 2023.

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

[42] Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. Codet: Code generation with generated tests. *arXiv preprint arXiv:2207.10397*, 2022.

[43] Boyuan Chen, Fei Xia, Brian Ichter, Kanishka Rao, Keerthana Gopalakrishnan, Michael S Ryoo, Austin Stone, and Daniel Kappler. Open-vocabulary queryable scene representations for real world planning. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11509–11522, 2023.

[44] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading Wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051*, 2017.

[45] David Chen and Raymond Mooney. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 25, pages 859–865, 2011.

[46] 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, 2021.

[47] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.

[48] Xinyun Chen, Chang Liu, and Dawn Xiaodong Song. Execution-guided neural program synthesis. In *International Conference on Learning Representations*, 2018.

[49] Yangyi Chen, Lifan Yuan, Ganqu Cui, Zhiyuan Liu, and Heng Ji. A close look into the calibration of pre-trained language models. *arXiv preprint arXiv:2211.00151*, 2022.

[50] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023.

[51] Noam Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.

[52] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[53] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[54] Alonzo Church. A set of postulates for the foundation of logic. *Annals of mathematics*, pages 346–366, 1932.

[55] Colin B. Clement, Dawn Drain, Jonathan Timcheck, Alexey Svyatkovskiy, and Neel Sundaresan. Pymt5: multi-mode translation of natural language and python code with transformers, 2020.

[56] Grace Colverd, Paul Darm, Leonard Silverberg, and Noah Kasmanoff. Floodbrain: Flood disaster reporting by web-based retrieval augmented generation with an llm. *ArXiv*, abs/2311.02597, 2023.

[57] Marc-Alexandre Côté, Akos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, et al. Textworld: A learning environment for text-based games. In *Computer Games: 7th Workshop, CGW 2018*, pages 41–75. Springer, 2019.

[58] C. Cowan, S. Arnold, S. Beattie, C. Wright, and J. Viega. Defcon capture the flag: defending vulnerable code from intense attack. In *Proceedings DARPA Information Survivability Conference and Exposition*, volume 1, pages 120–129 vol.1, 2003.

[59] Antonia Creswell and Murray Shanahan. Faithful reasoning using large language models, 2022.

[60] Antonia Creswell, Murray Shanahan, and Irina Higgins. Selection-inference: Exploiting large language models for interpretable logical reasoning, 2022.

[61] Antonia Creswell, Murray Shanahan, and Irina Higgins. Selection-inference: Exploiting large language models for interpretable logical reasoning. In *The Eleventh International Conference on Learning Representations*, 2023.

[62] Gautier Dagan, Frank Keller, and Alex Lascarides. Dynamic Planning with a LLM. *arXiv preprint arXiv:2308.06391*, 2023.

[63] Ishita Dasgupta, Christine Kaeser-Chen, Kenneth Marino, Arun Ahuja, Sheila Babayan, Felix Hill, and Rob Fergus. Collaborating with language models for embodied reasoning. In *Second Workshop on Language and Reinforcement Learning*, 2022.

[64] Nathaniel D Daw, Yael Niv, and Peter Dayan. Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature neuroscience*, 8(12):1704–1711, 2005.

[65] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. *arXiv preprint arXiv:2306.06070*, 2023.

[66] Nate Derbinsky, Justin Li, and John Laird. A multi-domain evaluation of scaling in a general episodic memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, pages 193–199, 2012.

[67] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT (1)*, 2019.

[68] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019.

[69] David Dohan, Winnie Xu, Aitor Lewkowycz, Jacob Austin, David Bieber, Raphael Gontijo Lopes, Yuhuai Wu, Henryk Michalewski, Rif A Saurous, Jascha Sohl-Dickstein, et al. Language model cascades. *arXiv preprint arXiv:2207.10342*, 2022.

[70] Li Dong and Mirella Lapata. Language to logical form with neural attention, 2016.

[71] Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. Self-collaboration code generation via chatgpt. *arXiv preprint arXiv:2304.07590*, 2023.

[72] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

[73] Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

[74] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

[75] Kevin Ellis, Maxwell Nye, Yewen Pu, Felix Sosa, Josh Tenenbaum, and Armando Solar-Lezama. Write, execute, assess: Program synthesis with a repl, 2019.

[76] Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B Tenenbaum. Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 835–850, 2021.

[77] Erich Elsen, Augustus Odena, Maxwell Nye, Sağnak Taşırlar, Tri Dao, Curtis Hawthorne, Deepak Moparthi, and Arushi Somani. Releasing Persimmon-8B, 2023.

[78] Angela Fan, Yacine Jernite, Ethan Perez, David Grangier, Jason Weston, and Michael Auli. ELI5: Long form question answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3558–3567, Florence, Italy, July 2019. Association for Computational Linguistics.

[79] Shangbin Feng, Chan Young Park, Yuhan Liu, and Yulia Tsvetkov. From pretraining data to language models to downstream tasks: Tracking the trails of political biases leading to unfair nlp models. *arXiv preprint arXiv:2305.08283*, 2023.

[80] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. Codebert: A pre-trained model for programming and natural languages, 2020.

[81] Charles Fernyhough. Vygotsky, luria, and the social brain. *Self and social regulation: Social interaction and the development of social understanding and executive functions*, pages 56–79, 2010.

[82] Meire Fortunato, Melissa Tan, Ryan Faulkner, Steven Hansen, Adrià Puigdomènech Badia, Gavin Buttimore, Charles Deck, Joel Z Leibo, and Charles Blundell. Generalization of reinforcement learners with working and episodic memory. *Advances in neural information processing systems*, 32, 2019.

[83] Daocheng Fu, Xin Li, Licheng Wen, Min Dou, Pinlong Cai, Botian Shi, and Y. Qiao. Drive like a human: Rethinking autonomous driving with large language models. *ArXiv*, abs/2307.07162, 2023.

[84] Deep Ganguli, Amanda Askell, Nicholas Schiefer, Thomas Liao, Kamilė Lukošiūtė, Anna Chen, Anna Goldie, Azalia Mirhoseini, Catherine Olsson, Danny Hernandez, et al. The capacity for moral self-correction in large language models. *arXiv preprint arXiv:2302.07459*, 2023.

[85] Chen Gao, Xiaochong Lan, Zhihong Lu, Jinzhu Mao, Jinghua Piao, Huandong Wang, Depeng Jin, and Yong Li. S3: Social-network simulation system with large language model-empowered agents. *arXiv preprint arXiv:2307.14984*, 2023.

[86] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser,

and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling, 2020.

