A Survey on LLM Test-Time Compute via Search: Tasks, LLM Profiling, Search Algorithms, and Relevant Frameworks

Anonymous authors
Paper under double-blind review

Abstract

LLM test-time compute (or LLM inference) via search has emerged as a promising research area with rapid developments. However, current frameworks often adopt distinct perspectives on three key aspects—task definition, LLM profiling, and search procedures—making direct comparisons challenging. Moreover, the search algorithms employed often diverge from standard implementations, and their specific characteristics are not thoroughly specified. In this survey, we provide a comprehensive technical review that unifies task definitions and provides modular definitions of LLM profiling and search procedures. The definitions enable precise comparisons of various LLM inference frameworks while highlighting their departures from conventional search algorithms. We also discuss the applicability, performance, and efficiency of these methods.

1 Introduction

Scaling test-time compute via search has recently enhanced the LLMs' power to a new level on reasoning tasks (Yao et al., 2023a; Hao et al., 2023), sequential decision-making tasks (e.g., robotics) (Putta et al., 2024), and graph-traversal tasks (e.g., path finding) (Meng et al., 2024). This survey aims to provide a comprehensive but integrated survey on existing frameworks for **L**LM **I**nference via **S**earch (**LIS**). The focus is on the work that the search processes are coupled with LLMs' test time compute rather than those using search and LLMs separately. For example, the plans (commonly in the form of PDDL) are prepared by LLMs to perform local search (Valmeekam et al., 2023b; Guan et al., 2023; Valmeekam et al., 2023a).

1.1 Existing Surveys

Current reviews on LLM search are limited from the following perspectives.

No dedicated, Detailed Survey Current surveys only contain paragraphs or sections to roughly touch on both technical aspects and their practical applicability, as summarized in Table 1.

Limited Mention on LLM-Side Design Specifically, most of existing surveys (Huang et al., 2024; Wang et al., 2024b) mention little or a few implementations and dimensions for LLM profiling, which is not suitable for all the frameworks. Besides, the lack of examples hinders understanding.

Limited Mention on Search Li (2024) give more detail regarding LLM profiling but lack details on search processes. Nonthelessness, most of existing surveys (Huang et al., 2024; Wang et al., 2024b) give a general sense of the computation process by mentioning which classical search algorithms the frameworks are built upon (e.g., depth-first search). However, details should be given because of their nuanced differences. Besides, many untypical twists to classic search algorithms are hidden. The deviations are not friendly for newcomers in the newly-developed area, e.g., those computer science graduates educated with typical search algorithms.

Table 1: Comparisons with other surveys related to LLM inference via search (LIS). **"Coverage"** indicates the number of related papers, **"Sections"** refers to the sections of the survey manuscripts that discuss LLM inference via search, and **"Def."** is the abbreviation for Definition. The URLs for the reference versions are given at the footnotes. **"Mentioned"** refers to the mention of the LLM function, while **"Limited"** means that a formal definition is given without specific distinctions. The numerical values correspond to the number of implementations (**impl.**) or dimensions (**dim.**) identified for LLM-Profiled Roles (LMPRs); different dimensions may lead to a combinatorial number of implementations. The "11" in **"20+"** specifically refers to the frameworks we describe in detail.

	Coverage	Sections	Task Def.	Search Algorithms		LLM Profiling		
				Details	Deviations	Policy	Value	Transition
Huang et al. (2024)	5	✓(§4)	✗	✗	✗	1 impl.	1 impl.	✗
Wang et al. (2024b)	3	✓(§2.1.3)	✗	✗	✗	✗	✗	Mentioned
Li (2024)	8	✓(§4.3)	✗	Limited	✗	2 impl.	2 dim.	NA
Ours	20+	All	✓	✓	✓	4 impl.	4 dim.; 12 impl.	2 impl.

1.2 Survey Structure

To solve the above limitations, we provide unified **task definitions** and decouple the **LLM-specific design (mainly prompting)** from **the control program (search procedures/algorithms)**. There exists a hierarchical structure between them: the low-level definitions provide a unified interface for the high-level components. The overall structure, accompanied by illustrative examples, is presented in Figure 1.

Introducing a Unified Task Definition Based on MDPs (§ 2) Our definition standardizes different tasks in MDP structure. While MDPs naturally align with AI domains like robotics, special attention is given to adapting this definition for tasks traditionally not modeled as MDPs, such as graph traversal, reasoning, dialogue systems, and code generation. Notably, this MDP-based definition is also applicable to other LLM inference frameworks beyond search, including works like Li et al. (2022), Zhao et al. (2023), and Hu et al. (2024).

Comprehensively Summarizing LLM Profiling and Implementations (§ 3) The design and implementation of LLM profiling and prompting can be modularized into components commonly used in solving MDPs (Sutton & Barto, 2018): policies, value functions, and transition models. Correspondingly, 3 types of LLM-Profiled Roles (LMPRs) are defined.

Defining Modular Search Procedures (§ 4) Rather than directly showcasing individual search-based frameworks for LLM inference, we focus on modular and reusable components to reduce redundancy and enable more straightforward comparisons across frameworks. This approach promotes flexibility and minimizes overhead when adapting or extending search methods.

Reviewing Individual Frameworks (§ 5) Based on the unified task and LMPR interface, we provide a comprehensive review of individual frameworks, organized by the search algorithms they are built upon. Our analysis highlights how LLM integration either diverges from or enhances traditional search algorithms. We identify and clearly present 11 frameworks, summarized in Table 7. This count exclusively includes frameworks that focus on test-time computation through search detailed in Section 5. Additionally, we highlight other test-time frameworks that function as components within search processes, such as ReAct (Yao et al., 2023b), CoT (Wei et al., 2022), and Self-Consistency (Wang et al., 2023), along with those discussed in Section 8.

Analyzing Key Perspectives of LIS Frameworks (§ 6) We critically examine these methods from four key perspectives: **deviation**, **applicability**, **performance**, and **efficiency**. For deviations, we compare the structural and functional differences between search procedures in LIS frameworks and standard search algorithms as described in foundational texts such as Russell & Norvig (2010) and Sutton & Barto (2018). This analysis highlights how LLMs modify or enhance traditional search processes, providing a deeper understanding of their impact and potential.

Other LLM Inference + Search Directions (§ 7) This survey primarily focuses on LLM inference via search, where downstream tasks are formulated as sequential decision-making problems, and LLMs serve as integral components. This focus allows us to present a detailed, concise, and systematic analysis. However, confining our discussion solely to these frameworks might give the impression that the title overstates our scope. Hence, Section 7 covers additional research directions that also involve LLM inference and search.

1.3 Intended Audience and Use Cases

Reusable Modules While we strive to provide comprehensive coverage of the latest research, we acknowledge the rapid pace of advancements in the field, where variations in LLM profiling and search implementations may not be covered. Nonetheless, this survey offers a collection of classical and reusable implementations that can serve as solid foundations for future research and development.

Anchoring Purpose Our work serves as a valuable reference in two key ways: **1) Incorporating Novel Designs with Minimal Adjustments**: During the development of this survey, we seamlessly integrated emerging methods with minor modifications. For example, we separated single-step simulation from path simulation in MCTS to accommodate LATS (Zhou et al., 2024a) and decoupled value-based selection from LMPE+ or state-based evaluation to allow specialized ways to assign values, e.g., using parents' states for evaluation Koh et al. (2024). **2) Expanding with Additional Details**: Although we intentionally omit details unrelated to search procedures—such as how the reflection module is incorporated in LATS (Zhou et al., 2024a)—our structured presentation of search control flows serves as a stable anchor for integrating such complex components. Although many of these design elements are non-trivial, our framework simplifies readers' understanding of their integration by using the search process as a guiding anchor.

For Research Engineers To support practical implementation, search procedures (Section 4) are presented in an Object-Oriented Programming (OOP) style, promoting modularity and ease of integration for research engineers.

Limitations We adopt Markov Decision Process (MDP) definitions to unify and compare various methods due to their comprehensive nature. However, this formalism may feel excessive for readers focused on specific frameworks or tasks where transition and action definitions are unnecessary. For instance, the Tree-of-Thoughts (Yao et al., 2023a) approach could be more intuitively understood without relying on MDP foundations.

2 Task (Re)formulation

Tasks solved by LLM-integrated search are inherited from both the "LLM" side (human language tasks) and the "search" side (structured tasks): 1) language reasoning: LLMs are naturally applied to reasoning tasks in language (Wei et al., 2022). 2) structured tasks: On the other hand, search algorithms are more conventionally utilized for structured tasks, such as web navigation, robotic navigation, gaming, and graph traversal (Russell & Norvig, 2010; Sutton & Barto, 2018). The convergent nature is that all of them belongs to sequential decision-making, e.g., reasoning often involves generating and evaluating sequences of logical steps or decisions to arrive at a conclusion.

A MDP-Like Formulation To enable a clear comparison across different frameworks, this section formulates the tasks in Markov Decision Processes (MDPs) $\langle S, A, \mathcal{T}, R \rangle$. In addition, observations O are often

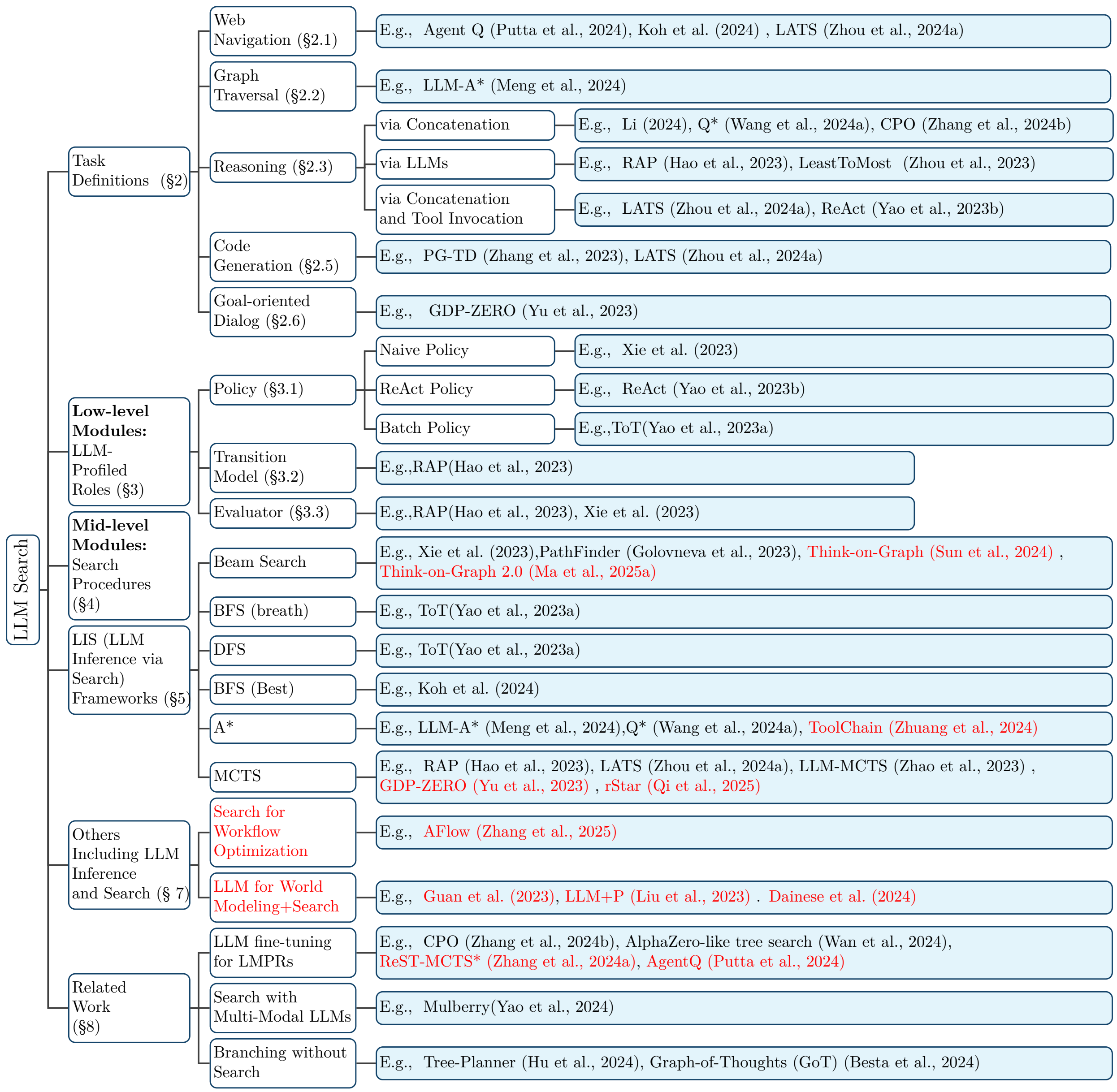

Figure 1: Survey structure. This table shows a selected subset of the works reviewed. Related work concerning **search procedures** is not included here, as these procedures are largely consistent across different frameworks and listing them in the graph would not aid in differentiating the works. Comprehensive lists of related work are provided in the following tables: Table 2 for **task definitions**, Table 4 for LLM-Profiled Evaluator, and Table 7 for **LMPRs** and **search procedures**.

considered in Partially Observable Markov Decision Processes (POMDPs) (Li et al., 2022; Zhao et al., 2023; Hu et al., 2024):

- A set of states S, including the goal state s_g;
- A set of observations O, where o_t is the partial observations of the state s_t at time step t;
- A set of actions A, where $a_t \in A$ is the action on s_t;
- Transitions $\mathcal{T}(s_{t+1} \mid s_t, a_t)$, which define the dynamics from one state to another after executing an action;

Table 2: A unification of various tasks for LLM Inference via Search (LIS). The column "Used By" denotes the LIS frameworks associated with the stated definition.

