Tree of Thoughts as a Classical Heuristic Search Problem: Formal Foundations and Design Patterns

Guni Sharon
Texas A&M University
guni@tamu.edu

Abstract

Large Language Models (LLMs) have demonstrated remarkable reasoning capabilities, yet their standard generation process—auto-regressive token prediction—is inherently myopic and prone to cascading errors. To address this, the *Tree-of-Thoughts* (ToT) framework creates a search space over intermediate reasoning steps, allowing search models to explore, look ahead, and backtrack. However, current ToT research remains fragmented across *Natural Language Processing* and *Automated Planning* communities, often using inconsistent terminology and ad-hoc implementations. Consequently, we synthesize the ToT landscape through a unified taxonomy based on *classical heuristic search terminology*. We map LLM-based reasoning to classical search components: state representation (granularity of thoughts), successor generation (prompting operators), and heuristic evaluation (self-assessment of progress). We analyze existing work within the context of our taxonomy and identify emerging design patterns: systematic search (Best-First Search) for shallow, deterministic tasks and lookahead-heavy strategies (DFS, MCTS) for deep multi-step reasoning. We conclude by identifying open algorithmic challenges at the intersection of heuristic search and LLM reasoning, and call on the heuristic search community to engage with this emerging domain.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities in reasoning and problem-solving across a wide range of domains, from creative writing to mathematical proofs. Central to this success is the *Chain of Thought* (CoT) prompting paradigm (Wei et al. 2022), which encourages models to generate intermediate reasoning steps before arriving at a final answer. However, CoT is fundamentally linear: once a reasoning step is generated, the model commits to it, often propagating early errors into final failures. These limitations are particularly pronounced in planning domains; Valmeekam et al. (2023) conclude that "LLMs' ability to generate executable plans autonomously is rather limited."

To address this, recent work introduced the *Tree-of-Thoughts* (ToT) framework (Yao et al. 2023), which generalizes CoT by maintaining multiple active reasoning paths and allowing lookahead and backtracking, effectively defining a tree search problem over a semantic state space. While ToT demonstrated significant accuracy improvements over CoT on various reasoning tasks, it lacks a unified taxonomy and consistent terminology. This lack of consistency limits (i) meaningful comparison between methods, (ii) understanding of design trade-offs, and (iii) transfer of established techniques from the heuristic search literature.

Addressing this gap, we propose a unified taxonomy mapping ToT to classical search components:

- **State Representation:** a sequence of "thoughts".
- **Successor Generation:** LLMs as generative successor functions.
- **Heuristic Evaluation:** Leveraging LLMs to estimate progress toward a goal.

By grounding ToT in classical search terminology, we enable systematic transfer of established search techniques to a domain with novel characteristics: **(1) Stochastic partial expansion:** Repeated expansion of the same state yields different successors; **(2) Implicit heuristics:** State evaluation emerges from learned patterns rather than domain-specific functions; **(3) High-dimensional linguistic state spaces:** States are text sequences, requiring new approaches to symmetry detection and state abstraction; and **(4) Hybrid objectives:** optimization targets combine solution correctness with qualitative measures such as fluency or diversity. These characteristics present novel algorithmic challenges that the search community is uniquely positioned to address. Beyond the algorithmic interest, LLM-based reasoning sits at the frontier of AI, with direct impact on automated decision-making, scientific discovery, and software engineering. Consequently, advances in ToT search algorithms may directly improve the reliability and effectiveness of LLM-based planning and reasoning systems.

2 Preliminaries

We briefly review *Large Language Models* (LLMs) and structured prompting techniques, introduce the ToT framework (Yao et al. 2023), and establish the heuristic search concepts that will be applied throughout the paper.

2.1 LLMs and Autoregressive Generation

LLMs are autoregressive models that generate text by iteratively predicting the next token conditioned on a prefix sequence (Vaswani et al. 2017).

Tokens. A *token* is the atomic unit of text processed by a language model. Tokens typically correspond to subword units produced by a fixed tokenizer (e.g., Byte Pair Encoding (Sennrich, Haddow, and Birch 2016)) and may represent words, word fragments, punctuation, or special symbols. Higher-level textual constructs, such as sentences or "thoughts", are therefore represented as sequences of tokens.

Given an input prompt x and a generated prefix $y_{1:t-1}$,[1] the model defines a distribution $p(y_t \mid x, y_{1:t-1})$, which represents the probability of the next token y_t conditioned on the prompt and previously generated tokens. During generation, the next token is drawn from this distribution according to a chosen *decoding strategy*. Repeating this process produces a token sequence.

Decoding Strategies. A *decoding strategy* specifies how the next output token is selected from the model's conditional distribution at each generation step. Common strategies include greedy decoding (Graves 2012), beam search (Freitag and Al-Onaizan 2017), top-k sampling (Fan, Lewis, and Dauphin 2018), and nucleus (top-p) sampling (Holtzman et al. 2020). These strategies trade off determinism, diversity, and computational cost, and implicitly define a local search policy over the token-level decision space.

Standard decoding strategies operate at the token level—at each step, one or more candidate tokens are selected and appended to growing sequences. Even strategies that maintain multiple candidates, such as beam search, evaluate alternatives based on token-level likelihood rather than semantic reasoning. Consequently, they have limited capacity to pause at intermediate reasoning steps, evaluate progress toward a goal, or backtrack at the level of meaningful semantic reasoning units ("thoughts").

2.2 Prompting and Intermediate Reasoning

Prompting strategies (Liu et al. 2023b) are methods of structuring the input prompt x to shape the LLM's conditional token distribution $p(y_t \mid x, y_{1:t-1})$. They commonly include specific instructions, examples, or output format specifications with the objective of directing the LLM toward accurate and task-appropriate output distributions, from which a given decoding strategy will sample the output. A prominent example is *Chain-of-Thought* (CoT) prompting (Wei et al. 2022; Kojima et al. 2022), which encourages the model to generate explicit intermediate reasoning steps before producing a final answer. CoT has been shown to improve performance on arithmetic, logical, and symbolic reasoning tasks.

