Inference-Time Compute Scaling in Large Language Models

A First-Principles Algorithmic and Complexity-Theoretic Survey

Anjan Goswami

General Manager, SmartInfer.com

January 2026

Abstract: This survey provides a rigorous theoretical treatment of inference-time compute scaling in large language models. We establish formal computational models for transformers under various precision regimes, characterize their expressivity through circuit complexity, and analyze how chain-of-thought reasoning extends computational power from $\mathbf{TC}^0$ to $\mathbf{P}$. The survey presents exact theorem statements with proof sketches for the key results: Merrill & Sabharwal's complexity characterization showing $\mathbf{TIME}(t(n)) \subseteq \mathbf{CoT}(t(n)) \subseteq \mathbf{SPACE}(t(n) + \log n)$, and Li et al.'s circuit simulation theorem proving $\mathbf{SIZE}[T(n)] \subseteq \mathbf{CoT}[T(n), \log n, 1]$. We develop a formal taxonomy of inference-time algorithms—including self-consistency, tree-of-thought, and Monte Carlo tree search—with complexity analysis for each. The survey further examines scaling laws, the generation-verification gap, and fundamental impossibility results. We conclude with a hierarchy of open problems ranked by tractability and impact, aimed at guiding future theoretical research in this rapidly evolving field.

Contents

1. Introduction

The emergence of inference-time compute scaling represents a paradigm shift in our understanding of large language model capabilities. Rather than viewing model performance as fixed at deployment, recent theoretical and empirical work demonstrates that *test-time computation acts as a tradeable resource*—performance can be systematically improved by allocating additional inference-time FLOPs through mechanisms such as chain-of-thought reasoning, parallel sampling, and guided search.

This survey provides a first-principles theoretical treatment of inference-time scaling, emphasizing algorithmic foundations and complexity-theoretic analysis. Our goal is to move beyond empirical observations to establish rigorous characterizations: What computational problems become solvable with additional inference steps? What are the fundamental limits? When does test-time compute substitute for model parameters, and when does it fail?

1.1 Motivation and Scope

The practical importance of inference-time scaling is now well-established. OpenAI's o1 and o3 models, DeepSeek-R1, and Anthropic's extended thinking demonstrate that "reasoning models" with extended inference achieve state-of-the-art results on mathematical olympiad problems, competitive programming, and scientific reasoning tasks. These systems use 10-100$\times$ more inference compute than standard models, raising fundamental questions about the nature of this tradeoff.

From a theoretical perspective, inference-time scaling touches core questions in computational complexity:

- **Expressivity:** What complexity classes can transformers recognize with varying amounts of chain-of-thought? The answer—$\mathbf{TC}^0$ without CoT, $\mathbf{P}$ with polynomial CoT—provides the first exact characterization of transformer power in standard complexity terms.
- **Learnability vs. Expressivity:** Can transformers *learn* to use CoT effectively, or does the gap between what they can express and what they can learn constitute a fundamental barrier?
- **Verification:** Many inference-time methods rely on verifiers to guide search. When can a model verify its own outputs? The connection to $\mathbf{NP}$ structure (generation-verification gaps) is deep and underexplored.
- **Scaling Laws:** Do principled functional forms govern the relationship between inference compute and performance? Can we derive compute-optimal allocation strategies?

This survey addresses these questions with mathematical rigor, presenting theorem statements with explicit assumptions and proof sketches for key results.

1.2 Contributions and Organization

The contributions of this survey are threefold:

1. **Unified Computational Framework:** We establish precise definitions for transformers as computational models, clarifying the roles of precision, attention semantics, and architectural variants. This framework (Section 2) provides the foundation for all subsequent complexity results.
2. **Complexity-Theoretic Synthesis:** We present a unified treatment of expressivity results (Section 3), showing how chain-of-thought extends transformer power through the hierarchy $\mathbf{TC}^0 \to \mathbf{L} \to \mathbf{P}$. We include the key proof techniques—layer-norm hashing and circuit simulation—that enable these characterizations.
3. **Algorithmic Taxonomy with Complexity Analysis:** We develop a formal taxonomy of inference-time methods (Section 4), analyzing time complexity, space complexity, and approximation guarantees for each class.

The survey continues with scaling laws and compute-optimal analysis (Section 5), verification theory and the generation-verification gap (Section 6), search-theoretic foundations (Section 7), training-inference tradeoffs (Section 8), open problems (Section 9), and conclusions (Section 10).

1.3 Related Surveys

Several surveys address adjacent topics. Merrill's "What Formal Languages Can Transformers Express?" (TACL 2024) provides comprehensive coverage of transformer expressivity without inference-time focus. The empirical surveys on chain-of-thought prompting document practical techniques but lack theoretical grounding. To our knowledge, this is the first survey providing a unified complexity-theoretic treatment of inference-time compute scaling.

1.4 Notation and Terminology

Throughout this survey, we use the following notation consistently:

Table 1: Notation Reference

Symbol	Meaning
Models and Computation	
$\mathcal{M}$	Language model
$\mathcal{T}$	Transformer architecture
n	Input length
d	Model dimension
L	Number of layers
H	Number of attention heads
$\|\Sigma\|$ or $\|V\|$	Vocabulary size
Chain-of-Thought	
T	Number of CoT steps/tokens
$\mathbf{CoT}(T)$	Class of problems solvable with T CoT steps
$y_{1:T}$	Sequence of CoT tokens
Sampling and Search	
N	Number of samples/candidates
K	Number of comparisons (in knockout)
b	Branching factor
d	Search depth
Probabilities	
p_{gen}	Probability of generating correct solution
p_{ver}	Probability of correct verification
p_{win}	Probability correct beats incorrect in comparison
TPR, FPR	True/False positive rates
Complexity Classes	
$\mathbf{TC}^{00}$	Constant-depth threshold circuits
$\mathbf{NC}^{1}$	Log-depth, polynomial-size circuits
$\mathbf{L}$	Log-space computable functions
$\mathbf{P}$	Polynomial-time computable functions
Compute	
C_{fwd}	Forward pass compute
C_{gen}	Generation compute
C_{ver}	Verification compute
FLOPs	Floating point operations

Terminology. We use "inference-time compute," "test-time compute," and "inference scaling" interchangeably to refer to computational resources expended during model inference (as opposed to training). "Chain-of-thought" (CoT) refers to any intermediate reasoning tokens generated before the final answer, whether prompted or emergent.

2. Computational Model

We begin by establishing the formal computational model for transformers. The choice of precision regime and attention semantics critically determines expressive power, and conflating different formalizations leads to confusion in the literature. This section provides precise definitions that underpin all subsequent results.

2.1 Transformer Architecture: Formal Definition

We adopt the formalization of Merrill et al. (2022), which captures decoder-only autoregressive transformers with sufficient generality for complexity analysis.