Tasks	Actions	States / Observations	Transitions	Rewards	Action Reversible?	Used By
Web Navigations	Discrete, constrained, heterogeneous	Discrete, infinite	Dynamic or Deterministic	1 if the goal is achieved, e.g., buying a coffee mug	Maybe	Agent Q (Putta et al., 2024); Koh et al. (2024)
Graph traversal	Discrete, constrained, homogeneous	Discrete, finite (commonly)	Deterministic	1 if $s \in G$	Maybe	LLM-A* (Meng et al., 2024)
Reasoning ($\mathcal{T}$ via Concatenation)	Open (A thought expressed in one or more tokens)	Open, unknown until reached (Problem description + concatenated thoughts)	Deterministic (Concatenating)	1 if the final results = groundtruth	✓	Q* Wang et al. (2024a) CPO (Zhang et al., 2024b); ToT (Yao et al., 2023a); RStar (Qi et al., 2025); MindStar (Kang et al., 2024)
Reasoning via QAs	Open	Open, unknown until reached	Dynamic (Question answering then concatenating Q&A)	1 if the final results = groundtruth	✓	RAP (Hao et al., 2023); RStar (Qi et al., 2025) ; LeastToMost (Zhou et al., 2023);
Reasoning ($\mathcal{T}$ via concatenations and tool invocation)	Open	Open, unknown until reached	Deterministic (Concatenating); dynamic or deterministic (tool invocation)	1 if the final results = groundtruth	Maybe	LATS (Zhou et al., 2024a)
Reasoning Over Knowledge Graph	Discrete, constrained, heterogeneous	Open thoughts + discrete and finite entity-relation triplets	Deterministic (Concatenating)	1 if the final results = groundtruth	✓	Think-on-Graph (Sun et al., 2024); Think-on-Graph 2.0 (Ma et al., 2025b)
Tool-based tasks	Discrete, constrained, heterogeneous	Problem description + concatenated actions	Deterministic (Concatenating)	1 if the task is completed	Maybe	ToolChain* (Zhuang et al., 2024)
Code Generation	Open (A single token)	Open, unknown until reached (Problem description + concatenated tokens)	Deterministic	pass rate of the complete program	✓	PG-TD (Zhang et al., 2023)
Goal-oriented Dialog	Discrete, constrained (An intent)	A sequence of intents, agent/user utterances	Dynamic	1 if the conversational goal is achieved	✗	GDP-Zero (Yu et al., 2023)

- Rewards R: A rewards evaluate the "quality" of a state-action pair or a trajectory towards the desired outcomes or goals.

This section only discuss task elements that are external to agents and exist independently of how agents operate or learn. We will see in the next section how the POMDP setting fits in LMPRs and agent definitions, where the Markovian assumption is broke.

A Summary of Concrete Tasks Some structured tasks (e.g., recycling robot, gridworld, and chess) are always modeled as MDPs. These typical MDP tasks are actively studied in the domains of reinforcement learning (Sutton & Barto, 2018), intelligent agents, and control theory. Actions and states are always explicitly defined. Commonly, a physical environment or a rule set defines a discrete and finite action space. And states are commonly finite and can be enumerated, e.g., all the possible configurations of chess board or the grid areas robot can travel. However, others, e.g., graph traversal and reasoning tasks, are not commonly formalized as MDPs until the emergent of LLM-based agents. In particular, the MDP elements in graph traversal and reasoning tasks are not explicit. The rest of this section mainly discusses how the these tasks fit the following MDP notations, as summarized in Table 2.

2.1 Web Navigation

Another type of tasks is to navigate on websites for shopping and retrieving information (Zhou et al., 2024b)

- States/observations: A state/observation is normally a web page. For example, the beginning state can be the homepage. The transition is governed by a deterministic transition function. The state is not always accessible to the agent. For example, "s_t may include private information such as database entries of the site" (Koh et al., 2024)).
- Reward: The reward is given when the goal g is reached, e.g., successfully ordering a product.

2.2 Graph Traversal in MDPs

A graph traversal problem, e.g., robotic navigation, is represented as a graph $G = (V, E)$, where V is a set of vertices (or nodes), and $E \subseteq V \times V$ represents the transitions between them. However, this definition is the common task settings for uninformed and informed search. However, these algorithms integrated with LLMs can be generalized beyond this definition. This is why recent work (Yao et al., 2023a) that uses these search algorithms along with LLM often uses MDP terminology, but does not include formal unification. Hence, we propose a conceptual framework that views graph traversal as a simplified, deterministic form of an MDP, where actions and transitions are predefined by the graph structure. This framework can be described as follows:

- States: Each node $v \in V$ is a state.
- Actions: Actions are represented by edges E. The action space is generally considered homogeneous because the type of action is uniform, such as "MOVE[arg]", where "[arg]" represents parameters like direction or target node.
- Transitions: Following an edge from a node s via an action (edge) $(s, s') \in E$ always leads to the same node s'. Hence, the transitions $\mathcal{T}$ are deterministic.
- Rewards $R(s)$ are typically binary, with $R(s) = 1$ if $s \in G$ (i.e., s is a goal state) and $R(s) = 0$ otherwise.
- Heuristics for nodes $h(s)$ can be interpreted as estimates of the value function in the MDP formulation, representing an estimated cost-to-go from node s to a goal node $g \in G$.

As best as we know, this conceptualization is not explicitly stated in any peer-reviewed literature.

2.3 Language Reasoning in MDP

The formulations of language reasoning tasks are more diverse and creative. Although not exhaustive, the following paragraphs summarize forms that are particularly used in the current study of LLM-integrated search.

Reasoning ($\mathcal{T}$ via Concatenation) A reasoning process can be concretized as a Chain of Thoughts (CoT) $T_1, T_2, \ldots$ (Wei et al., 2022), each expressed as a sequence of tokens via LLM generation. The reasoning steps can be not only naturally evolved but also deliberately specified. For creative writing, the first step can be specified as planning, and the second step is to generate according to the plan (Yao et al., 2023b) Following previous work (Li, 2024; Wang et al., 2024a), the MDP formulation includes:

- Actions: An action is a thought consisting of several tokens, i.e., $a_1 = T_1$.
- States: The initial state s_1 is defined by the task information, e.g., a user query, a problem description or the goal. The following states are defined as the concatenation of the following thoughts:

$$s_t = (s_{t-1}, a_{t-1}) = (s_1, a_1, \ldots, a_{t-1}) \tag{1}$$

  Apparently, directly concatnenating open actions leads to the open state space. When the reasoning is naturally evolved, the final state s_g comes when the final thought T_g or the entire chain expresses a valid response. It can be known in which step s_g is reached for deliberate reasoning steps.
- Transition $\mathcal{T}$: The deterministic state transition is defined for reasoning tasks. The next state s_{t+1} is equal to the concatenation of s_t and a_t.
- Reward R: It is given when the final answer matches the ground truth and fits in human preference. This is normally integrated as the training objective of recent LLMs (Ouyang et al., 2022).

Reasoning via QAs Some other works deliberately formulate an action space. A given task is decomposed into sequentially dependent subtasks (actions) requiring an Execute function, which relies on LLMs to solve. In other words, LLMs can be considered as transitions. Recent work on LLM search formulates a subtask as a question and use LLMs to answer the question. These subtasks (i.e., actions) can be generated all at once (Zhou et al., 2023) or in sequential order (Hao et al., 2023), each can be defined as an action a_t. s_t is the concatenation of the task information s_0 and all the questions already answered with their answers: $\text{question}_1, \text{answer}_1, \ldots, \text{question}_t, \text{answer}_t$.

Reasoning ($\mathcal{T}$ via Concatenation and Tool Invocations) Once tool definitions are given to LLMs. During reasoning, LLMs can generate specific tokens to invoke tools. This integration of tool invocation into reasoning is firstly proposed by ReAct Yao et al. (2023b) and recently adapted to the LIS framework (Zhou et al., 2024a). Based on the definitions for *Reasoning ($\mathcal{T}$ via Concatenation)*, the following things are added:

- Actions: Tool-related actions are commonly discrete and constrained. For example, when Wikipedia is used, the possible actions include search[arg], lookup[arg]. Although not always, most works on LLM tool use only include the reading-only actions, the actions are reversible.
- Transitions: Due to the change of the action space, the transitions can be either dynamic or deterministic, depending on the tools.
- States: a state now concatenates not only the LLM generation but also tool responses. Moreover, LLM generation is not only about the direct thoughts for tasks but also the actions for tool invocations.

$$s_t = (s_1, a_1, o_1, \ldots, a_{t-1}, o_{t-1}) \tag{2}$$

Reasoning Over Knowledge Graph Before defining reasoning over a knowledge graph, we clarify why this task is distinct from both graph traversal and reasoning under tool invocations.

- **Not a Tool Invocation:** In reasoning under tool invocations, LLMs autonomously decide whether to use external tools. In contrast, a knowledge graph is an integral part of the task environment on which the LLM agent operates.

- **Different from Graph Traversal:** In typical graph traversal tasks, the graph directly models a visible or observable world. A knowledge graph, however, is a carefully designed structure that captures heterogeneous semantic relationships between entities or concepts.

We now define the task of reasoning over a knowledge graph:

- States: In graph traversal, each node $v \in V$ represents a state. In this context, the state is composed of all explored nodes (entities) and their interrelationships, which collectively inform subsequent decisions and the final outcome. Additionally, any relevant LLM-generated knowledge that contributes to these decisions is incorporated into the state.
- Actions: The action space consists of exploring new relations and entities within the knowledge graph, as well as generating new knowledge through LLM outputs.
- Transitions: A transition deterministically concatenates the new information (e.g., a discovered relation or entity) with the existing state, reflecting the edge $(s, s') \in E$ between the current state and the new information.
- Rewards $R(s)$ are typically binary, with $R(s) = 1$ if $s \in G$ (i.e., s is a goal state) and $R(s) = 0$ otherwise.

2.4 Tool-Based Tasks in MDP

Unlike language reasoning under tool invocations, where tools are optionally provided and agents autonomously decide whether to use them, this kind of tasks inherently requires tool invocation. For example, an agent may need to clean a room using APIs designed for a robotic arm or send an email using an email API.

- Actions: An action is formalized using predicates and arguments, e.g., *Pick_Up[Apple]*. Since predicates can have various definitions, the action space is both discrete and heterogeneous.
- State and transition: While the properties of actions are different, the definition of state and transition are the same as language reasoning ($\mathcal{T}$ via concatenation). Specifically, the initial state s_1 is the given task description, and the following state is the concatenation of s_1 and actions in the previous steps.
- State and Transition: The definitions of state and transition remain the same as in language reasoning ($\mathcal{T}$ via concatenation). Specifically, the initial state s_1 is the given task description, and subsequent states are obtained by concatenating s_1 with actions from previous steps, as expressed in Equation 1.
- Reward: The reward is binary, where a value of 1 is given upon reaching the goal; otherwise, it remains 0.

