UNIVERSITY OF CALIFORNIA SAN DIEGO

Reasoning beyond Imitating Human Language

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Data Science

by

Shibo Hao

Committee in charge:

Professor Zhiting Hu, Chair
Professor Mikhail Belkin
Professor Julian McAuley
Professor Lianhui Qin
Professor Jingbo Shang

2026

The Dissertation of Shibo Hao is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2026

DEDICATION

To my parents and to all who have supported this journey.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my advisor, Professor Zhiting Hu. His guidance, insight, and unwavering support have shaped every part of my doctoral journey. He gave me the freedom to pursue ambitious and unconventional ideas while always offering sharp questions and encouragement when I needed them most. I could not have asked for a better mentor, and the way he approaches research will continue to influence me throughout my career.

I am sincerely grateful to the members of my dissertation committee, Professors Mikhail Belkin, Julian McAuley, Lianhui Qin, and Jingbo Shang, for their time, thoughtful feedback, and encouragement. Their perspectives, spanning the theory and practice of machine learning, have strengthened this dissertation and broadened how I think about my own work.

During my time as a researcher at FAIR at Meta, I was fortunate to be mentored by Yuandong Tian and Jason Weston. Their mentorship was instrumental to the work on latent-space reasoning that appears in this dissertation, and our many discussions reshaped how I think about what reasoning in language models can be. I am also grateful to my collaborators there for making the experience both productive and enjoyable.

This dissertation would not exist without my collaborators and co-authors, whose generosity with ideas and time meant a great deal to me, and who filled these projects with long nights, shared excitement, and friendship. I am grateful to my labmates and to the broader research community at UC San Diego for creating an environment in which ideas could grow.

I gratefully acknowledge the support of the Bloomberg Data Science Ph.D. Fellowship, which gave me the freedom to pursue this research.

To my friends, near and far, thank you for the encouragement, the distractions, and the perspective that kept me whole through these years.

Finally, I owe everything to my parents, whose love and sacrifice have supported me long before this journey began. This dissertation is dedicated to them.

Chapter 2, in full, is a reprint of the material as it appears in Proceedings of the 2023

Conference on Empirical Methods in Natural Language Processing. Hao, Shibo; Gu, Yi; Ma, Haodi; Hong, Joshua Jiahua; Wang, Zhen; Wang, Daisy Zhe; Hu, Zhiting, Association for Computational Linguistics, 2023. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in full, is a reprint of the material as it appears in Advances in Neural Information Processing Systems 36. Hao, Shibo; Liu, Tianyang; Wang, Zhen; Hu, Zhiting, Curran Associates, Inc., 2023. The dissertation author was the primary investigator and author of this paper. This work was partially supported by DARPA ECOLE HR00112390063.

Chapter 4, in full, is a reprint of the material as it appears in Proceedings of the Conference on Language Modeling 2025. Hao, Shibo; Sukhbaatar, Sainbayar; Su, DiJia; Li, Xian; Hu, Zhiting; Weston, Jason; Tian, Yuandong, 2025. The work was conducted while the dissertation author was a visiting researcher at FAIR at Meta. The dissertation author was the primary investigator and author of this paper. The authors thank Jihoon Tack for valuable discussions.

VITA

2018–2022 Bachelor of Science in Computer Science, Peking University

2026 Doctor of Philosophy in Data Science, University of California San Diego

PUBLICATIONS

1. 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." In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

2. Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. "ToolkenGPT: Augmenting Frozen Language Models with Massive Tools via Tool Embeddings." In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, 2023.

3. Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. "Training Large Language Models to Reason in a Continuous Latent Space." In *Proceedings of the Conference on Language Modeling (COLM)*, 2025.

FIELDS OF STUDY

Major Field: Data Science

ABSTRACT OF THE DISSERTATION

Reasoning beyond Imitating Human Language

by

Shibo Hao

Doctor of Philosophy in Data Science

University of California San Diego, 2026

Professor Zhiting Hu, Chair

Large language models (LLMs) have achieved remarkable progress on a wide range of tasks by learning from vast collections of human-written text. This training paradigm instills in LLMs the ability to produce fluent, contextually appropriate language—but it also imposes a fundamental constraint: the model learns to imitate the *outcomes* of human reasoning, as recorded in text, rather than the cognitive processes of reasoning itself. As a result, LLMs face persistent difficulties with tasks that require structured exploration, grounded symbolic computation, or deliberate planning—capabilities that, in humans, operate partly outside of language.

This dissertation explores three distinct approaches to move language model reasoning

beyond the boundaries imposed by language imitation. First, we reframe LLM reasoning as a planning problem. Reasoning via Planning (RAP) repurposes the same language model as both a reasoning agent and a world model, enabling structured exploration of the reasoning space via Monte Carlo Tree Search. By explicitly simulating world states and evaluating rewards, RAP enables LLMs to backtrack from wrong reasoning steps and find high-quality solutions—achieving results that surpass GPT-4 with only a 33B-parameter model on plan generation benchmarks.

Second, we address the limitation that language text cannot teach symbolic computation. ToolkenGPT augments frozen LLMs with external cognitive tools by treating each tool as a special "toolken" embedding appended to the model's vocabulary. This lightweight approach—training only the toolken embeddings while keeping the LLM frozen—allows models to master arbitrarily many tools in a plug-and-play fashion, improving performance on numerical reasoning, knowledge-based question answering with over 200 API tools, and embodied plan generation.

Third, we question whether language tokens are the right medium for reasoning at all. Chain of Continuous Thought (COCONUT) trains LLMs to reason in a continuous latent space by feeding the last hidden state directly back as the next input embedding, bypassing the language model head. This simple modification gives rise to an emergent breadth-first-search-like reasoning pattern: the continuous thoughts can encode multiple candidate reasoning paths simultaneously, allowing the model to delay commitment and converge on correct answers with fewer tokens than language-based chain-of-thought.

Together, these three works demonstrate a systematic progression away from language imitation as the sole basis for machine reasoning: from structured search over language reasoning paths, to grounding in symbolic external tools, to reasoning freed from language tokens entirely.

Chapter 1

Introduction

1.1 Reasoning by Imitating Human Language

The rise of large language models (LLMs) represents a watershed moment in artificial intelligence. Models such as GPT-4 [73], LLaMA [92], and their successors have demonstrated remarkable competence across diverse tasks—from writing assistance and code generation to scientific question answering and logical inference. Crucially, these capabilities emerge not from any explicit representation of reasoning rules, but from a single training objective applied to enormous corpora of human-written text: predict the next token.

This training paradigm places language at the center of everything. Text data is the medium through which world knowledge, linguistic structure, and—implicitly—reasoning patterns are transmitted to the model. When a mathematician writes a proof, or a programmer explains a debugging strategy, or a teacher works through a word problem, the language they produce captures the *outcome* of their cognitive process. LLMs are trained to reproduce that output, and in doing so acquire a form of apparent reasoning ability.

A compelling illustration of this mechanism is chain-of-thought (CoT) prompting [97]: by prompting LLMs to generate intermediate reasoning steps before a final answer, one can extract substantially better performance on multi-step problems. The model produces language that *looks like* the reasoning process a human would write down—and, because that process is correlated with correct answers in the training data, it often leads to correct answers. This is

language imitation at its most powerful.

Yet the mechanism is also the limitation. The web corpus records how humans verbalize their thinking, not how they actually think. Neuroimaging studies consistently show that the brain's language network remains largely *inactive* during many reasoning tasks [24]. Clinical evidence reveals that people with aphasia—who have lost the ability to use language—can still engage in sophisticated non-verbal reasoning. And linguists have long recognized that syntactic and lexical constraints in natural language are shaped by the demands of communication, not optimized for logical inference. Language, in short, is a lossy and distorted record of thought.

This mismatch has concrete consequences for LLMs. Models trained purely on language text:

- generate reasoning traces autoregressively—one token at a time, with no ability to look ahead, backtrack, or explore alternative paths [94];
- cannot reliably perform symbolic operations (arithmetic, database queries, program execution) that humans routinely delegate to external tools, because such operations leave only their results in text [19];
- are forced to express every intermediate reasoning step as a natural language token, even though most tokens in a reasoning chain serve fluency rather than reasoning [30].

1.2 Reasoning beyond Language Imitation

This dissertation investigates three approaches to reasoning that move beyond the constraints of language imitation. Each approach targets a different bottleneck.

Approach 1: Simulating the Structure of Human Deliberate Thought

Human deliberate reasoning—what cognitive scientists call "System 2" thinking [50]—involves constructing an internal model of the world, simulating possible future states, evaluating outcomes, and backtracking from dead ends. These operations are not faithfully encoded in

language text, which typically records only the final, cleaned-up output of the deliberative process.

Chapter 2 addresses this by reframing LLM reasoning as a *planning* problem. The key insight is that the same LLM can be repurposed as a *world model*: given the current state of a reasoning problem, the LLM predicts what the world would look like after an action is taken. Monte Carlo Tree Search (MCTS) then uses this world model to systematically explore the reasoning space—sampling action sequences, evaluating their rewards, and backpropagating value estimates to refine earlier decisions. This gives the LLM the ability to plan, explore, and recover from mistakes, producing reasoning behavior that is structurally closer to human deliberation. On plan generation benchmarks, Reasoning via Planning (RAP) with a 33B-parameter model surpasses CoT with GPT-4 by 33% relative improvement.

Approach 2: Augmenting Cognition with External Tools

A second limitation is that language text is a poor medium for transmitting grounded, symbolic competencies. A textbook may describe how to compute the least common multiple of two numbers, but the relevant skill is computational, not linguistic. Similarly, factual knowledge is better accessed from a structured database than retrieved from model weights. Human experts routinely use external tools—calculators, search engines, code interpreters, knowledge bases—to augment their reasoning beyond what language-based thought alone can achieve.

Chapter 3 proposes ToolkenGPT, which augments frozen LLMs with massive external tools by treating each tool as a special token—a "toolken"—appended to the model's vocabulary. A lightweight toolken embedding (with the LLM parameters frozen) is trained for each tool. At inference time, when the model generates a toolken, it switches to argument-generation mode, executes the tool, and injects the result back into the generation stream. This approach scales to hundreds of tools without fine-tuning the LLM, and substantially outperforms in-context learning baselines on numerical reasoning, knowledge-based question answering with 234 tools, and embodied plan generation.

Approach 3: Freeing Reasoning from Language Tokens

The most fundamental departure from language imitation is to ask: does reasoning need to happen in language at all? Every step of CoT reasoning must be expressed as a natural language word token, yet language tokens are optimized for communication, not for internal computation. Many tokens in a reasoning chain are purely for fluency. Worse, the commitment to one token at a time in autoregressive decoding prevents the model from maintaining multiple hypotheses simultaneously.

Chapter 4 introduces Chain of Continuous Thought (COCONUT), which replaces language tokens in the reasoning chain with continuous thought vectors. Rather than decoding the hidden state into a discrete word token, COCONUT feeds the last hidden state directly back as the next input embedding—entirely bypassing the language model head. The model reasons in a high-dimensional continuous latent space and only decodes into language when producing the final answer. A multi-stage training curriculum, which progressively replaces language reasoning steps with continuous thoughts, enables effective learning. Remarkably, continuous thoughts develop an emergent breadth-first-search-like property: they simultaneously encode multiple candidate next reasoning steps, allowing the model to explore a tree of possibilities within a single forward pass rather than committing to a single greedy path.

1.3 Summary of Contributions

The three works in this dissertation address different aspects of the same fundamental challenge: enabling language models to reason in ways that go beyond imitating the surface form of human language.

Reasoning via Planning (RAP) (Chapter 2) introduces a framework where LLM reasoning is formulated as a Markov decision process, with the LLM simultaneously serving as the reasoning agent and the world model. MCTS enables structured exploration of the reasoning space. Key contributions include: (1) a general formulation of state, action, reward, and world

model applicable to diverse reasoning tasks; (2) empirical demonstration that structured planning substantially outperforms CoT on plan generation, math reasoning, and logical reasoning; and (3) evidence that RAP with LLaMA-33B surpasses GPT-4 on plan generation—establishing inference-time search as a powerful axis for scaling reasoning.

ToolkenGPT (Chapter 3) introduces a lightweight method for augmenting frozen LLMs with massive external tools. Key contributions include: (1) the toolken embedding framework, which treats tools as vocabulary extensions and learns only their embeddings; (2) a training procedure that can use either supervised demonstrations or synthetic data; and (3) demonstration that ToolkenGPT outperforms fine-tuning and in-context learning baselines on three diverse tasks, scaling gracefully to hundreds of tools.

COCONUT (Chapter 4) introduces a new reasoning paradigm in continuous latent space. Key contributions include: (1) the COCONUT architecture, which feeds continuous hidden states back as input embeddings; (2) a multi-stage curriculum training procedure; (3) an analysis of the emergent latent tree search behavior; and (4) empirical results showing better efficiency and accuracy tradeoffs compared to language-based chain-of-thought reasoning.

1.4 Dissertation Organization

The remainder of this dissertation is organized as follows. Chapter 2 presents Reasoning via Planning. Chapter 3 presents ToolkenGPT. Chapter 4 presents COCONUT. Chapter 5 synthesizes the three contributions, discusses their connections, and outlines directions for future research.

Chapter 2

Reasoning via Planning: LLM Reasoning as Planning with a World Model

2.1 Introduction

Large language models (LLMs) have exhibited emergent reasoning abilities across a wide range of tasks [8, 13, 73]. Recent approaches further boost their ability by prompting LLMs to generate intermediate reasoning steps, e.g., Chain-of-Thought (CoT) [97], or to answer a series of subquestions, e.g., least-to-most prompting [104]. However, LLMs still face difficulties with tasks that humans find easy. For example, in creating action plans to move blocks to a target state, GPT-3 [8] achieves a success rate of only 1%, compared to 78% for humans [94]; these models also struggle with complex tasks that require multiple steps of math, logical, or commonsense reasoning [41, 70].

Humans possess an internal **world model**, a mental representation of the environment [28, 48, 49], which enables humans to simulate actions and their effects on the world's state for deliberate **planning** during complex tasks of motor control, imagery, inference, and decision making [7, 55, 91]. For example, to make an action plan towards a goal, planning with the world model involves exploring various alternative courses of actions, assessing the likely outcomes by rolling out possible future scenarios, and iteratively refining the plan based on the assessment [38, 45]. This is in stark contrast to the current LLM reasoning, which instinctively generates a reasoning trace in an autoregressive manner. In particular, we identify several key

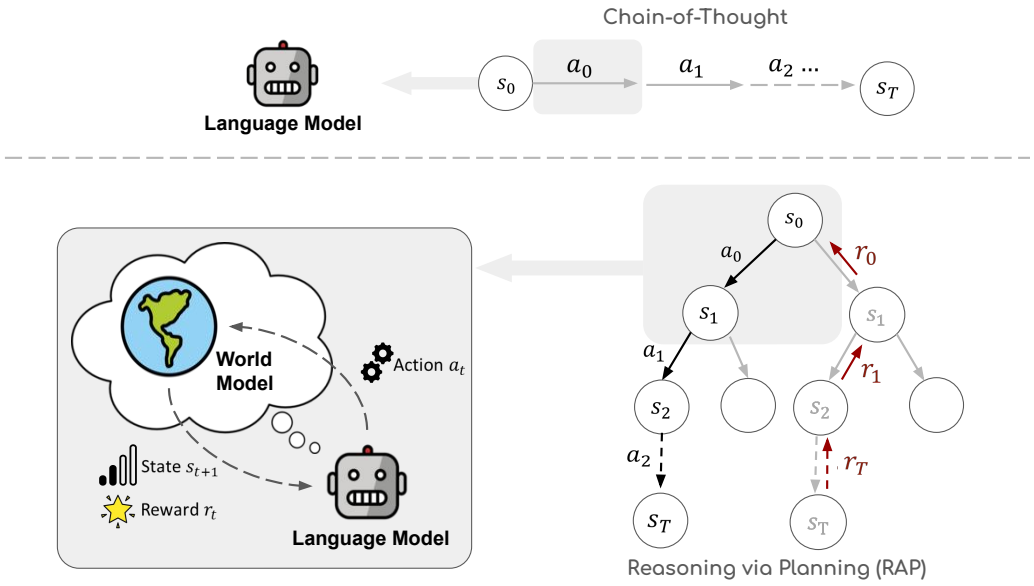

Figure 2.1. An overview of Reasoning via Planning (RAP). Compared with previous LLM reasoning methods like Chain-of-Thought [97], we explicitly model the world state from a world model (repurposed from the language model), and leverage advanced planning algorithms to solve the reasoning problems.

limitations of current reasoning with LLMs, including **(1)** the lack of an internal world model to simulate the *state* of the world (e.g., the configuration of blocks, the values of intermediate variables), which is the foundation of human planning; **(2)** the absence of a *reward* mechanism to assess and guide the reasoning towards the desired state; and due to both limitations, **(3)** the incapability of balancing *exploration vs. exploitation* to efficiently explore the vast reasoning space.