Definition 2.1 (Transformer)**.** A *transformer* is a tuple $\mathcal{T} = \langle \Sigma, \mathbb{D}, \alpha, L, H, \phi, \{s_{\ell,h}\}, \{f_\ell\} \rangle$ where:

- Σ is a finite input alphabet
- $\mathbb{D}$ is a scalar datatype (floating-point $\mathbb{F}$ or rationals $\mathbb{Q}$)
- $\alpha : \mathbb{D}^n \to \Delta^n$ is an attention function mapping scores to a probability distribution
- $L \in \mathbb{N}$ is the number of layers
- $H \in \mathbb{N}$ is the number of attention heads per layer
- $\phi : \Sigma \times \mathbb{N} \to \mathbb{D}^m$ is a position-aware embedding function
- $s_{\ell,h} : \mathbb{D}^m \times \mathbb{D}^m \to \mathbb{D}$ computes attention scores for layer ℓ, head h
- $f_\ell : (\mathbb{D}^m)^{H+1} \to \mathbb{D}^m$ is the feedforward function for layer ℓ

The computation proceeds as follows. Given input $x = x_1 \cdots x_n \in \Sigma^n$:

1. **Embedding:** Compute initial representations $v_i^{(0)} = \phi(x_i, i)$ for $i \in [n]$.
2. **Attention:** For each layer $\ell \in [L]$ and head $h \in [H]$, compute:

$$a_{\ell,h,i} = \alpha\left(s_{\ell,h}(v_i^{(\ell-1)}, v_1^{(\ell-1)}), \ldots, s_{\ell,h}(v_i^{(\ell-1)}, v_n^{(\ell-1)})\right) \tag{1}$$

$$u_{\ell,h,i} = \sum_{j=1}^{n} [a_{\ell,h,i}]_j \cdot v_j^{(\ell-1)} \tag{2}$$

3. **Feedforward:** Update representations:

$$v_i^{(\ell)} = f_\ell(v_i^{(\ell-1)}, u_{\ell,1,i}, \ldots, u_{\ell,H,i}) \tag{3}$$

4. **Output:** The final representation $v_n^{(L)}$ determines the output.

For decoder-only models with causal masking, position i can only attend to positions $j \leq i$, enforced by setting $s_{\ell,h}(v_i, v_j) = -\infty$ for $j > i$.

2.2 Precision Regimes

The precision of numerical representations fundamentally constrains transformer expressivity. We distinguish three regimes:

Definition 2.2 (p-Precision Floating Point)**.** A *p-precision float* is a tuple $\langle m, e\rangle$ where the mantissa m and exponent e are signed integers using p bits total. The represented value is $m \cdot 2^{e-|m|+1}$.

Definition 2.3 (Log-Precision Transformer)**.** A transformer has *log-precision* if its numerical precision satisfies $p = O(\log n)$ for inputs of length n. This is the minimal precision sufficient to:

- Represent position indices $i \in [n]$
- Compute uniform attention weights $1/n$
- Accumulate sums over n positions without overflow

Definition 2.4 (Constant-Precision Transformer)**.** A transformer has *constant precision* if $p = O(1)$ independent of input length.

The distinction is crucial: log-precision transformers can implement counting and position-dependent operations, while constant-precision transformers cannot reliably distinguish positions in long sequences.

2.3 Attention Semantics

The attention function α admits several formalizations with different computational properties:

Definition 2.5 (Softmax Attention)**.** Standard softmax attention computes:

$$[\alpha_{\text{soft}}(s)]_i = \frac{\exp(s_i)}{\sum_{j=1}^{n} \exp(s_j)} \tag{4}$$

Definition 2.6 (Saturated (Average-Hard) Attention)**.** *Saturated attention* assigns uniform weight to all maximum-score positions:

$$[\alpha_{\text{sat}}(s)]_i = \frac{\mathbf{1}[i \in M(s)]}{|M(s)|} \tag{5}$$

where $M(s) = \{i : s_i = \max_j s_j\}$ is the set of positions achieving the maximum score.

Definition 2.7 (Leftmost-Hard Attention)**.** *Leftmost-hard attention* selects only the leftmost maximum:

$$[\alpha_{\text{left}}(s)]_i = \mathbf{1}[i = \min M(s)] \tag{6}$$

Saturated attention arises as the limit of softmax attention as temperature approaches zero when ties exist. It is the standard idealization for complexity analysis because it admits clean circuit simulations.

Theorem 2.8 (Attention Hierarchy, Merrill et al. 2022)**.** *For transformer language recognition:*

$$\textit{Hard Attention} \subsetneq \textit{Saturated Attention} \subseteq \mathbf{TC}^0 \tag{7}$$

Specifically, hard attention transformers are contained in $\mathbf{AC}^0$*, while saturated attention transformers can compute threshold functions and thus reach* $\mathbf{TC}^0$*.*

2.4 Chain-of-Thought as Computation

We now formalize chain-of-thought as an extension of the transformer computational model.

Definition 2.9 (Autoregressive Generation)**.** Given a transformer $\mathcal{T}$ and input $x \in \Sigma^n$, *autoregressive generation* proceeds:

1. Initialize context $c_0 = x$
2. For $t = 1, 2, \ldots$:
   (a) Compute $\mathcal{T}(c_{t-1})$ to obtain distribution over next token
   (b) Sample or select $y_t \in \Sigma$
   (c) Update $c_t = c_{t-1} \cdot y_t$ (concatenation)
3. Terminate when stop condition is met

Definition 2.10 (Chain-of-Thought Complexity Class)**.** For functions $T, d, s, e : \mathbb{N} \to \mathbb{N}$, define $\mathbf{CoT}[T(n), d(n), s(n), e(n)]$ as the class of languages $L \subseteq \Sigma^*$ such that there exists a constant-depth transformer with:

- Embedding dimension $d(n)$
- Mantissa precision $s(n)$ bits
- Exponent precision $e(n)$ bits

that on input $x \in \Sigma^n$, generates at most $T(n)$ tokens, with the final token indicating acceptance/rejection.

When parameters are clear from context, we write $\mathbf{CoT}(T(n))$ for the class with $T(n)$ chain-of-thought steps.

This definition captures the key insight: each generated token adds one "step" of computation, transforming the bounded-depth transformer into an unbounded-time computational model.

2.5 Architectural Variants

Several architectural details affect theoretical results:

Definition 2.11 (Pre-Norm vs. Post-Norm)**.** In *post-norm* transformers, layer normalization is applied after the residual connection:

$$v^{(\ell)} = \text{LayerNorm}(v^{(\ell-1)} + \text{Sublayer}(v^{(\ell-1)})) \tag{8}$$

In *pre-norm* transformers, normalization precedes the sublayer:

$$v^{(\ell)} = v^{(\ell-1)} + \text{Sublayer}(\text{LayerNorm}(v^{(\ell-1)})) \tag{9}$$

Definition 2.12 (Projected Pre-Norm)**.** A transformer uses *projected pre-norm* if each sublayer can apply layer normalization to a linear projection of its input:

$$\text{proj\_layer\_norm}(v; M) = \text{LayerNorm}(Mv) \tag{10}$$

for learnable projection matrices M.