[87] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models, 2023.

[88] Tianyu Gao, Adam Fisch, and Danqi Chen. Making pre-trained language models better few-shot learners. *arXiv preprint arXiv:2012.15723*, 2020.

[89] Samuel J Gershman, Eric J Horvitz, and Joshua B Tenenbaum. Computational rationality: A converging paradigm for intelligence in brains, minds, and machines. *Science*, 349(6245):273–278, 2015.

[90] Amelia Glaese, Nat McAleese, Maja Trebacz, John Aslanides, Vlad Firoiu, Timo Ewalds, Maribeth Rauh, Laura Weidinger, Martin Chadwick, Phoebe Thacker, Lucy Campbell-Gillingham, Jonathan Uesato, Po-Sen Huang, Ramona Comanescu, Fan Yang, Abigail See, Sumanth Dathathri, Rory Greig, Charlie Chen, Doug Fritz, Jaume Sanchez Elias, Richard Green, Soňa Mokrá, Nicholas Fernando, Boxi Wu, Rachel Foley, Susannah Young, Iason Gabriel, William Isaac, John Mellor, Demis Hassabis, Koray Kavukcuoglu, Lisa Anne Hendricks, and Geoffrey Irving. Improving alignment of dialogue agents via targeted human judgements, 2022.

[91] Thomas L Griffiths. Understanding human intelligence through human limitations. *Trends in Cognitive Sciences*, 24(11):873–883, 2020.

[92] Jiatao Gu, Yong Wang, Kyunghyun Cho, and Victor OK Li. Search engine guided neural machine translation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[93] Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. *arXiv preprint arXiv:2305.14909*, 2023.

[94] Guidance. Guidance, 2023.

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

[96] Izzeddin Gur, Natasha Jaques, Kevin Malta, Manoj Tiwari, Honglak Lee, and Aleksandra Faust. Adversarial Environment Generation for Learning to Navigate the Web. *arXiv preprint arXiv:2103.01991*, 2021.

[97] Izzeddin Gur, Ulrich Rueckert, Aleksandra Faust, and Dilek Hakkani-Tur. Learning to Navigate the Web. *arXiv preprint arXiv:1812.09195*, 2018.

[98] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938, 2020.

[99] Pouya Hamadanian, Behnaz Arzani, Sadjad Fouladi, Siva Kesava Reddy Kakarla, Rodrigo Fonseca, Denizcan Billor, Ahmad Cheema, Edet Nkposong, and Ranveer Chandra. A holistic view of ai-driven network incident management. In *ACM Workshop on Hot Topics in Networks*, 2023.

[100] Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Tobias Pfaff, Theophane Weber, Lars Buesing, and Peter W Battaglia. Combining q-learning and search with amortized value estimates. In *International Conference on Learning Representations*, 2019.

[101] Austin W. Hanjie, Victor Zhong, and Karthik Narasimhan. Grounding language to entities and dynamics for generalization in reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2021.

[102] Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*, 2023.

[103] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[104] Masum Hasan, Cengiz Ozel, Sammy Potter, and Ehsan Hoque. Sapien: Affective virtual agents powered by large language models. *arXiv preprint arXiv:2308.03022*, 2023.

[105] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, Christian Muise, Ronald Brachman, Francesca Rossi, and Peter Stone. *An introduction to the planning domain definition language*, volume 13. Springer, 2019.

[106] Matthew Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. Interactive fiction games: A colossal adventure. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7903–7910, 2020.

[107] Matthew Hausknecht, Ricky Loynd, Greg Yang, Adith Swaminathan, and Jason D Williams. Nail: A general interactive fiction agent. *arXiv preprint arXiv:1902.04259*, 2019.

[108] Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. Deep reinforcement learning with a natural language action space. *arXiv preprint arXiv:1511.04636*, 2015.

[109] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[110] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with apps. *NeurIPS*, 2021.

[111] Abdelhadi Hireche, Abdelkader Nasreddine Belkacem, Sadia Jamil, and Chao Chen. Newsgpt: Chatgpt integration for robot-reporter. *ArXiv*, abs/2311.06640, 2023.

[112] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, et al. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.

[113] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. *arXiv preprint arXiv:2312.08914*, 2023.

[114] Ehsan Hosseini-Asl, Bryan McCann, Chien-Sheng Wu, Semih Yavuz, and Richard Socher. A simple language model for task-oriented dialogue. *Advances in Neural Information Processing Systems*, 33:20179–20191, 2020.

[115] Alexandra Hotti, Riccardo Sven Risuleo, Stefan Magureanu, Aref Moradi, and Jens Lagergren. The Klarna Product Page Dataset: A RealisticBenchmark for Web Representation Learning. *arXiv preprint arXiv:2111.02168*, 2021.

[116] Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. *arXiv preprint arXiv:2210.11610*, 2022.

[117] Junjie Huang, Chenglong Wang, Jipeng Zhang, Cong Yan, Haotian Cui, Jeevana Priya Inala, Colin Clement, and Nan Duan. Execution-based evaluation for data science code generation models. In *Proceedings of the Fourth Workshop on Data Science with Human-in-the-Loop (Language Advances)*, pages 28–36, Abu Dhabi, United Arab Emirates (Hybrid), December 2022. Association for Computational Linguistics.

[118] Siyuan Huang, Zhengkai Jiang, Hao Dong, Yu Qiao, Peng Gao, and Hongsheng Li. Instruct2Act: Mapping Multi-modality Instructions to Robotic Actions with Large Language Model. *arXiv preprint arXiv:2305.11176*, 2023.

[119] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 162:9118–9147, 17–23 Jul 2022.

[120] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

[121] Peter C Humphreys, David Raposo, Toby Pohlen, Gregory Thornton, Rachita Chhaparia, Alistair Muldal, Josh Abramson, Petko Georgiev, Alex Goldin, Adam Santoro, et al. A data-driven approach for learning to control computers. *arXiv preprint arXiv:2202.08137*, 2022.

[122] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. Codesearchnet challenge: Evaluating the state of semantic code search, 2020.

[123] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.

[124] Fergus Imrie, Paulius Rauba, and Mihaela van der Schaar. Redefining digital health interfaces with large language models. *ArXiv*, abs/2310.03560, 2023.

[125] Geoffrey Irving, Paul Christiano, and Dario Amodei. AI safety via debate. *arXiv preprint arXiv:1805.00899*, 2018.

[126] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*, 2021.

[127] Sheng Jia, Jamie Kiros, and Jimmy Ba. Dom-q-net: Grounded RL on Structured Language. *arXiv preprint arXiv:1902.07257*, 2019.