To address these limitations, this chapter proposes a new framework, **Reasoning via Planning (RAP)**, that enables LLMs to reason in a manner close to humans' conscious planning. RAP augments the LLM with a world model, and reasons with principled planning (specifically *Monte Carlo Tree Search*, MCTS) to produce high-reward reasoning traces after efficient exploration (Figure 2.1). Notably, we acquire the world model by repurposing the LLM itself with appropriate prompts. During the reasoning, the LLM strategically builds a reasoning tree by iteratively considering the most promising reasoning steps (*actions*) and using the world model (the same, repurposed LLM) to look ahead for future outcomes. The estimated future rewards

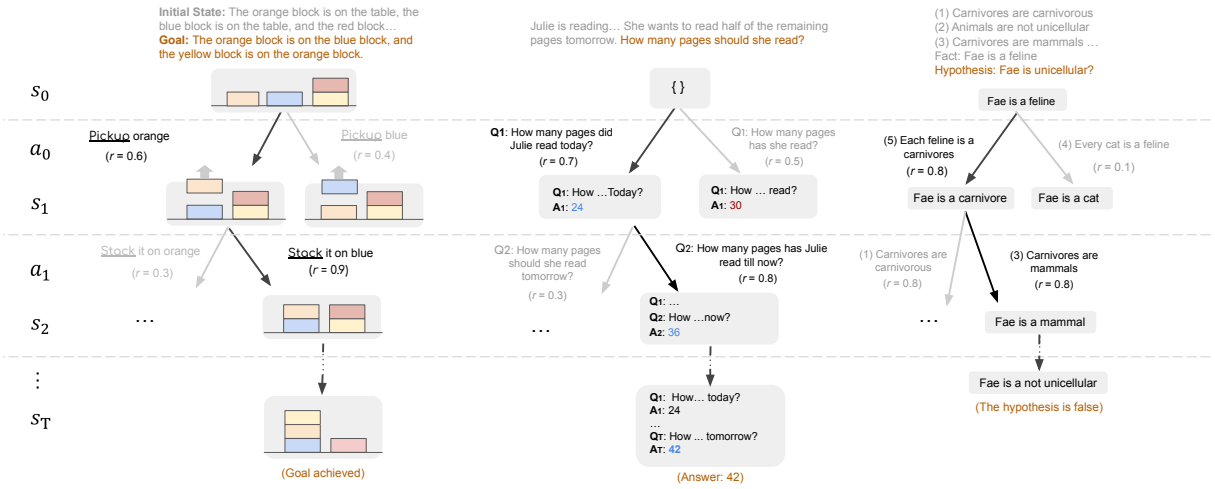

Figure 2.2. RAP for plan generation in Blocksworld (left), math reasoning in GSM8K (middle), and logical reasoning in PrOntoQA (right). The world model formulation enables a flexible, task-specific definition of states and actions.

are then back-propagated to update the LLM's beliefs about the current reasoning steps, guiding it to refine the reasoning by exploring better alternatives. Our MCTS-based planning effectively maintains a proper balance between exploration (of unvisited reasoning traces) and exploitation (of the best reasoning steps identified so far).

We show RAP is a general framework applicable to a diverse range of challenging problems and achieves substantial improvements over recent popular LLM reasoning methods. For plan generation, particularly in 2/4/6-step problems of Blocksworld [95], RAP achieves an average success rate of 64% while CoT fails almost completely. Moreover, LLaMA-33B with RAP surpasses GPT-4 with CoT by 33% relative improvement. In the domains of mathematical reasoning, such as GSM8K [14], and logical inference exemplified by PrOntoQA [81], RAP also consistently improves over strong baselines, including CoT, least-to-most prompting, and their self-consistency variants.

2.2 Related Work

Reasoning with LLMs. LLM reasoning typically involves decomposing complex questions into sequential intermediate steps (a.k.a. chains) before producing the final answer, exemplified by

Chain-of-Thought (CoT) prompting and its variants [54, 97]. The basic CoT generates chains all at once and can induce additional errors as the step count increases. Self-Consistency [96] samples multiple chains to choose the best answer via majority voting. Least-to-most prompting [104] reduces the question into simpler subquestions and answers them sequentially. Similar to our reward formulation, recent works have explored self-evaluation approaches to provide feedback for intermediate steps [76, 86, 98]. Aligned with our state formulation, Li et al. [59] incorporate latent "situations" into LLMs, referring to the state of entities from the context. More relevantly, recent works have started to explore more complex structures guided by search algorithms. For instance, CoRe [106] fine-tunes a reasoning step generator and verifier for math word problems with MCTS for decoding. Concurrently to our work, Yao et al. [101] apply heuristic-based search, like depth-/breadth-first search, for better reasoning paths. However, none of the above methods formally introduce a world model, nor do they instantiate the reward and state in a unified framework. Compared with these search-guided methods, RAP is a more principled framework that combines a world model and reward with advanced planning.

Planning with LLMs. Planning, a central ability in intelligent agents, involves generating a series of actions to achieve a specific goal [10, 68]. Classical planning methods have been widely adopted in robotics and embodied environments [11, 47]. Recently, prompting LLMs to plan directly has gained attention and shown potential [18, 43, 89]. Moreover, based on LLMs' programming ability [63, 65], recent works first translate natural language instructions into executable programming languages, such as the Planning Domain Definition Language (PDDL), and run classical planning algorithms, e.g., LLM+P [63]. However, code-based planning is constrained by its narrow domains and the environment, while RAP can handle open-domain problems, such as math and logical reasoning.

World models and planning. Traditional reinforcement learning (RL) relies heavily on interaction with the environment. To improve sample efficiency, some works leverage a **world model** that predicts state transitions, and directly learn a policy within the world model [33]. With latent imagination in a world model, an RL agent can be trained to solve long-horizon

tasks [34, 35]. Recent years have witnessed successful applications of **planning** algorithms in RL, such as AlphaZero [88] and MuZero [83]. These algorithms are typically based on tree search and are designed to maintain a balance between exploration and exploitation. Compared with previous works, we use LLMs as world models and apply a planning algorithm to search for better reasoning paths, solving various open-domain reasoning tasks. To the best of our knowledge, RAP is the first general framework repurposing an LLM as a world model to tackle a broad range of LLM reasoning problems.

2.3 Reasoning via Planning (RAP)

In this section, we present the RAP framework that enables LLMs to strategically plan a coherent reasoning trace for solving a wide range of reasoning tasks. We first build the world model by repurposing the LLM with prompting (Section 2.3.1). The world model serves as the foundation for deliberate planning, allowing the LLM to plan ahead and seek out expected future outcomes. We then introduce the rewards for assessing each state during reasoning (Section 2.3.2). Guided by the world model and rewards, planning with MCTS efficiently explores the vast reasoning space and finds high-reward reasoning traces (Section 2.3.3). Finally, when multiple promising reasoning traces are acquired during planning, we introduce an aggregation method (Section 2.3.4) that yields an ensembled result.

2.3.1 Language Model as World Model

In general, a world model predicts the next *state* of the reasoning after applying an *action* to the current state [33, 67]. RAP enables us to instantiate the general concepts of state and action in different ways depending on the specific reasoning problem at hand (Figure 2.2). For example, in Blocksworld, it is natural to define a state as the configuration of blocks (described in natural language), and an action as a behavior of moving a block (e.g., ‘‘`pick up the orange block`’’). In a math reasoning problem, we use the state to represent the values of intermediate variables, and set an action to be a subquestion that drives the reasoning to derive new values. In

logical reasoning, a state is a fact we are focusing on, and an action is to choose a rule for the next deduction.

With the definition of state and action, the reasoning process can be described as a Markov decision process (MDP): given the current state s_t, $t = 0, 1, \ldots, T$, e.g., the initial state s_0, the LLM (as a reasoning agent) generates an action space by sampling from its generative distribution $a_t \sim p(a \mid s_t, c)$, where c is a proper prompt (e.g., in-context demonstrations). Once an action is chosen, the world model then predicts the next state s_{t+1} of the reasoning. Specifically, we repurpose the *same* LLM to obtain a state transition distribution $p(s_{t+1} \mid s_t, a_t, c')$, where c' is another prompt to guide the LLM to generate a state. For instance, in Blocksworld, the LLM (as the world model) generates text s_{t+1} to describe the new configuration of blocks, given the previous state s_t and the action a_t.

Continuing the process results in a reasoning trace, which consists of a sequence of interleaved states and actions $(s_0, a_0, s_1, \ldots, a_{T-1}, s_T)$. This differs from previous reasoning methods, such as Chain-of-Thought [97], where the intermediate reasoning steps consist of only a sequence of actions, e.g., ($a_0 =$ ``pick up the red block'', $a_1 =$ ``stack on the yellow block'', ...); see comparisons in Figure 2.1. Augmenting the reasoning with the (predicted) world states helps the LLM with a more grounded and coherent inference. Note that the full reasoning trace is simulated by the LLM itself (as a reasoning agent with an *internal* world model) without interacting with the *external* real environment. This resembles humans contemplating a possible plan in their minds. The capability of simulating future states, by introducing the world model, allows us to incorporate principled planning algorithms to efficiently explore the vast reasoning space.

2.3.2 Reward Design

During reasoning, we want to assess the feasibility and desirability of each reasoning step, and guide the reasoning based on the assessment (Section 2.3.3). The assessment of each reasoning step (i.e., applying an action a_t to the state s_t) is performed by a *reward* function

$r_t = r(s_t, a_t) \in \mathbb{R}$. Similar to the state and action, the reward function can be specified in different ways to accommodate any knowledge or preferences about the reasoning problem of interest. Here we introduce several common rewards applicable to different tasks and shown to be effective in our experiments.

Likelihood of the action. When an action is generated by the LLM conditioned on the in-context demonstration and the current state, the probability of the specific action reflects the LLM's preference. We thus incorporate the log probability of the action as a reward. This reward reflects the "instinct" of the LLM as an agent, and can also be used as a prior for which action to explore.

Confidence of the state. State prediction is nontrivial in some problems; e.g., in math reasoning (Figure 2.2, middle), given an action (i.e., a subquestion), the world model predicts the next state by answering the subquestion. We incorporate the confidence of the state (i.e., the answer in this case) as a reward. Specifically, we draw multiple sample answers from the world model, and use the proportion of the most frequent answer as the confidence. Higher confidence indicates that the state prediction is more consistent with the world knowledge of the LLM [37], which typically leads to a more reliable reasoning step.

Self-evaluation by the LLM. It is sometimes easier to recognize errors in reasoning than to avoid generating them in advance. Thus, it is beneficial to allow the LLM to criticize itself with the question "`Is this reasoning step correct?`", and use the next-word probability of the token "`Yes`" as a reward. The reward evaluates the LLM's own estimation of the correctness of reasoning. The specific questions used for self-evaluation can differ depending on the task.

Task-specific heuristics. RAP also allows us to flexibly plug in other task-specific heuristics into the reward function. For example, in plan generation for Blocksworld, we compare the predicted current configuration of blocks with the goal to calculate a reward (Section 2.4.1). This reward encourages the plan of movements to actively pace towards the target.

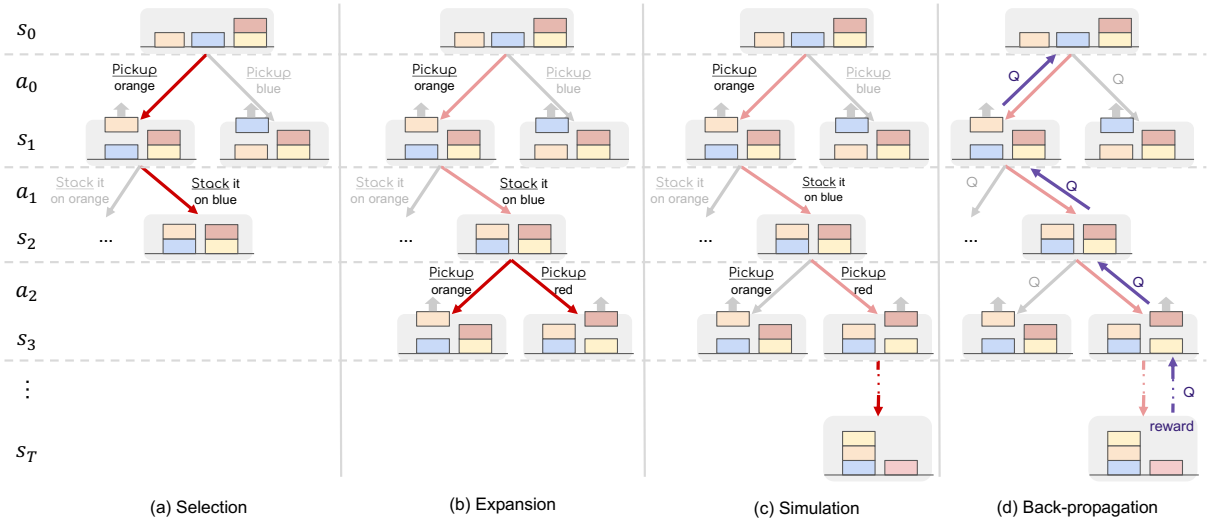

Figure 2.3. An illustration of the four phases in an iteration of MCTS planning (Section 2.3.3): selection, expansion, simulation, and back-propagation.

2.3.3 Planning with Monte Carlo Tree Search

Once equipped with the world model (Section 2.3.1) and rewards (Section 2.3.2), the LLM can reason with any planning algorithm. We adopt Monte Carlo Tree Search (MCTS) [15, 53], a powerful planning algorithm that strategically explores the space of reasoning trees and strikes a proper balance between exploration and exploitation to find high-reward reasoning traces efficiently.

Specifically, MCTS builds a reasoning tree iteratively, where each node represents a state, and each edge represents an action and the transition from the current state to the next state after applying the action (Figure 2.1). To guide the LLM agent to expand and explore the most promising nodes, the algorithm maintains a state-action value function $Q : \mathscr{S} \times \mathscr{A} \mapsto \mathbb{R}$, where $Q(s, a)$ estimates the *expected future reward* of taking action a in state s. Figure 2.3 illustrates the four operations performed in each iteration to expand the tree and update Q values. The process continues until a specified computational budget (e.g., the number of iterations) is reached, and the resulting traces are acquired from the tree. The full procedure is given in Algorithm 1.

Selection. The first phase selects a portion of the existing tree that is most promising for further

expansion. Starting from the root node (i.e., the initial state s_0), at each level of the tree the algorithm selects a child node as the next node, finishing when a leaf node of the current tree is reached. To balance between exploration (of less-visited nodes) and exploitation (of high-value nodes), we use the well-known Upper Confidence bounds applied to Trees (UCT) [53] to select each child node. Specifically, at node s, we select the action by considering both the Q value (for exploitation) and uncertainty (for exploration):

$$a^* = \arg\max_{a \in A(s)} \left[Q(s,a) + w\sqrt{\frac{\ln N(s)}{N(c(s,a))}} \right], \tag{2.1}$$

where $N(s)$ is the number of times node s has been visited in previous iterations, and $c(s,a)$ is the child node of applying a in state s. The less a child node was visited (i.e., the more uncertain we are about it), the higher the second term. The weight w controls the balance between exploration and exploitation.

Expansion. This phase expands the tree by adding new child nodes to the leaf node selected above. Given the state of the leaf node, we use the LLM (as agent) to sample d possible actions (e.g., subquestions in math reasoning), and then use the LLM (as world model) to predict the respective next state, resulting in d child nodes. If the selected leaf node is already a terminal node (the end of a reasoning chain), we skip expansion and jump to back-propagation.

Simulation. To estimate the expected future reward (Q value), this phase simulates the future of the current node using the world model. Starting from the current node, at each node s we create an action following a *roll-out policy* and use the world model to predict the next state. The roll-out continues until a terminal state is reached. In our experiments, for simplicity and reduced noise, we follow a process similar to expansion: generating d candidate actions and picking the one with the largest local reward, $a' = \arg\max_a r(s,a)$. For efficiency, we discard the computationally costly components in r (e.g., the confidence reward, which requires sampling the answer multiple times) during simulation.

Back-propagation. Once we reach a terminal state, we obtain a reasoning path from the root to

Algorithm 1. RAP-MCTS

Require: Initial state s_0, state transition function p_θ, reward function r_θ, action generator p_ϕ, number of actions d, depth limit L, number of roll-outs N, exploration weight w

Initialize memory of actions A, children c, rewards r, value function Q, visit counter $N(\cdot)$
for $n \leftarrow 0, \ldots, N-1$ **do**
  $t \leftarrow 0$
  while $N(s_t) > 0$ **do** ▷ Selection
    $N(s_t) \leftarrow N(s_t) + 1$
    $a_t \leftarrow \arg\max_{a \in A(s_t)} \left[Q(s_t, a) + w\sqrt{\ln N(s_t)/N(c(s_t, a))} \right]$
    $r_t \leftarrow r(s_t, a_t), \quad s_{t+1} \leftarrow c(s_t, a_t), \quad t \leftarrow t+1$
  end while
  while s_t is not terminal $\wedge\, t \leq L$ **do**
    for $i \leftarrow 1, \ldots, d$ **do** ▷ Expansion
      Sample $a_t^{(i)} \sim p_\phi(a \mid s_t)$, $s_{t+1}^{(i)} \sim p_\theta(s_t, a_t^{(i)})$, $r_t^{(i)} \sim r_\theta(s_t, a_t^{(i)})$
      Update $A(s_t)$, $c(s_t, a_t^{(i)})$, $r(s_t, a_t^{(i)})$
    end for
    $a_t \leftarrow \arg\max_{a \in A(s_t)} r(s_t, a)$ ▷ Simulation
    $r_t \leftarrow r(s_t, a_t), \quad s_{t+1} \leftarrow c(s_t, a_t), \quad t \leftarrow t+1$
  end while
  for $t' \leftarrow t, \ldots, 0$ **do** ▷ Back-propagation
    Update $Q(s_{t'}, a_{t'})$ with $\{r_{t'}, r_{t'+1}, \ldots, r_t\}$
  end for
end for

the terminal node. We then back-propagate the rewards along this path to update the Q value of each state-action pair. Specifically, we update $Q(s,a)$ by aggregating the rewards in all future steps of node s.

Once a predetermined number of iterations is reached, we terminate the algorithm and select the final reasoning trace from the constructed tree. There are various ways to do so: starting from the root and iteratively choosing the action with the highest Q value; directly selecting the path from the iteration that yielded the highest reward; or choosing the most-visited root-to-leaf path. In practice, we observed that the second strategy often yields the best results.

2.3.4 RAP-Aggregation

For problems such as math reasoning (Section 2.4.2) where only the final answer is required, RAP can produce multiple traces and answers from different MCTS iterations, which

are aggregated to produce the final answer. We refer to this mechanism as RAP-Aggregation. Note that problems like plan generation or logical inference require a complete reasoning trace as output; thus, RAP-Aggregation is not applied to them.

2.4 Experiments

In this section, we demonstrate the flexibility and effectiveness of RAP by applying it to three problem families: plan generation in an embodied environment (Section 2.4.1), mathematical reasoning (Section 2.4.2), and logical reasoning (Section 2.4.3). By default, we use the LLaMA-33B model [92] as the base LLM for both our methods and baselines, with a sampling temperature of 0.8. We primarily compare RAP with Chain-of-Thought (CoT) [97] and its variants like least-to-most prompting [104], as well as their self-consistency [96] variants. We also compare with GPT-4 [73] when computational resources allow.