The projected pre-norm condition is required for the strongest expressivity results (Theorem 3.4), as it enables the layer-norm hash construction for dynamic memory access.

3. Complexity-Theoretic Foundations

We now present the central complexity-theoretic results characterizing transformer expressivity with and without chain-of-thought. These results establish that CoT steps act as a fundamental computational resource, enabling a precise hierarchy from $\mathbf{TC}^0$ to $\mathbf{P}$.

3.1 The TC^0 Barrier Without Chain-of-Thought

The foundational negative result establishes that transformers without intermediate generation are severely limited:

Theorem 3.1 (Log-Precision Upper Bound, Merrill & Sabharwal 2023)**.** *Transformers with $O(\log n)$-precision arithmetic can be simulated by constant-depth logspace-uniform threshold circuits. Formally:*

$$\mathbf{T}[\mathit{poly}(n), \log n, \log n] \subseteq \mathbf{L}\text{-}\mathit{uniform}\ \mathbf{TC}^0 \tag{11}$$

where $\mathbf{T}[d, s, e]$ denotes transformers with embedding dimension d, s-bit mantissa, and e-bit exponent.

Proof Sketch. Each transformer layer computes a function expressible as a constant-depth circuit with threshold gates. The attention mechanism computes weighted sums (threshold-computable), and feedforward networks with ReLU activations implement piecewise-linear functions (also threshold-computable). Composing $O(1)$ layers yields a constant-depth threshold circuit. Log-space uniformity follows from the fact that circuit descriptions can be computed in log-space given the layer structure. □

This upper bound has immediate consequences for what transformers *cannot* compute:

Corollary 3.2 (Impossibility Results)**.** *Unless complexity-theoretic separations fail, log-precision transformers without CoT cannot:*

1. *Recognize all regular languages (since $\mathbf{TC}^0 \not\supseteq$ REG unless $\mathbf{TC}^0 = \mathbf{NC}^1$)*
2. *Solve directed graph connectivity ($\mathbf{NL}$-complete)*
3. *Evaluate Boolean formulas ($\mathbf{NC}^1$-complete)*
4. *Compute iterated multiplication ($\mathbf{NC}^1$-complete)*

Li et al. (2024) refine the bounds for different precision regimes:

Theorem 3.3 (Precision-Dependent Bounds, Li et al. 2024)**.** *1. $\mathbf{T}[\mathit{poly}(n), O(1), O(1)] \subseteq \mathbf{AC}^0$. Constant-precision transformers with constant exponent bits cannot compute PARITY.*
2. $\mathbf{T}[\mathit{poly}(n), \log n, O(1)] \subseteq \mathbf{TC}^0$. Log-precision with fixed-point arithmetic remains in $\mathbf{TC}^0$.

3.2 The Merrill-Sabharwal Characterization

The central positive result shows that chain-of-thought steps precisely correspond to computational time:

Theorem 3.4 (Main Complexity Characterization, Merrill & Sabharwal 2024)**.** *For any time-constructible function $t : \mathbb{N} \to \mathbb{N}$:*

$$\mathbf{TIME}(t(n)) \subseteq \mathbf{CoT}(t(n)) \subseteq \mathbf{SPACE}(t(n) + \log n) \tag{12}$$

This yields tight characterizations at key thresholds:

Table 2: Complexity Classes Achievable with Chain-of-Thought

CoT Steps	Lower Bound	Upper Bound	Notable Capabilities
0	$\mathbf{TC}^0$	$\mathbf{TC}^0$	Threshold functions only
$O(\log n)$	$\mathbf{TIME}(\log n)$	$\mathbf{L}$	Limited sequential computation
$O(n)$	$\mathbf{TIME}(n)$	$\mathbf{SPACE}(n)$	All regular languages
$O(n^k)$	$\mathbf{P}$	$\mathbf{P}$	**Exact characterization**

Corollary 3.5 (Polynomial CoT Equals P)**.** $\mathbf{CoT}(\mathit{poly}(n)) = \mathbf{P}$. *This is the first exact characterization of any transformer class in terms of standard complexity classes.*

3.3 Proof Technique: The Layer-Norm Hash

The proof of Theorem 3.4 relies on a novel construction for dynamic memory access within the attention mechanism.

Definition 3.6 (Layer-Norm Hash)**.** For $x, y \in \mathbb{R}$, define:

$$\varphi(x, y) \triangleq \text{LayerNorm}(x, y, -x, -y) \tag{13}$$

This produces a unit vector in $\mathbb{R}^4$. Define the position hash $\varphi_x \triangleq \varphi(x, 1)$.

Lemma 3.7 (Scale Invariance)**.** *For any $x \in \mathbb{R}$ and $i > 0$: $\varphi(x/i, 1/i) = \varphi_x$.*

Lemma 3.8 (Equality Detection)**.** *For any $q, k \in \mathbb{R}$: $\varphi_q \cdot \varphi_k = 1$ if and only if $q = k$.*

Proof. The layer-norm operation produces unit vectors. Two unit vectors have dot product 1 iff they are identical. The construction ensures $\varphi_q = \varphi_k$ iff $q = k$. □

These lemmas enable attention to implement *content-addressable memory*. The Turing machine simulation proceeds as follows:

1. **Tape Representation:** Store tape "diffs" at each step—each CoT token records (position, value, timestamp) for modified cells.
2. **Memory Retrieval:** To read tape position h at time t:
   - Use query $\langle \varphi_h, \mathbf{e}_1 \rangle$
   - Use keys $\langle \varphi_j, -\psi_j \rangle$ where ψ is monotonically decreasing (recency tiebreaker)
   - Attention retrieves the most recent write to position h
3. **State Transition:** The feedforward network implements the finite-state transition function.

Remark 3.9. The projected pre-norm condition (Definition 2.8) is essential—it allows computing φ on projections of the input, enabling the selective retrieval needed for tape reconstruction.

3.4 The Li et al. Circuit Simulation

An alternative proof technique establishes a complementary result through direct circuit simulation:

Theorem 3.10 (Circuit Simulation, Li et al. 2024)**.** *For any polynomial $T : \mathbb{N}^+ \to \mathbb{N}^+$:*

$$\mathbf{SIZE}[T(n)] \subseteq \mathbf{CoT}[T(n), \log n, 1] \tag{14}$$

Consequently: $\mathbf{P}/poly = \mathbf{CoT}[poly(n), \log n, 1]$.