However, CoT is fundamentally linear and non-backtracking: the model commits to a single reasoning trajectory in a greedy fashion. Within the framework of combinatorial search, this corresponds to exploring a single reasoning path where errors in early search nodes (intermediate steps) inevitably lead to suboptimal or incorrect leaf nodes. This limitation motivates approaches that explicitly branch, evaluate, and explore multiple intermediate reasoning paths.

2.3 Tree of Thoughts (ToT)

ToT (Yao et al. 2023) extends Chain-of-Thought by framing reasoning as a structured search process over intermediate textual steps, referred to as *thoughts*.

Thoughts. A *thought* is a contiguous sequence of tokens generated by a language model under a specified decoding strategy until a designated stopping condition is reached (e.g., a delimiter token, newline, or length threshold). Each node in the ToT is associated with a candidate thought generated from its parent node. The segmentation of text into thoughts is not uniquely defined and may be specified explicitly via prompt structure or stopping conditions, or implicitly induced by the language model during generation. For example, in planning domains (Valmeekam et al. 2023), a thought may correspond to a single action (e.g., `"pick block A and place on block B"`) or to a sequence of related actions (a macro-operator).

Instead of generating a single linear sequence of thoughts as in CoT prompting, ToT maintains multiple candidate reasoning paths organized in a tree structure. At each step, the model proposes multiple candidate thoughts, evaluates them, and selectively expands promising branches. Conceptually, ToT separates the reasoning process into four components:

1. A *proposal mechanism* that generates candidate next thoughts from a given sequence of thoughts.
2. An *evaluation mechanism* that estimates the quality or promise of partial solutions (sequences of thoughts).
3. A *goal test mechanism* that determines whether a given sequence of thoughts constitutes a valid solution. Note that some prior ToT formulations implicitly conflate evaluation and goal testing; we separate them for clarity and consistency with classical search formulations.
4. A *search strategy* that determines how the ToT is explored (e.g., depth-first, breadth-first, best-first).

This decomposition allows ToT to incorporate classical search algorithms while operating over a high-dimensional, linguistically-defined state space.

Note that the ToT literature (Yao et al. 2023) often uses the term *intermediate reasoning step* inherited from Chain-of-Thought prompting (Wei et al. 2022). While commonly used, this term is ambiguous in the context of heuristic search—it may refer to an atomic reasoning unit (a thought),

[1] We use the slice notation $y_{i:j}$ to denote the sequence $y_i, y_{i+1}, \ldots, y_j$ throughout the paper.

a partial solution (reasoning path), or a ToT node. Consequently, for precision in our formalization, we avoid this term and instead use *thought* to denote the atomic unit of reasoning (as defined above), *state* to denote a sequence of thoughts, and *node* to denote a ToT element containing a state and associated metadata (e.g., g and h values). Note that a state here is represented as a sequence of thoughts rather than a world configuration. This is because a sequence of thoughts implicitly defines a world configuration—namely, the one resulting from applying the sequence of thoughts to the initial configuration—while also serving as the context required by the LLM for next-thought generation. The formal details of the state space are given in Section 3.1.

2.4 Classical Search Terminology

A search problem is defined over a directed, weighted *state-space graph* $G = (S, E)$, where S is a set of states (vertices) and $E \subseteq S \times S$ is a set of directed edges. The state-space graph is often too large to store explicitly. In such cases, a search problem is often defined implicitly via a tuple $(s_0, \mathcal{O}, \mathcal{G})$, where $s_0 \in S$ is the initial state, $\mathcal{O} : S \to 2^S$ is the successor function, and $\mathcal{G} : S \to \{\text{True}, \text{False}\}$ is the goal test.

A search problem induces a *search tree* rooted at s_0, where each layer is obtained by applying $\mathcal{O}$ to all states in the previous layer. Each branch in the search tree corresponds to a unique path in G. Following notation by Russell and Norvig (2020), a search tree node n is defined by:

- $n.s$: the state in G to which the node corresponds.
- n.parent: the predecessor node that generated n.
- $n.g$: the path cost (sum of edge weights) from s_0 to $n.s$.

For informed search (Pearl 1983), a heuristic value $n.h$—an estimate of the cost-to-go from $n.s$ to a goal state—is also defined. Informed search algorithms typically use an evaluation function $n.f$ that combines g and h values (e.g., $f = g + h$ in A* (Hart, Nilsson, and Raphael 1968)) to guide node expansion order. Classical properties such as admissibility and consistency of the heuristic provide optimality and completeness guarantees for algorithms like A*.

In contrast to traditional search domains, the ToT state space presents several challenges: the successor function is often stochastic, and both g-values and h-values are not naturally defined. Nevertheless, adapting heuristic search algorithms to ToT can provide principled approaches for balancing exploration, exploitation, and computational cost in LLM reasoning.

2.5 Planning and Search with Language Models

Prior work on the intersection of LLMs and planning presents two competing paradigms. The **first** treats LLMs as *end-to-end planners*, directly prompting them to generate complete plans (Huang et al. 2022). This approach is fundamentally limited: LLMs fail to reliably produce valid plans even in simple domains, as autoregressive generation provides no guarantee of precondition satisfaction or goal reachability (Valmeekam et al. 2023; Kambhampati et al. 2024).

The **second** paradigm, exemplified by LLM+P (Liu et al. 2023a), uses LLMs solely as a *translation layer*: the LLM converts a natural language problem description into a PDDL specification, which is handed off to a classical planner. While effective, this approach requires well-structured domains admitting clean PDDL representations and forgoes the generative flexibility of LLMs for open-ended tasks.

ToT-style methods occupy a middle ground. Rather than treating the LLM as a standalone planner or a mere translator, they introduce a separation of responsibilities between two distinct entities: a *search agent* and the LLM. The search agent manages and explores the ToT—maintaining the search frontier, selecting nodes for expansion, and tracking search progress—while the LLM serves as a stateless component invoked at specific interaction points: generating candidate next thoughts from a given state and possibly estimating the promise of partial solutions. The search agent thus retains full control over the global search strategy, while the LLM contributes its generative and evaluative capabilities locally (Hao et al. 2023; Yao et al. 2023). A concrete specification of the prompt-based interfaces through which the search agent communicates with the LLM—thought generation, heuristic evaluation, goal testing, and state validation—is provided in Appendix A.[2]

This separation of responsibilities reveals a significant gap in current ToT research. The planning and heuristic search communities have developed decades of algorithmic machinery—admissible and consistent heuristics, pruning strategies, anytime algorithms, and learned cost-to-go functions (Ferber, Helmert, and Hoffmann 2020; Ferber et al. 2022; Agostinelli et al. 2019)—that current ToT implementations largely ignore.