2.4.1 Plan Generation

The plan generation task aims to produce a sequence of actions to achieve a given goal, possibly with additional constraints. The ability to generate plans is important for intelligent embodied agents, e.g., household robots [78].

Task setup. We adapt and evaluate RAP on the Blocksworld benchmark [94], where an agent is asked to rearrange blocks into stacks in a particular order. We define a **state** as the current orientation of the blocks and an **action** as an instruction that moves blocks. Specifically, an action is composed of one of four verbs (STACK, UNSTACK, PUT, and PICKUP) and the manipulated objects. We generate the currently valid actions given the domain restrictions and current orientation. To transition between states, we take the current action and query the LLM to predict the changes to the relevant blocks, updating the state by adding the new conditions and removing those that are no longer true. Once a state has met all goal conditions or the depth limit is reached, we terminate the associated node.

To assess the quality of actions, we use two rewards. First, we prompt the LLM with

example test cases and their solutions and compute the log probability of the action given the current state (the *likelihood of action* reward, r_1). This reflects the intuition of the LLM as a reasoning agent; it is typically indicative when there are few steps left to the goal, but less reliable for a distant goal. Second, we compare the new state after performing an action with the goal and provide a reward, r_2, scaling with the number of conditions met (the *task-specific heuristics* reward). When all conditions are met, we assign a very large reward to ensure the plan is selected as the solution.

Results. We use test cases from the Blocksworld dataset [95] and group them by the minimum number of actions required, resulting in 30 cases solvable within 2 steps, 57 within 4 steps, and 114 within 6 steps. There are at most 5 blocks in each test case. As the baseline, we prompt the LLM with 4 test cases and corresponding solutions and ask it to generate a plan for a new question, following the setup of Valmeekam et al. [94]; we denote this as CoT. For RAP, the same prompt is used to help the LLM calculate r_1.

As shown in Table 2.1, CoT with LLaMA-33B can only solve a few 2-step cases and fails almost completely on harder problems. RAP substantially improves over CoT, nearly solving all problems within 4 steps and a portion of 6-step problems, achieving an average success rate of 64%. It is worth noting that the search space of 6-step problems can be as large as 5^6, yet our algorithm finds a successful plan 42% of the time within 20 iterations. Even more strikingly, RAP allows LLaMA-33B to outperform GPT-4 by a 33% relative gain, despite GPT-4 having much stronger reasoning ability [9].

Case study. We compare the reasoning paths from CoT and RAP in Figure 2.4. We summarize the reasons for the improvement: (1) by maintaining the world state during reasoning, RAP can recognize valid actions for the current state, avoiding illegal plans; (2) RAP is capable of backtracking and trying other solutions when the first intuition from the LLM does not work—in the example, CoT achieves the second goal ("orange on red") with its first two steps, but doing so prevents the first goal from being satisfied, whereas RAP makes the same mistake in early iterations yet explores other paths and finally generates a successful plan; (3) when calculating

Table 2.1. Results on Blocksworld. RAP$^{(10)}$ and RAP$^{(20)}$ refer to our method with 10 and 20 iterations, respectively. "pass@10" means 10 plans are sampled for each test case, and the case is regarded as solved if at least one plan is correct. All other settings, including RAP, evaluate only a single plan.

Method	2-step	4-step	6-step
CoT	0.17	0.02	0.00
CoT (pass@10)	0.23	0.07	0.00
CoT (GPT-4)	0.50	0.63	0.40
RAP$^{(10)}$	1.00	0.86	0.26
RAP$^{(20)}$	**1.00**	**0.88**	**0.42**

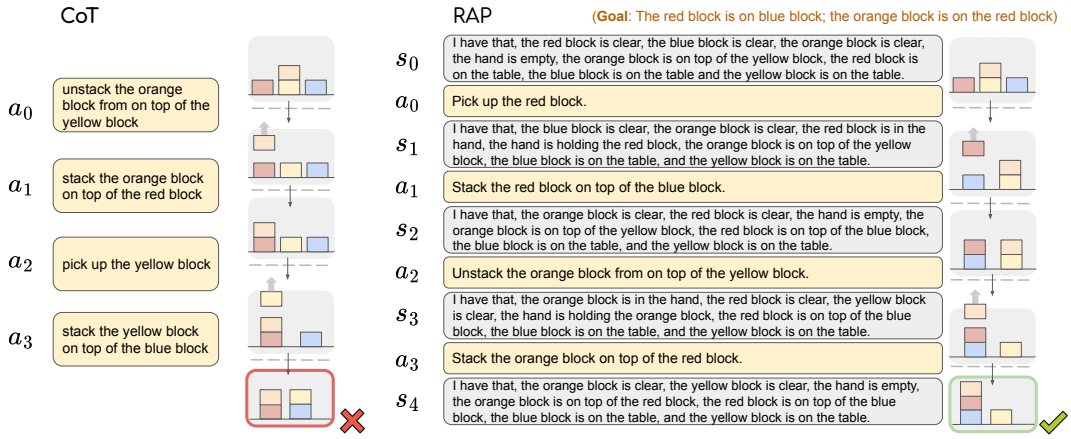

Figure 2.4. Comparing reasoning traces in Blocksworld from CoT (left) and RAP (right). CoT commits to an incorrect early action and cannot recover. RAP backtracks and finds a successful plan.

the reward, we can feed only the current state to the LLM and hide the history, which significantly lowers the difficulty of the last few steps and saves more iterations for the harder early decisions.

2.4.2 Math Reasoning

Task setup. Math reasoning tasks, such as GSM8K [14], often include a description and a final question. Arriving at the answer requires multi-step calculations based on the problem's context. It is thus natural to decompose the final question into a sequence of smaller sub-questions (Figure 2.2, middle). We define a **state** as the values of intermediate variables, and an **action** as proposing an incremental sub-question about an unknown intermediate variable.

The world model responds to the sub-question using the intermediate variables and the problem description, adding the new variable value to the next state. We combine the self-evaluation of helpfulness by the LLM, $r_{t,1}$, and the confidence of the state, $r_{t,2}$, using a weighted geometric mean $r_t = r_{t,1}^{\alpha} \cdot r_{t,2}^{1-\alpha}$ as the **reward**. This encourages more relevant and useful sub-questions. To account for the impact of the reasoning path's length, we compute the Q value using the maximum of average future rewards:

$$Q^*(s_t, a_t) = \max_{s_t, a_t, r_t, \ldots, s_l, a_l, r_l, s_{l+1}} \text{avg}(r_t, \ldots, r_l). \tag{2.2}$$

As a related work, least-to-most prompting [104] shares a similar idea of sub-question decomposition, but it generates all sub-questions at once. In contrast, RAP considers each action a_t based on the current state s_t, which enables more informed decisions.

Results. We evaluate on GSM8K, a dataset of grade-school math word problems, with CoT [97], least-to-most prompting [104], and their self-consistency variants as baselines, using the same 4-shot demonstrations for all methods. As shown in Table 2.2, RAP answers 48.6% of problems correctly, outperforming both CoT and least-to-most prompting with self-consistency—and this is achieved while RAP selects only one reasoning trace based on the reward. RAP-Aggregation further improves accuracy by about 3%. Figure 2.5 shows accuracy as a function of the number of MCTS iterations (for RAP) or samples (for self-consistency baselines): across all budgets, RAP-Aggregation consistently outperforms baselines, indicating that RAP is significantly better at finding reliable reasoning paths when only a few iterations/samples are allowed.

2.4.3 Logical Reasoning

Task setup. A logical reasoning task (e.g., PrOntoQA [81]) typically provides a set of *facts* and *logical rules*, and a model is required to verify whether a *hypothesis fact* is true or false by applying the rules to the given facts (Figure 2.2, right). These tasks require not only the correct final answer (true/false), but also a detailed proof. To apply RAP, we define the **state** as a fact

Table 2.2. Results on GSM8K. Superscripts denote the number of samples (self-consistency, SC) or MCTS iterations (RAP).

Method	**Accuracy (%)**
Chain-of-Thought	29.4
+ SC$^{(10)}$	46.8
Least-to-Most	25.5
+ SC$^{(10)}$	42.5
RAP$^{(1)}$	40.0
RAP$^{(10)}$	48.6
+ Aggregation	**51.6**

we are focusing on, analogous to the human's working memory [3] for inference. An **action** is defined as selecting a rule from the fact set. The world model performs a one-hop reasoning step to derive a new fact as the next state. The **reward** is computed via self-evaluation: we prompt the LLM with a few labeled examples to help it judge the quality of reasoning steps. We use the average reward of future steps to update the Q function, as in Equation (2.2).

Results. We assess RAP on PrOntoQA [81], adopting their "true" ontology (real-world knowledge) and "random" ordering of rules. We mix examples requiring 3, 4, and 5 reasoning hops to prevent the LLM from memorizing when to finish, sampling 500 examples. We compare both the prediction accuracy of the final answer and the accuracy of the entire proof, running 20 iterations for MCTS and 20 samples for self-consistency. As shown in Table 2.3, RAP achieves 94.2% prediction accuracy and 78.8% proof accuracy, surpassing CoT by 14% on proof accuracy and self-consistency CoT by 4.4% on prediction accuracy. As illustrated in Figure 2.2, RAP can effectively recognize when a reasoning chain reaches a dead end and propagate the signal back to earlier steps, with the planning algorithm allowing it to explore alternatives. The self-evaluation reward further helps RAP recognize potential incorrect steps.

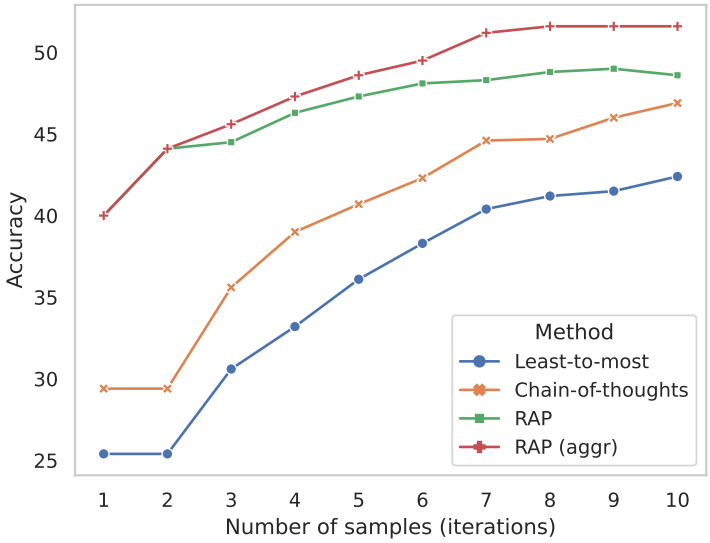

Figure 2.5. Accuracy on GSM8K with different numbers of sampled paths (self-consistency) or MCTS iterations (RAP). RAP-Aggregation consistently outperforms baselines at every budget.

Table 2.3. Results on PrOntoQA. We report both the prediction accuracy of the final answer and the accuracy of the full proof.

Method	**Pred. Acc. (%)**	**Proof Acc. (%)**
CoT	87.8	64.8
CoT + SC	89.8	—
RAP (Ours)	**94.2**	**78.8**

2.4.4 Analysis: Scaling to More Complex Problems

To study whether RAP can help stronger LLMs solve more complex problems, we conduct experiments on the full Blocksworld [95] dataset (602 test cases) using a more capable LLM, Llama-2 70B [93]. We group cases by the minimum number of actions required and consider two settings. In the `Easy` setting, we assume prior knowledge of the minimum number of actions for each case and use demonstration cases sharing the same minimum number of actions; for each group, we randomly select 10 cases as a demonstration pool and sample 4-shot demonstrations from it. In the `Hard` setting, we randomly select 10 cases from the full dataset to form a single global demonstration pool, sampling 4-shot demonstrations from it irrespective of the test case's difficulty.

Table 2.4. Results on the full Blocksworld with Llama-2 70B. Groups are defined by the minimum number of required actions.

Setting	Method	2	4	6	8	10	12	All
Easy	CoT	0.49	0.18	0.06	0.01	0.01	0.00	0.08
	RAP$^{(10)}$	1.00	0.99	0.75	0.61	0.32	0.32	0.65
Hard	CoT	0.22	0.14	0.02	0.02	0.00	0.00	0.05
	RAP$^{(10)}$	0.67	0.76	0.74	0.48	0.17	0.09	0.51

We employ CoT as a baseline and evaluate RAP$^{(10)}$ with an improved prompting technique. Results are summarized in Table 2.4. In both settings, RAP demonstrates superior performance over CoT by a substantial margin. Notably, when the test case requires six or more steps, CoT exhibits a severe drop in success rate, whereas RAP maintains a relatively high rate. Combined with the results of Section 2.4.1, we conclude that RAP is a general framework able to enhance the reasoning abilities of LLMs regardless of their intrinsic capabilities.

Reward choice. In our main experiments, we chose the combination of rewards based on heuristics and exploratory experiments. Generally, combining multiple rewards contributes to performance, but the effect of a reward depends on the nature of the task. For example, the action-likelihood reward is essential for plan generation but less helpful for mathematical reasoning, where the self-evaluation and state-confidence rewards play a larger role. This flexibility in reward design is a key strength of the RAP framework.

2.5 Conclusion

In this chapter, we presented Reasoning via Planning (RAP), a novel LLM reasoning framework that equips LLMs with an ability to reason akin to human-like strategic planning. Our framework, which repurposes the LLM to act as both a world model and a reasoning agent, enables the LLM to simulate states of the world and anticipate action outcomes, and to achieve an effective balance between exploration and exploitation via Monte Carlo Tree Search. Extensive experiments on a variety of challenging reasoning problems demonstrate RAP's superiority

over several contemporary CoT-based reasoning approaches, and even the advanced GPT-4 in certain settings. The key insight is that reasoning quality is improved not by more sophisticated language imitation, but by organizing the reasoning process according to the structural principles of deliberate planning. RAP is among the earliest works to demonstrate that inference-time structured search can dramatically improve reasoning; subsequent work has further validated this direction as an additional axis for scaling, and modern reasoning systems incorporate related ideas.

Acknowledgements

Chapter 2, in full, is a reprint of the material as it appears in Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. Hao, Shibo; Gu, Yi; Ma, Haodi; Hong, Joshua Jiahua; Wang, Zhen; Wang, Daisy Zhe; Hu, Zhiting, Association for Computational Linguistics, 2023. The dissertation author was the primary investigator and author of this paper.

Chapter 3

ToolkenGPT: Augmenting Frozen Language Models with Massive Tools via Tool Embeddings

3.1 Introduction

Large language models (LLMs) [8, 13, 73, 92] have established themselves as powerful tools for diverse real-world applications, ranging from writing assistance to automated customer support [5, 9, 21]. As these models continue to evolve, there is growing interest in their potential to interact with the real world and enhance their functionality through integration with external tools, such as calculators, databases, etc. [75, 79, 82, 90]. The capability of these models to master and control a wide array of tools not only serves as an indicator of their intelligence, but also signals a promising path to overcome some of their fundamental weaknesses. These include updating the latest world knowledge [72], reducing hallucinations [80, 87], and executing symbolic operations [19, 27]. However, the rapid emergence of new tools—advanced software libraries, novel APIs, or domain-specific utilities [60, 62]—accentuates the importance of empowering LLMs with the ability to adapt to and master massive new tools swiftly.

Recent advancements have witnessed two primary paradigms for tool integration with LLMs (Table 3.1). The first involves *fine-tuning* LLMs to learn specific tools [75]. For example, Toolformer [82] fine-tuned GPT-J to learn five tools. While this can yield promising results, it

Table 3.1. Comparison of different tool learning paradigms. "Plug-&-Play" means the LLM can be equipped and unequipped with a tool flexibly (it does not indicate zero-shot tool learning).

Tool Learning Paradigm	Frozen LMs	Massive Tools	Plug-&-Play	Extensive Data
Fine-tuning [75, 82]	✗	✗	✗	✓
In-context learning [79, 100]	✓	✗	✓	✗
TOOLKENGPT (Ours)	✓	✓	✓	✓

is computationally expensive and lacks adaptability to new tools. The second approach relies on *in-context learning* [74, 79, 100], where LLMs learn how to use tools through in-context demonstrations. This allows LLMs to handle newly introduced tools and drives applications like LangChain [12] and ChatGPT Plugins. However, in-context learning struggles with the inherent limitation of context length, making it impossible to demonstrate massive tools in the context; and mastering new tools simply via few-shot examples can be challenging—even GPT-4 faces difficulties with unusual tools [9].

In this chapter, we introduce TOOLKENGPT, an alternative solution that enables LLMs to master massive tools without any LLM fine-tuning, while still allowing for quick adaptation to new tools. The key idea is to represent each tool as a new token (a "*toolken*") to augment the vocabulary. Specifically, each tool is associated with an embedding inserted into the LLM head like a regular word token embedding. During generation, once a toolken is predicted, the LLM temporarily switches into a special mode (through prompting) to produce input arguments for the tool to execute, and injects the outputs back into the generation (Figure 3.1). This offers an efficient way for LLMs to master tools by only learning the lightweight toolken embeddings. Consequently, TOOLKENGPT combines the strengths of both fine-tuning and in-context learning while avoiding their limitations: compared to in-context learning, it allows massive tools (by simply inserting respective toolkens) and can use extensive demonstration data; in contrast to fine-tuning, the tool embeddings require minimal training cost and provide a convenient means for plugging in arbitrary new tools on the fly.