Proof Sketch. The construction simulates one Boolean gate per CoT step. The embedding stores four components:

1. Gate type: AND, OR, NOT, TRUE, FALSE
2. Two input gate IDs (integers in $[T(n)]$)
3. Current gate ID

At each step:

1. Attention retrieves input gate values (previously computed or original inputs)
2. Feedforward network computes the gate's Boolean operation
3. The output token becomes the input for the next step

The key insight: "Writing the output token back to the next input position resets the 'depth' of the intermediate output to 0." This circumvents the constant-depth limitation by serializing computation through the autoregressive loop. □

3.5 Inherently Serial Problems

The circuit simulation theorem motivates the study of *inherently serial* problems—those requiring sequential computation and thus benefiting from CoT:

Definition 3.11 (Inherently Serial Problem)**.** A problem is *inherently serial* if it lies in $\mathbf{P}$ (or $\mathbf{NC}^1$) but is conjectured outside $\mathbf{TC}^0$.

Table 3: Inherently Serial Problems Solvable with CoT

Problem	Complexity	Required CoT
Permutation composition (S_5)	$\mathbf{NC}^1$-complete	$O(n)$
Boolean formula evaluation	$\mathbf{NC}^1$-complete	$O(n)$
Circuit value problem	$\mathbf{P}$-complete	$\text{poly}(n)$
Iterated squaring mod p	Conjectured serial	$\text{poly}(n)$

By Barrington's theorem, if $\mathbf{TC}^0 \subsetneq \mathbf{NC}^1$ (widely believed), then permutation composition over S_5 is solvable with linear CoT but *not* by any polynomial-size transformer without CoT.

3.6 Impossibility Results and Lower Bounds

Complementing the positive results, several lines of work establish fundamental limitations:

3.6.1 Communication Complexity Barriers

Sanford, Hsu, and Telgarsky (2023-2024) model self-attention as a communication protocol:

Theorem 3.12 (Induction Head Separation)**.** *One-layer transformers require exponentially larger size than two-layer transformers to implement induction heads. The proof uses communication complexity lower bounds.*

3.6.2 Sensitivity and PARITY

Theorem 3.13 (PARITY Hardness, Hahn & Rofin 2024)**.** *While PARITY $\in \mathbf{TC}^0$, parameter settings achieving high-sensitivity functions (like parity) must inhabit steep, brittle minima. Close neighbors in parameter space define much less sensitive functions, making parity-like computations difficult to learn.*

3.6.3 The Globality Barrier

Abbe et al. (NeurIPS 2024) introduce a fundamental distinction between expressivity and learnability:

Definition 3.14 (Globality Degree)**.** For distribution $\mathcal{D}$ on $\mathcal{A}^n \times \mathcal{A}$, the *globality degree* $\text{glob}(\mathcal{D})$ is the smallest k such that there exists $S \subseteq [n]$ with $|S| = k$ satisfying:

$$I(X[S], \hat{P}_X; Y) = n^{-O(1)} \tag{15}$$

where $(X, Y) \sim \mathcal{D}$ and $\hat{P}_X$ is the empirical token histogram.

Theorem 3.15 (Globality Barrier, Abbe et al. 2024)**.** *For the cycle task (distinguishing two disjoint cycles from one large cycle), trained with population gradient descent on T-regular transformers using polynomial hyperparameters and differentiable loss, the network fails to weakly learn because glob$(\text{Cycle}(n)) \geq n$.*

This establishes that high-globality problems cannot be learned even if they are expressible—a fundamental expressivity-learnability gap.

3.7 Summary of Complexity Landscape

Complexity Hierarchy for Transformers with Chain-of-Thought

$$\begin{aligned}\mathbf{TC}^0 &\subset \mathbf{CoT}(0) = \mathbf{TC}^0 \\ &\subset \mathbf{CoT}(O(\log n)) \subseteq \mathbf{L} \\ &\subset \mathbf{CoT}(O(n)) \subseteq \mathbf{SPACE}(n) \\ &\subset \mathbf{CoT}(\text{poly}(n)) = \mathbf{P} \\ &\subset \mathbf{CoT}(\exp(n)) \subseteq \mathbf{PSPACE}\end{aligned}$$

Key Insight: Each CoT step adds one unit of "time," transforming bounded-depth parallel computation into unbounded sequential computation.

Figure 1: The complexity hierarchy induced by chain-of-thought steps

4. Algorithmic Taxonomy

We now develop a formal taxonomy of inference-time compute methods, analyzing each algorithm class with respect to time complexity, space complexity, and theoretical guarantees. The taxonomy organizes methods along two axes: *sequential vs. parallel* generation, and *flat vs. tree-structured* exploration.

Table 4: Taxonomy of Inference-Time Compute Methods

Method	Structure	Selection	Time Complexity
Chain-of-Thought	Sequential	Greedy	$O(T \cdot C_{\text{fwd}})$
Self-Consistency	Parallel	Majority Vote	$O(N \cdot T \cdot C_{\text{fwd}})$
Best-of-N	Parallel	Verifier	$O(N \cdot T \cdot C_{\text{fwd}} + N \cdot C_{\text{ver}})$
Tree-of-Thought (BFS)	Tree	LLM Evaluation	$O(b^d \cdot C_{\text{fwd}} + b^d \cdot C_{\text{eval}})$
Tree-of-Thought (DFS)	Tree	LLM Evaluation	$O(b \cdot d \cdot C_{\text{fwd}} + b \cdot d \cdot C_{\text{eval}})$
MCTS (UCT)	Tree	UCB + Rollout	$O(N_{\text{iter}} \cdot (d \cdot C_{\text{fwd}} + C_{\text{rollout}}))$

Here T denotes maximum generation length, N the number of samples, b the branching factor, d the tree depth, C_{fwd} the cost of a forward pass, C_{ver} the verifier cost, and C_{eval} the LLM self-evaluation cost.

4.1 Chain-of-Thought: Sequential Deepening

Chain-of-thought (CoT) prompting, introduced by Wei et al. (2022), represents the simplest form of inference-time scaling: generate intermediate reasoning steps before producing the final answer.

Definition 4.1 (Chain-of-Thought Generation)**.** Given prompt x and transformer $\mathcal{T}$, CoT generation produces a sequence $y_1, y_2, \ldots, y_T$ where each y_t is sampled from $\mathcal{T}(\cdot|x, y_1, \ldots, y_{t-1})$. The final answer is extracted from y_T or a designated answer token.

Algorithm 1 Chain-of-Thought Generation

Require: Prompt x, transformer $\mathcal{T}$, max length T, temperature τ
$y \leftarrow \emptyset$
for $t = 1$ to T **do**
  $p \leftarrow \mathcal{T}(\cdot|x, y)$ ▷ Forward pass
  $y_t \leftarrow \text{Sample}(p, \tau)$ ▷ Temperature sampling
  $y \leftarrow y \cdot y_t$
  if $y_t = \texttt{EOS}$ **then break**
  end if
end for
return ExtractAnswer(y)

Complexity Analysis.