2.5 Code Generation in MDP

This is similar to language reasoning under Deterministic $\mathcal{T}$ via Concatenation. The only difference is that an action is a token in the vocabulary set of the LLM (rather than a thought consisting of several tokens). Such definition is originally proposed by Zhang et al. (2023). Under their definition, "the reward of state s is the pass rate of the program on the public test cases. The reward of a partial program is always 0."

2.6 Goal-Oriented Dialog in MDP

Previous work (Wang et al., 2020) frames goal-oriented dialog as MDP. Yu et al. (2023) begin using such formulation for LLM-integrated search. The formulation is demonstrated below.

- Actions: An action $a \in A$ indicates the intent, which is predefined. For example, the intent to convince the Persuadee using reasoning and factual evidence is defined as "Logical Appeal". This is commonly termed "dialog act" (Wang et al., 2020).

- States: s_t is defined as the dialogue history of previous t turns, containing dialog act and agent/user utterances

$$h = \left(a_0^{\text{agent}}, \mathrm{u}_1^{\text{agent}}, \mathrm{u}_1^{\text{usr}}, \ldots, a_{t-1}^{\text{agent}}, \mathrm{u}_t^{\text{agent}}, \mathrm{u}_t^{\text{usr}}\right) \tag{3}$$

, where $\mathrm{u}_i^{\text{agent}}$, $\mathrm{u}_i^{\text{usr}}$ are the utterances of the agent and user, respectively at the $i-$th turn.

- Transitions: It "represents the dialogue state updates according to stochastic responses from the user to the agent." (Wang et al., 2020)

- Rewards: it represents the immediate feedback of a desired conversational outcome, such as in-process, success, or failure of persuading a user to donate to a charity

2.7 Discussion

Accessibility of Action-State Transition In environments under deterministic transitions (e.g., dialog, code generation), the next state s' can be directly derived based on the selected action. In a dynamic environment, s' can be either sampled over the probability distribution or generated from $\mathrm{lmpr}_{\text{transition}}$. Section 4 will demonstrate how this property affects search procedures.

Overhead of Using MDP Definition Although the comprehensive definition provides a unified interface to discuss LIS frameworks, it increases the overhead when applied to graph traversal tasks, since several defining characteristics of an MDP are not necessary, e.g., state transitions and explicit definitions of actions.

Why Is There No Previous Work Defining Reasoning Tasks as MDPs? Typical MDPs are often defined for decision-making models which can only handle the tasks whose action space is constrained and finite. However, general-purpose models like LLMs naturally deal with infinite or/and hard-to-define action space, since LLMs can infer plausible actions with world knowledge and commonsense.

Do Tasks Enable Action Undoing and State Back-Up? Some environments allow going back up to an earlier step after executing a sequence of actions (e.g., reasoning tasks), while other tasks may not (such as robotic tasks). This property is particularly important to discuss the applicability of LLM-integrated search methods. For environments under such property, the LIS agent can feel free to simulate future states for planning without worrying that the change of environments is irreversible. Details will be discussed in § 5).

3 LLM-Profiled Roles

Following standard reinforcement learning terminology (Sutton & Barto, 2018), an agent designed to solve Markov Decision Processes (MDPs) typically incorporates the following components:

- **Policy** $\pi(a_t \mid g, s_t)$: Determines the action a_t to take given the current state s_t.

- **Value Function** $V^\pi(s) \mapsto R$: Estimates the expected return of state s under policy π.

- **Transition Model** $\mathcal{T}(s_{t+1} \mid s_t, a_t)$: Represents the dynamics of the environment, predicting the next state s_{t+1} given the current state s_t and action a_t.

These definitions are broadly applicable across different agent designs. In this work, we adapt them to LLM-based search and focus on how to profile LLMs to work as/for these agentic components.

Background of LLM-Profiled Policy, Evaluator and Transition Model This section outlines the implementation of the three core components using three types of LMPRs. These roles are defined by Li (2024) as the LLM-profiled policy ($\text{lmpr}_{\text{policy}}$), evaluator ($\text{lmpr}_{\text{eval}}$), and transition model ($\text{lmpr}_{\text{transition}}$). For brevity, these notations are commonly adopted throughout this work.

While prior studies such as Spiegel et al. (2024) and Feng et al. (2024) explored these LMPRs primarily in theoretical contexts and toy environments for reinforcement learning, this section extends these ideas by presenting detailed implementations in real-world tasks.

Presentation of Prompting Examples To illustrate how LLMs are configured for different LMPR roles, we provide prompting examples throughout the paper. Model outputs are visually distinguished using shadow boxes for clarity. For example:

> An output from LMPR

To maintain brevity, placeholders enclosed in angle brackets (e.g., `<demos>` for few-shot demonstrations and `<task desc.>` for task descriptions) are used to represent verbal components within prompts.

Table 3: LLM-profiled policy. Note that what we really require is only a_t. N: sample size; T: length of action sequence.

	Outputs	Example Works
$\text{lmpr}_{\text{naive\_policy}}$	a_t	Xie et al. (2023)
$\text{lmpr}_{\text{reasoning\_policy}}$	Reasoning, a_t	ReAct (Yao et al., 2023b)
$\text{lmpr}_{\text{batch\_policy}}$	$a_t^1, a_t^2, \ldots, a_t^N$	Yao et al. (2023a), Jacob et al. (2024)

3.1 LLM-Profiled Policy (LMPP)

$\text{lmpr}_{\text{naive\_policy}}$ Given the observation o_t, $\text{lmpr}_{\text{naive\_policy}}$ directly generates the next action a_t.

$\text{lmpr}_{\text{reasoning\_policy}}$ To generate a_t, this policy first produces a complete reasoning path that explains or justifies the generation of a_t. The reasoning path serves as an explicit intermediate step, enhancing interpretability and illuminating the decision-making process for a_t.

$\text{lmpr}_{\text{react\_policy}}$ In contrast to $\text{lmpr}_{\text{reasoning\_policy}}$, $\text{lmpr}_{\text{react\_policy}}$ separates the reasoning step and the action-generation step into distinct inference passes. Each pass corresponds to an uninterrupted generation session. The reasoning text may include a planning path (e.g., $\tilde{a}_{t+1}, \ldots, \tilde{a}_T$), but only $\tilde{a}_t$ is used for search. Another distinguishing feature is the more autonomous behavior of this policy, which does not strictly adhere to a fixed reasoning-then-acting sequence. Instead, it can dynamically alternate between reasoning and acting steps, such as reasoning-acting-acting. For example:

Your task is to: put a cool tomato in microwave.
>

> think: To solve the task, I need to find a tomato, then cool it with the fridge, and finally put it in the microwave. *<more thoughts>*

OK.

> > go to countertop 1

<observation>

> > go to countertop 2

Prompting Example 1

The term "react" is attributed to the work of ReAct (Yao et al., 2023b). However, in their formulation, each thought is not explicitly treated as an action; instead, only tool invocations are considered actions in reasoning tasks. This distinction highlights the broader applicability of $\text{lmpr}_{\text{react\_policy}}$ in our definition.

$\text{lmpr}_{\text{batch\_policy}}$ This policy generates a batch of actions simultaneously, in contrast to other $\text{lmpr}_{\text{policy}}$ approaches, which generate actions step-by-step. $\text{lmpr}_{\text{batch\_policy}}$ can sample all actions in one pass.

1. $\text{lmpr}_{\text{batch\_policy1}}$: In this variant, one inference generates multiple candidate actions in text form, with candidates separated by a special token for subsequent extraction. As noted by Yao et al. (2023a), this avoids duplication in constrained action spaces, such as selecting a word in a crossword puzzle. By leveraging a global view of the action space, $\text{lmpr}_{\text{batch\_policy}}$ improves efficiency and coherence in tasks.

2. $\text{lmpr}_{\text{batch\_policy2}}$: This variant leverages the output logits of the LLM over the vocabulary to compute a probability distribution across a set of candidates. Actions are then sampled directly from this distribution.

3.2 LLM-Profiled Transition Model (LMPT)

Transition models are especially beneficial in dynamic environments, while in deterministic settings, where transitions are predictable or actions easily reversible, their utility is limited. $\text{lmpr}_{\text{transition}}$ predicts outcomes according to LLMs' internal knowledge. The profiling can be categorized as generating: **1) Full state**: The final goal is to return a full state/observation at the current step, as exemplified in Example 2.

<profile information>
[STATE 0] I have that, the white block is clear, the cyan block is clear, <more detail>
[ACTION] Pick up the brown block.
[CHANGE]

> The hand was empty and is now holding the brown block, the brown block was on the table and is now in the hand, and the brown block is no longer clear. [STATE 1] I have that, the white block is clear, the cyan block is clear, <more detail>

Prompting Example 2

2) Partial observation: The partial observation would be further processed to form the full state. One obvious task is reasoning via QAs.

Given a question, please decompose it into sub-questions. For each sub-question, please answer it in a complete sentence, ending with "The answer is". When the original question is answerable, please start the subquestion with "Now we can answer the question:
<few shot demos>
Question 1: James writes a 3-page letter to 2 different friends twice a week. How many pages does he write a year?
Question 1.1: How many pages does he write every week?

> *Answer 1.1:* James writes a 3-page letter to 2 different friends twice a week, so he writes 3 * 2 * 2 = 12 pages every week. The answer is 12.

Prompting Example 3

Table 4: LLM-profiled evaluators $\text{lmpr}_{\text{eval}}$. The columns of "$V$?" and "$Q$?" indicate whether the LMPR configuration can work as state value function and action value function, respectively.

	Prompting Tasks	Outputs	V?	Q?	Example Works
$\text{lmpr}_{\text{eval1}}$	Binary/Multi-class Classification	Discrete values mapped by $\text{lmpr}_{\text{eval}}$ generations	✓	✓	RAP (Hao et al., 2023), ToT (Yao et al., 2023a), Koh et al. (2024), LATS (Zhou et al., 2024a)
$\text{lmpr}_{\text{eval2}}$	Binary/multi-class Classification	Logits of $\text{lmpr}_{\text{eval}}$ generations	✓	✓	RAP (Hao et al., 2023), Tree-BeamSearch (Xie et al., 2023)
$\text{lmpr}_{\text{eval3}}$	Multi-choice QA	Choices of top-N actions	✗	✓	ToT (Yao et al., 2023a), Think-on-Graph (Sun et al., 2024)
$\text{lmpr}_{\text{eval4}}$	Classification (Implicit)	Logits of given continuation	✗	✓	RAP (Hao et al., 2023)
$\text{lmpr}_{\text{eval5}}$	Multi-choice QA	Logits of given continuation (choices)	✓	✗	Kadavath et al. (2022), Xie et al. (2023)
$\text{lmpr}_{\text{eval6}}$	Scoring	Generated continuous scores	✓	✗	Think-on-Graph (Sun et al., 2024)
$\text{lmpr}_{\text{policy\&eval1}}$	NA	Logits of $\text{lmpr}_{\text{policy}}$ generations	✗	✓	RAP (Hao et al., 2023), Tree-BeamSearch (Xie et al., 2023)
$\text{lmpr}_{\text{policy\&eval2}}$	NA	Self-consistency scores of $\text{lmpr}_{\text{policy}}$ generations	✗	✓	LATS (Zhou et al., 2024a), rStar (Qi et al., 2025), ToolChain* (Zhuang et al., 2024),
$\text{lmpr}_{\text{policy\&eval3}}$	NA	Consistency of $\text{lmpr}_{\text{policy}}$-generated steps and a candidate plan	✓	✗	LLM-A* (Meng et al., 2024)
$\text{lmpr}_{\text{policy\&eval4}}$	NA	Q-Values based on a stroing $\text{lmpr}_{\text{policy}}$	✗	✓	Q* (Wang et al., 2024a)
$\text{lmpr}_{\text{transition\&eval1}}$	NA	Logits of $\text{lmpr}_{\text{transition}}$ generations	✓	✗	RAP (Hao et al., 2023)
$\text{lmpr}_{\text{transition\&eval2}}$	NA	Logits of $\text{lmpr}_{\text{transition}}$ generations	✓	✗	/

3.3 LLM-Profiled Evaluator (LMPE)

LLMs can serve as flexible evaluators (*LMPEs*) by leveraging their generative and probabilistic capabilities. We propose categorizing these evaluators along three key dimensions:

- **Task Formulation**: Whether the evaluation is a binary classification, multi-choice QA, or a free-form judgment influences how the LLM's output or logits can be interpreted. These tasks are always formulated in LLMs' system-level prompts.
- **State vs. Action Evaluation**: This is analogous to state/action-state value functions in reinforcement learning. Depending on whether the evaluator is assessing a static state s_t or a transition (s_t, a_t), the LLM must parse different context inputs to provide a valid judgment.
- **Output Types**: Unlike $\text{lmpr}_{\text{policy}}$ or $\text{lmpr}_{\text{transition}}$, the output of an evaluator $\text{lmpr}_{\text{eval}}$ can produce not only text-based outputs (continuous text or discrete labels) but also raw logits and self-

consistency scores. This flexibility allows for nuanced scoring and confidence measurement that can be mapped to discrete classes or continuous values.