We demonstrate the flexibility and effectiveness of TOOLKENGPT across a diverse set of

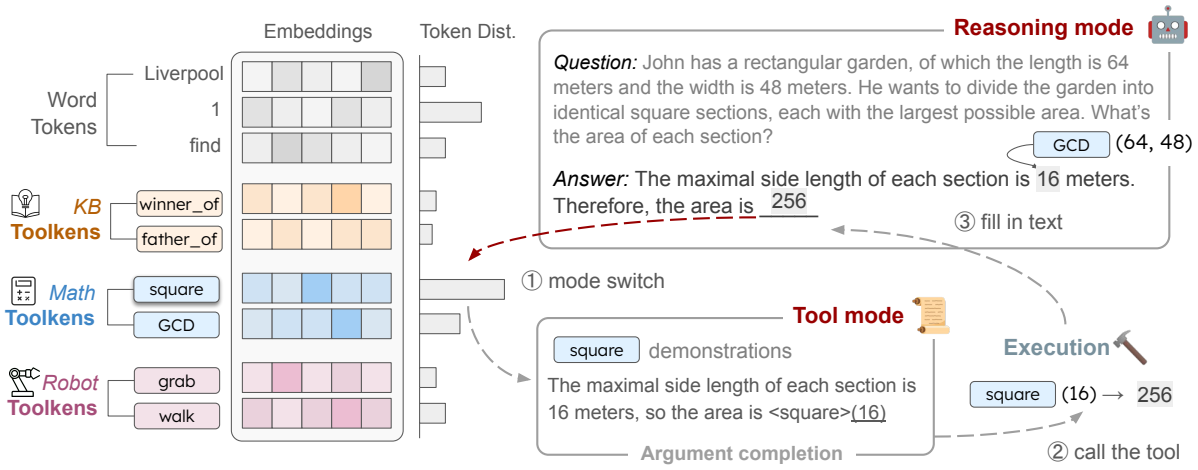

Figure 3.1. Overview of the TOOLKENGPT framework. Toolken embeddings are appended to the language model head like regular word tokens. In the "reasoning mode" for solving the problem, the LLM generates text as usual, except that any plugged-in toolkens are also considered for the next-token generation. Once a toolken is predicted, (1) the LLM switches to the "tool mode," which provides a few demonstrations of the same tool to complete the arguments; then (2) the tool call is executed, and (3) the result is sent back to the text to continue the reasoning mode until the final answer is generated.

problems, spanning numerical reasoning, knowledge-based question answering, and embodied plan generation. In complex numerical reasoning involving mathematical tools, TOOLKENGPT effectively utilizes these tools during reasoning, outperforming Chain-of-Thought [97] and ReAct [100]. For knowledge-based question answering, TOOLKENGPT accommodates over 200 relation APIs from a knowledge base, facilitating factual predictions. Finally, for embodied plan generation, TOOLKENGPT offers better grounding by learning toolken embeddings for 58 grounded actions and objects than previous in-context learning and specialized decoding methods.

3.2 Related Work

Fine-tuning LLMs to use tools. Early research relied heavily on fine-tuning to augment LMs with tools, mostly to use one or a few tools in a specific domain. The retriever has been a crucial tool for augmenting LLMs with external knowledge; prominent works include REALM [32],

RAG [58], and RETRO [6]. WebGPT [72] fine-tuned GPT-3 on human web-search behaviors. With the advancement of LLMs, there has been growing interest in tuning models on a collection of general tools (QA model, calculator, translator, etc.), e.g., TALM [75] and Toolformer [82]. However, fine-tuning is costly and the tuned LLMs struggle to generalize to emergent or updated tools. TOOLKENGPT learns lightweight toolken embeddings for new tools without any gradient calculation for the LLM parameters, enabling efficient adaptation at a GPU cost similar to LLM inference.

In-context learning for tools. LLMs exhibit a strong in-context learning ability [8], which has become a prevalent method to use tools by showing tool descriptions and demonstrations in context [70, 79]. Reasoning chains can be incorporated to tackle more complex problems [52, 100]. This paradigm has given rise to products such as ChatGPT Plugins and LangChain [12]. For instance, a code interpreter can address the LLM's shortcomings in symbolic operations [27, 65], and by calling tools that affect the virtual or physical world, the LLM can guide embodied agents [42, 44, 89]. Nevertheless, all in-context-learning methods suffer from inferior performance in complex scenarios, where the tools are unfamiliar or numerous. TOOLKENGPT is fundamentally different in that toolken embeddings can encode the implicit semantics of tools from extensive demonstrations, which can never be inferred from the surface text.

Efficient tuning of large language models. Adapting frozen LLMs efficiently to new tasks has led to a surge of interest in parameter-efficient fine-tuning (PEFT) methods [61, 64]. The idea is to fine-tune only a small subset of parameters while freezing most of the LLM, which bears similarity to our toolken embedding method. Adapters [39] insert trainable layers, prompt tuning [57] appends parameters to the input embedding layer, and LoRA [40] learns low-rank matrices within dense layers. However, existing PEFT methods have not proven suitable for efficient tool learning. To the best of our knowledge, we are the first to explore efficient tuning methods for predicting tools as tokens for the tool learning of massive tools.

3.3 ToolkenGPT for Mastering Massive Tools

In this section, we present TOOLKENGPT, which enables LLMs to learn and use massive tools for complex problem-solving without heavily fine-tuning the LLM. Typically, LLMs model the probability of a sequence of word tokens $s = (t_1, t_2, \ldots, t_n)$ as $P(s) = \prod_{i=1}^{n} P(t_i \mid t_{<i})$, where each token comes from the vocabulary $\mathscr{V}$. The distribution of the next token is predicted as $P(t_i \mid t_{<i}) = \text{softmax}(W_{\mathscr{V}} \cdot h_{i-1})$, where $h_{i-1} \in \mathbb{R}^d$ is the last hidden state of the current context and $W_{\mathscr{V}} \in \mathbb{R}^{|\mathscr{V}| \times d}$ is the embedding matrix for word tokens (the language model head).

Given a set of useful tools $\mathscr{T} = \{\tau_1, \tau_2, \ldots\}$, our goal is to enable LLMs to call a subset of these tools to solve a complex problem. Our flexible formulation allows tools to either return results that help with text generation (e.g., a calculation) or affect the real-world environment (e.g., a robot action). To call a tool during generation, the LLM first selects a tool and then inputs the arguments. In the running example in Figure 3.1, during the "reasoning mode" a math operator `square` is selected, and an operand `16` is generated as the argument in the "tool mode"; once the external tool receives the call, it executes and returns the result `256` back to the reasoning mode.

3.3.1 Framework Overview

The core idea of TOOLKENGPT is explicitly formulating tools as tokens ("toolkens"). Each toolken is parameterized as a *toolken embedding* vector, and we denote the set of toolken embeddings as a matrix $W_\tau \in \mathbb{R}^{|\mathscr{T}| \times d}$. Assuming we have trained the toolken embeddings (Section 3.3.2), the LLM is in the reasoning mode by default. Our framework allows the LLM to consider word tokens and toolkens uniformly: the tool embedding matrix is concatenated with $W_{\mathscr{V}}$, so the LLM predicts the next token with probability

$$P(t_i \mid t_{<i}) = \text{softmax}\big([W_{\mathscr{V}}; W_\tau] \cdot h_{i-1}\big), \tag{3.1}$$

where the next token can be either a word token or a toolken, i.e., $t_i \in \mathscr{V} \cup \mathscr{T}$, and $[;]$ is the concatenation operation. This formulation naturally allows for the fast adaption of new tools by expanding the toolken embedding matrix.

To execute a tool, the LLM switches into the "tool mode" once its toolken is predicted as the next token (the "mode switch" in Figure 3.1). Specifically, the LLM pauses generation and appends the current generated context to another prompt. The prompt in tool mode consists of in-context demonstrations for the predicted tool, showing how to generate the tool arguments by quoting the tool calls in a special syntax of `[tool](arguments)`. The LLM then follows the pattern in the demonstrations to complete the arguments. In contrast to previous methods [79, 100] that fully rely on in-context learning for tool learning, our framework only leaves the easy work of completing arguments to in-context learning, with abundant context space for extensive demonstrations of a single specified tool. This design shares similarities with classic divide-and-conquer methods [52]. Finally, the arguments are sent to the specified tool for execution, and the returned value is sent back to the text in the reasoning mode.

3.3.2 Learning Toolken Embeddings

Our framework keeps the original LLM parameters frozen and introduces minimal additional training overhead with the toolken embeddings, W_τ. This embedding matrix contains the only parameters to optimize. Unlike other efficient tuning methods, e.g., prompt tuning [57] or prefix tuning [61], it does not require gradients flowing through the major body of LLM parameters, leading to much more stable and efficient training—the tuning of toolken embeddings maintains nearly the same GPU memory as LLM inference. Whenever a new tool is added, the toolken embedding can be conveniently expanded, and subsequent training on tool demonstration data involving the new tool gradually refines its embedding.

Drawing parallels to how infants learn a new tool through demonstrations from adults [22], we primarily focus on learning toolken embeddings with tool demonstrations, which can be either in-domain training data or synthetic data generated by LLMs. Consider the exam-

ple from Figure 3.1: "*the area is 256 square feet ...*" can be tokenized into a word token sequence $s = (\text{“the”}, \text{“area”}, \text{“is”}, \text{“2”}, \text{“5”}, \text{“6”}, \text{“square”}, \text{“feet”}, \dots)$. To indicate when to predict the toolkens, we need a parallel sequence mixed with word tokens and toolkens, i.e., $s' = (\text{“the”}, \text{“area”}, \text{“is”}, \texttt{[square]}, \texttt{[N/A]}, \texttt{[N/A]}, \text{“square”}, \text{“feet”}, \dots)$. The subsequence (“2”, “5”, “6”) in s is where the returned tool results should fill in, and we choose the corresponding first token in s' as the toolken for the tool call, with the following tokens filled with `[N/A]` to indicate neglect in loss calculation. Thus, given a dataset of paired sequences $\mathscr{D} = \{(s, s')\}$, the training objective of TOOLKENGPT is

$$\mathscr{L}(W_\tau) = \sum_{(s,s')\in\mathscr{D}} \sum_{i=1}^{N} -\log P(t'_i \mid t_{<i})\, \mathbb{1}_{t'_i \neq \texttt{[N/A]}}, \tag{3.2}$$

where $P(t'_i \mid t_{<i})$ is defined in Eq. (3.1), and the indicator function signals that we ignore the `[N/A]` tokens during training. Thus, our training process is largely consistent with inference in the reasoning mode: to call a tool, the only job for the LLM is to predict a toolken at the beginning, and then the returned value is filled back into the text.

There are two primary ways to obtain the paired data. First, some datasets provide ground-truth tool calls along with natural language sequences, e.g., the facts in a KB supporting an answer (Section 3.4.2), or the calculation trace for solving a math problem (Section 3.4.1). Second, we explore synthesizing tool demonstrations with LLMs, sharing a similar idea to self-instruct. An intuitive interpretation is to distill the knowledge inside the LLM into the new toolken embeddings: we prompt the LLM with the tool documentation and a few demonstrations with a special syntax indicating tool calling, e.g., *The capital of U.S. is ¡capital¿("U.S.")="Washington D.C.*"; conditioned on that, the LLM generates new use cases that quote the tool call with the same syntax, which we then process into paired data.

3.4 Experiments

In this section, we apply TOOLKENGPT to three distinct applications characterized by meaningful tool-use scenarios: arithmetic tools for numerical reasoning, database APIs for knowledge-based question answering, and robot actions for embodied plan generation. Our experiments show that TOOLKENGPT can efficiently master massive tools while leveraging them to solve complex problems with improved performance, consistently better than advanced prompting techniques.

3.4.1 Numerical Reasoning

LLMs often struggle with mathematical tasks since the models are inherently designed for probabilistic estimation rather than symbolic operations. In this section, we assess the tool-learning capabilities of TOOLKENGPT, compared with in-context tool learning (e.g., ReAct [100]). We first demonstrate that TOOLKENGPT consistently matches or outperforms in-context learning with four basic arithmetic functions ($+$, $-$, $\times$, $\div$). Moreover, to benchmark the tool-handling capability in more complex problems, we include an expanded set of 13 functions and create a set of synthetic data.

Datasets. We curate two new test datasets. (1) **GSM8K-XL**, an enhanced version of GSM8K [14]. In the original dataset, the numbers for calculations are typically small; in the test set, we magnify the numbers to increase the computational difficulty, resulting in 568 test cases with much larger numbers requiring the four basic arithmetic operators. (2) **FuncQA**, a synthetic dataset we created involving 13 arithmetic operators (e.g., `power`, `sqrt`, `lcm`), which is challenging for both humans and LLMs to solve without an external calculator. FuncQA is categorized into 68 one-hop questions ($\text{FuncQA}_{\text{one}}$) solvable with one operation, and 60 multi-hop questions ($\text{FuncQA}_{\text{multi}}$) requiring a few reasoning steps. To train the toolken embeddings for GSM8K-XL, we preprocess the original GSM8K training set (which has calculation annotations) into 5,054 training and 1,000 validation examples. For FuncQA, we prompt ChatGPT to generate one-hop

Table 3.2. Results on GSM8K-XL and FuncQA. Numbers in parentheses indicate how many tools are available. For GSM8K-XL and $\text{FuncQA}_{\text{one}}$, accuracy is evaluated by exact match (floats rounded to two decimals). For $\text{FuncQA}_{\text{multi}}$, we allow a margin of 0.1% to account for errors at each step of multi-hop reasoning.

Method	GSM8K-XL (4)	FuncQA (13)	
		One-Hop	Multi-Hop
0-shot ChatGPT	0.17	0.55	0.09
CoT [97]	0.18	0.20	0.03
ReAct [100]	0.32	0.57	0.06
TOOLKENGPT (Ours)	**0.33**	**0.73**	**0.15**

QA patterns for each operator, yielding 611 training and 39 validation samples.

Comparison methods. We train toolken embeddings for each available operator. During inference, we prompt the LLM with 4-shot Chain-of-Thought [97] examples to enhance reasoning. We compare against: (1) *0-shot ChatGPT*, with no examples and no tools; (2) *Chain-of-Thought (CoT)* [97], with the same reasoning chains as ours but no functions; and (3) *ReAct* [100], which generates verbal reasoning traces and tool calls in an interleaved manner using a special syntax. LLaMA-33B [92] is the base LLM for all settings other than zero-shot prompting.

Result analysis. Table 3.2 shows the results. On GSM8K-XL, 0-shot ChatGPT and CoT struggle to calculate large numbers without tools, while ReAct and TOOLKENGPT increase accuracy by a large margin; with only four basic operators, both can call the correct tools. However, for FuncQA, learning to call applicable tools becomes challenging for ReAct as the number of tools increases: it is infeasible to include demonstrations of every tool in the limited context (we provide 4 examples including 5 tool demonstrations), so ReAct is susceptible to missing tool calls, making wrong tool calls, and predicting wrong arguments—especially for tools not demonstrated in context. TOOLKENGPT outperforms all baselines across both one-hop and multi-hop scenarios. Notably, even though the toolken embeddings are trained solely with *one-hop synthetic data* and *without any CoT examples*, they still enhance performance in multi-hop contexts and integrate effectively with CoT prompting, implying a degree of generalization that

lowers the requirements for in-domain training data.

3.4.2 Knowledge-based Question Answering

LLMs are known to often make factual errors and hallucinate [46] because of their limited or outdated parametric knowledge [37]. Equipping them with access to knowledge bases (KBs) is a promising direction to reduce hallucinations [87]. We formulate access to the KB as APIs querying the database: each relational query is a tool whose input argument is a subject entity and whose output is the corresponding tail entity. An example tool call is `P1346`(2005-06 FA CUP) $\rightarrow$ LIVERPOOL F.C., where "P1346" is a relation identifier in Wikidata (referred to as `winner_of` below). In this section, we show that TOOLKENGPT can accurately query a large knowledge base of up to 234 tools (relations), and that even with synthetic data alone we can train strong toolken embeddings that outperform popular tool-learning methods.

Dataset. KAMEL [51] is a question-answering dataset built with facts in Wikidata, containing knowledge about 243 relations, each associated with a question template (e.g., `winner_of`: "*Who is the winner of [S]?*"). We have 234 tools in total. To analyze performance with different numbers of tools, we create four subsets by sampling questions related to 30, 60, 100, and 234 relations, each of size 500.

Comparison methods. We set up two variants of our framework: (1) TOOLKENGPT *(sup)*, where we sample 200 examples per relation from the KAMEL training set and train toolken embeddings via supervised learning, representing scenarios with sufficient in-domain data; and (2) TOOLKENGPT *(syn)*, a more challenging setting where in-domain data is unavailable, so we use the text description of each relation to synthesize training data with ChatGPT (on average 40 examples per toolken). We compare against: (1) *Prompting* [51], which answers using the LLM's internal knowledge; (2) *In-context Learning (ICL)* [79], listing tool demonstrations and descriptions of all available tools; and (3) *ICL (desc)* [79], providing only descriptions of all available tools plus 8 demonstrations of tools not in the test subset. The base model for all methods is LLaMA-13B [92].

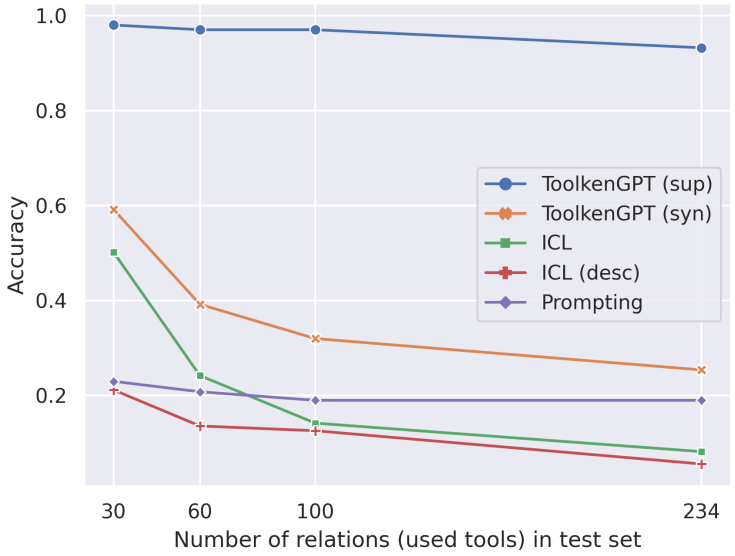

Figure 3.2. Performance of TOOLKENGPT and baselines on 4 test sets involving different numbers of tools (relations) from KAMEL. ICL is short for In-context Learning [79]. Due to the context length limit of 2048 tokens, we list the descriptions and demonstrations of up to 30 relations for ICL, and up to 60 relation descriptions for ICL (desc).