- **Time:** $O(T \cdot C_{\text{fwd}})$ where $C_{\text{fwd}} = O(L \cdot d^2 \cdot n)$ for transformer with L layers, dimension d, and context length n. With KV-caching, incremental generation costs $O(L \cdot d^2)$ per token.
- **Space:** $O(T \cdot d)$ for the KV-cache, plus $O(L \cdot d^2)$ for model parameters.
- **Context Window:** CoT consumes $|x| + T$ tokens of context, limiting reasoning depth.

Theoretical Guarantees. By Theorem 3.4, T CoT steps enable recognition of $\mathbf{TIME}(T)$. The practical implication: longer chains unlock harder problems, but with diminishing returns as context fills.

4.2 Self-Consistency: Parallel Sampling with Marginalization

Self-consistency (Wang et al., 2023) generates multiple independent reasoning paths and selects the answer by majority vote, leveraging the intuition that correct answers admit multiple valid derivations.

Definition 4.2 (Self-Consistency)**.** Given prompt x, sample N independent CoT traces $\{(y^{(i)}, a^{(i)})\}_{i=1}^{N}$ where $a^{(i)} = \text{ExtractAnswer}(y^{(i)})$. Return:

$$\hat{a} = \arg\max_{a \in \mathcal{A}} \sum_{i=1}^{N} \mathbf{1}[a^{(i)} = a] \tag{16}$$

Algorithm 2 Self-Consistency Decoding

Require: Prompt x, transformer $\mathcal{T}$, sample count N, temperature $\tau > 0$

votes $\leftarrow \{\}$ ▷ Answer $\rightarrow$ count mapping
for $i = 1$ to N **do** ▷ Parallelizable
  $y^{(i)} \leftarrow \text{CoT}(x, \mathcal{T}, \tau)$
  $a^{(i)} \leftarrow \text{ExtractAnswer}(y^{(i)})$
  $\text{votes}[a^{(i)}] \leftarrow \text{votes}[a^{(i)}] + 1$
end for
return $\arg\max_a \text{votes}[a]$

Complexity Analysis.

- **Time:** $O(N \cdot T \cdot C_{\text{fwd}})$ sequential, or $O(T \cdot C_{\text{fwd}})$ with N-way parallelism.
- **Space:** $O(N \cdot T \cdot d)$ to store all traces, or $O(T \cdot d)$ if answers are extracted online.

Theoretical Analysis. Let p be the probability that a single CoT trace yields the correct answer. Under independence:

Proposition 4.3 (Self-Consistency Success Probability)**.** *If each sample is correct independently with probability $p > 0.5$, then the probability that majority voting returns the correct answer satisfies:*

$$P(\textit{correct}) \geq 1 - \exp\left(-2N(p - 0.5)^2\right) \tag{17}$$

by Hoeffding's inequality. For $p > 0.5$, this converges to 1 exponentially in N.

Remark 4.4 (Failure Mode). If $p < 0.5$—i.e., the model is more likely to produce incorrect answers—majority voting *amplifies* errors. Chen et al. (2024) prove that self-consistency can converge to the *wrong* answer as $N \to \infty$ when incorrect answers have higher generation probability than correct ones.

Empirical Scaling. Wang et al. report diminishing returns beyond $N \approx 40$ samples, with most gains from $N = 1$ to $N = 5$. This suggests a ceiling imposed by the base model's coverage of valid reasoning paths.

4.3 Best-of-N with Verifiers

Best-of-N (BoN) sampling uses an external verifier (reward model) to select among candidate generations, decoupling generation quality from selection quality.

Definition 4.5 (Best-of-N Sampling)**.** Given prompt x, verifier $V : \mathcal{Y} \to \mathbb{R}$, sample N responses $\{y^{(i)}\}_{i=1}^{N}$ and return:

$$\hat{y} = \arg\max_{y \in \{y^{(1)}, \ldots, y^{(N)}\}} V(y) \tag{18}$$

Verifier Types.

- **Outcome Reward Models (ORMs):** $V(y) = r(y)$ scores the final answer only.
- **Process Reward Models (PRMs):** $V(y) = \sum_{t=1}^{T} r_t(y_{\leq t})$ aggregates step-level rewards.
- **Self-Verification:** $V(y) = \mathcal{T}(\text{"Is } y \text{ correct?"} | x, y)$ uses the LLM itself.

Complexity Analysis.

- **Time:** $O(N \cdot T \cdot C_{\text{fwd}} + N \cdot C_{\text{ver}})$. For learned verifiers, $C_{\text{ver}} \approx C_{\text{fwd}}$.
- **Space:** $O(N \cdot T)$ to store candidates for verification.

Theoretical Guarantees. Chen et al. (2024) establish:

Theorem 4.6 (BoN with Imperfect Verifier)**.** *Let p_{gen} be the probability of generating a correct solution, and let the verifier have true positive rate TPR and false positive rate FPR. Then BoN accuracy is characterized by the verifier's ROC curve:*

$$P(\textit{correct}) = \frac{p_{gen} \cdot \textit{TPR}^N}{p_{gen} \cdot \textit{TPR}^N + (1 - p_{gen}) \cdot \textit{FPR}^N} \tag{19}$$

As $N \to \infty$: if TPR $>$ FPR, accuracy $\to 1$; if TPR $<$ FPR, accuracy $\to 0$.

This reveals a critical failure mode: imperfect verifiers with high false positive rates can cause BoN to *degrade* with more samples.

4.4 Tree-of-Thoughts: Structured Search

Tree-of-Thoughts (ToT), introduced by Yao et al. (2023), generalizes CoT by exploring multiple reasoning paths organized as a tree, with LLM-based evaluation guiding search.

Definition 4.7 (Tree-of-Thoughts)**.** A ToT search maintains:

- **State space** $\mathcal{S}$: partial solutions ("thoughts")
- **Generator** $G : \mathcal{S} \to 2^{\mathcal{S}}$: proposes next thoughts
- **Evaluator** $E : \mathcal{S} \to \mathbb{R}$: scores thought quality
- **Search algorithm**: BFS, DFS, or beam search

Algorithm 3 Tree-of-Thoughts with BFS

Require: Initial state s_0, generator G, evaluator E, beam width b, depth d
frontier $\leftarrow \{s_0\}$
for $\ell = 1$ to d **do**
  candidates $\leftarrow \bigcup_{s \in \text{frontier}} G(s)$ ▷ Expand all frontier nodes
  scores $\leftarrow \{E(s) : s \in \text{candidates}\}$ ▷ Evaluate candidates
  frontier $\leftarrow$ Top-b(candidates, scores) ▷ Keep best b
end for
return $\arg\max_{s \in \text{frontier}} E(s)$

Complexity Analysis.