- **Use of Reasoning**: As LMPPs, some reasoning techniques can be generalized to LMPEs.

As summarized in Table 4, various works have shown that LLMs configured under these dimensions can support diverse evaluation goals. The paragraphs below illustrate the four dimensions through concrete prompts and output examples.

Task Formulations Three types are commonly used for evaluation. **1) Binary/Multi-class Classification**: As shown below, a prompt can explicitly request a binary judgment, e.g., yes/no or failure/success:

<prompt>

> Status: "failure"

Prompting Example 4

In some cases, more fine-grained judgments are required with multiple labels. For example, the "status" tag is defined for LMPE to indicate partial successes in Koh et al. (2024):

<prompt>

> Status: "failure"
> On the right track to success: "yes"

Prompting Example 5

This can be considered as multi-class classification, where "yes" yields an intermediate class. A more direct multi-class classification is specified in the example below, where the prompt assesses whether a given set of numbers can reach 24:

Evaluate if given numbers can reach 24 (sure/likely/impossible)
10 14
10 + 14 = 24
sure

1 3 3

> 1 * 3 * 3 = 9
> (1 + 3) * 3 = 12
> 1 3 3 are all too small
> impossible

Prompting Example 6

2) Multi-choice QA: This is often advantageous when directly scoring an action/state is difficult to compute in contrast to comparing multiple candidates. For example, it is difficult to judge whether a give passage is coherent, while it is easy to judge whether Passage A is more or less coherent than Passage B. Another way is to implicitly compare different solutions via voting through self-consistency scores of $\mathrm{lmpr}_{\mathrm{policy}}$, which belongs to the next formulation type. **3) Scoring**: A continuous score is given for each candidate option. Example 7 demonstrates how to prompt LLMs for scoring in the task of reasoning over a knowledge graph.

Please rate thee contribution of the relations on a scale from 0 to 1 (the sum of the scores of the relations is 1)
Q: <Query>
Topic Entity: <Topic Entity>
Relations: <list of relations>

Prompting Example 7

4) No Explicit Evaluator Definition: Evaluation can be inferred from the $\text{lmpr}_{\text{policy}}$'s generative process itself. In such cases, no separate system-level prompt is required for task formulation. Likewise, $\text{lmpr}_{\text{transition}}$ can be used for evaluation. This LLM-based evaluation will be detailed from the perspective of output types.

State vs. State-Action Function **1) State-Value Evaluator**: A state-based evaluator accepts s_t as its input to produce a judgment:

$$\text{discrete judge} = \text{lmpr}_{\text{eval}}(s_t) \tag{4}$$

Example 6 is one of the example. **2) State-Action Evaluator**: the evaluator assesses whether taking action a_t is appropriate at the current state s_t:

$$\text{discrete judge} = \text{lmpr}_{\text{eval}}(s_t, a_t) \tag{5}$$

This setup is exemplified in Example 10, where the new sub-question (a_t)'s usefulness depends on the prior state (s_t). Another example of BlocksWorld from Hao et al. (2023):

[STATE]
As initial conditions I have that, the blue block is clear, the orange block is in the hand, the red block is clear, the hand is holding the orange block, the red block is on top of the yellow block, the blue block is on the table, and the yellow block is on the table. My goal is to have have that the red block is on top of the yellow block and the orange block is on top of the blue block.
[ACTION]
stack the orange block on top of the red block
[EVALUATION]

bad

Prompting Example 8

Outputs Finally, the outputs of $\text{lmpr}_{\text{eval}}$ can take one of several forms, depending on how we wish to interpret or utilize the evaluator's opinion:

1. **Mapping $\text{lmpr}_{\text{eval}}$ generation to discrete values**: For instance, "impossible" or "Yes" may be mapped to numeric scores (0 or 1) in Quotes 6 and 10.
2. **Using logits of $\text{lmpr}_{\text{eval}}$**: The probability of generating a specific token (e.g., "impossible") can serve as the confidence score.
3. **Using logits of given continuations**: Rather than having the LLM generate the evaluation tokens, one can provide the exact sequence to be evaluated (e.g., "good" in Quote 8 or "(A) good"). The log probability of each token is then summed to indicate how well (i.e., how confidently) the model "accepts" that evaluation in the given context. A higher cumulative log-likelihood suggests that the LLM finds the provided evaluation more plausible. Moreover, multiple predefined options (e.g., "(A) Correct" vs. "(B) Incorrect") can be separately fed in as continuations. The log probabilities of each option can then be compared as self-evaluation scores (Xie et al., 2023; Kadavath et al., 2022).
4. **Using logits of $\text{lmpr}_{\text{policy}}$ or $\text{lmpr}_{\text{transition}}$**: Alternatively, the evaluation can be derived from the policy or transition model's token probabilities. This method can avoid additional inference steps

by reusing existing logits. However, the fundamental flaw is that when multiple plausible answers exist, individual logits can be low even if overall confidence in the correct answer set is high (Lin et al., 2022).

5. **Using self-consistency scores**: By sampling multiple trajectories from $\text{lmpr}_{\text{policy}}$ or $\text{lmpr}_{\text{transition}}$, one can gauge confidence via how often a particular outcome (state or action) appears. More frequent outcomes can be assumed more likely (or better). One challenge is how to distinguish different outcomes. For example, Zhuang et al. (2024) fine-tune a natural language inference (NLI) model to distinguish the generated actions.

6. **Comparing consistency of $\text{lmpr}_{\text{policy}}$-generated steps and candidate states**: The actions/plan generated by actions can be compared to the candidate for evaluation. This is suitable for tasks with limited successor states.

7. **Comparing Consistency of $\text{lmpr}_{\text{policy}}$-Generated Steps and Candidate States**: The actions or plans produced by the policy can be compared against candidate states to assess consistency. This approach is particularly suitable for tasks with a limited number of successor states, e.g., sovling a maze (Meng et al., 2024).

8. **Using a Stronger $\text{lmpr}_{\text{policy}}$ as a Proxy Optimal Policy to Approximate Q-Values**: When rewards are only obtained at the terminal state, the Q-value can be approximated by discounting the rewards along the path. This approach assumes that all subsequent actions are optimal, which necessitates the use of a more robust $\text{lmpr}_{\text{policy}}$ as a proxy for the optimal policy.

Use of Reasoning Similar to $\text{lmpr}_{\text{reasoning\_policy}}$, a reasoning process can be required before generating the final judgment. This reasoning process provides a logical justification to augment the evaluation. For example:

<prompt>

> Thoughts: <your thoughts and reasoning process>
> Status: "failure"

Prompting Example 9

3.4 Discussion

Inference Cost of $\text{lmpr}_{\text{policy}}$ In practice, the overall computational cost follows the pattern

$$\text{lmpr}_{\text{react\_policy}} > \text{lmpr}_{\text{reasoning\_policy}} > \text{lmpr}_{\text{naive\_policy}}.$$

The gap between $\text{lmpr}_{\text{reasoning\_policy}}$ and $\text{lmpr}_{\text{naive\_policy}}$ arises from the additional output tokens produced for reasoning. More importantly, when commercial API is used, $\text{lmpr}_{\text{react\_policy}}$ exhibits an even higher cost because each *separate* reasoning or action-generation pass is effectively stateless with respect to the cached K–V pairs from previous passes, thereby preventing token-reuse optimizations.

Applicability of $\text{lmpr}_{\text{react\_policy}}$. A central requirement for ReAct-style prompting ($\text{lmpr}_{\text{react\_policy}}$) is the availability of step-wise observations after each action. This imposes two prevalent scenarios:

1. **Tasks relying on simulators**: When direct interaction with the real environment is impractical (e.g., actions on tasks are irreversible), a simulator can be substituted to generate the observation following each action. For instance, an LLM-based simulator ($\text{lmpr}_{\text{transition}}$) might use commonsense knowledge to model environmental responses (e.g., turning on a water tap in a sealed sink). However, such simulators are unsuitable for tasks involving external or private data—like querying proprietary databases or retrieving up-to-date information—since an LLM's internal knowledge typically cannot replicate these data sources.

2. **Action-reversible tasks.** Certain problems can be retried or backtracked, allowing the agent to iteratively act, observe, and refine its actions, as discussed in Section 2. In Section 5, for example, LLM-based search frameworks such as LATS (Zhou et al., 2024a) leverage this property when interacting with real environments across multiple search steps to perform monte-carlo simulation.

Risk of $\mathrm{lmpr}_{\mathrm{value}}$ Although $\mathrm{lmpr}_{\mathrm{value}}$ can effectively evaluate state or action quality, two challenges stand out:

1. **Mediocre discrimination abilities**: As shown by Chen et al. (2024b), using logits as dense rewards (e.g., in $\mathrm{lmpr}_{\mathrm{policy\&eval}}$ or $\mathrm{lmpr}_{\mathrm{transition\&eval}}$) can reveal that many open-source LLMs struggle to reliably distinguish "good" from "bad" examples.[1]

2. **In-Context Reward Hacking (ICRH)**: According to Pan et al. (2024), an LLM evaluator ($\mathrm{lmpr}_{\mathrm{value}}$) may attempt to "explain away" negative feedback by globally altering its reasoning and actions, potentially violating constraints. For example, to fix an InsufficientBalanceError, the LLM might suggest unauthorized money transfers from other accounts, thus compromising safety or policy compliance.

Not All Generation with "Reasoning" is Truly Augmented. By design, LLMs generate tokens in an auto-regressive manner, meaning earlier tokens are not influenced by later ungenerated tokens. Hence, although reasoning tokens after actions (or evaluation) can make the model outputs more interpretable, they do not always alter subsequent decisions or evaluations. In Xie et al. (2023), for instance, a chain of thoughts

$$a_t,\ \tilde{a}_{t+1},\ \ldots,\ \tilde{a}_T$$

is produced, where $\tilde{a}_{t+1}, \ldots, \tilde{a}_T$ are "unrecorded" actions. Crucially, a_t is unaffected by any future $\tilde{a}$ tokens, making this effectively a naive policy rather than a true reasoning-augmented approach.

Similarly, consider the evaluator in Example 10:

> Given a question and some sub-questions, determine whether the last sub-question is useful to answer the question. Output 'Yes' or 'No', and a reason.
> **Question 1:** Four years ago, Kody was only half as old as Mohamed. If Mohamed is currently twice as 30 years old, how old is Kody?
> **Question 1.1:** How old is Mohamed?
> **Question 1.2:** How old was Mohamed four years ago?
> **New question 1.3:** How old was Kody four years ago?
> Is the new question useful?
>
> Yes. We need the answer to calculate how old is Kody now.
>
> *Prompting Example 10*

Although the model's output includes a short "reason," that intermediate reasoning does not necessarily *inform* the generation of 'Yes' or 'No'.

4 Search Procedures

This section presents the reusable search procedures applied across various frameworks, including both non-LMPR-specific and LMPR-based procedures. Unlike Section 3, which focused on configuring LMPRs, here we demonstrate how these LMPRs are integrated into the operational processes. However, some content may overlap slightly for coherence. Table 6 summarizes the dependencies between these search procedures and the LMPR components.

[1] GPT-4 turbo was the most advanced model at the time of evaluation.

Table 5: Examples of combining LLM-Profiled Evaluators with heuristics. $\|s = s_g\|_1$ indicate whether the agent reaches the goal state at s.