Result analysis. Figure 3.2 shows the results across test sets involving different numbers of relations. For all test sets, the accuracy of Prompting is about 20%, indicating LLMs still struggle to store accurate facts in their parameters. TOOLKENGPT (sup) achieves the highest results by a large margin, showing that learning toolken embeddings is effective when there is massive in-domain data. On the contrary, even though in-context learning also sees in-domain training data in the context, it still gets confused about which tools to call, and the context length limit leads to drastic performance drops when there are more than 30 tools—revealing the fundamental limitation of the in-context learning paradigm. TOOLKENGPT (syn) also outperforms all other baselines in all subsets without seeing any in-domain training data; the synthetic data, often in very different expression styles from the dataset, still helps the LLM understand these relations. In-context learning (desc) generally fails in this task, because the LLM has difficulty memorizing text descriptions shown in context and mapping them to relation identifiers. Based on this observation, it is reasonable to speculate that LLMs mostly *recall* the tools from their identifiers instead of really *learning* to use tools from their descriptions.

Table 3.3. Results on VirtualHome. "Grounding" is the proportion of scripts in which all actions and objects can be grounded to the environment; "Executable" is the proportion executable without violating any rules; "Success" is the proportion leading to the correct final state; "Success (R)" is a relaxed variant counting scripts that reached the correct state but did not necessarily end with it.

Method	Grounding	Executable	Success	Success (R)
In-context Learning	0.74	0.42	0.20	0.30
+ Translation [42]	**1.00**	0.52	0.24	0.32
+ Grounded Decoding [44]	**1.00**	0.66	0.38	0.42
TOOLKENGPT (Ours)	**1.00**	**0.82**	**0.68**	**0.70**

3.4.3 Embodied Plan Generation

There have been many attempts to utilize LLMs as the controller of embodied agents [42, 44, 89]. Despite preliminary success, teaching LLMs about an environment and enabling them to make grounded predictions remains challenging. As discussed in Mialon et al. [70], tools that gather information (e.g., math or KB tools) and tools that have an effect on the physical world (e.g., actions taken by embodied agents) can be called in similar styles by the LLM. Compared to previous methods that prompt LLMs, TOOLKENGPT can understand the environment better by learning toolken embeddings for agent actions and objects.

Dataset. VirtualHome [78] is a simulation platform for typical household activities, and its ActivityPrograms knowledge base consists of many tasks with plans executable in VirtualHome. We derive a subset of 297 tasks. For each task, the model is given a high-level **goal** (e.g., "*Read book*"), a detailed **instruction** (e.g., "*I would go lie down in my bed and open the book and start reading.*"), and a description of the **environment** (the initial state of the agent and the object list). The model is expected to output an executable plan, which is an ordered list of verb-object instructions (e.g., "*[FIND] ¡novel¿*"). Each task comes with an initial and final state graph, enabling verification of the generated plans with the simulator. We split the dataset into 247 training and 50 test tasks, with 25 verbs and 32 objects.

Comparison methods. We consider all actions and objects in VirtualHome as tools. With an

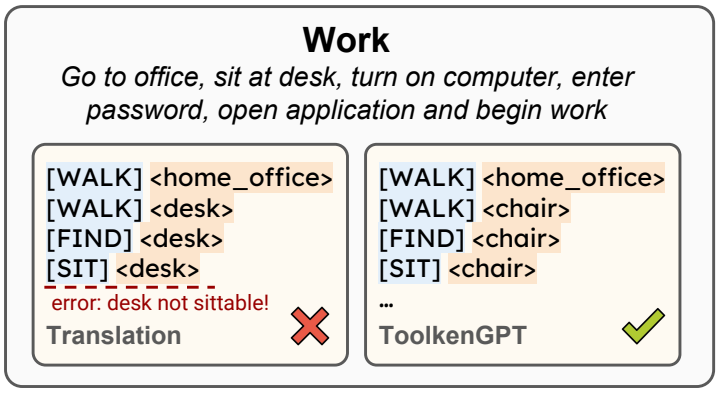

Figure 3.3. Case study on VirtualHome. TOOLKENGPT predicts a successful script while baselines fail to produce an executable one due to their misunderstanding of the `SIT` action.

additional `[END]` function indicating the end of a plan, we have 58 toolkens in total. We do not need the argument generation process because these tools do not take arguments; during inference, TOOLKENGPT alternately generates action toolkens and object toolkens, ending with `[END]`. We compare against: (1) *In-context Learning*, prompting the LLM with the action list, 3 demonstration plans, and the new task; (2) *Translation* [42], which uses a SentenceRoBERTa model to translate the LLM's generation to admissible instructions; and (3) *Grounded Decoding* [44], a decoding-stage grounding method that predicts the next token considering both LLM logits and an affordance grounding function. The base model is LLaMA-13B [92].

Result analysis. Table 3.3 lists the results. Even though all valid actions and objects are explicitly listed in the context for In-context Learning, it sometimes fails to ground its prediction to admissible instructions, and even valid plans often violate physical rules in VirtualHome, resulting in a low success rate. Translation [42] solves some shallow grounding problems (e.g., `[tv]` $\rightarrow$ `[television]`), and Grounded Decoding [44] further improves executability by considering grounding earlier in decoding. However, neither significantly improves the LLM's understanding of actions and objects. TOOLKENGPT not only predicts valid actions and objects by design, but also achieves the highest success rate by learning toolken embeddings from more training tasks. A concrete example is shown in Figure 3.3: all baselines predict `[SIT] <desk>`, presumably guided by the description "sit at desk," but in VirtualHome `[SIT]` refers to "sit on,"

Table 3.4. Comparison between TOOLKENGPT and fine-tuning (LoRA) in terms of training cost and performance on FuncQA. Both methods are based on LLaMA-7B.

Method	One-hop	Multi-hop	Computing Resource	Training Time
ReAct	0.40	0.03	—	—
Prompting	0.10	0.00	—	—
Fine-tune w/ LoRA [40]	0.62	0.07	8 × A100 (80G)	40 min
TOOLKENGPT	0.55	0.06	1 × RTX3090 (24G)	2 min

and a desk is regarded as not sittable; TOOLKENGPT is the only method to successfully learn this rule from demonstrations and instead predict `[SIT] <chair>`.

3.4.4 Analysis

Computational cost. We compare TOOLKENGPT with fine-tuning, specifically LoRA [40], in terms of computational efficiency and performance. Due to the cost of fine-tuning LLMs, we implement both methods on LLaMA-7B (Table 3.4). Fine-tuning results in slightly better performance than TOOLKENGPT on FuncQA, but the time consumption exceeds significantly when compared to training toolken embeddings (40 minutes on 8 A100s vs. 2 minutes on a single RTX3090). TOOLKENGPT also enjoys additional benefits, especially the plug-and-play of massive tools, thanks to the decoupled parameters for different tools.

Ablation study. The design of TOOLKENGPT benefits both tool selection and argument completion (the tool mode in Figure 3.1). To understand their respective contributions, we implement a baseline combining ReAct-style prompting with the argument-completion sub-routine (tool mode). As shown in Table 3.5, adding a tool mode does improve vanilla ReAct by enhancing argument completion, but TOOLKENGPT still outperforms this improved baseline by a large margin, indicating that toolken embeddings effectively help LLMs decide *when* and *which* tool to call.

Training data. We explore the effects of training data on learning toolken embeddings, extending our KAMEL experiments since there are two sources of training data. We sample 10/20/40

Table 3.5. Ablation study on FuncQA with LLaMA-30B.

Method	One-hop	Multi-hop
ReAct	0.57	0.06
+ Tool mode	0.60	0.07
TOOLKENGPT	**0.73**	**0.15**

Table 3.6. TOOLKENGPT with different configurations of training data on KAMEL.

# Examples	Synthetic	Supervised
10	0.36	0.56
20	0.46	0.90
40	0.52	0.95

training examples per tool for both TOOLKENGPT (sup) and TOOLKENGPT (syn) and report accuracy on the test set involving 30 tools (Table 3.6). Under the same data budget, training with supervised data leads to better performance; though we do not observe obvious mistakes in most synthetic instances, the distribution gap between synthetic data and the test set may prevent the toolken embedding from performing well. A larger training set benefits performance for both data sources.

3.5 Conclusion

We presented TOOLKENGPT, a new approach for augmenting frozen LLMs with massive external tools without expensive fine-tuning. Our method introduces the concept of a toolken embedding for each tool, enabling LLMs to call and use different tools as easily as generating word tokens. This approach overcomes the limitations of current fine-tuning and in-context learning paradigms, enabling LLMs to accommodate a much larger set of tools and to use extensive demonstration data for learning toolken embeddings. On a series of tasks—numerical reasoning, knowledge-based question answering, and embodied plan generation—we observed a significant enhancement in LLM performance with the help of toolken embeddings. More importantly, TOOLKENGPT is able to rapidly adapt and leverage new tools, demonstrating its capacity to keep pace with the constantly evolving landscape of massive tools. We expect future research to learn robust toolken embeddings not only from demonstration data but also from other rich forms of experience, such as tool descriptions and input–output records, and to explore the integration of toolken embeddings with advanced planning techniques such as those

in Chapter 2.

Acknowledgements

Chapter 3, in full, is a reprint of the material as it appears in Advances in Neural Information Processing Systems 36. Hao, Shibo; Liu, Tianyang; Wang, Zhen; Hu, Zhiting, Curran Associates, Inc., 2023. The dissertation author was the primary investigator and author of this paper. This work was partially supported by DARPA ECOLE HR00112390063.

Chapter 4

Training Large Language Models to Reason in a Continuous Latent Space

4.1 Introduction

Large language models (LLMs) have demonstrated remarkable reasoning abilities, emerging from extensive pretraining on human languages [1, 20]. While next-token prediction is an effective training objective, it imposes a fundamental constraint on the LLM as a reasoning machine: the explicit reasoning process must be generated in word tokens. For example, a prevalent approach known as chain-of-thought (CoT) reasoning [97] involves prompting or training LLMs to generate solutions step-by-step using natural language. However, this is in stark contrast with certain human-cognition results. Neuroimaging studies have consistently shown that the language network—a set of brain regions responsible for language comprehension and production—remains largely inactive during various reasoning tasks [2, 23, 71]. Further evidence indicates that human language is optimized for communication rather than reasoning [24].

A significant issue arises when LLMs use language for reasoning: the amount of reasoning required for each token varies greatly, yet current architectures allocate nearly the same compute budget for predicting every token. Most tokens in a reasoning chain are generated solely for fluency, contributing little to the actual reasoning process, while some critical tokens require complex planning and pose huge challenges. While previous work has attempted to fix these problems by prompting LLMs to generate succinct reasoning chains [66], or performing addi-

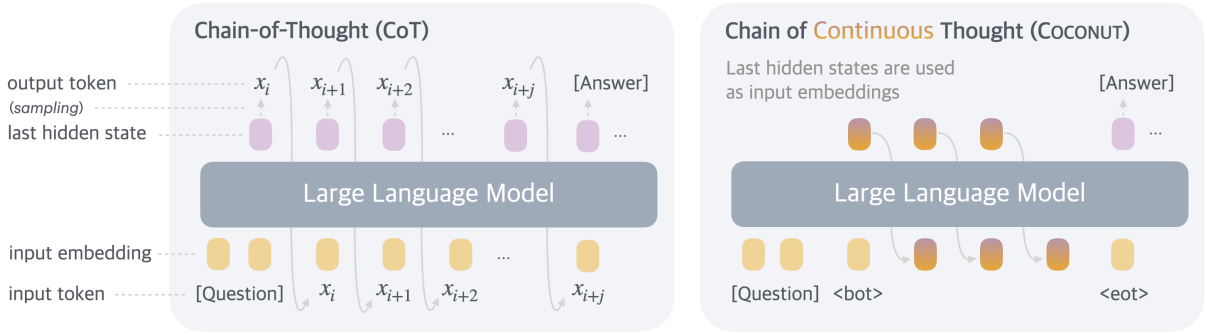

Figure 4.1. A comparison of Chain of Continuous Thought (COCONUT) with Chain-of-Thought (CoT). In CoT, the model generates the reasoning process as a word-token sequence (e.g., $[x_i, x_{i+1}, \ldots, x_{i+j}]$). COCONUT regards the last hidden state as a representation of the reasoning state (a "continuous thought"), and directly uses it as the next input embedding. This allows the LLM to reason in an unrestricted latent space instead of a language space.

tional reasoning before generating some critical tokens [103], these solutions remain constrained within the language space. On the contrary, it would be ideal for LLMs to have the freedom to reason without language constraints, and then translate their findings into language only when necessary.

In this chapter, we explore LLM reasoning in a latent space by introducing a novel paradigm, COCONUT (Chain of Continuous Thought). It involves a simple modification to the CoT process: instead of mapping between hidden states and language tokens using the language model head and embedding layer, COCONUT directly feeds the last hidden state (a "continuous thought") as the input embedding for the next token (Figure 4.1). This frees the reasoning from being within the language space, and the system can be optimized end-to-end by gradient descent, as continuous thoughts are fully differentiable. To enhance the training of latent reasoning, we employ a multi-stage training strategy inspired by Deng et al. [17], which effectively utilizes language reasoning chains to guide the training process.

Interestingly, our proposed paradigm leads to an efficient reasoning pattern. Unlike language-based reasoning, continuous thoughts in COCONUT can encode multiple potential next steps simultaneously, allowing for a reasoning process akin to breadth-first search (BFS). While the model may not initially make the correct decision, it can maintain many possible

options within the continuous thoughts and progressively eliminate incorrect paths through reasoning, guided by some implicit value functions. This advanced reasoning mechanism surpasses traditional CoT, even though the model is not explicitly trained or instructed to operate in this manner. Empirically, COCONUT outperforms CoT on certain logical reasoning tasks that require substantial search during planning, and shows a better trade-off between accuracy and efficiency, generating fewer tokens during inference.

4.2 Related Work

Chain-of-thought (CoT) reasoning. We use the term chain-of-thought broadly to refer to methods that generate an intermediate reasoning process in language before outputting the final answer. This includes prompting LLMs [52, 97, 104], or training LLMs to generate reasoning chains, either with supervised finetuning [102] or reinforcement learning [85]. Recent theoretical analyses have demonstrated the usefulness of CoT from the perspective of model expressivity [25, 69]: by employing CoT, the effective depth of the transformer increases because the generated outputs are looped back to the input [25]. These analyses, combined with the established effectiveness of CoT, motivated our design of feeding the continuous thoughts back into the LLM as input embeddings. While CoT has proven effective, its autoregressive generation nature makes it challenging to mimic human reasoning on more complex problems [36, 55], which typically require planning and search. There are works that equip LLMs with explicit tree search algorithms [36, 101], or train the LLM on search dynamics and trajectories [26, 56]. In our analysis, we find that after removing the constraint of a language space, a new reasoning pattern similar to BFS emerges, even though the model is not explicitly trained this way.

Latent reasoning in LLMs. Previous works mostly define latent reasoning in LLMs as the hidden computation in transformers [4, 99]. To enhance the latent reasoning of LLMs, previous research proposed to augment it with additional tokens. Goyal et al. [30] pretrained the model by randomly inserting learnable `<pause>` tokens into the training corpus. Pfau et al. [77] further

explored the usage of filler tokens, e.g., ". . .", and concluded that they work well for highly parallelizable problems, but do not extend the expressivity of the LLM like CoT. Recently, it has been found that one can "internalize" CoT reasoning into latent reasoning with knowledge distillation [16] or a special training curriculum that gradually shortens the CoT [17]. We find that breaking down the learning of continuous thoughts into multiple stages, inspired by iCoT [17], is very beneficial for training. Other work explores alternative architectures for latent reasoning, including looped transformers [29]. Different from these works, we focus on general multi-step reasoning tasks and aim to investigate the unique properties of latent reasoning in comparison to language space. Building on COCONUT, Zhu et al. [105] developed a theoretical framework demonstrating that continuous CoT can be more efficient than discrete CoT on certain tasks by encoding multiple reasoning paths in superposition states.

4.3 Chain of Continuous Thought

In this section, we introduce our new paradigm COCONUT for reasoning in an unconstrained latent space. We begin by introducing the notation we use for language models. For an input sequence $x = (x_1, \ldots, x_T)$, the standard LLM $\mathscr{M}$ can be described as:

$$H_t = \text{Transformer}(E_t), \tag{4.1}$$

$$\mathscr{M}(x_{t+1} \mid x_{\leq t}) = \text{softmax}(W h_t), \tag{4.2}$$

where $E_t = [e(x_1), e(x_2), \ldots, e(x_t)]$ is the sequence of token embeddings up to position t; $H_t \in \mathbb{R}^{t \times d}$ is the matrix of last hidden states for all tokens up to position t; $h_t = H_t[t,:]$ is the last hidden state of position t; $e(\cdot)$ is the token embedding function; and W is the parameter of the language model head.

Method overview. In COCONUT, the LLM switches between a "language mode" and a "latent mode" (Figure 4.1). In language mode, the model operates as a standard language model, autoregressively generating the next token. In latent mode, it directly utilizes the last hidden state

as the next input embedding—this hidden state represents the current reasoning state, termed a "continuous thought." Special tokens <bot> and <eot> mark the beginning and end of the latent mode, respectively. Assuming latent reasoning occurs between positions i and j (i.e., $x_i =$ <bot> and $x_j =$ <eot>), when the model is in latent mode ($i < t < j$) we use the last hidden state from the previous token to replace the input embedding:

$$E_t = [e(x_1), \ldots, e(x_i), h_i, h_{i+1}, \ldots, h_{t-1}]. \tag{4.3}$$

After the latent mode finishes ($t \geq j$), the input reverts to using the token embedding, i.e., $E_t = [e(x_1), \ldots, e(x_i), h_i, \ldots, h_{j-1}, e(x_j), \ldots, e(x_t)]$. The last hidden states have been processed by the final normalization layer, so they are not too large in magnitude. Note that $\mathscr{M}(x_{t+1} \mid x_{\leq t})$ is not defined when $i < t < j$, since the latent thought is not intended to be mapped back to language space; however, $\text{softmax}(Wh_t)$ can still be calculated for probing purposes (Section 4.4).