We aim to bridge this gap by formalizing ToT as a classical search problem, clarifying the design space for successor generation, cost assignment, and heuristic definitions, and demonstrating how different design choices give rise to distinct search behaviors and algorithmic trade-offs.

Iterative Repair. ToT is a constructive search process where solutions are built incrementally. Alternative paradigms such as *Iterative Repair* or *Local Search* take a different approach: a complete solution is generated first and refined through iterative LLM calls. Prominent examples include Reflexion (Shinn et al. 2023) and Self-Refine (Madaan et al. 2023). These are discussed in Appendix B, along with their trade-offs compared to ToT-based approaches.

3 Search Tree Formalization for ToT

We formalize ToT as a heuristic search problem defined by a state space S, successor function $\mathcal{O}$, cost function (g-value), heuristic function (h-value), and goal test $\mathcal{G}$.

A concrete instantiation of all components defined in this section is provided in Appendix D for readers who prefer a running example alongside the formalism.

[2] All appendices referenced in this paper are available in the full version, at arXiv:2605.28566 (Sharon 2026).

3.1 State Space S

A state $s \in S$ is defined as a sequence $s = [z_0, z_1, z_2, \ldots, z_t]$, where z_0 is the *problem prompt* and each z_i for $i > 0$ is a thought. The problem prompt is a fixed token sequence provided by the user that encodes the problem description, any relevant context, the initial world configuration, and the goal conditions. The initial state is $s_0 = [z_0]$, a singleton sequence containing only the problem prompt. The state space S is the set of all reachable thought sequences.

Formally, a state is represented as a full sequence of thoughts rather than a single thought. This is because the sequence implicitly defines a unique world configuration—the one resulting from applying all thoughts in order to the initial world configuration—and therefore carries the same information as a classical world state, albeit implicitly. A more memory-efficient implementation could store only the single last thought introduced at each ToT node, reconstructing the full state when needed by following backpointers to the root and concatenating thoughts along the path. However, current work typically stores full states per ToT node, as it simplifies implementation, and state memory complexity is less of a concern since the ToT's memory footprint is generally dominated by the LLM itself. In line with common ToT conventions, we assume storing full states per ToT node.

Note that while each thought sequence is unique by construction (making the ToT a tree over thought-sequence states), distinct sequences may induce the *same* world configuration. Consequently, classical graph-search optimization techniques (e.g., duplicate detection, symmetry breaking) may become applicable if the implicit world-state is recoverable.

3.2 Successor Function

For any state $s = [z_0, z_1, \ldots, z_t]$, the successor function $\mathcal{O}(s)$ returns a set of successor states of the form $s' = s + [z_{t+1}]$, where z_{t+1} is a proposed next thought appended to the sequence s and the '+' operator is sequence concatenation. In principle, the number of distinct next thoughts is exponential in the thought length (approximately $|V|^\ell$, where V is the vocabulary size and ℓ is the thought length in tokens). Generating all potential next thoughts is therefore intractable in many cases. To address this, ToT employs *proposal mechanisms* (Yao et al. 2023) that generate a small subset of candidate thoughts. This is akin to Partial Expansion of search nodes (Goldenberg et al. 2014), where only a subset of successors are generated and added to the search frontier. We characterize proposal mechanisms along two orthogonal dimensions: *Sampling Strategy* and *Structural Constraints*.

Sampling Strategy. Defines how candidate thoughts are generated from the LLM:

S1. Independent Sampling. Generate each thought independently by sampling from the LLM's output distribution using a token-level decoding strategy (e.g., top-k sampling, nucleus/top-p sampling). This process is repeated b times to produce b candidate successor thoughts. The randomness at the token level induces diversity in the generated thoughts.

S2. Explicit Diversity Sampling. Extend independent sampling with explicit diversity mechanisms that bias generation to encourage semantic or lexical diversity among candidates. The goal here is to reduce redundancy and improve exploration coverage of alternative reasoning paths. Common approaches include penalizing tokens that would produce n-gram overlap with previously generated thoughts (Li et al. 2016), enforcing embedding-based dissimilarity thresholds (Reimers and Gurevych 2019), or using diverse beam search variants (Vijayakumar et al. 2018).

S3. LLM-Enumerated Proposals. Prompt the LLM to explicitly generate a list of b distinct candidate thoughts in a single forward pass. In practice, this is implemented via prompts such as `"Generate 5 different options for the next step"`, where the LLM response is parsed into 5 thoughts. This leverages the model's ability to produce several alternatives in parallel while maintaining coherence with the current state (Yao et al. 2023). However, it is particularly susceptible to the 'mode-seeking' nature of aligned LLMs, which tend to collapse toward a few high-probability reasoning paths (Kirk et al. 2024), often resulting in lower semantic diversity than independent sampling.

Structural Constraints. Optionally imposed to further restrict the space of valid thoughts:

C1. Domain-Specific Constraints. Restrict thoughts to valid elements of a domain's action schema (Hao et al. 2023), such as legal moves in a game or valid operators in a planning domain. Unlike grammar-based constraints (**C2.**), which enforce adherence to a formal language grammar, domain-specific constraints restrict generation to a defined set of domain actions, enforced either via the LLM prompt (e.g., `"Suggest a valid move from: {move list}"`) or by the search agent filtering out thoughts that do not appear in the provided move list post-generation. Note that whether a valid domain action is also contextually applicable given the current world configuration is a separate concern, addressed by semantic constraints (**C4.**).

C2. Grammar-Based Constraints. Enforce adherence to a formal grammar or structured format (e.g., arithmetic expressions, logical formulas, or formal languages such as PDDL or SQL). This can be implemented via constrained LLM decoding, where the decoder is restricted to only produce tokens that yield a valid parse according to the grammar (e.g., ensuring generated SQL queries conform to valid SQL syntax (Scholak, Schucher, and Bahdanau 2021)), or by the search agent filtering out thoughts that fail to parse post-generation.

C3. Length Constraints. Bound the token length of generated thoughts, either through prompt instructions or explicit truncation, to control computational cost and maintain thought granularity.