Task	Value
Reasoning-QA in RAP (Hao et al., 2023)	$\text{lmpr}_{\text{eval1}} + \text{lmpr}_{\text{policy\&eval1}}$
Game (Blockworld) in RAP (Hao et al., 2023)	$\text{lmpr}_{\text{eval4}} + \text{lmpr}_{\text{eval2}} + \|s = s_g\|_1$ (weights ignored)
Graph traversal in LLM-A* (Meng et al., 2024)	Euclidean distance to the next state + glm_{eval}
Reasoning in Xie et al. (2023)	$\text{lmpr}_{\text{value5}} + \text{lmpr}_{\text{policy\&eval1}}$
Reasoning-QA/Code Gen/Web Nav. in LATS (Zhou et al., 2024a)	$\text{lmpr}_{\text{value1}} + \text{lmpr}_{\text{policy\&eval2}}$

Table 6: Overview of dependencies in search procedures. Sampl.: Sampling; Exp.: Expansion; Eval.: Evaluation; Sel.: Selection; Sim.: Simulation; Backprop.: Backpropagation.

(a) Dependency of first-order procedures on LMPRs.

	LMPP	LMPE	LMPT
LMPP Sampl.	✓	✗	✗
LMPE+ Eval.	✗	✓	✗
LMPT Sim.	✗	✗	✓
Multi-Choice LMPE Sel.	✗	✓	✗
Single-Step UCT Sel.	✗	✗	✗
Exhaustive Action Retrieval	✗	✗	✗

(b) Dependency of higher-order procedures on LMPR-based, first-order procedures.

	LMPP Sampl.	LMPE+ Eval.	LMPT Sim.
Value-Based Sel.	✗	maybe	✗
LMPP Exp.	✓	✗	maybe
Path Sim.	✓	maybe	maybe
MCTS Sel.	✗	✗	✗
MCTS Backprop.	✗	✗	✗

Search Nodes: Integrating States, Action, and Rewards In this section, we shift our focus to search and clarify how the fundamental search "node" is defined with respect to states and actions. Some methods (e.g., ToT (Yao et al., 2023a)) treat a node as a particular state in a search tree, with transitions determined by the actions taken. To ensure generality, we unify states, actions, and even their estimated values (or rewards) in a single node structure (e.g., RAP (Hao et al., 2023)), facilitating partial expansions or multi-step lookahead. To align with object-oriented design, we represent a node as n with attributes n.action, n.state, n.parent, and n.val , representing the action, state, parent node, and value, respectively.

4.1 First-Order Procedures

First-order procedures operate independently, without relying on other procedures. They serve as the foundational components upon which more complex procedures are built, ensuring a modular and scalable framework for LLM-based search operations. The first three are based on LLM-Profiled Policy (LMPP), evaluator (LMPE), and transition model (LMPT), respectively, while others not necessarily depend on LMPRs.

LMPP Sampling The sampling procedure involves generating multiple actions (assuming N actions) for a given state s_t. Generally, there are two approaches to sampling actions: one based on generating actions sequentially using the single-action policy ($\text{lmpr}_{\text{policy}}$), and another based on generating all actions simultaneously using the batch policy ($\text{lmpr}_{\text{batch\_policy}}$). These approaches are detailed in Procedures 1 and 2, respectively.

Procedure 1 LMPP Sampling: Sample Actions One at a Time

procedure SAMPLE_LMPP_ONE_AT_A_PASS(s_t, N)
  $\mathbf{a}_t \leftarrow \{\}$
  for $i \in \{1, \ldots, N\}$ **do**
    $a_i \sim \text{lmpr}_{\text{policy}}(a \mid s_t)$
    $\mathbf{a}_t \leftarrow \mathbf{a}_t \cup \{a_i\}$
  end for
  return $\mathbf{a}_t$
end procedure

Procedure 2 LMPP Sampling: Sample All Actions at Once

procedure SAMPLE_LMPP_ALL_AT_ONCE(s_t, N)
  $\mathbf{a}_t \sim \text{lmpr}_{\text{batch\_policy}}(a \mid s_t, N)$
  return $\mathbf{a}_t$
end procedure

LMPE+ Evaluation LMPEs can be used to estimate the value (or reward) of a state. The evaluator's output—whether in textual or numerical form—can then be combined with rule-based heuristics to refine the overall assessment. For instance, Table 5 illustrates how numeric outputs from LMPEs are incorporated into a heuristic that balances both LLM-based scoring and domain-specific constraints. Such integrated approaches are particularly relevant when neither pure heuristic nor pure LLM-based evaluation alone is sufficient for robust decision-making. Based on the input, the procedure can be defined as V_EVAL for taking a state as input and Q_EVAL for taking both a state and an action as input.

Multi-Choice LMPE Selection $\text{lmpr}_{\text{eval5}}$ (an LLM profiled for multiple-choice tasks) can be leveraged for top-k selection to directly select K nodes.

Procedure 3 TopK Selection via $\text{lmpr}_{\text{eval5}}$

procedure TOPK_SELECT_LMPE(N, k)
  $A \leftarrow \{n.\text{action} \mid n \in N\}$ ▷ Extract actions from each node in N
  $A^* \leftarrow \text{lmpr}_{\text{eval5}}(A, k)$ ▷ Select the top k actions from A
  $N^* \leftarrow \{n \in N \mid n.\text{action} \in A^*\}$ ▷ Return nodes corresponding to the selected actions
  return N^*
end procedure

Single-Step UCT Selection The objective of UCT selection is to choose an action that balances exploitation and exploration. This is captured by the Upper Confidence Tree (UCT) formula:

$$\text{UCT}(s,a) = Q(s,a) + w\sqrt{\frac{\ln N(s)}{N(c(s,a))}}, \tag{6}$$

where c is a constant controlling exploration, $N(s)$ is the number of times state s has been visited, and $N(s,a)$ is the number of times action a has been selected under s. A related variant, Predictor Upper Confidence Tree (PUCT), incorporates the prior probability $P(s,a)$ into the exploration term to further guide the action selection.

$$\text{PUCT}(s,a) = Q(s,a) + \text{c}P(s,a)\frac{\sqrt{N(s)}}{1+N(s,a)} \tag{7}$$

$P(s, a)$ is normally a domain-specific predictor. Based on the two estimates, the procedure is just to iterate over the given actions A, along with their values Q, and extract the one with the highest value, as summarized in Procedure 4.

Procedure 4 Single-Step UCT Selection

procedure UCT_SELECT(s, A)
  $a^* = \arg\max_{a \in A} [\text{UCT}(s, a)]$
end procedure
procedure PUCT_SELECT(s, A)
  $a^* = \arg\max_{a \in A} [\text{PUCT}(s, a)]$
end procedure

LMPT Simulation This procedure is straightforward: given the current state s_t and an action a_t, the LMPT directly outputs the next state s_{t+1}. Unlike LMPP sampling (which may loop through multiple actions) or LMPE+ Evaluation (which may incorporate additional heuristics), no further processing or components are involved.

Exhaustive Action Retrieval When the action space is small and well-defined (e.g., in BlockWorld), all possible actions can be retrieved exhaustively. This procedure is primarily used to facilitate the subsequent expansion or simulation steps.

4.2 Higher-Order Procedures

Value-Based Selection The first type is top-k selection. The top k states or actions are picked from a large pool of candidates based on their estimated values. A state-value function $V(s')$ or an action-value function $Q(s, a)$ is used to generate values. Commonly, they are implemented by LMPE+ evaluation. The detail is illustrated in Procedures 5. Note that the **if** statement for value assignment also allows specialized ways to assign values without necessarily relying on n'.state.

Procedure 5 Value-Based TopK Selection

procedure TOPK_SELECT(N, k)
  for each node $n' \in N'$ **do**
    if n'.val **is uninitialized then**
      $n'.\text{val} \leftarrow \text{ValueFunc}(n'.\text{state})$
    end if
  end for
  $N^* \leftarrow \arg\text{top}_k \{n'.\text{val} \mid n' \in N'\}$
  return N^*
end procedure

Once $k = 1$, the $\arg\text{top}_k \{n'.\text{val} \mid n' \in N'\}$ reduces to $\arg\max_{n' \in N'} n'.\text{val}$.

Another type is threshold-based selection. Here, the procedure repeatedly samples one action from the current node's state, simulates the new state, and evaluates its value. If the value surpasses a threshold θ, the procedure returns the newly created node; otherwise, it continues sampling.

LMPP Expansion This expansion procedure adds one or more child nodes under the given leaf node(s), typically visualized at the next depth level in a tree search. The procedure is based on the following components:

1. **LMPP Sampling**: This step provides the n.action attribute for each node, while leaving n.state and n.val uninitialized.

Procedure 6 Value-Based Threshold Selection

```
procedure ThresholdSelect(n, θ)
    while true do
        a ← Sample_Action(n.state)
        s′ ← Simulate(n.state, a)
        v ← ValueFunc(s′)
        if v ≥ θ then                      ▷ Value exceeds threshold; accept this node
            Instantiate a new node n′
            n′.state ← s′
            n′.action ← a
            n′.val ← v
            return n′
        end if                             ▷ Otherwise, continue sampling another action
    end while
end procedure
```

2. **Simulating States Based on Sampled Actions**: To enable further expansion, the node's state (n.state) must be generated. This can be accomplished via simulators (e.g., using LMPT simulation), direct action execution, or even hard-coded methods (e.g., simply concatenating actions).
3. **(Optional) Expanding Multiple Nodes Simultaneously**: In many search strategies (such as beam search with a beam size greater than 1 or breadth-first search), multiple leaf nodes are expanded concurrently.

A general form of the expansion procedure is specified in Procedure 7.

Procedure 7 Expansion

```
procedure Expand_LMPP(N_t)
    N_{t+1} ← {}
    for each node n_t ∈ N_t do
        A_t ← Sample_LMPP(n_t.state, k)
        for each action a_t ∈ A_t do
            Instantiate a new node n_{t+1} with n_{t+1}.action ← a_t
            n_{t+1}.state ← Simulate(n_t.state, a_t)
            N_{t+1} ← N_{t+1} ∪ n
        end for
    end for
    return N_{t+1}
end procedure
```

Here, Sample_LMPP refers to Procedure 1 or Procedure 2. The total number of N_{t+1} equals the product of the number of nodes in N_t and the number of actions k sampled per node.

Path Simulation Procedure 8 demonstrates the overall process for rolling out a path until either a goal state or a terminal state is reached. In this procedure, the following steps are executed sequentially and iteratively:

1. **Sampling**: At each step, actions are sampled from the current state using either LMPP sampling or a random sampling method.
2. **Value-Based Selection**: TopK_Select is applied with only the highest-valued node preserved (i.e., $k = 1$), since only a single path is rolled out. The evaluation can be performed using LMPE+ evaluation or a heuristic-only approach.

   3) Single-Step Simulation: The next state is generated either by either using LMPT simulation, employing domain-specific simulators, or directly executing the selected action in the environment to obtain the subsequent state (or observation).

Procedure 8 Path Simulation

```
procedure SIMULATE(n_t)
    path ← {}
    while n_t.state is not terminal do
        A ← SAMPLE_ACTION(n_t.state)
        a* ← Q_SELECT(n_t.state, A', k = 1)
        s ← SIMULATE(s, a*)
        path ← path ∪ s
    end while
    return path
end procedure
```

MCTS Selection However, during MCTS for planning, before going to the expansion phase, Procedure 4 (One-step UCT Selection) should be used multiple times to traverse from s_0 to a leaf node s_{leaf}.

MCTS Backpropagation - Value Update After each simulation returns a reward r, update the Q value as:

$$Q_{\text{new}} = \frac{r + Q_{\text{old}} \cdot \text{Count}_{\text{new}}}{\text{Count}_{\text{new}}}, \tag{8}$$

- r, depending on the task, can be a reward at the terminal state. In some cases,it can be an aggregated one, if each simulation step yields a reward. Specifically, if rewards r_t are discounted by γ, then the final sample reward r for backpropagation is:

  $$r = G = \sum_{t=0}^{T-1} \gamma^t r_t.$$

  In many implementations (e.g., RAP (Hao et al., 2023)), the step-wise rewards are obtained via LMPE+ or a heuristic evaluation during path simulation. These rewards from simulated nodes are then employed to update the Q values for non-simulated nodes, including the leaf nodes and those above.

- Q_{old} are the previous Q-value.

- $\text{Count}_{\text{new}}$ is the total visit count after the current update.

During some implementations (Hao et al., 2023), each reward $r \in R_{\text{cum}}$ propagating from the terminal node can be stored in a list R_{cum} , and average them when used. The procedure of value estimate, as summarized in Table 5, provides the initialized values for new actions or resulting states.