[128] Zhengbao Jiang, Jun Araki, Haibo Ding, and Graham Neubig. How can we know when language models know? on the calibration of language models for question answering. *Transactions of the Association for Computational Linguistics*, 9:962–977, 2021.

[129] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

[130] Zhijing Jin, Sydney Levine, Fernando Gonzalez Adauto, Ojasv Kamal, Maarten Sap, Mrinmaya Sachan, Rada Mihalcea, Joshua B. Tenenbaum, and Bernhard Schölkopf. When to make exceptions: Exploring language models as accounts of human moral judgment. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[131] Shi Jinxin, Zhao Jiabao, Wang Yilei, Wu Xingjiao, Li Jiawen, and He Liang. Cgmi: Configurable general multi-agent interaction framework. *arXiv preprint arXiv:2308.12503*, 2023.

[132] Randolph M Jones, John E Laird, Paul E Nielsen, Karen J Coulter, Patrick Kenny, and Frank V Koss. Automated intelligent pilots for combat flight simulation. *AI magazine*, 20(1):27–27, 1999.

[133] Jaehun Jung, Lianhui Qin, Sean Welleck, Faeze Brahman, Chandra Bhagavatula, Ronan Le Bras, and Yejin Choi. Maieutic prompting: Logically consistent reasoning with recursive explanations. *arXiv preprint arXiv:2205.11822*, 2022.

[134] Daniel Kahneman. *Thinking, fast and slow*. Macmillan, 2011.

[135] Daniel Kahneman, Shane Frederick, et al. Representativeness revisited: Attribute substitution in intuitive judgment. *Heuristics and biases: The psychology of intuitive judgment*, 49(49-81):74, 2002.

[136] Siddharth Karamcheti, Megha Srivastava, Percy Liang, and Dorsa Sadigh. Lila: Language-informed latent actions. In *CoRL*, pages 1379–1390, 2021.

[137] Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive NLP. *arXiv preprint arXiv:2212.14024*, 2022.

[138] Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks, 2023.

[139] James R Kirk and John E Laird. Interactive task learning for simple games. *Advances in Cognitive Systems*, 3(13-30):5, 2014.

[140] James R Kirk, Wray Robert, Peter Lindes, and John E. Laird. Improving Knowledge Extraction from LLMs for Robotic Task Learning through Agent Analysis. *arXiv preprint arXiv:2306.06770*, 2023.

[141] Kenneth R Koedinger, John R Anderson, William H Hadley, Mary A Mark, et al. Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, 8(1):30–43, 1997.

[142] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 35:22199–22213, 2022.

[143] Mojtaba Komeili, Kurt Shuster, and Jason Weston. Internet-augmented dialogue generation. *arXiv preprint arXiv:2107.07566*, 2021.

[144] 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.

[145] Shuvendu K. Lahiri, Aaditya Naik, Georgios Sakkas, Piali Choudhury, Curtis von Veh, Madanlal Musuvathi, Jeevana Priya Inala, Chenglong Wang, and Jianfeng Gao. Interactive code generation via test-driven user-intent formalization, 2022.

[146] Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Scott Wen tau Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data science code generation. *ArXiv*, abs/2211.11501, 2022.

[147] Cassidy Laidlaw, Stuart Russell, and Anca Dragan. Bridging rl theory and practice with the effective horizon. *arXiv preprint arXiv:2304.09853*, 2023.

[148] John E Laird. *The Soar cognitive architecture*. MIT press, 2019.

[149] John E Laird. Introduction to Soar. *arXiv preprint arXiv:2205.03854*, 2022.

[150] John E Laird, Allen Newell, and Paul S Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[151] John E Laird, Paul S Rosenbloom, and Allen Newell. Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, 1:11–46, 1986.

[152] John Edwin Laird, Keegan R Kinkade, Shiwali Mohan, and Joseph Z Xu. Cognitive robotics using the Soar cognitive architecture. In *CogRob @ AAAI*, 2012.

[153] Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people, 2016.

[154] Andrew Lampinen, Stephanie Chan, Andrea Banino, and Felix Hill. Towards mental time travel: a hierarchical memory for reinforcement learning agents. *Advances in Neural Information Processing Systems*, 34:28182–28195, 2021.

[155] LangChain. LangChain, 2022.

[156] Angeliki Lazaridou, Elena Gribovskaya, Wojciech Stokowiec, and Nikolai Grigorev. Internet-augmented language models through few-shot prompting for open-domain question answering. *ArXiv*, abs/2203.05115, 2022.

[157] Angeliki Lazaridou, Anna Potapenko, and Olivier Tieleman. Multi-agent Communication meets Natural Language: Synergies between Functional and Structural Language Learning. In *ACL*, 2020.

[158] Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems*, 35:21314–21328, 2022.

[159] Yann LeCun. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, 62, 2022.

[160] Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. KaggleDBQA: Realistic evaluation of text-to-SQL parsers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2261–2273, Online, August 2021. Association for Computational Linguistics.

[161] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[162] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[163] Belinda Z Li, William Chen, Pratyusha Sharma, and Jacob Andreas. Lampp: Language models as probabilistic priors for perception and action. *arXiv preprint arXiv:2302.02801*, 2023.

[164] Bowen Li, Wenhan Wu, Ziwei Tang, Lin Shi, John Yang, Jinyang Li, Shunyu Yao, Chen Qian, Binyuan Hui, Qicheng Zhang, et al. Devbench: A comprehensive benchmark for software development. *arXiv preprint arXiv:2403.08604*, 2024.

[165] Chunyuan Li, Zhe Gan, Zhengyuan Yang, Jianwei Yang, Linjie Li, Lijuan Wang, and Jianfeng Gao. Multimodal foundation models: From specialists to general-purpose assistants. *arXiv preprint arXiv:2309.10020*, 2023.

[166] Huayang Li, Yixuan Su, Deng Cai, Yan Wang, and Lemao Liu. A survey on retrieval-augmented text generation. *arXiv preprint arXiv:2202.01110*, 2022.

[167] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, and et al. Starcoder: may the source be with you!, 2023.

[168] Shuang Li, Xavier Puig, Chris Paxton, Yilun Du, Clinton Wang, Linxi Fan, Tao Chen, De-An Huang, Ekin Akyürek, Anima Anandkumar, Jacob Andreas, Igor Mordatch, Antonio Torralba, and Yuke Zhu. Pre-trained language models for interactive decision-making, 2022.