Training procedure. We focus on a problem-solving setting where the model receives a question as input and is expected to generate an answer through a reasoning process. We leverage language CoT data to supervise continuous thought by implementing a multi-stage training curriculum inspired by Deng et al. [17]. As shown in Figure 4.2, in the initial stage, the model is trained on regular CoT instances. In subsequent stages, at the k-th stage, the first k reasoning steps in the CoT are replaced with $k \times c$ continuous thoughts, where c is a hyperparameter controlling the number of latent thoughts replacing a single language reasoning step. Following Deng et al. [17], we reset the optimizer state when training stages switch. We insert <bot> and <eot> tokens (not counted towards c) to encapsulate the continuous thoughts.

During training, we optimize the normal negative log-likelihood loss, but mask the loss on questions and latent thoughts. It is important to note that the objective does *not* encourage the continuous thought to *compress the removed language thought*, but rather to *facilitate the prediction of future reasoning*. Therefore, it is possible for the LLM to learn more effective representations of reasoning steps than human language.

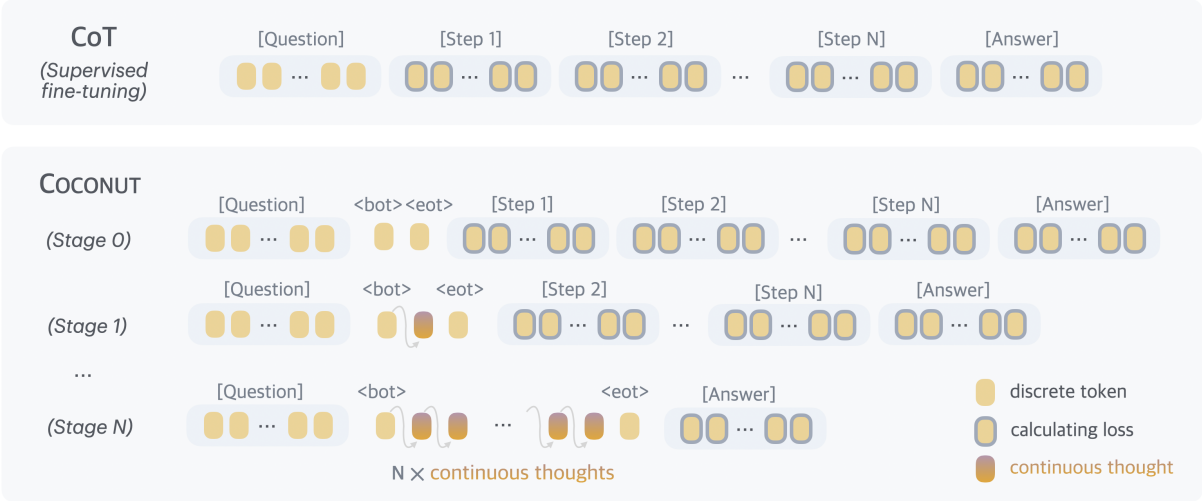

Figure 4.2. Training procedure of COCONUT. Given training data with language reasoning steps, at each training stage we integrate c additional continuous thoughts ($c = 1$ in this example) and remove one language reasoning step. The cross-entropy loss is then applied to the remaining tokens after the continuous thoughts.

Training details. Our continuous thoughts are fully differentiable and allow back-propagation. We perform $n + 1$ forward passes when n latent thoughts are scheduled in the current training stage, computing a new latent thought with each pass and finally conducting an additional forward pass to obtain a loss on the remaining text sequence. While we can save repetitive computation by using a KV cache, the sequential nature of the multiple forward passes poses challenges for parallelism; further optimizing the training efficiency of COCONUT remains an important direction for future research.

Inference process. The inference process is analogous to standard language model decoding, except that in latent mode we directly feed the last hidden state as the next input embedding. A challenge lies in determining when to switch between latent and language modes. As we focus on the problem-solving setting, we insert a `<bot>` token immediately following the question. For `<eot>`, we consider two strategies: (a) train a binary classifier on latent thoughts to enable the model to autonomously decide when to terminate latent reasoning, or (b) always pad the latent thoughts to a constant length. We found both approaches work comparably well, and use the second for simplicity unless specified otherwise.

4.4 Continuous Space Enables Latent Tree Search

To understand the behavior of COCONUT, we introduce ProsQA (Proof with Search Question-Answering), a logical reasoning dataset where each instance involves finding a path through a directed acyclic graph (DAG) of concept relationships, presented as natural language statements. The task requires models to determine logical relationships by finding valid paths through this graph, demanding sophisticated planning and search. Unlike previous datasets like ProntoQA [81], ProsQA's DAG structure introduces complex exploration paths with distracting branches, making it particularly challenging to identify the correct reasoning chain.

4.4.1 Experimental Setup

We use a pre-trained GPT-2 model as the base model. The learning rate is 1×10^{-4} and the effective batch size is 128. We train a COCONUT model following the training procedure in Section 4.3. Since the maximum number of reasoning steps in ProsQA is 6, we set the number of training stages to $N = 6$. As reference, we report the performance of (1) *CoT*, where the model is trained with CoT data and generates a complete reasoning chain; and (2) *no-CoT*, where the model is trained with only question–answer pairs.

To understand the properties of latent and language reasoning space, *we manipulate the model to switch between fully latent reasoning and fully language reasoning* by manually setting the position of the `<eot>` token during inference. When we enforce COCONUT to use k continuous thoughts, the model is expected to output the remaining reasoning chain in language, starting from the $(k+1)$-th step. We test variants with $k \in \{0, 1, 2, 3, 4, 5, 6\}$; all these variants differ only at inference time while sharing the same model weights.

We apply two sets of evaluation metrics. One is based on the correctness of the *final answer*. To enable fine-grained analysis, we define another metric on the *reasoning process*, classifying a reasoning chain into: **Correct Path** (a shortest path to the correct answer), **Longer Path** (a valid but non-shortest correct path), **Hallucination** (includes nonexistent edges or is

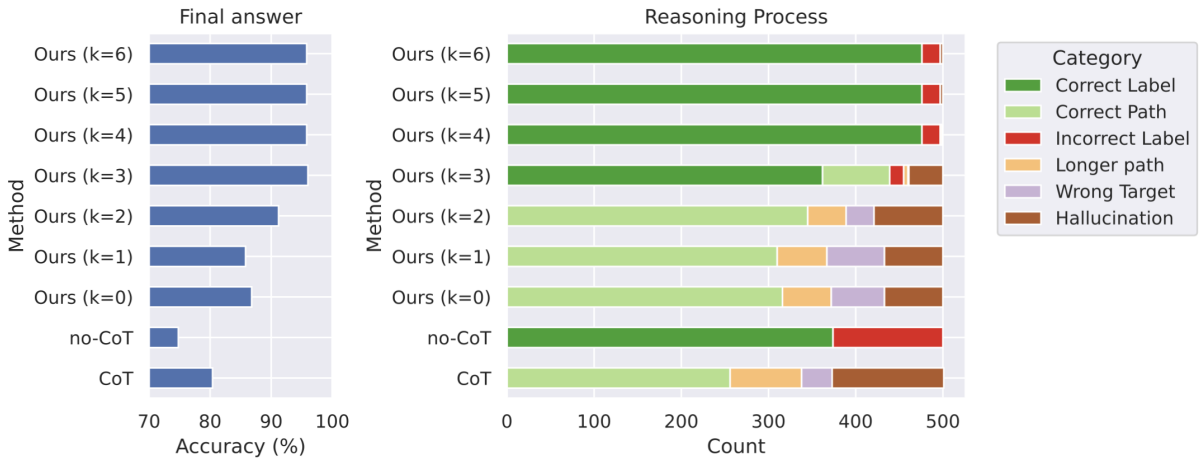

Figure 4.3. The accuracy of the final answer (left) and the reasoning process (right) of multiple variants of COCONUT and baselines on ProsQA. More continuous thoughts (larger k) lead to higher accuracy and fewer hallucinations / wrong-target errors.

disconnected), **Wrong Target** (a valid path to the wrong node), and—for outputs without a partial path—**Correct Label** or **Incorrect Label**.

4.4.2 Overall Results

Figure 4.3 presents a comparative analysis on ProsQA. The model trained with *CoT* frequently hallucinates non-existent edges or outputs paths leading to incorrect targets, resulting in lower answer accuracy. In contrast, COCONUT, which leverages continuous-space reasoning, demonstrates improved accuracy as it uses an increasing number of continuous thoughts. The rate of correct reasoning processes ("Correct Label" and "Correct Path") significantly increases, while there is a notable reduction in "Hallucination" and "Wrong Target"—issues that typically emerge when the model makes mistakes early in the reasoning process. As the case study in Figure 4.4 shows, models operating in language space often fail to plan ahead or backtrack: once they commit to an incorrect path, they either hallucinate unsupported edges or terminate with irrelevant conclusions. In contrast, latent reasoning avoids such premature commitments by enabling the model to iteratively refine its decisions across multiple steps.

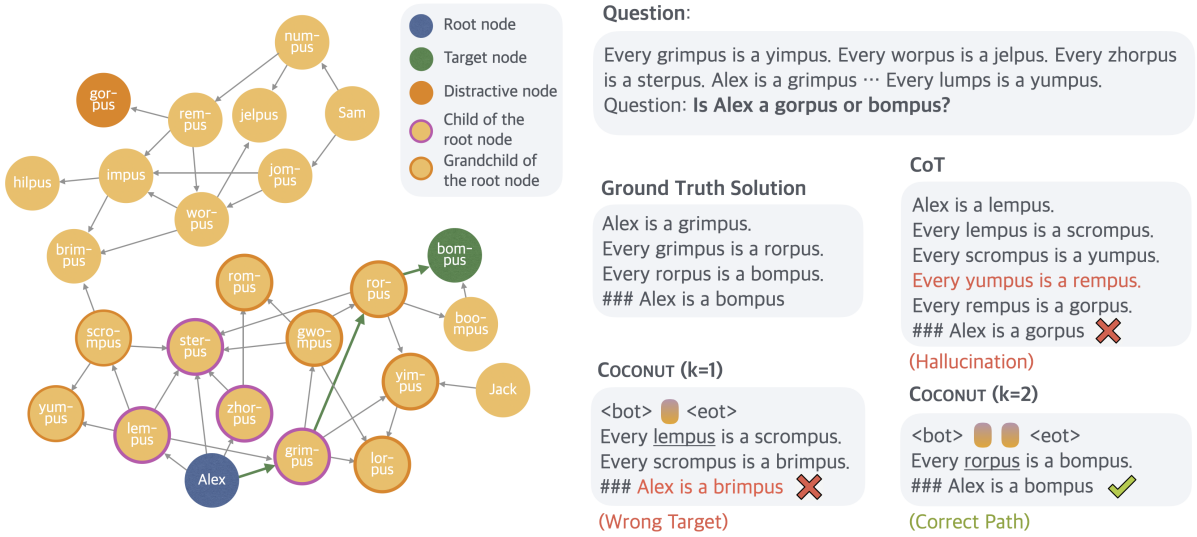

Figure 4.4. A case study of ProsQA. The model trained with *CoT* hallucinates an edge ("*Every yumpus is a rempus*") after getting stuck in a dead end. COCONUT (k=1) outputs a path that ends with an irrelevant node, while COCONUT (k=2) solves the problem correctly.

4.4.3 Interpreting the Latent Reasoning as Tree Search

To better understand COCONUT, we probe the latent reasoning process by forcing the model to explicitly generate language reasoning steps following intermediate continuous thoughts (Figure 4.5). Using the example in Figure 4.4, at the initial reasoning step the model must select which immediate child node of "Alex" to consider next; in the subsequent step, these nodes expand further into an extended set of potential paths. We define the predicted probability of a concept following continuous thoughts as a value function, estimating each node's potential for reaching the correct target. Interestingly, the strategy employed by COCONUT is not greedy search: while "lempus" initially has the highest value (0.33) at the first step, the model subsequently assigns the highest value (0.87) to "rorpus," a child of "grimpus," rather than following "lempus." This resembles a breadth-first search (BFS) approach, contrasting sharply with the greedy decoding typical of CoT. The inherent capability of continuous representations to encode multiple candidate paths enables the model to avoid making immediate deterministic decisions.

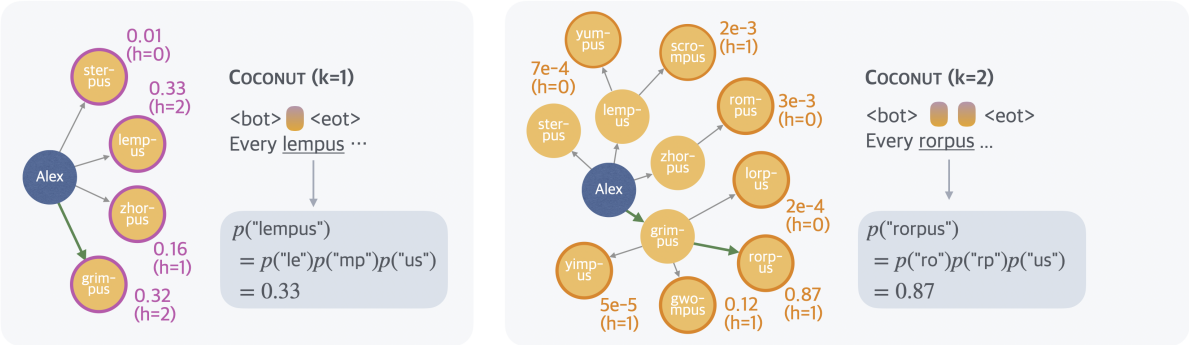

Figure 4.5. An illustration of the latent search trees, for the same test case as Figure 4.4. The height h of a node is the longest distance to any leaf node. We show the probability of the first concept predicted by the model following the latent thoughts; this can be interpreted as an implicit value function estimated by the model, assessing the potential of each node to lead to the correct answer.

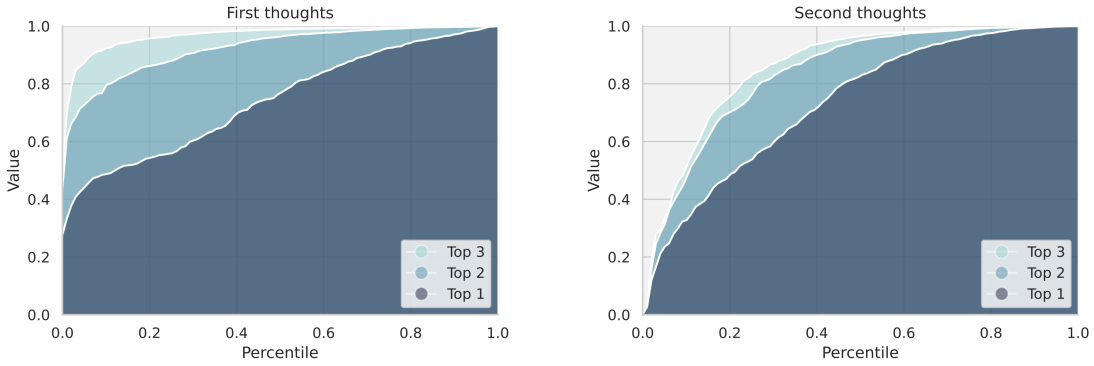

Figure 4.6. Analysis of parallelism in the first two steps of the latent tree search. The three curves in each panel depict the cumulative value of the top-1, top-2, and top-3 candidate nodes across the test set.

Figure 4.6 analyzes the parallelism in the model's latent reasoning across the first and second thoughts. For the first thoughts (left), the cumulative values of the top-1, top-2, and top-3 candidate nodes show noticeable gaps, indicating the model maintains significant diversity in its reasoning paths—a broad exploration of alternatives. In contrast, the second thoughts (right) show a narrowing of these gaps, suggesting the model transitions from parallel exploration to more focused reasoning as it gains certainty about the most promising paths.

4.4.4 Why Is Latent Space Better for Planning?

Building on the tree-search perspective, we examine why maintaining multiple candidate paths and postponing deterministic decisions enhances reasoning. Our hypothesis is that nodes explored in early reasoning stages are inherently more challenging to evaluate accurately because they are farther from the final target. In contrast, nodes closer to potential targets, having fewer subsequent exploration possibilities, can be assessed with higher confidence. To test this, we define the height of a node as its shortest distance to any leaf node and analyze the relationship between node height and the model's estimated value (Figure 4.7). Empirical results support our hypothesis: nodes with lower heights consistently receive more accurate and definitive probability evaluations, while nodes with greater heights exhibit more ambiguous evaluations. By delaying deterministic decisions and allowing exploration to proceed toward terminal states, latent reasoning significantly enhances the model's ability to differentiate correct from incorrect paths.

4.5 Empirical Results with COCONUT

After analyzing the promising parallel-search pattern of COCONUT, we validate the feasibility of LLM reasoning in a continuous latent space through more comprehensive experiments, highlighting its reasoning efficiency over language space, as well as its potential for test-time scaling.

4.5.1 Experimental Setup

Math reasoning. We use GSM8K [14] for math reasoning. To train the model, we use a synthetic dataset generated by Deng et al. [16]. We use two continuous thoughts for each reasoning step ($c = 2$), with 3 stages besides the initial stage, plus an additional stage where the remaining language reasoning chain is removed to handle the long-tail distribution of chains longer than 3 steps.

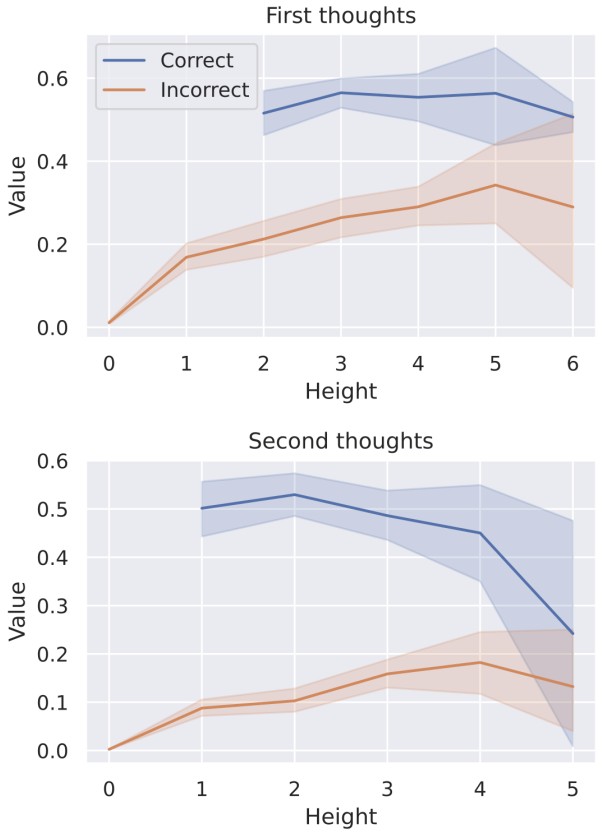

Figure 4.7. The correlation between the predicted value of correct/incorrect nodes and their heights. Nodes farther from the goal (higher height) are harder to evaluate correctly, motivating the multi-path exploration in latent space.