- **BFS Time:** $O(b \cdot d \cdot k \cdot C_{\text{gen}} + b \cdot d \cdot k \cdot C_{\text{eval}})$ where k is candidates per expansion.
- **DFS Time:** $O(d \cdot k \cdot C_{\text{gen}} + d \cdot k \cdot C_{\text{eval}})$ per path, with backtracking.
- **Space:** BFS requires $O(b \cdot d)$ states; DFS requires $O(d)$ on the stack.

Connection to Classical Search. ToT instantiates classical AI search with LLM-based heuristics:

- **A* analogy:** The evaluator E serves as an admissible heuristic if it upper-bounds solution quality.
- **Completeness:** ToT with DFS is complete (will find solution if one exists) given sufficient depth and branching.
- **Optimality:** Not guaranteed—LLM evaluators provide noisy, non-admissible heuristics.

Empirical Results. On Game of 24, ToT achieves 74% accuracy vs. 4% for standard CoT (GPT-4), demonstrating that structured search recovers capabilities inaccessible to linear generation. However, ToT requires 5-100$\times$ more tokens than CoT.

4.5 Monte Carlo Tree Search

MCTS combines tree search with stochastic sampling and bandit-based exploration, providing theoretical guarantees via the UCB framework.

Definition 4.8 (UCT for LLM Reasoning)**.** Upper Confidence bounds for Trees (UCT) selects actions by:

$$a^* = \arg\max_{a \in A(s)} \left[Q(s,a) + c\sqrt{\frac{\ln N(s)}{N(s,a)}} \right] \tag{20}$$

where $Q(s,a)$ is the empirical mean reward, $N(s)$ is visit count for state s, $N(s,a)$ is visit count for action a at s, and c is the exploration constant.

Algorithm 4 MCTS for LLM Reasoning (MCTSr)

Require: Root state s_0, LLM $\mathcal{T}$, iterations N_{iter}, exploration constant c
1: **for** $i = 1$ to N_{iter} **do**
2: **Selection:** Traverse tree using UCT until leaf s_ℓ
3: **Expansion:** Generate child states $G(s_\ell)$ using $\mathcal{T}$
4: **Evaluation:** Score new states via $\mathcal{T}$ self-evaluation or rollout
5: **Backpropagation:** Update Q and N values along path to root
6: **end for**
7: **return** $\arg\max_a Q(s_0, a)$

Theoretical Guarantees. The classical UCT theorem extends to LLM settings:

Theorem 4.9 (UCT Convergence, Kocsis & Szepesvári 2006)**.** *Consider UCT on a game tree of depth D and branching factor K with stochastic payoffs in $[0,1]$. The bias of the estimated expected payoff satisfies:*

$$\textit{Bias}(\bar{X}_n) = O\left(\frac{K^D \log n + K^D}{n}\right) \tag{21}$$

The failure probability at the root converges to zero as $n \to \infty$.

Theorem 4.10 (Regret Bound)**.** *UCB1 achieves expected regret:*

$$\mathbb{E}[R_n] \leq \sum_{i:\mu_i<\mu^*} \left(\frac{8\ln n}{\Delta_i} + \left(1 + \frac{\pi^2}{3}\right)\Delta_i\right) \tag{22}$$

where $\Delta_i = \mu^* - \mu_i$ is the gap between optimal and arm i mean rewards.

LLM-Specific Adaptations. MCTSr (Zhang et al., 2024) modifies standard MCTS for LLM reasoning:

- **Self-Refine Expansion:** Children are generated via iterative refinement, not random sampling.
- **LLM Self-Evaluation:** Rollouts replaced by LLM-based quality assessment.
- **Modified UCB:** $Q(a) + c\sqrt{\ln N(\text{parent})/N(a)} + \epsilon$ with tie-breaking constant ϵ.

MCTSr achieves GPT-4-level performance on mathematical olympiad problems using LLaMA-3 8B, demonstrating that search can substitute for model scale.

4.6 Comparative Analysis

We summarize the tradeoffs across methods:

Table 5: Comparative Analysis of Inference-Time Methods

Method	Parallel?	Backtrack?	Verifier?	Best For
CoT	No	No	No	Simple reasoning
Self-Consistency	Yes	No	No	Discrete answers
Best-of-N	Yes	No	Yes	With good verifier
ToT	Partial	Yes	Self	Multi-step planning
MCTS	No	Yes	Self/External	Complex search

When to Use Which Method.

- **CoT:** When compute is limited and problem is within model capability.
- **Self-Consistency:** When answers are discrete and model has $p > 0.5$ per-sample accuracy.
- **Best-of-N:** When a reliable verifier exists (e.g., unit tests for code, numerical checks for math).
- **ToT:** When problems require explicit planning and dead-end detection.
- **MCTS:** When optimal exploration-exploitation tradeoff is needed and compute budget is large.

5. Scaling Laws

We now examine the quantitative relationship between inference-time compute and model performance. Unlike pretraining scaling laws—which characterize how loss decreases with model size and data—inference scaling laws govern the tradeoff between test-time FLOPs and task accuracy. This section presents both empirical observations and provable theoretical guarantees.

5.1 Empirical Scaling Observations

Snell et al. (ICLR 2025) provide the first systematic study of inference-time scaling across multiple strategies. Their central finding: *optimal allocation of test-time compute can substitute for model parameters at favorable exchange rates*.

Theorem 5.1 (Compute-Optimal Inference, Snell et al. 2025)**.** *Given a fixed inference compute budget C, there exists an optimal allocation between:*

1. *Model size (parameters)*
2. *Number of samples/search iterations*
3. *Verifier compute*

such that a smaller model with sophisticated inference can match or exceed a 14× larger model with naive inference.

This result has profound implications: inference-time scaling provides a new axis for capability improvement beyond pretraining scale.

Key Empirical Findings.

- **Strategy Dependence:** The optimal inference strategy varies with problem difficulty. On easy problems, Best-of-N suffices; on hard problems, beam search or MCTS with process reward models dominates.
- **Verifier Quality:** Performance scales reliably only when verifier (PRM) quality is sufficient. With weak verifiers, scaling can plateau or even degrade.
- **Diminishing Returns:** All methods eventually saturate, but the saturation point depends on model capability and verifier quality.

Wu et al. (ICLR 2025) complement this with the REBASE algorithm, demonstrating Pareto-optimal cost-performance tradeoffs:

Proposition 5.2 (REBASE Pareto Optimality)**.** *On MATH benchmarks, Llemma-7B with REBASE tree search consistently outperforms Llemma-34B with standard majority voting across all compute budgets. Smaller models with advanced inference achieve better cost-accuracy Pareto frontiers.*

5.2 Functional Forms of Scaling Laws

Several functional forms characterize inference scaling:

Definition 5.3 (Logarithmic Scaling)**.** For many methods, accuracy improves logarithmically with compute:

$$\text{Accuracy}(C) = a + b \log C \tag{23}$$

This form arises when additional compute provides diminishing marginal improvements.