[169] Xiujun Li, Xi Yin, Chunyuan Li, Pengchuan Zhang, Xiaowei Hu, Lei Zhang, Lijuan Wang, Houdong Hu, Li Dong, Furu Wei, et al. Oscar: Object-semantics aligned pre-training for vision-language tasks. In *European Conference on Computer Vision*, pages 121–137. Springer, 2020.

[170] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Ré mi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, dec 2022.

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

[172] Paul Pu Liang, Chiyu Wu, Louis-Philippe Morency, and Ruslan Salakhutdinov. Towards understanding and mitigating social biases in language models. In *International Conference on Machine Learning*, pages 6565–6576, 2021.

[173] Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Zhaopeng Tu, and Shuming Shi. Encouraging divergent thinking in large language models through multi-agent debate. *arXiv preprint arXiv:2305.19118*, 2023.

[174] Falk Lieder and Thomas L Griffiths. Resource-rational analysis: Understanding human cognition as the optimal use of limited computational resources. *Behavioral and Brain Sciences*, 43:e1, 2020.

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

[176] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[177] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. Pyserini: An Easy-to-Use Python Toolkit to Support Replicable IR Research with Sparse and Dense Representationss. *arXiv preprint arXiv:2102.10073*, 2021.

[178] 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, May 2018. European Language Resources Association (ELRA).

[179] Peter Lindes and John E Laird. Toward integrating cognitive linguistics and cognitive language processing. In *Proceedings of the 14th International Conference on Cognitive Modeling (ICCM)*, 2016.

[180] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+p: Empowering large language models with optimal planning proficiency, 2023.

[181] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. LLM+P: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

[182] Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement Learning on Web Interfaces using Workflow-Guided Exploration. *arXiv preprint arXiv:1802.08802*, 2018.

[183] Hao Liu, Carmelo Sferrazza, and Pieter Abbeel. Languages are rewards: Hindsight finetuning using human feedback. *arXiv preprint arXiv:2302.02676*, 2023.

[184] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *NeurIPS*, 2023.

[185] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What Makes Good In-Context Examples for GPT-3 ? *arXiv preprint arXiv:2101.06804*, 2021.

[186] 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. *ACM Computing Surveys*, 55(9), 2023.

[187] Ruibo Liu, Jason Wei, Shixiang Shane Gu, Te-Yen Wu, Soroush Vosoughi, Claire Cui, Denny Zhou, and Andrew M. Dai. Mind's eye: Grounded language model reasoning through simulation. In *The Eleventh International Conference on Learning Representations*, 2023.

[188] Ruibo Liu, Ruixin Yang, Chenyan Jia, Ge Zhang, Denny Zhou, Andrew M Dai, Diyi Yang, and Soroush Vosoughi. Training socially aligned language models in simulated human society. *arXiv preprint arXiv:2305.16960*, 2023.

[189] LlamaIndex. LlamaIndex, 2023.

[190] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. Codexglue: A machine learning benchmark dataset for code understanding and generation. *CoRR*, abs/2102.04664, 2021.

[191] Ximing Lu, Sean Welleck, Peter West, Liwei Jiang, Jungo Kasai, Daniel Khashabi, Ronan Le Bras, Lianhui Qin, Youngjae Yu, Rowan Zellers, Noah A.

Smith, and Yejin Choi. Neurologic a*esque decoding: Constrained text generation with lookahead heuristics. In *North American Chapter of the Association for Computational Linguistics*, 2021.

[192] Aleksandr Romanovich Luria. Ls vygotsky and the problem of localization of functions. *Neuropsychologia*, 3(4):387–392, 1965.

[193] Lucy Ellen Lwakatare, Aiswarya Raj, Ivica Crnkovic, Jan Bosch, and Helena Holmström Olsson. Large-scale machine learning systems in real-world industrial settings: A review of challenges and solutions. *Information and software technology*, 127:106368, 2020.

[194] Zilin Ma, Yiyang Mei, and Zhaoyuan Su. Understanding the benefits and challenges of using large language model-based conversational agents for mental well-being support. *arXiv preprint arXiv:2307.15810*, 2023.

[195] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.

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

[197] Aman Madaan and Amir Yazdanbakhsh. Text and patterns: For effective chain of thought, it takes two to tango, 2022.

[198] Andrei Andreevich Markov. The theory of algorithms. *Trudy Matematicheskogo Instituta Imeni VA Steklova*, 42:3–375, 1954.

[199] Marcelo G Mattar and Nathaniel D Daw. Prioritized memory access explains planning and hippocampal replay. *Nature Neuroscience*, 21(11):1609–1617, 2018.

[200] Sahisnu Mazumder and Oriana Riva. FLIN: A Flexible Natural Language Interface for Web Navigation. *arXiv preprint arXiv:2010.12844*, 2020.

[201] James L McClelland, Felix Hill, Maja Rudolph, Jason Baldridge, and Hinrich Schütze. Extending machine language models toward human-level language understanding. *arXiv preprint arXiv:1912.05877*, 2019.

[202] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[203] Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*, 2023.

[204] Vincent Micheli and François Fleuret. Language models are few-shot butlers. *arXiv preprint arXiv:2104.07972*, 2021.

[205] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[206] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[207] Shiwali Mohan and John Laird. Learning goal-oriented hierarchical tasks from situated interactive instruction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

[208] Shiwali Mohan, Aaron H Mininger, James R Kirk, and John E Laird. Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, 2:113–130, 2012.

[209] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. WebGPT: Browser-Assisted Question-Answering with Human Feedback. *arXiv preprint arXiv:2112.09332*, 2021.

[210] Karthik Narasimhan, Regina Barzilay, and Tommi Jaakkola. Deep transfer in reinforcement learning by language grounding. In *Journal of Artificial Intelligence Research (JAIR)*, 2018.

[211] Karthik Narasimhan, Adam Yala, and Regina Barzilay. Improving Information Extraction by Acquiring External Evidence with Reinforcement Learning. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2355–2365, 2016.

[212] Anjali Narayan-Chen, Prashant Jayannavar, and Julia Hockenmaier. Collaborative dialogue in Minecraft. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5405–5415. Association for Computational Linguistics, 2019.

[213] Shelley Nason and John E Laird. Soar-RL: Integrating reinforcement learning with Soar. *Cognitive Systems Research*, 6(1):51–59, 2005.

[214] Allen Newell. Studies in problem solving: Subject 3 on the crypt-arithmetic task DONALD+ GERALD= ROBERT. Technical report, Carnegie Mellon University, 1967.

[215] Allen Newell. Physical symbol systems. *Cognitive science*, 4(2):135–183, 1980.