Logical reasoning. We use ProntoQA [81] and our newly proposed ProsQA, which is more challenging due to more distracting branches. We use one continuous thought per reasoning step ($c = 1$), with 6 training stages in addition to the initial stage, after which the model fully reasons with continuous thoughts. For all datasets, after the standard schedule, the model stays in the final training stage until reaching 50 epochs. We use greedy decoding for all experiments.

4.5.2 Baselines and Variants of COCONUT

We consider the following baselines: (1) *CoT*; (2) *No-CoT*, direct answer prediction; (3) *iCoT* [17], which is trained with language reasoning chains and follows a schedule that "internalizes" CoT by gradually removing tokens from the beginning of the chain; and (4) *Pause token* [30], where special `<pause>` tokens are inserted between the question and answer (same

Table 4.1. Results on GSM8K, ProntoQA, and ProsQA. Higher accuracy indicates stronger reasoning ability, while generating fewer tokens (#Tok) indicates better efficiency. *The result is from Deng et al. [17].

	GSM8k		ProntoQA		ProsQA	
Method	Acc. (%)	#Tok	Acc. (%)	#Tok	Acc. (%)	#Tok
CoT	42.9	25.0	98.8	92.5	77.5	49.4
No-CoT	16.5	2.2	93.8	3.0	76.7	8.2
iCoT	30.0*	2.2	99.8	3.0	98.2	8.2
Pause Token	16.4	2.2	77.7	3.0	75.9	8.2
COCONUT (Ours)	**34.1**	8.2	**99.8**	9.0	**97.0**	14.2
w/o curriculum	14.4	8.2	52.4	9.0	76.1	14.2
w/o thought	21.6	2.3	99.9	3.0	95.5	8.2
pause as thought	24.1	2.2	100.0	3.0	96.6	8.2

count as COCONUT's continuous thoughts). We also evaluate variants of COCONUT: (1) *w/o curriculum*, which directly trains in the last stage; (2) *w/o thought*, which keeps the multi-stage training but adds no continuous latent thoughts; and (3) *pause as thought*, which uses `<pause>` tokens to replace the continuous thoughts with the same multi-stage curriculum.

4.5.3 Results and Discussion

Table 4.1 presents the main results. Using continuous thoughts effectively enhances LLM reasoning over the No-CoT baseline; e.g., COCONUT achieves 34.1% on GSM8K, significantly outperforming No-CoT (16.5%). We highlight several findings.

"Chaining" continuous thoughts enhances reasoning. Generating more tokens serves as a way of inference-time scaling for reasoning. This property holds naturally for COCONUT. On GSM8K, COCONUT outperforms other architectures trained with similar strategies, including *pause as thought* and *w/o thought*, and surpasses the latest baseline *iCoT* [17], which requires a more carefully designed training schedule. Adjusting the hyperparameter c—the number of latent thoughts per reasoning step—from 0 to 1 to 2 steadily improves performance (Figure 4.8, II), validating the potential of continuous thoughts to scale to harder problems.

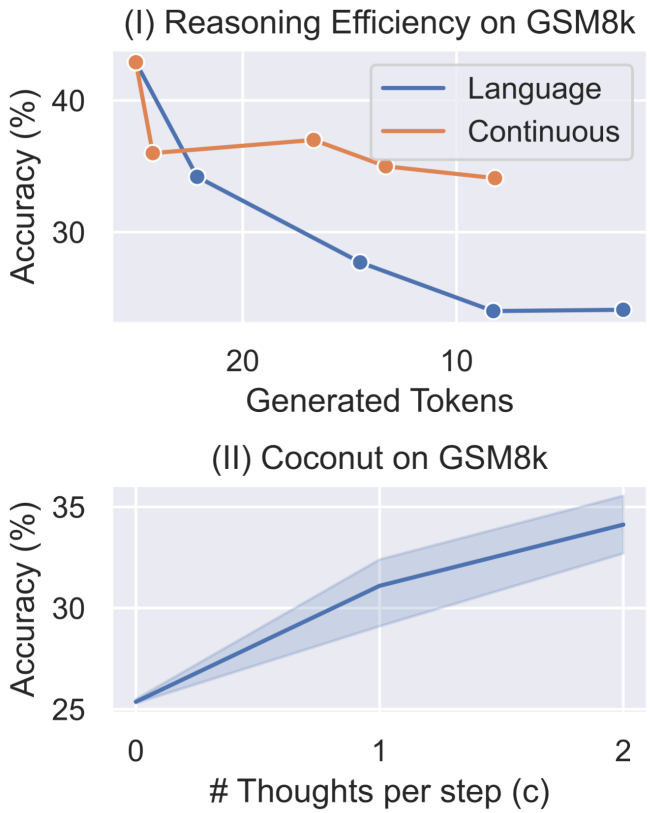

Figure 4.8. (I) Accuracy vs. number of generated tokens for language CoT (progressive internalization) and COCONUT on GSM8K; COCONUT degrades much more gracefully as fewer tokens are used. (II) Accuracy of COCONUT on GSM8K as the number of continuous thoughts per step c increases.

Continuous thoughts are efficient representations of reasoning. Compared to CoT, COCONUT generates fewer tokens while achieving higher accuracy on ProntoQA and ProsQA. Although COCONUT does not surpass CoT on GSM8K, it offers a superior trade-off between efficiency and accuracy (Figure 4.8, I): we train a series of CoT models that progressively "internalize" the initial reasoning steps, and find these models quickly lose accuracy as more steps are skipped, whereas COCONUT substantially mitigates the drop. Another interesting observation is that when we decode the first continuous thought, it often corresponds to possible intermediate variables in the calculation, suggesting that continuous thoughts are more efficient representations of reasoning.

The LLM still needs guidance to learn latent reasoning. In the ideal case, the model should

learn the most effective continuous thoughts automatically through gradient descent on questions and answers (i.e., COCONUT *w/o curriculum*). However, the models trained this way do not perform any better than No-CoT. On the contrary, with the multi-stage curriculum, COCONUT achieves top performance across various tasks. While the multi-stage training used for COCONUT has proven effective, further research is needed to develop better and more general strategies for learning reasoning in latent space, especially without supervision from language reasoning chains.

4.6 Conclusion

In this chapter, we introduced COCONUT, a new paradigm for reasoning in a continuous latent space. Experiments demonstrate that COCONUT effectively enhances LLM performance across a variety of reasoning tasks. Reasoning in latent space gives rise to advanced emergent behaviors, where continuous thoughts can represent multiple alternative next steps. This enables the model to perform a breadth-first search over possible reasoning paths, rather than prematurely committing to a single deterministic trajectory as in language-space CoT reasoning. Further research is needed to refine and scale latent reasoning to pretraining, which could improve generalization across a broader range of reasoning challenges. We hope our findings will spark continued exploration into latent reasoning, ultimately advancing the development of more capable machine reasoning systems—a direction connected to both the structured search of Chapter 2 and the broader question, raised throughout this dissertation, of whether language is the right substrate for machine reasoning.

Acknowledgements

Chapter 4, in full, is a reprint of the material as it appears in Proceedings of the Conference on Language Modeling 2025. Hao, Shibo; Sukhbaatar, Sainbayar; Su, DiJia; Li, Xian; Hu, Zhiting; Weston, Jason; Tian, Yuandong, 2025. The work was conducted while the dissertation

author was a visiting researcher at FAIR at Meta. The dissertation author was the primary investigator and author of this paper. The authors thank Jihoon Tack for valuable discussions.

Chapter 5

Conclusion and Future Directions

5.1 Summary of Contributions

This dissertation has investigated three approaches to LLM reasoning that move beyond the constraints of language imitation. Each chapter addressed a distinct bottleneck in the current paradigm and demonstrated that substantial improvements are possible when reasoning is not confined to imitating the surface form of human language text.

Chapter 2: Reasoning via Planning. We proposed the RAP framework, which reframes LLM reasoning as a planning problem. By repurposing the LLM as both a reasoning agent and a world model, and applying Monte Carlo Tree Search to explore the reasoning space, RAP enables structured exploration, backtracking from mistakes, and reward-guided search. On Blocksworld plan generation, RAP with LLaMA-33B surpasses GPT-4 with CoT by 33% relative improvement. On GSM8K math reasoning and PrOntoQA logical reasoning, RAP consistently outperforms strong chain-of-thought baselines. The key contribution is demonstrating that inference-time structured search, guided by an LLM world model, can dramatically improve reasoning quality without any additional training.

Chapter 3: ToolkenGPT. We introduced ToolkenGPT, which augments frozen LLMs with massive external tools through toolken embeddings—lightweight vectors appended to the LLM head, one per tool. At training time, only the toolken embeddings are updated, leaving the LLM frozen. At inference time, predicting a toolken triggers execution of the corresponding tool.

This plug-and-play approach scales to hundreds of tools without fine-tuning and outperforms both fine-tuning-based and in-context learning baselines on numerical reasoning (4 and 13 tools), knowledge-based QA (up to 234 tools), and embodied plan generation (58 tools). The key contribution is demonstrating that external symbolic and procedural cognition—which cannot be learned from text alone—can be efficiently grafted onto LLMs through lightweight learned interfaces.

Chapter 4: COCONUT. We presented the Chain of Continuous Thought (COCONUT) paradigm, which replaces language tokens in the reasoning chain with continuous latent vectors. Rather than decoding hidden states into words, COCONUT feeds them directly back as input embeddings. A multi-stage training curriculum, which progressively replaces language steps with continuous thoughts, enables effective learning. The resulting model develops an emergent BFS-like property: continuous thoughts simultaneously encode multiple candidate reasoning branches, allowing implicit tree search within a single forward pass. COCONUT achieves better efficiency–accuracy tradeoffs than CoT on logical reasoning benchmarks and significantly reduces token generation. The key contribution is demonstrating that language is not a necessary substrate for machine reasoning—a continuous latent space enables more powerful and efficient computation.

5.2 Connections and Unified Perspective

Viewed together, the three contributions address reasoning from a progression of increasing conceptual radicality.

RAP improves reasoning by changing *how* the model searches through language reasoning paths—replacing greedy autoregressive generation with principled MCTS exploration. Language remains the medium for both the reasoning steps and the world model's state descriptions. The departure from language imitation is at the *process* level: instead of imitating the linear, single-path structure of human written solutions, RAP simulates the exploratory,

backtracking structure of human deliberation.

ToolkenGPT extends reasoning by changing *what* can be reasoned about—incorporating symbolic and factual operations that language text cannot adequately represent. The language model remains the core reasoning engine, but its capabilities are augmented with grounded, executable tools. The departure from language imitation is at the *scope* level: language text teaches the model how to reason in language, but not how to perform symbolic computation; tools fill this gap.

COCONUT changes the *substrate* of reasoning—replacing discrete language tokens with continuous latent vectors. There is no longer a requirement that intermediate reasoning steps be expressible as natural language. The departure from language imitation is at the most fundamental level: the reasoning process itself is decoupled from the language interface, which is reserved only for the final communicative output.

A deeper connection ties these three works: all three can be understood through the lens of *exploring a reasoning tree more effectively*. RAP does so explicitly, through MCTS over language reasoning steps. ToolkenGPT does so implicitly, by expanding the reachable reasoning states to include results of external computations. COCONUT does so through emergent BFS encoded in the continuous latent representation. The question of how to best navigate the reasoning space—and what form that space should take—remains central to the study of machine reasoning.

5.3 Future Directions

The works in this dissertation open several directions for future research.

Combining the three approaches. Each chapter addresses a different aspect of the reasoning challenge. A natural next step is to combine them: training a model to reason in a continuous latent space (COCONUT), augmented with external tools (ToolkenGPT), with an explicit planning algorithm (RAP) applied over the latent reasoning process. The connection

between RAP and COCONUT is particularly interesting: RAP performs explicit tree search over language space, while COCONUT achieves implicit tree search in latent space. Can the two be combined to produce an explicit tree search over latent reasoning states?

Pretraining with continuous thoughts. COCONUT in this dissertation uses a fine-tuning curriculum initialized from a pre-trained language model. An exciting direction is to incorporate latent reasoning into the pretraining objective itself, allowing the model to develop latent reasoning strategies from scratch on large-scale data. Recent theoretical work suggests that continuous latent reasoning can be exponentially more efficient than discrete language reasoning for certain problems [69, 105].

Scalable tool-augmented reasoning. ToolkenGPT demonstrates scaling to 234 tools; real-world applications may involve thousands. Challenges include efficient toolken retrieval, compositional tool use, and tool discovery. Combining ToolkenGPT with RAP's planning capabilities could enable more complex tool-use strategies.

Reinforcement learning in latent space. Chapter 2 uses MCTS with reward signals at inference time. An important question is whether these reward signals can be used to further train the model—either via standard RL or by combining latent reasoning (COCONUT) with verifiable reward signals. Recent work on RL for LLM reasoning [31, 84] shows the power of outcome-based RL; applying it in continuous latent space is an open and promising direction.

Understanding and interpretability. The emergent BFS behavior in COCONUT suggests that continuous thoughts encode structured reasoning patterns. Developing better tools for understanding and interpreting latent reasoning—as begun in Section 4.4—is important for the safety and trustworthiness of future reasoning systems.

5.4 Closing Remarks

The central thesis of this dissertation is simple: LLMs learn to reason by imitating the text outputs of human thinking, and this creates fundamental limitations. Overcoming these

limitations requires moving beyond language imitation—either in the *process* of reasoning (RAP), the *scope* of what can be reasoned about (ToolkenGPT), or the *substrate* in which reasoning takes place (COCONUT).

Each of these directions has already had broad impact. Inference-time scaling via search—a direction RAP helped pioneer—has become central to the design of state-of-the-art reasoning models. Tool-augmented LLMs have become a standard component of production AI systems. And latent reasoning is an active area of research with deep connections to questions about the nature of machine cognition.

The gap between language imitation and genuine reasoning remains large. But the works in this dissertation demonstrate that it is possible to make meaningful progress—and that the right question to ask is not "how do we better imitate human language?" but rather "how do we build machines that reason?"

Appendix A

Supplementary Material for Reasoning via Planning

A.1 MCTS Planning Details

We adapt MCTS to search for the optimal reasoning path (Algorithm 1). Compared with traditional applications of MCTS, we are faced with a large reasoning space and the heavy computational cost of LLMs. Thus, we made several modifications to the classic MCTS in our implementation: (1) For open-domain problems, e.g., math problems, it is impossible to enumerate all actions (subquestions), so we reduce the action space by sampling a fixed number of potential actions from the LLM, conditioned on a prompt of the current state and in-context demonstrations. (2) In the selection phase, if there are actions that have not been visited before, we estimate the Q value with lightweight local rewards, e.g., the self-evaluation reward, and then select the action with UCT. This provides prior knowledge for exploration, which is crucial given the limited iteration budget.

A.2 Experiment Settings

Language model decoding. We use random sampling with a temperature of 0.8. The generation is cut off at the maximum length of 2048 tokens or a newline token.

Computing resources. All of our experiments run on 4 $\times$ NVIDIA A5000 GPUs with 24GB

memory.

A.3 Prompt Example: Plan Generation

We show the prompt used to calculate the action likelihood for RAP below. The same prompt is also applied in the CoT baseline. <init_state> and <goals> are instantiated by the problem to solve.

```
I am playing with a set of blocks where I need to arrange the blocks into
    stacks. Here are the actions I can do:
Pick up a block
Unstack a block from on top of another block
Put down a block
Stack a block on top of another block
I have the following restrictions on my actions:
I can only pick up or unstack one block at a time.
I can only pick up or unstack a block if my hand is empty.
I can only pick up a block if the block is on the table and the block is
    clear. A block is clear if the block has no other blocks on top of it and
    if the block is not picked up.
I can only unstack a block from on top of another block if the block I am
    unstacking was really on top of the other block.
I can only unstack a block from on top of another block if the block I am
    unstacking is clear.
Once I pick up or unstack a block, I am holding the block.
I can only put down a block that I am holding.
I can only stack a block on top of another block if I am holding the block
    being stacked.
I can only stack a block on top of another block if the block onto which I am
    stacking the block is clear.
```

```
Once I put down or stack a block, my hand becomes empty.


[STATEMENT]
As initial conditions I have that, the red block is clear, the yellow block is
   clear, the hand is empty, the red block is on top of the blue block, the
   yellow block is on top of the orange block, the blue block is on the table
   and the orange block is on the table.
My goal is to have that the orange block is on top of the red block.
My plan is as follows:
[PLAN]
unstack the yellow block from on top of the orange block
put down the yellow block
pick up the orange block
stack the orange block on top of the red block
[PLAN END]


[STATEMENT]
As initial conditions I have that, <init_state>
My goal is to have that <goals>.
My plan is as follows:
[PLAN]
```

For the next-state prediction with the world model, we apply prompts conditioned on the last action. The prompt instructs the LLM (as world model) to update the configuration of blocks given the current state `<state>` and the action `<action>`, removing conditions that are no longer true and adding conditions that newly hold.

Appendix B

Supplementary Material for ToolkenGPT

B.1 Details of Numerical Reasoning

For a fair comparison, we use the same prompts for CoT and the reasoning mode of TOOLKENGPT. With the same example questions, we label the calculation process in place to obtain the prompts for ReAct. For the tool mode of TOOLKENGPT, we randomly sample 4 examples of the specified tool from the training set and transform them into ReAct-style prompts.