Definition 5.4 (Power-Law Scaling)**.** Some methods exhibit power-law improvement:

$$\text{Error}(N) = c \cdot N^{-\alpha} \tag{24}$$

where N is the number of samples and $\alpha > 0$ is the scaling exponent.

Definition 5.5 (Exponential Decay)**.** Under favorable conditions, error decays exponentially:

$$P(\text{error}) = e^{-\beta N} \tag{25}$$

This is the strongest possible scaling, achievable only with specific algorithmic guarantees.

The functional form depends critically on the algorithm and task structure:

Table 6: Scaling Law Functional Forms by Method

Method	Typical Form	Conditions for Exponential
Best-of-N (perfect verifier)	Exponential	$p_{\text{gen}} > 0$
Best-of-N (imperfect verifier)	Logarithmic/Plateau	TPR $>$ FPR
Self-Consistency	Exponential (if $p > 0.5$)	Modal answer is correct
Majority Voting	Power-law or Failure	$p_{\text{correct}} > p_{\text{any incorrect}}$
Knockout Tournament	Exponential	Correct beats incorrect pairwise

5.3 Provable Scaling Laws

Chen et al. (NeurIPS 2025) establish the first *provable* scaling laws for inference-time compute, providing theoretical guarantees rather than empirical fits.

5.3.1 The Knockout Algorithm

Algorithm 5 Knockout Tournament for Solution Selection

Require: LLM $\mathcal{M}$, problem x, candidates N, comparisons per pair K
Generate N candidate solutions $\{s_1, \ldots, s_N\}$ via $\mathcal{M}$
while |candidates| > 1 **do**
  Pair candidates arbitrarily
  for each pair (s_i, s_j) **do**
    Compare K times via $\mathcal{M}$: "Which solution is better?"
    Winner $\leftarrow$ majority vote over K comparisons
  end for
  candidates $\leftarrow$ winners
end while
return remaining candidate

The key assumptions for provable guarantees:

Assumption 5.6 (Generation Success)**.** The LLM generates a correct solution with probability $p_{\text{gen}} > 0$.

Assumption 5.7 (Pairwise Comparison)**.** When comparing a correct solution s^+ against an incorrect solution s^-, the LLM selects s^+ with probability $p_{\text{win}} > 0.5$.

Under these assumptions:

Theorem 5.8 (Knockout Exponential Scaling, Chen et al. 2024)**.** *If Assumptions 5.6 and 5.7 hold, and both N (candidates) and K (comparisons) scale, then the failure probability satisfies:*

$$P(\textit{incorrect output}) \leq \exp\left(-\Omega(\min\{N \cdot p_{gen}, K \cdot (p_{win} - 0.5)^2\})\right) \tag{26}$$

Theorem 5.9 (Knockout Power-Law Scaling, Chen et al. 2024)**.** *If only N scales (with fixed K), and $N = 2^m$ for integer m, then:*

$$P(\textit{incorrect output}) \leq O\left(\frac{1}{N^\gamma}\right) \tag{27}$$

for some $\gamma > 0$ depending on p_{gen} and p_{win}.

Proof Sketch. The knockout tournament has $\log_2 N$ rounds. At each round, a correct solution advances with probability at least $p_{\text{win}}^K > 0.5$ (by assumption and majority vote concentration). The probability that *some* correct solution survives all rounds is:

$$P(\text{correct survives}) \geq 1 - (1 - p_{\text{gen}} \cdot p_{\text{win}}^{\log_2 N})^N \tag{28}$$

As $N, K \to \infty$, this converges to 1 exponentially. □

5.3.2 The League Algorithm

The league-style algorithm provides more robust guarantees by evaluating candidates against multiple opponents:

Algorithm 6 League Tournament for Solution Selection

Require: LLM $\mathcal{M}$, problem x, candidates N, opponents per candidate M
1: Generate N candidate solutions $\{s_1, \ldots, s_N\}$
2: **for** each candidate s_i **do**
3: Sample M opponents uniformly from $\{s_j : j \neq i\}$
4: score$[s_i] \leftarrow$ average win rate against opponents
5: **end for**
6: **return** $\arg\max_{s_i}$ score$[s_i]$

Assumption 5.10 (Correct-and-Strong Solutions)**.** There exist "correct-and-strong" solutions s^* such that:

$$\mathbb{E}[\text{win rate of } s^* \text{ against random opponent}] > \max_{s^- \text{ incorrect}} \mathbb{E}[\text{win rate of } s^-] \tag{29}$$

with gap $\Delta > 0$.

Theorem 5.11 (League Exponential Scaling, Chen et al. 2024)**.** *Under Assumption 5.10, the league algorithm's failure probability satisfies:*

$$P(\textit{incorrect output}) \leq \exp\left(-\Omega\left(\min\left\{\frac{N}{p_{cs}^{-1}}, M \cdot \Delta^2\right\}\right)\right) \tag{30}$$

where p_{cs} is the probability of generating a correct-and-strong solution.

Comparison of Assumptions. Neither knockout nor league assumptions strictly implies the other:

- Knockout requires *every* correct solution to beat *every* incorrect solution pairwise with probability > 0.5.
- League requires only that *some* correct solutions have higher *average* win rates than all incorrect solutions.

League is more robust to occasional comparison errors but requires a stronger "best solution" property.

5.4 Failure Modes and Limitations

Not all methods enjoy provable scaling. Understanding failure modes is essential:

Majority Voting Failure.

Proposition 5.12 (Majority Voting Can Fail, Chen et al. 2024)**.** *Even if $p_{correct} = 0.45$ (model generates correct answer 45% of the time), if there exists an incorrect answer with $p_{incorrect} = 0.46$, then:*

$$\lim_{N \to \infty} P(\textit{majority vote correct}) = 0 \tag{31}$$

Majority voting converges to the wrong *answer.*

Best-of-N with Imperfect Verifier. As noted by Cobbe et al. (2021): "The benefits of search are eventually outweighed by the risk of finding adversarial solutions that fool the verifier." With high false positive rates, Best-of-N can degrade with more samples.

Compute-Accuracy Tradeoff Ceiling. All methods eventually saturate due to:

- Model capability limits (cannot generate correct solution if $p_{\text{gen}} = 0$)
- Verifier quality limits (imperfect discrimination)
- Problem hardness (some problems require capabilities the model lacks)

5.5 Compute-Optimal Allocation

Given a total compute budget C, how should it be allocated? Let:

- C_{gen}: compute for generating candidates
- C_{ver}: compute for verification/comparison
- C_{search}: compute for tree search overhead

Proposition 5.13 (Optimal Allocation Depends on Difficulty)**.** *For easy problems (high p_{gen}): allocate more to generation (many cheap samples).*

For hard problems (low p_{gen}): allocate more to search and verification (fewer, higher-quality candidates with careful selection).