[216] Allen Newell. Précis of unified theories of cognition. *Behavioral and Brain Sciences*, 15(3):425–437, 1992.

[217] Allen Newell, Paul S Rosenbloom, and John E Laird. Symbolic architectures for cognition. *Foundations of cognitive science*, pages 93–131, 1989.

[218] Allen Newell, John C Shaw, and Herbert A Simon. Report on a general problem solving program. In *IFIP congress*, volume 256, page 64. Pittsburgh, PA, 1959.

[219] Allen Newell, Herbert Alexander Simon, et al. *Human problem solving*. Prentice-Hall, 1972.

[220] Khanh Nguyen and Hal Daumé III. Help, Anna! visual navigation with natural multimodal assistance via retrospective curiosity-encouraging imitation learning. *arXiv preprint arXiv:1909.01871*, 2019.

[221] Khanh Nguyen, Debadeepta Dey, Chris Brockett, and Bill Dolan. Vision-based navigation with language-based assistance via imitation learning with indirect intervention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12527–12537, 2019.

[222] Khanh X Nguyen, Yonatan Bisk, and Hal Daumé Iii. A framework for learning to request rich and contextually useful information from humans. In *International Conference on Machine Learning*, pages 16553–16568, July 2022.

[223] Khanh X Nguyen, Dipendra Misra, Robert Schapire, Miroslav Dudík, and Patrick Shafto. Interactive learning from activity description. In *International Conference on Machine Learning*, pages 8096–8108, 2021.

[224] Khanh Xuan Nguyen. Language models are bounded pragmatic speakers. In *First Workshop on Theory of Mind in Communicating Agents*, 2023.

[225] Thanh Tam Nguyen, Thanh Trung Huynh, Phi Le Nguyen, Alan Wee-Chung Liew, Hongzhi Yin, and Quoc Viet Hung Nguyen. A survey of machine unlearning. *arXiv preprint arXiv:2209.02299*, 2022.

[226] Ansong Ni, Srini Iyer, Dragomir Radev, Ves Stoyanov, Wen tau Yih, Sida I. Wang, and Xi Victoria Lin. Lever: Learning to verify language-to-code generation with execution, 2023.

[227] Daniel Ni. ScraperAPI, 2015.

[228] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. *ICLR*, 2023.

[229] Nils J. Nilsson. Shakey the robot. *Technical Note*, 1984.

[230] Rodrigo Nogueira and Kyunghyun Cho. End-to-End Goal-Driven Web Navigation. *Advances in Neural Information Processing Systems*, 29, 2016.

[231] Rodrigo Nogueira and Kyunghyun Cho. Task-Oriented Query Reformulation with Reinforcement Learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 574–583, 2017.

[232] Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. Document expansion by query prediction, 2019.

[233] Andrew M Nuxoll and John E Laird. Extending cognitive architecture with episodic memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1560–1564, 2007.

[234] 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, 2021.

[235] OpenAI. Function calling and other API updates, 2023.

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

[237] 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.

[238] Aishwarya Padmakumar, Jesse Thomason, Ayush Shrivastava, Patrick Lange, Anjali Narayan-Chen, Spandana Gella, Robinson Piramuthu, Gokhan Tur, and Dilek Hakkani-Tur. Teach: Task-driven embodied agents that chat. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 2017–2025, 2022.

[239] Norman Di Palo, Arunkumar Byravan, Leonard Hasenclever, Markus Wulfmeier, Nicolas Heess, and Martin Riedmiller. Towards a unified agent with foundation models. In *Workshop on Reincarnating Reinforcement Learning at ICLR 2023*, 2023.

[240] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Annual Meeting of the Association for Computational Linguistics*, 2002.

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

[242] Joon Sung Park, Joseph C O'Brien, Carrie J Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*, 2023.

[243] Panupong Pasupat, Tian-Shun Jiang, Evan Zheran Liu, Kelvin Guu, and Percy Liang. Mapping natural language commands to web elements. In *EMNLP*, 2018.

[244] Panupong Pasupat, Tian-Shun Jiang, Evan Zheran Liu, Kelvin Guu, and Percy Liang. Mapping natural language commands to web elements. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[245] Pat Pataranutaporn, Valdemar Danry, Joanne Leong, Parinya Punpongsanon, Dan Novy, Pattie Maes, and Misha Sra. AI-generated characters for supporting personalized learning and well-being. *Nature Machine Intelligence*, 3(12):1013–1022, 2021.

[246] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

[247] Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. Refiner: Reasoning feedback on intermediate representations, 2023.

[248] Andi Peng, Ilia Sucholutsky, Belinda Li, Theodore R. Sumers, Thomas L. Griffiths, Jacob Andreas, and Julie A. Shah. Language guided state abstractions. In *Workshop on Social Intelligence in Humans and Robots at RSS 2023*, 2023.

[249] Emil L Post. Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics*, 65(2):197–215, 1943.

[250] Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adria Puigdomenech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In *International conference on machine learning*, pages 2827–2836, 2017.

[251] Ruchir Puri, David S. Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, Veronika Thost, Luca Buratti, Saurabh Pujar, Shyam Ramji, Ulrich Finkler, Susan Malaika, and Frederick Reiss. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks, 2021.

[252] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[253] Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 2023.

[254] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

[255] Dorian Quelle and Alexandre Bovet. The perils & promises of fact-checking with large language models. *ArXiv*, abs/2310.13549, 2023.

[256] Morgan Quigley. Ros: an open-source robot operating system. In *IEEE International Conference on Robotics and Automation*, 2009.

[257] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. *OpenAI blog*, 2018.

[258] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[259] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[260] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. A generalist agent, 2022.

[261] Allen Z Ren, Anushri Dixit, Alexandra Bodrova, Sumeet Singh, Stephen Tu, Noah Brown, Peng Xu, Leila Takayama, Fei Xia, Zhenjia Xu, et al. Robots that ask for help: Uncertainty alignment for large language model planners. In *7th Annual Conference on Robot Learning*, 2023.

[262] Oscar J Romero, John Zimmerman, Aaron Steinfeld, and Anthony Tomasic. Synergistic integration of large language models and cognitive architectures for robust ai: An exploratory analysis. *arXiv preprint arXiv:2308.09830*, 2023.

[263] Armin Ronacher. Flask API, 2010.

[264] Ohad Rubin, Jonathan Herzig, and Jonathan Berant. Learning to retrieve prompts for in-context learning. *arXiv preprint arXiv:2112.08633*, 2021.