C4. Semantic Constraints. Filter thoughts based on contextual validity using external verifiers, simulators, or model-based checkers. Unlike domain-specific constraints, which check the syntactic validity of a thought in isolation, semantic constraints are state-dependent—they assess whether a thought is valid given the current world configuration induced by the preceding thoughts. For example, a syntactically valid planning action may still violate preconditions in the current state and should therefore be rejected. This filtering is performed exclusively by the search agent, which invokes an external validator after thought generation and discards any thoughts that fail the contextual validity check before adding them to the search frontier.

Importantly, these dimensions (sampling strategies and structural constraints) are *orthogonal and composable*: structural constraints (**C1.–C4.**) can be combined with any sampling strategy (**S1.–S3.**), and multiple constraints can be applied simultaneously.

The choice of proposal mechanism directly determines the branching factor, diversity, and computational cost, and implicitly defines which regions of the thought space are explored.

3.3 Pruning at the ToT Node Level

Beyond thought-level filtering, which bounds the branching factor of the ToT, node-level pruning (applied post-generation) is commonly used to further reduce the ToT size and enable practical runtimes for tree search algorithms. Node-level pruning is typically guided by a node priority value. To conform with classical search notation, we denote this priority as the node's f-value and adopt the decomposition $n.f = n.g + n.h$, where potential instantiations of g and h are discussed in Sections 3.4 and 3.5. However, unlike classical search where $n.g$ represents actual path cost and $n.h$ estimates cost-to-go, neither property is guaranteed in ToT settings: g may be uninformative and h often represents success likelihood rather than cost-to-go. Consequently, f-values in ToT should be interpreted as node ranking functions that guide search toward promising regions, rather than as estimates of the optimal path cost. As such, pruning nodes based on their f-value can effectively bias exploration toward high-quality solutions. Common pruning strategies for ToT nodes include:

P1. Beam Pruning. (Yao et al. 2023) At each depth layer, retain only the top-k nodes ranked by f-value across the entire layer, pruning all others. This corresponds to fixed-width best-first expansion without backtracking: once a node is pruned from the beam, it cannot be reconsidered in later expansions. Beam pruning results in linear growth of the search tree, with a total of $O(k \cdot d)$ nodes across all layers up to depth d. Recent beam pruning approaches (Elbaz and Salzman 2025) incorporate diversity objectives directly into the retention criterion, complementing diversity at the generation level (**S2.**).

P2. Local Branching Pruning. At each node expansion, retain only the top-b successors ranked by f-value, pruning the rest. Unlike beam pruning, which operates globally across a depth layer, local branching pruning operates on a per-node basis. When $b > 1$, this results in exponential tree growth ($O(b^d)$ nodes at depth d), but with a reduced branching factor compared to retaining all successors.

P3. Local Threshold Pruning. Prune any node with $n.f >$ `threshold`. Unlike local branching pruning, which retains a fixed number of successors regardless of quality, threshold pruning retains all nodes below the threshold. This preserves multiple promising branches when they exist—beneficial for solution diversity—but cannot bound the branching factor and may result in no pruning when f-values are poorly calibrated.

While necessary for tractable search, pruning methods may invalidate optimality guarantees. Nodes pruned early may lie on optimal solution paths, and the lack of re-expansion mechanisms prevents recovery from such errors.

Additionally, unlike classical successor functions, ToT proposal mechanisms are typically stochastic: repeated expansions of the same state may yield different successors. This stochasticity motivates the use of MCTS algorithms (Browne et al. 2012), which explicitly account for sampling variance by repeatedly resampling successors.

3.4 Cost Function (g-value)

In classical search, $n.g$ represents the cost of the path from the initial state s_0 to state $n.s$. In ToT settings, however, the notion of "cost" for a sequence of thoughts is not universally established. Unlike classical domains where edge costs are typically given, ToT cost functions must often be inferred from general task requirements. Common cost function instantiations include:

G1. Uniform Cost. $n.g$ is defined as the depth of the node in the ToT, equivalent to the number of thoughts in the sequence. For example, in the "Game of 24" task (see D1. in Appendix C.1), each thought corresponds to a single arithmetic operation, and minimizing depth corresponds to finding solutions with fewer operations.

G2. No Cost. In tasks where no natural notion of incremental cost exists—such as creative writing—all edges in the search tree may be assigned zero cost: $n.g = 0$ for all n.

G3. Negative Log-Likelihood (NLL) Cost. Define $n.g$ as the accumulated negative log-likelihood of the thought sequence in $n.s$:

$$g([z_0, z_1, \ldots, z_t]) = -\sum_{i=1}^{t} \log p(z_i \mid z_0, \ldots, z_{i-1}),$$

where $p(z_i \mid z_0, \ldots, z_{i-1})$ is the probability assigned by the LLM to thought z_i given the preceding context, and the probability of each thought is computed as the product of the probabilities of its constituent tokens. This cost function naturally emerges when converting from a probabilistic framework to a cost-based framework, as the negative log transformation converts multiplicative probabilities into additive costs (Meister, Cotterell, and Vieira 2020). Under this cost model,

search favors paths through high-probability thought sequences. Note that unnormalized NLL costs inherently penalize longer reasoning chains, potentially biasing the search toward shorter, shallower reasoning paths. Consequently, it is advisable to apply length normalization.

The choice of cost function significantly impacts search behavior: it defines what constitutes an "optimal" solution and determines how the search algorithm trades off solution quality against path length.

3.5 Heuristic Function (h-value)

In classical search, $h(s)$ estimates the cost-to-go from state s to the nearest goal state satisfying $\mathcal{G}$. In ToT settings, $h(s)$ is challenging to formalize: goals may be underspecified (e.g., `"write a creative story"`) and distance to a goal is not naturally quantifiable in linguistic spaces.