MCTS Backpropagation - Visit Update Except for the estimated value $Q(s, a)$, the backpropagation also updates the visit count for every state on the path from the root to the leaf and each edge (s, a) along that path, denoted as $\text{Count}\,(s_t)$ and $\text{Count}\,(s_t, a_{t+1})$, respectively. The increase in $\text{Count}(s, a)$ (the action-level visit count) reduces the exploration bonus for a in future selections, thus making ($\mathbf{s}, \mathbf{a}$) slightly less likely to be chosen purely for exploration next time, assuming the same or lower estimated value. Assuming $\text{Count}\,(s_t, a_{t+1}) -> \text{Count}\,(s_{t+1})$ in some deterministic environments, only $\text{Count}\,(s_{t+1})$ is tracked. For the same reason, Q values can be attached as an attribute of the state node. An example is the implementation of RAP (Hao et al., 2023) [2].

[2] https://github.com/maitrix-org/llm-reasoners/blob/main/reasoners/algorithm/mcts.py

[1] multimodal

[2] fine-tuning

Table 7: Search-based frameworks for LLM inference. We use **"*"** to indicate workshop publication. **EAR** means Exhaustive Action Retrieval. **Sel. for Expansion** indicates selection methods used to select from multiple candidate actions/nodes, which can be **UCT** or **PUCT**, **Multi-choice LMPE**, Value-based TopK (**V-TopK**), and **threshold** selection. Many MCTS-specific procedures are excluded, except for simulation (**Sim.**) to distinguish **LMPT**, **Simulator** and environment (**Env.**) simulation.

	Resemble	LMPP Exp.	LMPE+ Eval.	Sel. for Expansion	Sim.	EAR	Published Date
PG-TD (Zhang et al., 2023)	MCTS	✓	✓	PUCT	LMPT	✗	Mar 2023 (ICLR2023)
ToT (Yao et al., 2023a)	BFS (B for Breath); DFS	✓	✓	V-TopK for BFS; Threshold for DFS	✗	✗	May 2023 (NIPS2023)
Xie et al. (2023)	Beam Search	✓	✓	V-TopK	✗	✗	May 2023 (NIPS2023)
RAP (Hao et al., 2023)	MCTS	✓	✓	UCT	LMPT	✓	May 2023 (EMNLP2023)
GDP-ZERO (Yu et al., 2023)	MCTS	✓	✓	PUCT	LMPT	✓	Oct 2023 (EMNLP2023)
LATS (Zhou et al., 2024a)	MCTS	✓	✓	UCT	Env.	✓	Oct 2023 (ICML2024)
LLM-MCTS (Zhao et al., 2023)	MCTS	✓	✓	PUCT	Simulator	✓	May 2023 (NIPS2023)
LLM-A* (Meng et al., 2024)	A*	✗	✓	V-TopK	✗	✓	Jun 2024 (EMNLP2024)
Q* (Wang et al., 2024a)	A*	✓	optional	V-TopK	✗	✗	Jun 2024
Koh et al. (2024)	BFS (B for Best) [1]	✓	✓	V-TopK	Env.	✗	Jul 2024
PathFinder (Golovneva et al., 2023)	Beam Search	✓	✗	V-TopK	✗	✗	Dec 2023 (NeurIPS2023*)
rStar (Qi et al., 2025)	MCTS	✓	✓	UCT	LMPT	✗	Aug 2024 (ICLR2025)
Think-on-Graph 2.0 (Ma et al., 2025b)	Beam Search	✗	✓	Multi-Choice LMPE	✗	✓	Jul 2024 (ICLR2025)
Think-on-Graph (Sun et al., 2024)	Beam Search	✗	✓	Multi-Choice LMPE	✗	✓	Jul 2023 (ICLR2024)
ToolChain (Zhuang et al., 2024)	A*	✓	✓	V-TopK	✗	✗	Oct 2023 (ICLR2024)

5 Frameworks Based on Search Algorithms

This section summarizes how different frameworks utilize search algorithms, leveraging the LMPRs and search procedures introduced in Table 7. Note that, some MCTS-specific procedures (e.g., **MCTS selection**, and **MCTS backpropagation**) are not elaborated in the table. UCT selection is highlighted to distinguish between PUCT and UCT variants; **simulation** is included to clarify whether LMPT or environment-based simulation or simulator is used. Below, we discuss perspectives that are not fully captured in Table 7.

5.1 Beam Search

Xie et al. (2023) and PathFinder (Golovneva et al., 2023) adapt beam search for reasoning via concatenation, while Think-on-Graph (Sun et al., 2024) and Think-on-Graph 2.0 (Ma et al., 2025a) are applied for reasoning over knowledge graph. Generally, **Expansion** and **TopK-Based Selection** always alternate iteratively until reaching a terminal state.

Xie et al. (2023)

- **LMPP Expansion**: Each set of beam nodes N_t is passed to the LMPP expansion procedure. Internally, SAMPLE_LMPP calls either SAMPLE_ACTIONS_ONE_AT_A_PASS or SAMPLE_ACTIONS_BATCH, while SIMULATE can be a simple concatenation transition: each node $n_t \in N_t$ has its parent state n_t.parent expressed as a sequence of actions $(a_1, \ldots, a_{t-1})$, an action a_t assigned to n_t.action, and the new node's state (n_t.state) as $(a_1, \ldots, a_{t-1}, a_t)$. The resulting set of expanded nodes is denoted N_t^{sample}.

- **TopK-Based Selection**: From N_t^{sample}, a value-based selection procedure is applied to pick the top-k nodes (the beam size). This subset is returned as N_t. Specifically, it uses a value function implemented by $\text{lmpr}_{\text{value5}} + \text{lmpr}_{\text{policy\&eval1}}$.

PathFinder (Golovneva et al., 2023) This framework also applies **LMPP Expansion** and **Value-Based TopK Selection** iteratively. However, it computes a summed similarity score as the value for each candidate node, comparing its state with those of other beams N_t^{sample}. The similarity function can be as simple as n-gram overlap.

Think-on-Graph (Sun et al., 2024) In the graph traversal tasks described in Section 2, the search space is derived directly from explicit task specifications. In contrast, when searching over a knowledge graph, the search space is constructed from the graph itself.

- **Step 1: Entity Initialization**: An LLM is prompted twice to extract the topic entities in the question, and select the top-K entities (Procedure 3), where K is the beam size. The resulting entities $\mathcal{E}_0$ form the initial paths.

- **Step 2: Expansion via SPARQL Queries for Candidate Relations**: Unlike previous two frameworks, LMPP expansion and sampling are not necessary because neighboring nodes $N_t^{\text{candidate}}$ can be identified via simple SPARQL queries, along with candidate relations, i.e., Exhaustive Action Retrieval is applied. The relation set can be denoted as $\mathcal{R}_{\text{entity}}$, where topic $\in \mathcal{E}_{t-1} = \{n_{t-1}.\text{state.tail\_entity} | n \in N_{t-1}\}$. The queries are run for each tail entity $e \in \mathcal{E}_{t-1}$ to get candidate relations. The size of $\mathcal{E}_{t-1}$ equals to beam size B.

- **Step 3: TopK Selection from Candidate Relations**: Procedure 3 is applied to directly select the top K relations from candidates for each entity $e \in \mathcal{E}_{t-1}$. Note that both K here is not necessarily the beam size B or B divided by K, since the final path will be formed after the following entity expansion and selection phases.

- **Step 4: Expansion via SPARQL Queries for Candidate Entities**: Candidate entities are selected via SPARQL queries for each selected relations in the last step.

- **Step 5: TopK Selection from Candidate Entities**: To finish each triple tailed by the K relations selected above.$\text{lmpr}_{\text{eval6}}$ would be used to score each triple regarding their contributions to solve the given question. This defines ValueFunc in Procedure 5. Note that the k here is the beam size.

- **Step 6: LLM Reasoning for Terminal States**: At the end of each iteration, the LLM is prompted to judge whether a terminal state is reached, where the knowledge is enough to reach the final answer.

Think-on-Graph 2.0 (Ma et al., 2025a) Think-on-Graph 2.0 (Ma et al., 2025a) differs from Think-on-Graph in the following perspectives.

- **Entity Initialization**: The initial entities $\mathcal{E}_0$ are initialized and linked to the knowledge graph via an entity linking method.
- **Context Retrieval**: A dense retrieval model (DRM) is used to retrieve context with $\mathcal{E}_0$ as input.
- **LLM Reasoning for Terminal States**: Same as Step 6.
- **Relation Identification**: This is basically identical to Step 2-3 in Think-on-Graph. [3] One difference in Step 3 is that Procedure 3 and the LMPE in Step 3 will evaluate the candidate relations of all the entities $\mathcal{E}_{t-1}$ in one LLM inference.
- **Entity Identification**: This corresponds Step 4-5 in Think-on-Graph. Step 4 still locates potential entities on knowledge graph [4]. The main difference is that the the pruning process is based on a context retrieval process over unstructured documents.
- **LLM Reasoning for Terminal States**: Similar as Step 6. The only difference is that the retrieved context for each entity is added for prompting.

5.2 Breadth-First Search

Tree-of-Thoughts (ToT) (Yao et al., 2023a)

- Similar to beam search, breadth-first search (BFS) is performed by iteratively applying **LMPP Expansion** and **TopK Selection**.
- A key difference is that, in BFS, all nodes at depth t undergo the same number of actions before expanding further levels. This enforces uniform depths across expansions.

5.3 Depth-First Search

Tree-of-Thoughts (ToT) (Yao et al., 2023a) Yao et al. (2023a) also apply depth-first search (DFS) for LLM inference, relying on the **LMPP Expansion** and **Threshold-Based Selection**. Key points include:

- **Threshold Selection**: One action (node) is sampled at a time, but it is not compared with other nodes. Instead, once its value is evaluated by whether it surpasses a threshold, that path is followed to its conclusion.
- **LMPE Evaluation for Deadend**: The system uses a deadend judgment to halt exploration of unpromising paths, which can be considered as another LMPE evaluation.
- **Backtracking**: Upon reaching a deadend, the system reverts to an earlier node and continues exploring other previously expanded but untried branches.
- **Path Maintenance**: Because of backtracking, the framework must track partial paths, whereas BFS or beam search only needs to maintain the selected nodes at each depth.

5.4 Best-First Search

Best-first search typically uses a heuristic function $h(s)$ to estimate how promising a state will reach the goal.

[3]For Step2, they do not explicitly specify SPARQL implementations.
[4]Still, they do not explicitly specify SPARQL implementations.

Koh et al. (2024)

- Uses **LMPP Expansion** for the selected node or the initial root node, where actions are executed in the environment (web interface) to simulate the next states. The generated nodes are saved in N.
- Employs TopK_Select (k=1) through LMPE+ evaluation on each node in saved nodes $n \in N$. The node value n.val is derived from evaluating its parent's state.
- Continues until either the search tree reaches a specified budget β or the state value exceeds a threshold θ.

5.5 A*

A* is similar to best-first search but augments the heuristic $h(s)$ with the accumulated cost/utility $g(s)$ to reach a node from the start. The evaluation function is

$$f(s_t) = g(s_t) + \lambda\, h(s_t), \tag{9}$$

where λ balances the two terms. Table 8 summarizes how two frameworks implement this formula differently:

Table 8: LMPE+ Evaluation for LLM-A* and Q*. $\text{dist}(\cdot,\cdot)$ represents the actual distance between two points, while $\|\cdot\|$ denotes the Euclidean norm. $s_0, s_t, s_n, s_g, s_{\text{llm}}$ represent the initial, current, neighbor, goal, LMPP-generated states, respectively.

	$g(s)$		$h(s)$	Use of LMPE
	Agg	$R(s)$/Cost(s)		
LLM-A* (Meng et al., 2024)	$\sum$	$\text{dist}(s_0, s_n)$	$\|s_n - s_g\| + \|s_n - s_{\text{llm}}\|$	$\text{lmpr}_{\text{policy\&eval3}}$ (for $h(s)$)
Q* (Wang et al., 2024a)	min , max , $\sum$, Last	Human feedback, ground-truth, rules, LLM logits	$\max_{a_t \in \mathcal{A}} Q^*(s_t, a_t)$	$\text{lmpr}_{\text{policy\&eval4}}$ (for $h(s)$)
ToolChain (Zhuang et al., 2024)	$\sum$	Self-consistency scores, Longest Common Subsequence scores	Consistency scores, relative position scores	$\text{lmpr}_{\text{policy\&eval2}}$ (for $g(s)$), $\text{lmpr}_{\text{policy\&eval3}}$ (for $h(s)$)

LLM-A* (Meng et al., 2024)

- Designed for path-finding tasks (e.g., mazes).
- $\boldsymbol{g(s_t)}$: Computed incrementally as the path cost from s_0 to s_t. Formally,

$$g(s_t) = \sum_{i=1}^{t} \text{Cost}(s_i). \tag{10}$$

- $\boldsymbol{h(s)}$ under LMPE+ evaluation: The main modification beyond the typical A* is to integrate a LMPE to the $h(s)$. Specifically, $\text{lmpr}_{\text{policy\&eval3}}$ (see Table 4) is applied to evaluate the Euclidean distance from s_n back to the recently visited $s_{\text{llm}} \in \text{lmpr}_{\text{policy}}$, along with the typical Euclidean distance between s_n (expanded neighbour node) and s_g.