B.1.1 GSM8K-XL Data Synthesis

To build GSM8K-XL by magnifying numbers in GSM8K, we take the following steps. (1) We prompt ChatGPT (`gpt-3.5-turbo`) with two examples to replace the numbers with appropriate placeholders. (2) To validate the correctness of the number replacements, we develop a verification function for GSM8K that calculates the formulas embedded in the solutions (quoted with ¡¡ and ¿¿); by substituting the original numbers back into the rewritten question–answer pairs and comparing the execution results with the original answer, we identify and exclude any erroneous rewrites. (3) We then replace the original numbers with their respective cubic values, exponentially amplifying their magnitudes (preserving values such as percentages where cubic magnification would be inappropriate). (4) Finally, we substitute the placeholders within the rewritten pairs and use the solving function to obtain accurate solutions, resulting in the GSM8K-XL dataset.

B.1.2 Training Details

The toolken embeddings for GSM8K-XL are trained with a subset of 5,063 examples, with an additional 1,000 examples reserved for validation, using a learning rate of 5×10^{-4} and early stopping with a maximum of 10 epochs. For FuncQA, the toolken embeddings are trained using 611 examples, with 39 examples reserved for validation, using a learning rate of 1×10^{-4} and early stopping with a maximum of 20 epochs.

B.1.3 Prompt Examples

The prompt for Chain-of-Thought (CoT) and the TOOLKENGPT reasoning mode on GSM8K-XL:

```
Answer the following questions step by step.
Question: Mark has 3 tanks for pregnant fish. Each tank has 4 pregnant fish
    and each fish gives birth to 20 young. How many young fish does he have at
    the end?
Answer: He has 4*3=12 pregnant fish They give birth to 12*20=240 fish #### 240
Question: A clothing store sells 20 shirts and 10 pairs of jeans. A shirt
    costs $10 each and a pair of jeans costs twice as much. How much will the
    clothing store earn if all shirts and jeans are sold?
Answer: Twenty shirts amount to $10 x 20 = $200. The cost of each pair of
    jeans is $10 x 2 = $20. So 10 pairs of jeans amount to $20 x 10 = $200.
    Therefore, the store will earn $200 + $200 = $400 if all shirts and jeans
    are sold. #### 400
Question: [QUESTION]
Answer:
```

The corresponding prompt for ReAct, where the calculation process is labeled in place with tool calls:

```
Answer the following questions with <add>, <subtract>, <multiply>, <divide>
   operators
Question: Mark has 3 tanks for pregnant fish. Each tank has 4 pregnant fish
   and each fish gives birth to 20 young. How many young fish does he have at
   the end?
Answer: He has 4*3=<multiply>(4, 3)=12 pregnant fish They give birth to
   12*20=<multiply>(12, 20)=240 fish #### 240
Question: [QUESTION]
Answer:
```

For FuncQA, the toolken embeddings are trained on synthetic one-hop data. To create the training data, we manually craft two examples adhering to the desired format and then prompt ChatGPT to generate more examples for each operator (with subsequent filtering and de-duplication), where numbers are replaced by placeholders so that arbitrary values can be substituted.

B.2 Details of Knowledge-based Question Answering

For TOOLKENGPT (sup), we sample 200 examples per relation from the KAMEL training set. For TOOLKENGPT (syn), we use the text description of each relation to synthesize, on average, 40 examples per toolken with ChatGPT. The base model for all KAMEL experiments is LLaMA-13B. The four test subsets correspond to questions involving 30, 60, 100, and 234 relations respectively, each of size 500.

B.3 Details of Embodied Plan Generation

We consider all 25 verbs and 32 objects in our VirtualHome subset as tools, plus an `[END]` toolken, for 58 toolkens in total. Since these tools do not take arguments, TOOLKENGPT alternately generates action toolkens and object toolkens, ending with `[END]`. The toolken

embeddings are trained on the 247 training tasks and evaluated on 50 test tasks. The base model is LLaMA-13B. For the Translation baseline, we use SentenceRoBERTa-large to map predicted actions/objects to the available ones with the highest cosine similarity; for Grounded Decoding, we apply the affordance grounding function to encourage valid actions and objects.

Appendix C

Supplementary Material for COCONUT

C.1 Datasets

C.1.1 Construction of ProsQA

To construct ProsQA, we first compile a set of typical entity names, such as "Alex" and "Jack," along with fictional concept names like "lorpus" and "rorpus," following the setting of ProntoQA [81]. Each problem is structured as a binary question: "Is [Entity] a [Concept A] or [Concept B]?" Assuming [Concept A] is the correct answer, we build a directed acyclic graph (DAG) where each node represents an entity or a concept, such that a path exists from [Entity] to [Concept A] but not to [Concept B]. The DAG is incrementally built by adding nodes and randomly connecting them with edges. To preserve the validity of the binary choice, with some probability we enforce that a new node cannot simultaneously serve as a descendant of both the [Concept A] and [Concept B] families. This separation maintains distinct families of nodes and balances their sizes to prevent model shortcuts. After the graph is constructed, nodes without parents are assigned entity names, while other nodes receive concept names. To avoid positional biases, [Concept A] and [Concept B] are randomly permuted in each question.

Table C.1. Statistics of the graph structure in ProsQA (averaged over the dataset).

# Nodes	# Edges	Len. of Shortest Path	# Shortest Paths
23.0	36.0	3.8	1.6

C.1.2 Dataset Statistics

Table C.2 reports the size of all datasets used in our experiments.

Table C.2. Statistics of the datasets.

Dataset	Training	Validation	Test
GSM8K	385,620	500	1,319
ProntoQA	9,000	200	800
ProsQA	17,886	300	500

C.2 Clock-Time Reasoning Efficiency

We present a clock-time comparison to evaluate reasoning efficiency. The reported values represent the average inference time per test case (in seconds), with a batch size of 1, measured on an NVIDIA A100 GPU (Table C.3). For the No-CoT and CoT baselines, we employ the standard `generate` method from the `transformers` library. Our results show that clock time is generally proportional to the number of newly generated tokens.

Table C.3. Inference time (in seconds) comparison across tasks and methods.

Method	GSM8K	ProntoQA	ProsQA
No-CoT	0.03	0.03	0.08
CoT	0.26	0.85	0.47
COCONUT	0.09	0.11	0.15

C.3 Additional Discussion

C.3.1 Using More Continuous Thoughts

In Figure 4.8 (II), we present the performance of COCONUT on GSM8K using $c \in \{0, 1, 2\}$. When experimenting with $c = 3$, we observe a slight performance drop accompanied by increased variance. Analysis of the training logs indicates that adding three continuous thoughts at once—particularly during the final stage transition—leads to a sharp spike in training

loss, causing instability. Future work could explore finer-grained schedules, such as incrementally adding continuous thoughts one at a time while removing fewer language tokens, as in iCoT [17]. Additionally, combining language and latent reasoning—e.g., generating the reasoning skeleton in language and completing the reasoning process in latent space—could improve performance and stability.

C.3.2 COCONUT with Larger Models

We experimented with COCONUT on GSM8K using Llama 3.2-3B and Llama 3-8B [20] with $c = 1$, training for 3 epochs in Stage 0 followed by 1 epoch per subsequent stage (Table C.4). We observe consistent performance gains over the No-CoT baseline, though the improvements are not as pronounced as those demonstrated with GPT-2. One possible reason is that larger models have already undergone extensive language-focused pre-training, making the transition to latent reasoning more challenging. We emphasize that the primary goal of this work is to highlight the promising attributes of latent-space reasoning and to initiate exploration in this direction. Universally surpassing language-based CoT likely requires significant research dedicated to **latent-space pre-training**, and integrating recent advances in this area with COCONUT presents an exciting avenue for future research.

Table C.4. Experimental results of applying COCONUT to larger Llama models, comparing models without CoT reasoning (No-CoT) and our proposed COCONUT method.

Model	**No-CoT**	**COCONUT (Ours)**
Llama 3.2-3B	26.0	31.7
Llama 3-8B	42.2	43.6

Bibliography

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Marie Amalric and Stanislas Dehaene. A distinct cortical network for mathematical knowledge in the human brain. *NeuroImage*, 189:19–31, 2019.

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

[4] Eden Biran, Daniela Gottesman, Sohee Yang, Mor Geva, and Amir Globerson. Hopping too late: Exploring the limitations of large language models on multi-hop queries. *arXiv preprint arXiv:2406.12775*, 2024.

[5] Michael Bommarito II and Daniel Martin Katz. Gpt takes the bar exam. *arXiv preprint arXiv:2212.14402*, 2022.

[6] 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. PMLR, 2022.

[7] Robert Eamon Briscoe. Mental imagery and the varieties of amodal perception. *Pacific Philosophical Quarterly*, 92(2):153–173, 2011.

[8] 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.

[9] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

[10] Tom Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

[11] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer science & business media, 2013.

[12] Harrison Chase. LangChain, 10 2022. URL https://github.com/hwchase17/langchain.

[13] 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.

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

[15] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games: 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers 5*, pages 72–83. Springer, 2007.

[16] Yuntian Deng, Kiran Prasad, Roland Fernandez, Paul Smolensky, Vishrav Chaudhary, and Stuart Shieber. Implicit chain of thought reasoning via knowledge distillation. *arXiv preprint arXiv:2311.01460*, 2023.

[17] Yuntian Deng, Yejin Choi, and Stuart Shieber. From explicit cot to implicit cot: Learning to internalize cot step by step. *arXiv preprint arXiv:2405.14838*, 2024.

[18] Yan Ding, Xiaohan Zhang, Chris Paxton, and Shiqi Zhang. Task and motion planning with large language models for object rearrangement. *arXiv preprint arXiv:2303.06247*, 2023.

[19] Iddo Drori, Sarah Zhang, Reece Shuttleworth, Leonard Tang, Albert Lu, Elizabeth Ke, Kevin Liu, Linda Chen, Sunny Tran, Newman Cheng, et al. A neural network solves, explains, and generates university math problems by program synthesis and few-shot learning at human level. *Proceedings of the National Academy of Sciences*, 119(32): e2123433119, 2022.

[20] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[21] Tyna Eloundou, Sam Manning, Pamela Mishkin, and Daniel Rock. Gpts are gpts: An early look at the labor market impact potential of large language models. *arXiv preprint*

arXiv:2303.10130, 2023.

[22] Jacqueline Fagard, Lauriane Rat-Fischer, Rana Esseily, Eszter Somogyi, and JK O'Regan. What does it take for an infant to learn how to use a tool by observation? *Frontiers in psychology*, 7:267, 2016.

[23] Evelina Fedorenko, Michael K Behr, and Nancy Kanwisher. Functional specificity for high-level linguistic processing in the human brain. *Proceedings of the National Academy of Sciences*, 108(39):16428–16433, 2011.

[24] Evelina Fedorenko, Steven T Piantadosi, and Edward AF Gibson. Language is primarily a tool for communication rather than thought. *Nature*, 630(8017):575–586, 2024.

[25] Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: a theoretical perspective. *Advances in Neural Information Processing Systems*, 36, 2023.

[26] Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D Goodman. Stream of search (sos): Learning to search in language. *arXiv preprint arXiv:2404.03683*, 2024.

[27] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*, 2022.

[28] Dedre Gentner and Albert L Stevens. *Mental models*. Psychology Press, 2014.

[29] Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, pages 11398–11442. PMLR, 2023.

[30] Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. *arXiv preprint arXiv:2310.02226*, 2023.

[31] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[32] 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. PMLR, 2020.

[33] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*,

2018.

[34] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[35] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[36] 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. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[37] Shibo Hao, Bowen Tan, Kaiwen Tang, Bin Ni, Xiyan Shao, Hengzhe Zhang, Eric Xing, and Zhiting Hu. BertNet: Harvesting knowledge graphs with arbitrary relations from pretrained language models. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 5000–5015, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.309. URL https://aclanthology.org/2023.findings-acl.309.

[38] Mark K Ho, David Abel, Carlos G Correa, Michael L Littman, Jonathan D Cohen, and Thomas L Griffiths. Control of mental representations in human planning. *arXiv e-prints*, pages arXiv–2105, 2021.

[39] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[40] Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2021.

[41] Jie Huang and Kevin Chen-Chuan Chang. Towards reasoning in large language models: A survey. *arXiv preprint arXiv:2212.10403*, 2022.

[42] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147. PMLR, 2022.

[43] 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.

[44] Wenlong Huang, Fei Xia, Dhruv Shah, Danny Driess, Andy Zeng, Yao Lu, Pete Florence, Igor Mordatch, Sergey Levine, Karol Hausman, et al. Grounded decoding: Guiding text generation with grounded models for robot control. *arXiv preprint arXiv:2303.00855*, 2023.

[45] Quentin JM Huys, Neir Eshel, Elizabeth O'Nions, Luke Sheridan, Peter Dayan, and Jonathan P Roiser. Bonsai trees in your head: how the pavlovian system sculpts goal-directed choices by pruning decision trees. *PLoS computational biology*, 8(3):e1002410, 2012.

[46] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, 2023.

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

[48] Philip N Johnson-Laird. Mental models and human reasoning. *Proceedings of the National Academy of Sciences*, 107(43):18243–18250, 2010.

[49] Philip Nicholas Johnson-Laird. *Mental models: Towards a cognitive science of language, inference, and consciousness*. Number 6. Harvard University Press, 1983.

[50] Daniel Kahneman. *Thinking, fast and slow*. macmillan, 2011.

[51] Jan-Christoph Kalo and Leandra Fichtel. Kamel: Knowledge analysis with multitoken entities in language models. In *Proceedings of the Conference on Automated Knowledge Base Construction*, 2022.

[52] Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*, 2022.

[53] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006: 17th European Conference on Machine Learning Berlin, Germany, September 18-22, 2006 Proceedings 17*, pages 282–293. Springer, 2006.

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

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

[56] Lucas Lehnert, Sainbayar Sukhbaatar, Paul Mcvay, Michael Rabbat, and Yuandong Tian. Beyond a*: Better planning with transformers via search dynamics bootstrapping. *arXiv preprint arXiv:2402.14083*, 2024.

[57] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, 2021.

[58] 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.

[59] Belinda Z Li, Maxwell Nye, and Jacob Andreas. Language modeling with latent situations. *arXiv preprint arXiv:2212.10012*, 2022.

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

[61] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. 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 4582–4597, 2021.

[62] Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *arXiv preprint arXiv:2303.16434*, 2023.

[63] 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.

[64] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*, 2021.

[65] Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. Faithful chain-of-thought reasoning. *arXiv preprint arXiv:2301.13379*, 2023.

[66] Aman Madaan and Amir Yazdanbakhsh. Text and patterns: For effective chain of thought, it takes two to tango. *arXiv preprint arXiv:2209.07686*, 2022.

[67] Yutaka Matsuo, Yann LeCun, Maneesh Sahani, Doina Precup, David Silver, Masashi Sugiyama, Eiji Uchibe, and Jun Morimoto. Deep learning, reinforcement learning, and world models. *Neural Networks*, 2022.

[68] John McCarthy. Situations, actions, and causal laws. Technical report, STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1963.

[69] William Merrill and Ashish Sabharwal. The expresssive power of transformers with chain of thought. *arXiv preprint arXiv:2310.07923*, 2023.

[70] 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.

[71] Martin M Monti, Lawrence M Parsons, and Daniel N Osherson. Thought beyond language: neural dissociation of algebra and natural language. *Psychological science*, 23(8):914–922, 2012.

[72] 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.

[73] OpenAI. Gpt-4 technical report, 2023.

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

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

[76] Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. Refiner: Reasoning feedback on intermediate representations. *arXiv preprint arXiv:2304.01904*, 2023.

[77] Jacob Pfau, William Merrill, and Samuel R Bowman. Let's think dot by dot: Hidden computation in transformer language models. *arXiv preprint arXiv:2404.15758*, 2024.

[78] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs, 2018.

[79] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng

Qian, Runchu Tian, Kunlun Zhu, Shi Liang, Xingyu Shen, Bokai Xu, Zhen Zhang, Yining Ye, Bo Li, Ziwei Tang, Jing Yi, Yu Zhu, Zhenning Dai, Lan Yan, Xin Cong, Ya-Ting Lu, Weilin Zhao, Yuxiang Huang, Jun-Han Yan, Xu Han, Xian Sun, Dahai Li, Jason Phang, Cheng Yang, Tongshuang Wu, Heng Ji, Zhiyuan Liu, and Maosong Sun. Tool learning with foundation models. *ArXiv*, abs/2304.08354, 2023.

[80] Stephen Roller, Emily Dinan, Naman Goyal, Da Ju, Mary Williamson, Yinhan Liu, Jing Xu, Myle Ott, Eric Michael Smith, Y-Lan Boureau, et al. Recipes for building an open-domain chatbot. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 300–325, 2021.

[81] Abulhair Saparov and He He. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. *arXiv preprint arXiv:2210.01240*, 2022.

[82] 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.

[83] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588 (7839):604–609, 2020.

[84] Rulin Shao, Yicong Tian, et al. Spurious rewards: Rethinking training signals in rlvr. *Advances in Neural Information Processing Systems*, 2025.

[85] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, YK Li, Yu Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[86] Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic memory and self-reflection. *ArXiv*, abs/2303.11366, 2023.

[87] Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. Retrieval augmentation reduces hallucination in conversation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3784–3803, 2021.

[88] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[89] 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. *arXiv preprint arXiv:2209.11302*, 2022.

[90] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

[91] Edward C Tolman. Cognitive maps in rats and men. *Psychological review*, 55(4):189, 1948.

[92] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

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

[94] 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.

[95] Karthik Valmeekam, Sarath Sreedharan, Matthew Marquez, Alberto Olmo, and Subbarao Kambhampati. On the planning abilities of large language models (a critical investigation with a proposed benchmark). *arXiv preprint arXiv:2302.06706*, 2023.

[96] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

[97] 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.

[98] Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. Generating sequences by learning to self-correct. *arXiv preprint arXiv:2211.00053*, 2022.

[99] Sohee Yang, Elena Gribovskaya, Nora Kassner, Mor Geva, and Sebastian Riedel. Do large language models latently perform multi-hop reasoning? *arXiv preprint arXiv:2402.16837*, 2024.

[100] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and

Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

[101] 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. *arXiv preprint arXiv:2305.10601*, 2023.

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

[103] Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024.

[104] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

[105] Hanlin Zhu, Shibo Hao, Zhiting Hu, Jiantao Jiao, Stuart Russell, and Yuandong Tian. Reasoning by superposition: A theoretical perspective on chain of continuous thought. *arXiv preprint arXiv:2505.12514*, 2025.

[106] 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.