A key distinction from classical search is that common ToT implementations (Yao et al. 2023; Shinn et al. 2023) define $h(s)$ as a *success likelihood*—a score or probability indicating how promising a partial solution appears—rather than a cost-to-go estimate. We denote such heuristics as $h_{\text{success}}(s)$; nodes are ranked by maximizing h_{success} rather than minimizing it. Since h_{success} and g have incompatible scales and directions of preference, they cannot be directly summed to form a meaningful f-value. To align with the classical cost-minimization framework, $h_{\text{success}}(s)$ can be converted to a cost-based heuristic, denoted h_{cost}, via inversion. For example, assuming $h_{\text{success}} \in [0, 1]$:

- $h_{\text{cost}}(s) = (1/h_{\text{success}}(s)) - 1$
- $h_{\text{cost}}(s) = -\log(h_{\text{success}}(s))$

The logarithmic transformation is particularly relevant when h_{success} represents a probability, as it converts multiplicative probability relationships into additive costs, consistent with the NLL cost (**G3.**). In the remainder of this section, heuristic functions are described as they appear in the ToT literature (typically success-based).

H1. Scalar Value Estimation (type h_{success}). The search agent invokes a function approximator to assign a real-valued score to each state, estimating its promise of leading to a successful solution (Yao et al. 2023). The approximator may be the same LLM used for thought generation (e.g., prompted to rate a partial solution on a scale from 0 to 10) or a dedicated learned model such as a *process reward model* (Zhang et al. 2025b). Scores may also be elicited as categorical judgments mapped to numerical values (e.g., *"sure/maybe/impossible"* $\rightarrow$ $2/1/0$) (Yao et al. 2023). Unlike External Functions (**H3.**), the evaluation is model-based rather than grounded in domain-specific execution or symbolic verification.

H2. Thought Probability (type h_{success}). Define $h(s)$ as the joint probability of the thought sequence in s (Pendurkar and Sharon 2025). Formally, for a state $s = [z_0, z_1, \ldots, z_t]$,

$$h(s) = p(s) = \prod_{i=1}^{t} p(z_i \mid z_0, \ldots, z_{i-1})$$

Higher $h(s)$ indicates greater model confidence in the thought sequence. H2 rests on the assumption that $p(\text{success} \mid s) \approx p(s)$, i.e., that the probabilities of thought sequences approximate their likelihood of reaching a solution. Under a negative log transformation, H2 is functionally equivalent to NLL Cost (**G3.**)—both induce the same node ranking—and consequently using them jointly is redundant. Note that in (Pendurkar and Sharon 2025), $h(s)$ is used as part of a ratio cost function $\text{cost}(s) = g(s)/h(s)$ rather than the additive $n.f = n.g + n.h$ decomposition and $g(s)$ is set as **G1.**

H3. External Functions (potentially h_{cost}). The search agent invokes a domain-specific external tool—such as a simulator, code interpreter, or formal verifier—to evaluate state quality (Valmeekam et al. 2023). Unlike Scalar Value Estimation (**H1.**), the h-value is grounded in execution or symbolic reasoning rather than model-based approximation, providing more reliable feedback at the cost of requiring domain-specific tooling. Interestingly, recent work (Corrêa, Pereira, and Seipp 2025) demonstrated that LLMs can independently generate such external heuristic functions.

These evaluation methods serve as proxies for cost-to-go but differ fundamentally from classical heuristics. LLM-based h-values (**H1.**, **H2.**) are not derived from domain knowledge or problem relaxations but from the model's learned associations and pattern matching. Consequently: (1) they may incur significant computational overhead and can potentially benefit from lazy evaluation (Dellin and Srinivasa 2016); and (2) classical properties such as admissibility and consistency generally do not apply. However, several avenues exist for recovering formal guarantees in structured domains. First, admissible closed-form heuristics (e.g., gap heuristics (Helmert 2010), pattern databases (Culberson and Schaeffer 1998)) could be combined with LLM-based estimates, for instance, using the closed-form value as a floor, for recovering (bounded) admissibility guarantees. Second, consistency can often be recovered via pathmax (Mero 1984). Third, recent work on heuristic approximation provides relevant theoretical guarantees regarding admissibility (Ferber et al. 2022). Finally, we note that some effective search algorithms (Orseau et al. 2018; Browne et al. 2012) operate without admissibility guarantees.

3.6 Goal Test

The goal test $\mathcal{G} : S \rightarrow \{\text{True}, \text{False}\}$ determines whether a state constitutes a valid solution.

T1. Deterministic Verification. The search agent verifies solution correctness via direct programmatic computation on the final output, without requiring an external environment. For example, in the "Game of 24" task, the goal test evaluates whether the final arithmetic expression equals 24; in programming tasks, unit tests verify functional correctness (Austin et al. 2021). This approach provides objective, reproducible validation when solution correctness can be determined from the output alone.

T2. LLM-Based Evaluation. The search agent uses an LLM (potentially the same one used for generating states) to judge whether the final thought sequence satisfies the task requirements (Yao et al. 2023). For example, in creative writing tasks, the LLM may assess whether a generated story meets specified constraints (e.g., genre, theme). Notably, in such linguistic domains, the goal test is often a threshold over a continuous quality score which may be inconsistent across evaluations.

T3. External Environment Validation. The search agent verifies solutions by executing them in an external environment, such as a world simulator or interactive execution environment. Unlike Deterministic Verification (**T1.**), correctness here depends on the state of the environment rather than the output alone—for example, in Blocksworld, $\mathcal{G}(s)$ is often determined by executing the plan in a simulator and checking whether the resulting world state matches the goal configuration (Valmeekam et al. 2023).

In practice, hybrid goal tests combining multiple verifications are also possible. For example, a programming task might use both unit test execution (**T1.**) and LLM-based evaluation of code quality or style (**T2.**) (Liu et al. 2023a).

State Validator. Goal tests can often be extended to serve as a *state validator*—a function that determines whether a partial solution defined by s is valid or executable (Valmeekam et al. 2023). For example, in the Game of 24 domain, a state validator would reject partial solutions that lead to division by zero or produce non-numeric results. State validators enable early pruning of invalid branches (dead-end detection), improving search efficiency by avoiding expansion of states that cannot possibly lead to valid solutions. State validators are typically applied as semantic constraints (**C4.**) during successor generation.

4 ToT Design Choices

A practical implementation of ToT must commit to concrete instantiations of the interdependent components described above: (i) the proposal mechanism defining the successor function $\mathcal{O}(s)$, possibly with structural constraints (Section 3.2), (ii) node pruning strategies for tractable computation (Section 3.3), (iii) the path cost function $g(s)$ (Section 3.4), (iv) the heuristic function $h(s)$ (Section 3.5), (v) the goal test $\mathcal{G}$ (Section 3.6), and finally, (vi) the search algorithm that is deployed to traverse the resulting ToT.