[265] Evan Russek, Daniel Acosta-Kane, Bas van Opheusden, Marcelo G Mattar, and Tom Griffiths. Time spent thinking in online chess reflects the value of computation. *PsyArXiv*, 2022.

[266] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education Limited London, 2013.

[267] Dorsa Sadigh, Anca D. Dragan, Shankar Sastry, and Sanjit A. Seshia. Active preference-based learning of reward functions. In Nancy M. Amato, Siddhartha S. Srinivasa, Nora Ayanian, and Scott Kuindersma, editors, *Robotics: Science and Systems XIII*, 2017.

[268] William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Long Ouyang, Jonathan Ward, and Jan Leike. Self-critiquing models for assisting human evaluators. *arXiv preprint arXiv:2206.05802*, 2022.

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

[270] Imanol Schlag, Sainbayar Sukhbaatar, Asli Celikyilmaz, Wen tau Yih, Jason Weston, Jürgen Schmidhuber, and Xian Li. Large language model programs, 2023.

[271] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. Machine Learning: The High Interest Credit Card of Technical Debt. In *SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)*, 2014.

[272] Vibhor Sharma, Monika Goyal, and Drishti Malik. An intelligent behaviour shown by chatbot system. *International Journal of New Technology and Research*, 3(4):263312, 2017.

[273] Freda Shi, Daniel Fried, Marjan Ghazvininejad, Luke Zettlemoyer, and Sida I. Wang. Natural language to code translation with execution, 2022.

[274] Quan Shi, Michael Tang, Karthik Narasimhan, and Shunyu Yao. Can language models solve olympiad programming? *arXiv preprint arXiv:2404.10952*, 2024.

[275] Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of Bits: An Open-Domain platform for web-based agents. In *International Conference on Machine Learning*, pages 3135–3144. PMLR, 2017.

[276] Noah Shinn, Beck Cassano, Federico Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

[277] 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 *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749, 2020.

[278] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

[279] Kurt Shuster, Mojtaba Komeili, Leonard Adolphs, Stephen Roller, Arthur D. Szlam, and Jason Weston. Language models that seek for knowledge: Modular search & generation for dialogue and prompt completion. *ArXiv*, abs/2203.13224, 2022.

[280] Kurt Shuster, Jing Xu, Mojtaba Komeili, Da Ju, Eric Michael Smith, Stephen Roller, Megan Ung, Moya Chen, Kushal Arora, Joshua Lane, Morteza Behrooz, William Ngan, Spencer Poff, Naman Goyal, Arthur Szlam, Y-Lan Boureau, Melanie Kambadur, and Jason Weston. Blenderbot 3: a deployed conversational agent that continually learns to responsibly engage, 2022.

[281] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[282] Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B Tenenbaum, Leslie Pack Kaelbling, and Michael Katz. Generalized Planning in PDDL Domains with Pretrained Large Language Models. *arXiv preprint arXiv:2305.11014*, 2023.

[283] Tom Silver, Varun Hariprasad, Reece S Shuttleworth, Nishanth Kumar, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Pddl planning with pretrained large language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

[284] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530, 2023.

[285] Steven A Sloman. The empirical case for two systems of reasoning. *Psychological bulletin*, 119(1):3, 1996.

[286] Keith E Stanovich. *Who is rational? Studies of individual differences in reasoning*. Psychology Press, 1999.

[287] Yu Su, Ahmed Hassan Awadallah, Madian Khabsa, Patrick Pantel, Michael Gamon, and Mark Encarnacion. Building Natural Language Interfaces to Web APIs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 177–186, 2017.

[288] Yu Su, Ahmed Hassan Awadallah, Miaosen Wang, and Ryen W White. Natural Language Interfaces with Fine-Grained User Interaction: A Case Study on Web APIs. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 855–864, 2018.

[289] Theodore Sumers, Robert Hawkins, Mark K Ho, Tom Griffiths, and Dylan Hadfield-Menell. How to talk so AI will learn: Instructions, descriptions, and autonomy. *Advances in Neural Information Processing Systems*, 35:34762–34775, 2022.

[290] Theodore Sumers, Kenneth Marino, Arun Ahuja, Rob Fergus, and Ishita Dasgupta. Distilling internet-scale vision-language models into embodied agents. In *Proceedings of the 40th International Conference on Machine Learning*, pages 32797–32818, 2023.

[291] Theodore R Sumers, Mark K Ho, Robert D Hawkins, Karthik Narasimhan, and Thomas L Griffiths. Learning rewards from linguistic feedback. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6002–6010, 2021.

[292] Theodore R Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L Griffiths. Cognitive architectures for language agents. *arXiv preprint arXiv:2309.02427*, 2023.

[293] Ron Sun. Desiderata for cognitive architectures. *Philosophical Psychology*, 17(3):341–373, 2004.

[294] Yuqian Sun, Xingyu Li, Ze Gao, Ze Gao, Shunyu Yao, Jun Peng, Noura Howell, Tristan Braud, Chang Hee Lee, and Ali Asadipour. ORIBA: Supporting artistic development of original characters with conversational ai agents. *In Submission to CHI*, 2023.

[295] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[296] Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. Proofwriter: Generating implications, proofs, and abductive statements over natural language. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3621–3634, 2021.

[297] Ronen Tamari, Chen Shani, Tom Hope, Miriam R L Petruck, Omri Abend, and Dafna Shahaf. Language (re)modelling: Towards embodied language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6268–6281, Online, July 2020. Association for Computational Linguistics.

[298] Milind Tambe, W Lewis Johnson, Randolph M Jones, Frank Koss, John E Laird, Paul S Rosenbloom, and Karl Schwamb. Intelligent agents for interactive simulation environments. *AI magazine*, 16(1):15–15, 1995.

[299] Michael Tang, Shunyu Yao, John Yang, and Karthik Narasimhan. Referral augmentation for zero-shot information retrieval. *arXiv preprint arXiv:2305.15098*, 2023.

[300] Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. ToolAlpaca: Generalized Tool Learning for Language Models with 3000 Simulated Cases. *arXiv preprint arXiv:2306.05301*, 2023.

[301] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[302] Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew Walter, Ashis Banerjee, Seth Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 25, pages 1507–1514, 2011.

[303] Jesse Thomason, Michael Murray, Maya Cakmak, and Luke Zettlemoyer. Vision-and-dialog navigation. In *Conference on Robot Learning*, pages 394–406. PMLR, 2020.

[304] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. Fever: a large-scale dataset for fact extraction and verification. *arXiv preprint arXiv:1803.05355*, 2018.