Q* (Wang et al., 2024a)

- Targets reasoning and code-generation tasks.
- $\boldsymbol{g(s_t)}$: Aggregates rewards via

$$g(s_t) = \text{Agg}\left(\mathcal{R}(s_1), \ldots, \mathcal{R}(s_t)\right), \tag{11}$$

  where Agg $\in \{\min, \max, \sum, \text{Last}\}$. Under this definition, $\mathcal{R}(s)$ in $g(s_t)$ can be calculated as human feedback, ground-truth, rules or via LMPE+ evaluation, depending on tasks.

- $\boldsymbol{h(s)}$ optionally under LMPE+ evaluation: It is initialized as the optimal Q-value of s_t over all the possible actions:

$$\max_{a_t \in \mathcal{A}} Q^*\left(s_t, a_t\right) \tag{12}$$

  Optionally, $\text{lmpr}_{\text{policy\&eval4}}$ introduces a stronger LMPP to approximate an optimal policy.

ToolChain (Zhuang et al., 2024)

- Designed for tool-based tasks but can be generalized to language reasoning tasks ($\mathcal{T}$ via concatenation).
- $\boldsymbol{g(s_t)}$: Aggregated from two terms:
  1. **Self-consistency scores**: $\text{lmpr}_{\text{policy\&eval2}}$ is used to calculate self-consistency scores, normalized by the number of distinct actions.
  2. **Longest Common Subsequence (LCS) scores**: The LCS score between the current generated path s_t and each completed path m in the memory is computed and normalized by the length of the shorter path (m or s_t). Among the scores obtained for each memory path, only the maximum score is used for $\boldsymbol{g(s_t)}$.
- $\boldsymbol{h(s_t)}$: Also aggregated from two terms:
  1. **Consistency scores** based on $\text{lmpr}_{\text{policy\&eval3}}$.
  2. **Relative position scores**: Similar to the LCS scores in $\boldsymbol{g(s_t)}$, this metric is also based on memory. It computes the average relative position of a candidate action appearing in memory examples.

5.6 Monte Carlo Tree Search

Monte Carlo Tree Search typically involves selection, expansion, path simulation, and backpropagation. Most frameworks under review adhere to these steps, except where noted in the highlights that follow.

After MCTS completes its allotted iterations (or reaches its time/resource limit), a tree is left where each child of the root has associated statistics. The standard practice is to pick the move corresponding to the child node with the highest visit count. The reasoning is that more visits generally indicate a move that has been explored more thoroughly and is statistically more promising. In frameworks where final selection strategies are not specified, we assume this default approach.

Below are the highlights of notable frameworks:

RAP (Hao et al., 2023)

- Applicable to BlocksWorld, Crosswords, and other reasoning tasks.
- Sample_Action depends on whether the action space is finite and predefined (e.g., BlocksWorld, where exhaustive retrieval is used) or open-ended (e.g., Crosswords, where LMPP sampling is used).

LATS (Zhou et al., 2024a)

- Targets tasks with reversible actions (e.g., certain reasoning problems).
- Executes actions in the actual environment during path simulation, requiring actions to be reversible to allow repeated trials.

LLM-MCTS (Zhao et al., 2023)

- Designed for robotic tasks.
- Uses random sampling and a domain-specific simulator for path simulation, producing next states and rewards.
- Adopts a domain-specific $P(s, a)$ in PUCT, which is derived from LMPP sampling to form an action distribution.

PG-TD (Zhang et al., 2023)

- Specializes in code generation.
- Treats the prior distribution $P(a \mid s)$ in PUCT as the LMPP token probabilities for the next token, given the partial program.
- Uses LMPT simulation to generate partial programs, but internally adopts beam search to complete the path until a leaf node is reached. They refer to the whole process as "evaluation").

GDP-ZERO (Yu et al., 2023)

- Designed for goal-oriented dialogue.
- Skips path simulation (the dialogue may terminate at any step), so LMPE evaluation is performed directly to produce a reward for MCTS backpropagation after expansion.
- Still requires LMPT simulation for expansion. Particularly, two inference calls are needed to simulate both the system and user responses.

rStar (Qi et al., 2025)

- Designed for reasoning tasks.
- Defines a more heterogeneous set of actions, including proposing an intermediate thought, generating multiple thoughts until reaching a terminal state, or rephrasing the original question and questions.
- Acknowledging the limitations of small language models (SLMs) in serving as direct evaluators, it employs self-consistency scores derived from multiple simulation samples (i.e., $\text{lmpr}_{\text{policy\&eval2}}$).
- Instead of selecting only the next action at the root node, the framework selects an entire trajectory from all rollout trajectories. To verify candidates, the selected trajectory is pruned to assess whether another SLM would generate the same subsequent steps.

6 Discussion

In this section, we analyze how search frameworks for LLM inference deviate from traditional search algorithms, where and how they apply, and the resulting impact on performance and efficiency.

6.1 Deviations from Typical Search Algorithms

Beyond Finite, Fixed Search Space Typical search algorithms, e.g., BFS (breath), deals with fixed search space and needs to keep track of all possibilities at each depth level, the large or infinite search space can lead to excessive memory consumption. LMPP sampling based on LLM priors makes BFS overcome this limitation.

Beyond Finite, Fixed Search Spaces Classical BFS or DFS typically requires enumerating all successors at each depth, which can lead to massive memory usage in large or infinite search spaces. By contrast, **LMPP sampling** (based on LLM priors) can manage successor expansions more selectively, reducing the need to store every possibility at each level. Also, it is possible to handle tasks with an open and infinite action space.

Making "Uninformed" Search Informed Traditionally, BFS and DFS are considered *uninformed*, exploring the search space without heuristics. LLM-based frameworks labeled as BFS or DFS often incorporate **LMPP sampling** or **LMPE+ evaluation**, effectively introducing heuristic knowledge from the LLM. Moreover, **anticipating dead ends** in DFS is feasible with LLM-based heuristics. Classic DFS only identifies dead ends when it exhausts neighbor nodes. With an LLM, the search can backtrack early if the model predicts an unpromising or "dead-end" scenario.

Compromised Optimality in A* A* requires an *admissible* heuristic $h(s)$ to guarantee optimality. However, if $h(s)$ partially depends on a policy-generated state s_{llm} from $\text{lmpr}_{\text{policy}}$, the heuristic may be overestimated. This breaks admissibility assumptions, meaning the final solution may no longer be strictly optimal.

Terminology Deviation for Heuristic Search: Cost vs. Return/Value Many early applications of search algorithms, such as Best-First Search and A*, focused on minimizing quantities like distance, travel time, or energy expenditure. Generally, the cost-based evaluation can be considered as minimization objectives or eliminating "bad" states (or unpromising actions). However, in modern applications involving LLM-integrated search, such as web navigation or document retrieval, heuristics often reflect *value estimates* (positive polarity), especially when LLMEs tend to be defined to reflect the relevance or utility of states, which are better suited for maximization objectives or maintaining "good" states (or promising actions). We highlight this point for rigidity. However, these terms can be abstractly defined without indicating real-world semantics. For example, in game design, moving to a node which end up losing a life point can be given a negative reward, while a reward from gaining a key can be granted as a negative cost.

6.2 Applicability

Extending Uninformed Search to Dynamic Decision-Making BFS and DFS were originally designed to explore predefined or easily generated state spaces (e.g., enumerating all children in a graph). This can be: **1) No Need for Transition**: Traditional implementations of DFS and BFS do not require an explicit, computed transition model because they inherently rely on the graph structure where edges already define state transitions. For example, one task is to solve a maze. **2) Only Successor Function**: In some applications, especially in implicit state-space search, a minimal form of transition function (or successor function) is still required to generate successors when the full graph is not explicitly available.

However, when applied to LLM inference, they can be adapted for: **3) Tasks That Require Dynamic Decision-Making**: These tasks are based on state-dependent actions (as seen in planning or reinforcement learning), e.g., solving the Game of 24 and completing crossword puzzles.

Open-Loop vs. Closed-Loop Frameworks This property is important to discuss the applicability of frameworks for planning. **1) Open-Loop (Offline)**: After executing a_t, the open-loop agent does not adapt its future actions based on the actual new state. Instead, it continues to follow a pre-planned sequence of actions, which are based on the simulated state during planning. For example, ToT-DFS and ToT-BFS (Yao et al., 2023a) produce a predefined sequence of actions based on a search through a static search tree or graph. This sequence is intended to be executed exactly as planned. Ideally, open-loop planning requires that the environment will remain as anticipated throughout the execution. Such environments satisfies the following assumptions: a) environments are static, b)(no unforeseen events or changes will affect the execution (closed-world assumption), and c) deterministic transitions. **2) Closed-Loop (Online)**: In contrast, after the closed-loop agent executes an action a_t, it observes the outcome in the real world (i.e., the resulting state) and can adjust its future action a_{t+1} based on the new state s_{t+1}. The plan evolves dynamically as new information becomes available. MCTS-based frameworks, except for LATS (Zhou et al., 2024a), can be open-loop because it generates a sequence of actions based on simulations and a static model of the environment at a given state. However, the agent can operate in a closed-loop manner if MCTS is re-run at each new state: after a plan is generated for s_t, only a_t is selected and executed. Instead of executing the rest $a_{t+1}, \ldots$, the agent re-runs MCTS from the new state s_{t+1} to determine the next best action a_{t+1}. These frameworks are suitable for task execution in dynamic and interactive environments.

LATS Requires Action Undoing LATS (Zhou et al., 2024a) relies on direct environment interaction for path simulation. It assumes actions are reversible so that the agent can revisit earlier states if necessary. This limits its applicability to environments where undoing actions is viable.

6.3 Performance

MCTS May Degenerate in Early Stages Chen et al. (2024b) observe that if all candidate steps receive equal (or zero) scores initially, MCTS lacks a clear basis for distinguishing among branches, potentially leading to suboptimal partial-plan selection. The performance is worse than iterative refinement (Madaan et al., 2023).

6.4 Efficiency

General Running Time As noted by Chen et al. (2024b), tree search can be 10–20 times slower than iterative refinement, especially if the evaluator (LMPE) has less than 90% discrimination accuracy. High-accuracy LMPEs are essential to prune the search tree effectively, thereby reducing the number of iterations needed.

LMPP Expansion and Path Simulation with Memory Finally, maintaining a memory of explored nodes can avoid repeated sampling and simulation. If a given state s has already produced certain actions, those child nodes can be cached for subsequent expansions. Similarly, for simulation, previously simulated results can be stored for future simulation. By reusing these cached outcomes, the framework reduces redundant calls to LMPP or LMPT, thereby improving both inference cost.

Unnecessary LMPP Use in Some Cases Some tasks possess a small, tractable action space (e.g., the Game of 24). In such scenarios, *exhaustive* action retrieval may be cheaper than performing multiple LMPP inferences. Designers must weigh the inference costs of LMPP against potential benefits, as LLM-based sampling can be expensive relative to enumerating a finite set of actions. The potential cost of the following LMPE+ evaluation should also be considered.

7 LLM Inference + Search Beyond Sequential Decision Making

This survey examines frameworks designed for sequential decision-making tasks, which are detailed in Section 2. Additionally, this section explores other frameworks that include both LLM inference and search methods. However, rather than directly searching for intermediate actions in sequential decision-making tasks, these approaches either optimize other elements, such as model selection or inference workflows, or do not fit in the MDP formulation.

As a result, the notions of actions and states do not exist, at least during LLM inference. This distinction leads to several fundamental differences. For instance, the roles of LMPP and LMPT are not applicable, which is why LLM sampling, rather than LMPP sampling, is introduced in the frameworks below. Furthermore, state simulation is never required.