While these components are defined independently in the formal framework, in practice their choices are tightly coupled, and some combinations are more natural or effective than others. The task semantics often dictate whether cost-sensitive search is meaningful (i.e., whether minimizing g is a relevant objective), whether heuristic guidance should be scalar-valued or comparative, and whether successor generation should prioritize diversity or likelihood. As a result, ToT implementations typically adopt coherent design patterns rather than arbitrary combinations of components.

4.1 Design Choices in Prior Work

We survey representative ToT implementations from the literature and analyze their design choices along the axes established in Section 3. Table 1 summarizes these implementations, highlighting the diversity of approaches.

Analysis of Patterns. Table 1 reveals design patterns tailored to the structural properties of specific domains:

(1) Shallow, Deterministic Constraint Satisfaction (e.g., Game of 24, Math Reasoning). In domains like *Game of 24* (Yao et al. 2023) and *Math Reasoning* (Xie et al. 2023), the solution depth is shallow (typically 3–5 steps) but the branching factor is high. Success is binary and easily verifiable (**T1.**). Consequently, implementations favor **systematic search** strategies (Best-First Search/Levin Tree Search (Orseau et al. 2018)) that explore diverse 'thought' combinations in parallel, while using uniform or likelihood costs (**G1.**, **G3.**) to favor concise, high-probability reasoning paths. To maintain a **tractable** branching factor for systematic search, aggressive node-level pruning (**P1.**, **P2.**) and structural format constraints (**C1.**) are commonly invoked.

(2) Deep, Multi-Step Reasoning (e.g., Coding, Planning, Crosswords). Domains such as *Programming* (Zhou et al. 2024), *Blocksworld* (Hao et al. 2023), and *Crosswords* (Yao et al. 2023) are characterized by deep solution paths where early errors are often only detected much deeper in the tree. As a result, relevant implementations often apply **MCTS** or **DFS** to enable lookahead rollouts, or employ strict grammar constraints in beam search (Scholak, Schucher, and Bahdanau 2021). They rely heavily on **external feedback** (**T3.**) and structural constraints (**C4.**) to eliminate invalid branches early on.

(3) Open-Ended Creative Generation (e.g., Creative Writing). In tasks like *Creative Writing* (Yao et al. 2023), the goal state is often subjective (**T2.**). Unlike logical domains where structural constraints are hard (binary), here constraints are soft (stylistic). Accordingly, search is often guided by **LLM-based heuristics** (**H1.**) rather than formal verification, prioritizing the generation of high-quality partial drafts over the strict pruning of "incorrect" ToT branches.

5 Open Challenges and Research Directions

The formalization presented in this work exposes a set of concrete algorithmic gaps—places where classical search theory makes precise predictions, but ToT implementations lack the machinery to act on them. We organize these gaps into three research directions.

OC1. Adapting Classical Algorithms. Current ToT implementations can benefit from, yet often overlook, a wealth of classical search techniques. Promising candidates include anytime and bounded-suboptimal algorithms (e.g., weighted A^*, ARA^* (Likhachev, Gordon, and Thrun 2003)) for trading solution quality against computation time; pruning techniques such as partial-order reduction (Wehrle and Helmert 2012), Partial Expansion (Goldenberg et al. 2014), and symmetry breaking (Domshlak, Katz, and Shleyfman 2012) for

Work	Domain	Proposal	Pruning	Search Strategy	g-value	h-value	Goal Test
(Zhang et al. 2025a)	Logic Puzzles	**S3. + C4.**	**P3.**	BFS	**G2.**	**H1.**	**T2.**
(Xie et al. 2023)	Math Reasoning	**S2.**	**P1.**	Beam	**G3.**	**H1.**	**T1.**
(Scholak, Schucher, and Bahdanau 2021)	Text-to-SQL	**S1. + C2.**	**P1.**	Beam	**G3.**	—	**T3.**
(Yao et al. 2023)	Game of 24	**S3. + C1.**	**P1.**	Beam	**G1.**	**H1.**	**T1.**
(Yao et al. 2023)	Creative Writing	**S1. + C3.**	**P1.**	Beam	**G2.**	**H1.**	**T2.**
(Yao et al. 2023)	Crosswords	**S1.**	**P3.**	DFS	**G2.**	**H1.**	**T1.**
(Pendurkar and Sharon 2025)	Game of 24	**S1. + C1.**	**P2.**	LTS	**G1.**	**H2.**	**T1.**
(Pendurkar and Sharon 2025)	Blocksworld	**S1. + C1.**	**P2.**	LTS	**G1.**	**H2.**	**T3.**
(Jiang et al. 2024)	Math Reasoning	**S1.**	—	MCTS	**G2.**	**H3.**	**T1.**
(Hao et al. 2023)	Blocksworld	**S1. + C1.**	—	MCTS	**G2.**	**H1.**	**T3.**
(Zhou et al. 2024)	Programming	**S1.+C1.+C4.**	—	MCTS	**G2.**	**H1.**	**T3.**
(Qi et al. 2024)	Math Reasoning	**S2.+C1.**	—	MCTS	**G2.**	**H3.**	**T1.**

Table 1: Design choices in representative ToT implementations. Labels reference the taxonomy defined in Section 3. **BFS** denotes Best-First Search (f-value ranking). Beam denotes level-by-level Breadth-First Beam Search. **DFS** denotes Greedy Depth-First Search (Russell and Norvig 2020). **LTS** denotes Levin Tree Search (Orseau et al. 2018). **MCTS** (Monte Carlo Tree Search) is listed without a pruning strategy because it relies on probabilistic selection policies (e.g., UCT) to guide exploration rather than hard pruning thresholds. Details regarding the benchmark domains and evaluation metrics used in the listed work are provided in Appendix C.

reducing the effective branching factor; and lazy evaluation frameworks (Dellin and Srinivasa 2016) for deferring expensive LLM calls. Transferring these techniques is not straightforward: suboptimality bounds assume meaningful g-values, while ToT costs are often zero (**G2.**) or length-biased (**G3.**); and symmetry detection requires mapping thought sequences to world configurations (see OC3).