[305] 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 arXiv:2105.13231*, 2021.

[306] Alan Mathison Turing et al. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936.

[307] Klemen Tusar. sqlite3mysql, 2018.

[308] Jens Tuyls, Shunyu Yao, Sham Kakade, and Karthik Narasimhan. Multi-stage episodic control for strategic exploration in text games. In *ICLR*, 2022.

[309] Carnegie Mellon University. picoCTF, 2013.

[310] Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). *arXiv preprint arXiv:2206.10498*, 2022.

[311] Siddharth Verma, Justin Fu, Sherry Yang, and Sergey Levine. Chai: A chatbot ai for task-oriented dialogue with offline reinforcement learning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4471–4491, 2022.

[312] Lev S Vygotsky. Thinking and speech. *The collected works of LS Vygotsky*, 1:39–285, 1987.

[313] Eric Wallace, Nicholas Tomlin, Albert Xu, Kevin Yang, Eshaan Pathak, Matthew Ginsberg, and Dan Klein. Automated crossword solving. *arXiv preprint arXiv:2205.09665*, 2022.

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

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

[316] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models, 2023.

[317] Liang Wang, Nan Yang, and Furu Wei. Query2doc: Query expansion with large language models. *arXiv preprint arXiv:2303.07678*, 2023.

[318] Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? *arXiv preprint arXiv:2203.07540*, 2022.

[319] Sida I Wang, Percy Liang, and Christopher D Manning. Learning language games through interaction. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2368–2378, 2016.

[320] Xiao Wang, Craig Macdonald, and Iadh Ounis. Deep reinforced query reformulation for information retrieval. *arXiv preprint arXiv:2007.07987*, 2020.

[321] Xin Wang, Yasheng Wang, Yao Wan, Fei Mi, Yitong Li, Pingyi Zhou, Jin Liu, Hao Wu, Xin Jiang, and Qun Liu. Compilable neural code generation with compiler feedback, 2022.

[322] Xingyao Wang, Hao Peng, Reyhaneh Jabbarvand, and Heng Ji. Leti: Learning to generate from textual interactions, 2023.

[323] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2022.

[324] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. Rationale-augmented ensembles in language models. *arXiv preprint arXiv:2207.00747*, 2022.

[325] 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, 2021.

[326] Zekun Wang, Ge Zhang, Kexin Yang, Ning Shi, Wangchunshu Zhou, Shaochun Hao, Guangzheng Xiong, Yizhi Li, Mong Yuan Sim, Xiuying Chen, Qingqing Zhu, Zhenzhu Yang, Adam Nik, Qi Liu, Chenghua Lin, Shi Wang, Ruibo Liu, Wenhu Chen, Ke Xu, Dayiheng Liu, Yike Guo, and Jie Fu. Interactive natural language processing, 2023.

[327] Zhiruo Wang, Shuyan Zhou, Daniel Fried, and Graham Neubig. Execution-based evaluation for open-domain code generation, 2023.

[328] Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents, 2023.

[329] Zirui Wang, Jiahui Yu, Adams Wei Yu, Zihang Dai, Yulia Tsvetkov, and Yuan Cao. Simvlm: Simple visual language model pretraining with weak supervision. *arXiv preprint arXiv:2108.10904*, 2021.

[330] Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, Jack Rae, Piotr Mirowski, Joel Z Leibo, Adam Santoro, et al. Unsupervised predictive memory in a goal-directed agent. *arXiv preprint arXiv:1803.10760*, 2018.

[331] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022. Survey Certification.

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

[333] Lilian Weng. Llm-powered autonomous agents. *lilianweng.github.io*, Jun 2023.

[334] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[335] Alfred North Whitehead and Bertrand Russell. *Principia mathematica to* 56*, volume 2. Cambridge University Press, 1997.

[336] David E Wilkins. *Practical planning: extending the classical AI planning paradigm*. Elsevier, 2014.

[337] Kyle Williams, Seyyed Hadi Hashemi, and Imed Zitouni. Automatic Task Completion Flows from Web APIs. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1009–1012, 2019.

[338] Terry Winograd. Understanding natural language. *Cognitive psychology*, 3(1):1–191, 1972.

[339] Lionel Wong, Gabriel Grand, Alexander K Lew, Noah D Goodman, Vikash K Mansinghka, Jacob Andreas, and Joshua B Tenenbaum. From word models to world models: Translating from natural language to the probabilistic language of thought. *arXiv preprint arXiv:2306.12672*, 2023.

[340] Robert E Wray, James R Kirk, John E Laird, et al. Language models as a knowledge source for cognitive agents. *arXiv preprint arXiv:2109.08270*, 2021.

[341] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023.

[342] Tongshuang Wu, Ellen Jiang, Aaron Donsbach, Jeff Gray, Alejandra Molina, Michael Terry, and Carrie J Cai. Promptchainer: Chaining large language model prompts through visual programming. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, pages 1–10, 2022.

[343] Tongshuang Wu, Michael Terry, and Carrie Jun Cai. AI chains: Transparent and controllable human-AI interaction by chaining large language model prompts. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, pages 1–22, 2022.

[344] Yuanzhen Xie, Tao Xie, Mingxiong Lin, WenTao Wei, Chenglin Li, Beibei Kong, Lei Chen, Chengxiang Zhuo, Bo Hu, and Zang Li. Olagpt: Empowering llms with human-like problem-solving abilities. *arXiv preprint arXiv:2305.16334*, 2023.

[345] Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, Xu Zhao, Min-Yen Kan, Junxian He, and Qizhe Xie. Decomposition enhances reasoning via self-evaluation guided decoding, 2023.

[346] Benfeng Xu, An Yang, Junyang Lin, Quan Wang, Chang Zhou, Yongdong Zhang, and Zhendong Mao. ExpertPrompting: Instructing Large Language Models to be Distinguished Experts. *arXiv preprint arXiv:2305.14688*, 2023.

[347] Binfeng Xu, Xukun Liu, Hua Shen, Zeyu Han, Yuhan Li, Murong Yue, Zhiyuan Peng, Yuchen Liu, Ziyu Yao, and Dongkuan Xu. Gentopia: A collaborative platform for tool-augmented llms. *arXiv preprint arXiv:2308.04030*, 2023.

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

[349] John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. Intercode: Standardizing and benchmarking interactive coding with execution feedback. In *NeurIPS Datasets and Benchmarks Track*, 2023.

[350] Sherry Yang, Ofir Nachum, Yilun Du, Jason Wei, Pieter Abbeel, and Dale Schuurmans. Foundation models for decision making: Problems, methods, and opportunities, 2023.