LLM for World Modeling + Search In this paradigm, LLMs are employed to generate world models that serve as the basis for planning. In contrast, LLMs in the LIS frameworks operate as world models (e.g., LMPEs and LMPTs). The world models can be represented by the Planning Domain Definition Language (PDDL), which clearly defines action preconditions and effects, or Python code. For example, Guan et al. (2023); Liu et al. (2023) utilize LLMs to generate PDDL-based world models and then apply classical search-based planners (e.g., LPG) to solve tasks. In contrast, Dainese et al. (2024) prompt LLMs to generate Python code for world modeling and then solve tasks via Monte-Carlo Tree Search (MCTS).

Search for LLM-Inference Workflows AFlow (Zhang et al., 2025) employs MCTS to search for optimal inference workflows for solving downstream tasks. However, the resulting solutions may not strictly conform to a search-based workflow. Essentially, this approach focuses on the structure and efficiency of the overall inference process. In contrast, the LIS frameworks directly concentrate on constructing search workflows for downstream tasks.

Equilibrium Search in a Two-Player Game Generally, a discriminator (a special LMPE) and a generator (a special LMPP) are defined to engage in a game, where a correctness parameter for the generator is selected uniformly at random. Both the discriminator and the generator receive a payoff of 1 when they agree on the correctness parameter. This equilibrium objective is regularized by the prior, i.e., the inital generation of large language models (LLMs), as some equilibrium solutions may not align with commonsense reasoning. The final state is reached when the discriminator and the generator reconcile with each other. Unlike single-player games, two-player games involve theoretical frameworks under the umbrella of game theory and equilibrium analysis. While this aspect is beyond the scope of this survey, the fundamental steps of LLM sampling and LMPE evaluation remain essential prerequisites before executing equilibrium computations to achieve a regularized equilibrium.

- **LLM Sampling**: An LLM is prompted for sampling in batch. While this is similar to $\text{lmpr}_{\text{batch\_policy2}}$, the LLM is prompted with the value of the correctness parameter, which is either "correct" or "incorrect".
- **LMPE Evaluation**: Corresponding to the values of the correctness parameter, the LMPE (or discriminator) generates binary predictions (i.e., "correct" or "incorrect"). Specifically, the LMPE is implemented as $\text{lmpr}_{\text{eval1}}$.

Evolutionary Search for Code Generation EvoPrompting (Chen et al., 2023) employs evolutionary search to generate implementation code for neural architectures. However, it does not decompose complete code into smaller components for sequential decision making.

The key procedures in their algorithm are:

- **LLM Sampling**: An LLM is used during the cross-mutation phase to generate candidate codes. The generation process is conditioned on randomly selected codes from the population.
- **Value-Based Top-K Selection**: This method updates the population by selecting the top K candidates based on a value function. The ValueFunc is implemented by training a deep learning model with the generated code and evaluating its validation accuracy.

8 Related Work

Although the primary focus of this survey is on test-time compute via search, several related directions fall outside our current scope:

- **LLM Training/Fine-Tuning for LMPRs.** Recent methods adapt large language models through fine-tuning or preference optimization to enhance their policy, evaluation, or transition roles. Examples include Chain of Preference Optimization (Zhang et al., 2024b), AlphaZero-like tree search (Wan et al., 2024), ReST-MCTS* (Zhang et al., 2024a), and AgentQ (Putta et al., 2024).
- **Search with Multi-Modal LLMs.** Some work extends tree-based exploration and action selection to multi-modal contexts by incorporating visual features alongside textual reasoning steps, e.g., Mulberry (Yao et al., 2024)
- **Branching without Search.** Some frameworks utilize branching or tree-like expansions but do not incorporate a full-fledged search algorithm. Examples include Tree-Planner (Hu et al., 2024), Boost-of-Thoughts (Chen et al., 2024a), and Graph-of-Thoughts (GoT) (Besta et al., 2024). Although they adopt branching structures similar to traditional search, these methods rely on aggregation, sorting, or heuristics rather than explicit search procedures.

References

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17682–17690, 2024.

Angelica Chen, David Dohan, and David So. Evoprompting: Language models for code-level neural architecture search. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=ifbF4WdT8f`.

Sijia Chen, Baochun Li, and Di Niu. Boosting of thoughts: Trial-and-error problem solving with large language models. In *The Twelfth International Conference on Learning Representations*, 2024a. URL `https://openreview.net/forum?id=qBL04XXex6`.

Ziru Chen, Michael White, Ray Mooney, Ali Payani, Yu Su, and Huan Sun. When is tree search useful for LLM planning? it depends on the discriminator. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13659–13678, Bangkok, Thailand, August 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.738. URL `https://aclanthology.org/2024.acl-long.738`.

Nicola Dainese, Matteo Merler, Minttu Alakuijala, and Pekka Marttinen. Generating code world models with large language models guided by monte carlo tree search. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL `https://openreview.net/forum?id=9SpWvX9ykp`.

Xidong Feng, Ziyu Wan, Haotian Fu, Bo Liu, Mengyue Yang, Girish A. Koushik, Zhiyuan Hu, Ying Wen, and Jun Wang. Natural language reinforcement learning, 2024. URL `https://arxiv.org/abs/2411.14251`.

Olga Golovneva, Sean O'Brien, Ramakanth Pasunuru, Tianlu Wang, Luke Zettlemoyer, Maryam Fazel-Zarandi, and Asli Celikyilmaz. PATHFINDER: Guided search over multi-step reasoning paths. In *R0-FoMo:Robustness of Few-shot and Zero-shot Learning in Large Foundation Models*, 2023. URL `https://openreview.net/forum?id=5TsfEEwRsu`.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=zDbsSscmuj`.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 8154–8173, Singapore, December 2023. Association for Computational Linguistics. URL `https://aclanthology.org/2023.emnlp-main.507`.

Mengkang Hu, Yao Mu, Xinmiao Chelsey Yu, Mingyu Ding, Shiguang Wu, Wenqi Shao, Qiguang Chen, Bin Wang, Yu Qiao, and Ping Luo. Tree-planner: Efficient close-loop task planning with large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Glcsog6zOe.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. Understanding the planning of llm agents: A survey. *arXiv preprint arXiv:2402.02716*, 2024.

Athul Paul Jacob, Yikang Shen, Gabriele Farina, and Jacob Andreas. The consensus game: Language model generation via equilibrium search. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=n9xeGcI4Yg.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. Language models (mostly) know what they know. *CoRR*, abs/2207.05221, 2022. URL https://doi.org/10.48550/arXiv.2207.05221.

Jikun Kang, Xin Zhe Li, Xi Chen, Amirreza Kazemi, Qianyi Sun, Boxing Chen, Dong Li, Xu He, Quan He, Feng Wen, Jianye Hao, and Jun Yao. Mindstar: Enhancing math reasoning in pre-trained llms at inference time, 2024. URL https://arxiv.org/abs/2405.16265.

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents. *arXiv preprint arXiv:2407.01476*, 2024.

Shuang Li, Xavier Puig, Chris Paxton, Yilun Du, Clinton Wang, Linxi Fan, Tao Chen, De-An Huang, Ekin Akyürek, Anima Anandkumar, Jacob Andreas, Igor Mordatch, Antonio Torralba, and Yuke Zhu. Pre-trained language models for interactive decision-making. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=FWMQYjFso-a.

Xinzhe Li. A review of prominent paradigms for llm-based agents: Tool use (including rag), planning, and feedback learning. *arXiv preprint arXiv:2406.05804*, 2024.

Stephanie Lin, Jacob Hilton, and Owain Evans. Teaching models to express their uncertainty in words. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL https://openreview.net/forum?id=8s8K2UZGTZ.

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.

Shengjie Ma, Chengjin Xu, Xuhui Jiang, Muzhi Li, Huaren Qu, Cehao Yang, Jiaxin Mao, and Jian Guo. Think-on-graph 2.0: Deep and faithful large language model reasoning with knowledge-guided retrieval augmented generation. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=oFBu7qaZpS.

Shengjie Ma, Chengjin Xu, Xuhui Jiang, Muzhi Li, Huaren Qu, Cehao Yang, Jiaxin Mao, and Jian Guo. Think-on-graph 2.0: Deep and faithful large language model reasoning with knowledge-guided retrieval augmented generation. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=oFBu7qaZpS.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

Silin Meng, Yiwei Wang, Cheng-Fu Yang, Nanyun Peng, and Kai-Wei Chang. Llm-a*: Large language model enhanced incremental heuristic search on path planning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, December 2024.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Gray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=TG8KACxEON`.

Alexander Pan, Erik Jones, Meena Jagadeesan, and Jacob Steinhardt. Feedback loops with language models drive in-context reward hacking. In *Forty-first International Conference on Machine Learning*, 2024. URL `https://openreview.net/forum?id=EvHWlYTLWe`.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Ramesh Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. Agent q: Advanced reasoning and learning for autonomous ai agents. 2024. URL `https://api.semanticscholar.org/CorpusID:271865516`.

Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. Mutual reasoning makes smaller LLMs stronger problem-solver. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=6aHUmotXaw`.

Stuart J Russell and Peter Norvig. *Artificial intelligence a modern approach*. London, 2010.

Benjamin Adin Spiegel, Ziyi Yang, William Jurayj, Katie Ta, Stefanie Tellex, and George Konidaris. Informing reinforcement learning agents by grounding natural language to markov decision processes, 2024. URL `https://openreview.net/forum?id=P4op21eju0`.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel Ni, Heung-Yeung Shum, and Jian Guo. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=nnVO1PvbTv`.

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

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023a. URL `https://openreview.net/forum?id=YXogl4uQUO`.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models - a critical investigation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023b. URL `https://openreview.net/forum?id=X6dEqXIsEW`.

Ziyu Wan, Xidong Feng, Muning Wen, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training, 2024. URL `https://openreview.net/forum?id=fLO9VaAb3B`.

Chaojie Wang, Yanchen Deng, Zhiyi Lv, Shuicheng Yan, and An Bo. Q*: Improving multi-step reasoning for llms with deliberative planning. *arXiv preprint arXiv:2406.14283*, 2024a.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):1–26, 2024b.

Sihan Wang, Kaijie Zhou, Kunfeng Lai, and Jianping Shen. Task-completion dialogue policy learning via Monte Carlo tree search with dueling network. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 3461–3471, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.278. URL https://aclanthology.org/2020.emnlp-main.278.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, Xu Zhao, Min-Yen Kan, Junxian He, and Qizhe Xie. Self-evaluation guided beam search for reasoning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=Bw82hwg5Q3.

Huanjin Yao, Jiaxing Huang, Wenhao Wu, Jingyi Zhang, Yibo Wang, Shunyu Liu, Yingjie Wang, Yuxin Song, Haocheng Feng, Li Shen, et al. Mulberry: Empowering mllm with o1-like reasoning and reflection via collective monte carlo tree search. *arXiv preprint arXiv:2412.18319*, 2024.

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, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=WE_vluYUL-X.

Xiao Yu, Maximillian Chen, and Zhou Yu. Prompt-based monte-carlo tree search for goal-oriented dialogue policy planning. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=ZkR2bWvRpZ.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. ReST-MCTS*: LLM self-training via process reward guided tree search. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024a. URL https://openreview.net/forum?id=8rcFOqEud5.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=z5uVAKwmjf.

Shun Zhang, Zhenfang Chen, Yikang Shen, Mingyu Ding, Joshua B. Tenenbaum, and Chuang Gan. Planning with large language models for code generation. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=Lr8cOOtYbfL.

Xuan Zhang, Chao Du, Tianyu Pang, Qian Liu, Wei Gao, and Min Lin. Chain of preference optimization: Improving chain-of-thought reasoning in LLMs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024b. URL https://openreview.net/forum?id=2cczgOfMP4.

Zirui Zhao, Wee Sun Lee, and David Hsu. Large language models as commonsense knowledge for large-scale task planning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=Wjp1AYB8lH.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models, 2024a. URL https://openreview.net/forum?id=6LNTSrJjBe.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=WZH7099tgfM`.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*, 2024b. URL `https://openreview.net/forum?id=oKn9c6ytLx`.

Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A. Rossi, Somdeb Sarkhel, and Chao Zhang. Toolchain*: Efficient action space navigation in large language models with a* search. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=B6pQxqUcT8`.