OC2. Heuristic Design and Learning. Given their inherent output randomness, LLM-based heuristics (**H1.**, **H2.**) largely lack admissibility guarantees. While recent work (Ferber et al. 2022) provides such guarantees in specific domains, generalizing them remains an open challenge. Improved heuristic approximation can feed a positive loop where successful search traces enable fine-tuning the LLM's heuristic estimation—e.g., using STaR-style (Zelikman et al. 2022) or DPO (Rafailov et al. 2023) methods—which in turn yields better search traces.

OC3. Novel Search Challenges and Theory. ToT introduces unique algorithmic challenges. First, *Stochastic partial expansion*: repeated expansion of the same state yields different successors, violating determinism assumptions underlying most completeness proofs. While MCTS (Browne et al. 2012) partially addresses this via resampling, convergence proofs assume f-value estimates stabilize with repeated sampling—an assumption that may be violated when LLM outputs are non-stationary. Second, *Semantic state equivalence*: distinct ToT states (thought sequences) may encode identical world configurations. Reliable duplicate detection has significant pruning potential, yet it remains an open problem. Third, *Token-budget search*: LLM inference cost scales with token count; establishing performance bounds for anytime and bounded-suboptimal algorithms under a token budget requires new theoretical machinery. Finally, standardized evaluation requires benchmarks annotated with search-theoretic properties (optimal solution depth, branching factor), enabling controlled comparison of search efficiency across methods (see Appendix C).

6 Conclusion

We presented a unified taxonomy of the Tree-of-Thoughts framework grounded in classical heuristic search, mapping LLM-based reasoning to classical search components: state representation, successor generation, cost evaluation, and heuristic guidance. While ToT implementations vary in their prompting techniques, they converge on a small set of fundamental search patterns tailored to domain structure: systematic search (BFS) for shallow deterministic tasks and lookahead-heavy strategies (DFS, MCTS) for deep reasoning. This perspective allows us to view current LLM agents not as distinct "prompting tricks," but as specific instantiations of well-studied search algorithms.

A Call to the Search Community. Our formalization reveals ToT as more than an NLP technique—it is a heuristic search problem in a fundamentally new domain. Current implementations typically employ basic BFS or DFS without leveraging decades of advances in search algorithms, heuristic design, and pruning techniques. Meanwhile, the unique characteristics of ToT—stochastic successor generation, learned heuristics, linguistic state spaces, and computational budgeting—present novel algorithmic challenges that the heuristic search community is uniquely equipped to address. We hope this work serves as a foundation for systematic engagement between the heuristic search and LLM communities, enabling the transfer of algorithmic insights in both directions: proven search techniques can improve ToT implementations, while the novel characteristics of learned search spaces can inspire new theoretical and algorithmic advances in heuristic search.

Acknowledgments

We thank **Ariel Felner** for encouraging us to write this paper and for providing valuable structural and editorial comments. The research for this paper was supported in part by the NSF (IIS-2238979).

References

Agostinelli, F.; McAleer, S.; Shmakov, A.; and Baldi, P. 2019. Solving the Rubik's Cube with Deep Reinforcement Learning and Search. *Nature Machine Intelligence*, 1: 356–363.

Austin, J.; Odena, A.; Nye, M.; Bosma, M.; Michalewski, H.; Dohan, D.; Jiang, E.; Cai, C.; Terry, M.; Le, Q.; and Sutton, C. 2021. Program Synthesis with Large Language Models. arXiv:2108.07732.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1): 1–43.

Corrêa, A. B.; Pereira, A. G.; and Seipp, J. 2025. Classical Planning with LLM-Generated Heuristics: Challenging the State of the Art with Python Code. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Culberson, J. C.; and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence*, 14(3): 318–334.

Dellin, C.; and Srinivasa, S. 2016. A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors. In *Proceedings of the international conference on automated planning and scheduling*, volume 26, 459–467.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2012. Enhanced symmetry breaking in cost-optimal planning as forward search. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling*, ICAPS'12, 343–347. AAAI Press.

Elbaz, D.; and Salzman, O. 2025. Portfolio Beam Search: Diverse Transformer Decoding for Offline Reinforcement Learning Using Financial Algorithmic Approaches. In *ECAI 2025*, 3847–3854. IOS Press.

Fan, A.; Lewis, M.; and Dauphin, Y. 2018. Hierarchical Neural Story Generation. In Gurevych, I.; and Miyao, Y., eds., *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 889–898.

Ferber, P.; Geißer, F.; Trevizan, F. W.; Helmert, M.; and Hoffmann, J. 2022. Neural Network Heuristic Functions for Classical Planning: Bootstrapping and Comparison to Other Methods. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling*, 583–587.

Ferber, P.; Helmert, M.; and Hoffmann, J. 2020. Neural Network Heuristics for Classical Planning: A Study of Hyperparameter Space. In *Proceedings of the 24th European Conference on Artificial Intelligence*, volume 325, 2346–2353.

Freitag, M.; and Al-Onaizan, Y. 2017. Beam Search Strategies for Neural Machine Translation. In Luong, T.; Birch, A.; Neubig, G.; and Finch, A., eds., *Proceedings of the First Workshop on Neural Machine Translation*, 56–60.

Goldenberg, M.; Felner, A.; Stern, R.; Sharon, G.; Sturtevant, N.; Holte, R.; and Schaeffer, J. 2014. Enhanced Partial Expansion A*. *Journal of Artificial Intelligence Research*, 50: 141–187.

Graves, A. 2012. Sequence Transduction with Recurrent Neural Networks. arXiv:1211.3711.

Hao, S.; Gu, Y.; Ma, H.; Hong, J. J.; Wang, Z.; Wang, D. Z.; and Hu, Z. 2023. Reasoning with Language Model is Planning with World Model. In *Conference on Empirical Methods in Natural Language Processing*.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Helmert, M. 2010. Landmark heuristics for the pancake problem. In *Proceedings of the International Symposium on Combinatorial Search*, volume 1, 109–110.

Holtzman, A.; Buys, J.; Du, L.; Forbes, M.; and Choi, Y. 2020. The Curious Case of Neural Text Degeneration. In *International Conference on Learning Representations*.