[351] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.

[352] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. In *NeurIPS*, 2022.

[353] Shunyu Yao and Karthik Narasimhan. Language agents in the digital world: Opportunities and risks. *princeton-nlp.github.io*, Jul 2023.

[354] Shunyu Yao, Karthik Narasimhan, and Matthew Hausknecht. Reading and acting while blindfolded: The need for semantics in text game agents. In *NAACL*, 2021.

[355] Shunyu Yao, Binghui Peng, Christos Papadimitriou, and Karthik Narasimhan. Self-attention networks can process bounded hierarchical languages. In *ACL*, 2021.

[356] Shunyu Yao, Rohan Rao, Matthew Hausknecht, and Karthik Narasimhan. Keep calm and explore: Language models for action generation in text-based games. In *EMNLP*, 2020.

[357] Shunyu Yao, Theodore Sumers, Karthik Narasimhan, and Thomas L Griffiths. Cognitive architectures for language agents. *arXiv preprint arXiv:2309.02427*, 2023.

[358] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *NeurIPS*, 2023.

[359] Shunyu Yao, Mo Yu, Yang Zhang, Karthik R Narasimhan, Joshua B Tenenbaum, and Chuang Gan. Linking emergent and natural languages via corpus transfer. In *ICLR*, 2022.

[360] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *ICLR*, 2023.

[361] Pengcheng Yin, Wen-Ding Li, Kefan Xiao, Abhishek Rao, Yeming Wen, Kensen Shi, Joshua Howland, Paige Bailey, Michele Catasta, Henryk Michalewski, Alex Polozov, and Charles Sutton. Natural language to code generation in interactive data science notebooks, 2022.

[362] Pengcheng Yin and Graham Neubig. Reranking for neural semantic parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4553–4559, Florence, Italy, July 2019. Association for Computational Linguistics.

[363] Tao Yu, Rui Zhang, He Yang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter S Lasecki, and Dragomir Radev. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases, 2019.

[364] 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, October-November 2018. Association for Computational Linguistics.

[365] Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Jonathan Kraft, Vincent Zhang, Caiming Xiong, Richard Socher, and Dragomir Radev. SParC: Cross-domain semantic parsing in context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4511–4523, Florence, Italy, July 2019. Association for Computational Linguistics.

[366] Xingdi Yuan, Jie Fu, Marc-Alexandre Côté, Yi Tay, Christopher Joseph Pal, and Adam Trischler. Interactive machine comprehension with information seeking agents. In *ACL*, 2020.

[367] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STaR: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

[368] Andy Zeng, Adrian Wong, Stefan Welker, Krzysztof Choromanski, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, Johnny Lee, Vincent

Vanhoucke, et al. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*, 2022.

[369] Lu Zeng, Sree Hari Krishnan Parthasarathi, and Dilek Hakkani-Tur. N-best hypotheses reranking for text-to-sql systems, 2022.

[370] Cedegao Zhang, Lionel Wong, Gabriel Grand, and Josh Tenenbaum. Grounded physical language understanding with probabilistic programs and simulated worlds. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 45, 2023.

[371] Kechi Zhang, Zhuo Li, Jia Li, Ge Li, and Zhi Jin. Self-edit: Fault-aware code editor for code generation, 2023.

[372] Shun Zhang, Zhenfang Chen, Yikang Shen, Mingyu Ding, Joshua B. Tenenbaum, and Chuang Gan. Planning with large language models for code generation, 2023.

[373] Tianjun Zhang, Fangchen Liu, Justin Wong, Pieter Abbeel, and Joseph E Gonzalez. The wisdom of hindsight makes language models better instruction followers. *arXiv preprint arXiv:2302.05206*, 2023.

[374] Tianyi Zhang, Tao Yu, Tatsunori B. Hashimoto, Mike Lewis, Wen tau Yih, Daniel Fried, and Sida I. Wang. Coder reviewer reranking for code generation, 2022.

[375] Yizhe Zhang, Siqi Sun, Michel Galley, Yen-Chun Chen, Chris Brockett, Xiang Gao, Jianfeng Gao, Jingjing Liu, and Bill Dolan. DialoGPT: Large-scale generative pre-training for conversational response generation. *arXiv preprint arXiv:1911.00536*, pages 270–278, 2019.

[376] Wenjia Joyce Zhao, Russell Richie, and Sudeep Bhatia. Process and content in decisions from memory. *Psychological Review*, 129(1):73, 2022.

[377] Victor Zhong, Austin W Hanjie, Sida Wang, Karthik Narasimhan, and Luke Zettlemoyer. SILG: The Multi-domain Symbolic Interactive Language Grounding Benchmark. *Advances in Neural Information Processing Systems*, 34:21505–21519, 2021.

[378] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning, 2017.

[379] Corey Y Zhou, Deborah Talmi, Nathaniel Daw, and Marcelo G Mattar. Episodic retrieval for model-based evaluation in sequential decision tasks, 2023.

[380] 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, 2022.

[381] Hao Zhou, Minlie Huang, Tianyang Zhang, Xiaoyan Zhu, and Bing Liu. Emotional chatting machine: Emotional conversation generation with internal and external memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[382] Shuyan Zhou, Uri Alon, Sumit Agarwal, and Graham Neubig. Codebertscore: Evaluating code generation with pretrained models of code, 2023.

[383] Shuyan Zhou, Uri Alon, Frank F Xu, Zhengbao Jiang, and Graham Neubig. Docprompting: Generating code by retrieving the docs. In *The Eleventh International Conference on Learning Representations*, 2022.

[384] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. WebArena: A Realistic Web Environment for Building Autonomous Agents. *arXiv preprint arXiv:2307.13854*, 2023.

[385] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022.

[386] Xinyu Zhu, Junjie Wang, Lin Zhang, Yuxiang Zhang, Ruyi Gan, Jiaxing Zhang, and Yujiu Yang. Solving math word problem via cooperative reasoning induced language models. *arXiv preprint arXiv:2210.16257*, 2022.

[387] Yunchang Zhu, Liang Pang, Yanyan Lan, Huawei Shen, and Xueqi Cheng. Adaptive information seeking for open-domain question answering. *arXiv preprint arXiv:2109.06747*, 2021.

[388] Shengyao Zhuang, Houxing Ren, Linjun Shou, Jian Pei, Ming Gong, Guido Zuccon, and Daxin Jiang. Bridging the gap between indexing and retrieval for differentiable search index with query generation. *arXiv preprint arXiv:2206.10128*, 2022.