Huang, W.; Abbeel, P.; Pathak, D.; and Mordatch, I. 2022. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. arXiv:2201.07207.

Jiang, J.; Chen, Z.; Min, Y.; Chen, J.; Cheng, X.; Wang, J.; Tang, Y.; Sun, H.; Deng, J.; Zhao, W. X.; Liu, Z.; Yan, D.; Xie, J.; Wang, Z.; and Wen, J.-R. 2024. Enhancing LLM Reasoning with Reward-guided Tree Search. arXiv:2411.11694.

Kambhampati, S.; Valmeekam, K.; Guan, L.; Verma, M.; Stechly, K.; Bhambri, S.; Saldyt, L. P.; and B Murthy, A. 2024. Position: LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, 22895–22907.

Kirk, R.; Mediratta, I.; Nalmpantis, C.; Luketina, J.; Hambro, E.; Grefenstette, E.; and Raileanu, R. 2024. Understanding the Effects of RLHF on LLM Generalisation and Diversity. In *The Twelfth International Conference on Learning Representations*.

Kojima, T.; Gu, S. S.; Reid, M.; Matsuo, Y.; and Iwasawa, Y. 2022. Large language models are zero-shot reasoners. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22. ISBN 9781713871088.

Li, J.; Galley, M.; Brockett, C.; Gao, J.; and Dolan, B. 2016. A Diversity-Promoting Objective Function for Neural Conversation Models. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 110–119.

Likhachev, M.; Gordon, G. J.; and Thrun, S. 2003. ARA* : Anytime A* with Provable Bounds on Sub-Optimality. In *Advances in Neural Information Processing Systems*, volume 16. MIT Press.

Liu, B.; Jiang, Y.; Zhang, X.; Liu, Q.; Zhang, S.; Biswas, J.; and Stone, P. 2023a. LLM+P: Empowering Large Language Models with Optimal Planning Proficiency. arXiv:2304.11477.

Liu, P.; Yuan, W.; Fu, J.; Jiang, Z.; Hayashi, H.; and Neubig, G. 2023b. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *ACM Comput. Surv.*, 55(9).

Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegreffe, S.; Alon, U.; Dziri, N.; Prabhumoye, S.; Yang, Y.; Gupta, S.; Majumder, B. P.; Hermann, K.; Welleck, S.; Yazdanbakhsh, A.; and Clark, P. 2023. SELF-REFINE: iterative refinement with self-feedback. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23.

Meister, C.; Cotterell, R.; and Vieira, T. 2020. If beam search is the answer, what was the question? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2173–2185.

Mero, L. 1984. A heuristic search algorithm with modifiable estimate. *Artificial Intelligence*, 23(1): 13–27.

Orseau, L.; Lelis, L. H. S.; Lattimore, T.; and Weber, T. 2018. Single-agent policy tree search with guarantees. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, 3205–3215.

Pearl, J. 1983. Heuristics: intelligent search strategies for computer problem solving.

Pendurkar, S.; and Sharon, G. 2025. Policy-Guided Search on Tree-of-Thoughts for Efficient Problem Solving with Bounded Language Model Queries. *Transactions on Machine Learning Research*.

Qi, Z.; Ma, M.; Xu, J.; Zhang, L. L.; Yang, F.; and Yang, M. 2024. Mutual Reasoning Makes Smaller LLMs Stronger Problem-Solvers. arXiv:2408.06195.

Rafailov, R.; Sharma, A.; Mitchell, E.; Ermon, S.; Manning, C. D.; and Finn, C. 2023. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. *Advances in Neural Information Processing Systems*, 36.

Reimers, N.; and Gurevych, I. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. arXiv:1908.10084.

Russell, S.; and Norvig, P. 2020. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson. ISBN 9780134610993.

Scholak, T.; Schucher, N.; and Bahdanau, D. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In Moens, M.-F.; Huang, X.; Specia, L.; and Yih, S. W.-t., eds., *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 9895–9901. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics.

Sennrich, R.; Haddow, B.; and Birch, A. 2016. Neural Machine Translation of Rare Words with Subword Units. arXiv:1508.07909.

Sharon, G. 2026. Tree of Thoughts as a Classical Heuristic Search Problem: Formal Foundations and Design Patterns. arXiv:2605.28566.

Shinn, N.; Cassano, F.; Gopinath, A.; Narasimhan, K. R.; and Yao, S. 2023. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Valmeekam, K.; Marquez, M.; Sreedharan, S.; and Kambhampati, S. 2023. On the planning abilities of large language models: a critical investigation. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L. u.; and Polosukhin, I. 2017. Attention is All you Need. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Vijayakumar, A. K.; Cogswell, M.; Selvaraju, R. R.; Sun, Q.; Lee, S.; Crandall, D.; and Batra, D. 2018. Diverse Beam Search: Decoding Diverse Solutions from Neural Sequence Models. arXiv:1610.02424.

Wehrle, M.; and Helmert, M. 2012. About partial order reduction in planning and computer aided verification. In *Proceedings of the Twenty-Second International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'12, 297–305. AAAI Press.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E. H.; Le, Q. V.; and Zhou, D. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22. ISBN 9781713871088.

Xie, Y.; Kawaguchi, K.; Zhao, Y.; Zhao, X.; Kan, M.-Y.; He, J.; and Xie, Q. 2023. Self-Evaluation Guided Beam Search for Reasoning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T. L.; Cao, Y.; and Narasimhan, K. 2023. Tree of thoughts: deliberate problem solving with large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23.

Zelikman, E.; Wu, Y.; Mu, J.; and Goodman, N. D. 2022. STaR: self-taught reasoner bootstrapping reasoning with reasoning. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781713871088.

Zhang, Y.; Yang, J.; Yuan, Y.; and Yao, A. C.-C. 2025a. Cumulative Reasoning with Large Language Models. arXiv:2308.04371.

Zhang, Z.; Zheng, C.; Wu, Y.; Zhang, B.; Lin, R.; Yu, B.; Liu, D.; Zhou, J.; and Lin, J. 2025b. The Lessons of Developing Process Reward Models in Mathematical Reasoning. arXiv:2501.07301.

Zhou, A.; Yan, K.; Shlapentokh-Rothman, M.; Wang, H.; and Wang, Y.-X. 2024. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.