Snell et al. formalize this with a difficulty-dependent optimal strategy:

$$\text{Strategy}^*(x) = \begin{cases} \text{Best-of-N} & \text{if } p_{\text{gen}}(x) > \tau_{\text{easy}} \\ \text{Beam Search + PRM} & \text{if } \tau_{\text{hard}} < p_{\text{gen}}(x) \leq \tau_{\text{easy}} \\ \text{MCTS + PRM} & \text{if } p_{\text{gen}}(x) \leq \tau_{\text{hard}} \end{cases} \tag{32}$$

The thresholds $\tau_{\text{easy}}, \tau_{\text{hard}}$ depend on model and verifier quality.

5.6 Relationship to Pretraining Scaling Laws

The Chinchilla scaling law (Hoffmann et al., 2022) characterizes compute-optimal pretraining:

$$L(N, D) = \frac{A}{N^\alpha} + \frac{B}{D^\beta} + E \tag{33}$$

where N is parameters, D is data, and L is loss.

Inference scaling adds a third dimension:

$$\text{Accuracy}(N, D, C_{\text{inf}}) = f(N, D) + g(C_{\text{inf}}) \tag{34}$$

The key insight: *inference compute and pretraining compute are partially substitutable*. A smaller model with large C_{inf} can match a larger model with small C_{inf}. This has practical implications:

- For low-volume deployment: prefer larger models (amortize training cost)
- For high-volume deployment: prefer smaller models with inference scaling (reduce per-query cost)

6. Verification Theory

The effectiveness of inference-time scaling fundamentally depends on *verification*—the ability to distinguish correct from incorrect solutions. This section examines the theoretical foundations of verification in LLM systems, the generation-verification gap, and the surprising limitations of self-correction.

6.1 The Generation-Verification Gap

A central empirical observation motivates much of verification theory:

Definition 6.1 (Generation-Verification Gap)**.** For a model $\mathcal{M}$ and task distribution $\mathcal{D}$, define:

$$p_{\text{gen}} = P_{x \sim \mathcal{D}}[\mathcal{M} \text{ generates correct solution for } x] \tag{35}$$

$$p_{\text{ver}} = P_{(x, y^+, y^-) \sim \mathcal{D}}[\mathcal{M} \text{ correctly identifies } y^+ \text{ as better than } y^-] \tag{36}$$

The *generation-verification gap* is $\Delta_{\text{GV}} = p_{\text{ver}} - p_{\text{gen}}$.

Proposition 6.2 (Prevalence of Generation-Verification Gap, Song et al. 2025)**.** *Across model families (GPT, LLaMA, Mistral) and tasks (math, code, reasoning),* $\Delta_{GV} > 0$ *consistently. Moreover, a normalized variant of the gap scales monotonically with pretraining FLOPs.*

This gap is the foundation for inference-time scaling: if verification is easier than generation, we can generate many candidates and use verification to select the best.

Complexity-Theoretic Intuition. The generation-verification gap echoes the $\mathbf{P}$ vs. $\mathbf{NP}$ distinction: verifying a solution to an $\mathbf{NP}$-complete problem is efficient (polynomial time), while finding a solution may be hard (exponential time under $\mathbf{P} \neq \mathbf{NP}$). However, for LLMs, the analogy is imperfect—LLMs are not Turing machines, and their "verification" is approximate pattern matching, not logical proof checking.

6.2 Verifier Taxonomy

Verifiers vary in quality and implementation:

Definition 6.3 (Verifier Types)**.**
- **Oracle Verifier:** Perfect accuracy. Examples: unit tests for code, formal proof checkers (Lean, Coq), ground-truth numerical answers.
- **Outcome Reward Model (ORM):** Scores final answers only. Trained on (problem, solution, correctness) triples.
- **Process Reward Model (PRM):** Scores intermediate steps. Trained on step-level annotations (e.g., OpenAI's PRM800K dataset).
- **LLM-as-Judge:** Prompts an LLM to evaluate solutions. No additional training required but inherits LLM limitations.
- **Self-Verification:** The generator LLM verifies its own outputs.

Table 7: Verifier Comparison

Verifier Type	Accuracy	Scalability	Generality	Cost
Oracle	Perfect	Limited domains	Low	Free
Human	High	Low	High	Very High
PRM	Good	High	Medium	Training cost
ORM	Medium	High	Medium	Training cost
LLM-as-Judge	Variable	High	High	Inference cost
Self-Verification	Poor	High	High	Low

6.3 Process vs. Outcome Reward Models

The choice between PRMs and ORMs has significant theoretical implications:

Definition 6.4 (Reward Model Formulations)**.** Given problem x and solution trajectory $y = (y_1, \ldots, y_T)$:

$$\text{ORM:} \quad R_{\text{out}}(x, y) = r(x, y_T) \quad \text{(final answer only)} \tag{37}$$

$$\text{PRM:} \quad R_{\text{proc}}(x, y) = \sum_{t=1}^{T} r_t(x, y_{\leq t}) \quad \text{(step-level rewards)} \tag{38}$$

Theorem 6.5 (PRM Advantage for Search, Lightman et al. 2023)**.** *For tree search algorithms, PRMs provide denser reward signals that enable:*

1. *Earlier pruning of incorrect branches*
2. *Better credit assignment for intermediate steps*
3. *More efficient exploration of solution space*

Empirically, PRM-guided Best-of-N achieves higher accuracy than ORM-guided Best-of-N at the same compute budget on MATH benchmarks.

However, PRMs require expensive step-level annotations. Recent work addresses this:

Definition 6.6 (Process Advantage Verifier, Setlur et al. 2024)**.** A *Process Advantage Verifier (PAV)* measures *progress*—the change in likelihood of reaching a correct solution before and after a step:

$$A(s_t) = V^{\pi_{\text{prover}}}(s_t) - V^{\pi_{\text{prover}}}(s_{t-1}) \tag{39}$$

where V^π is the value function under a prover policy π_{prover}.

PAVs can be trained without human step-level annotations by using Monte Carlo rollouts to estimate value functions.

6.4 Self-Correction: Theoretical Limits

A natural question: can LLMs verify and correct their own outputs without external feedback? The answer is largely negative.

Definition 6.7 (Intrinsic Self-Correction)**.** *Intrinsic self-correction* is a process where an LLM:

1. Generates initial response y_0 to input x
2. Generates critique c of y_0 using only (x, y_0)
3. Generates refined response y_1 using (x, y_0, c)

without access to ground truth, external tools, or other feedback.

Theorem 6.8 (Self-Correction Impossibility, Huang et al. ICLR 2024)**.** *For reasoning tasks (arithmetic, symbolic, commonsense):*

1. *LLMs cannot reliably improve performance through intrinsic self-correction*
2. *Performance often* degrades *after self-correction*
3. *The degradation occurs because LLMs are as likely to "correct" correct answers to incorrect ones as vice versa*

Intuition. If an LLM could reliably self-correct, why didn't it produce the correct answer initially? The self-correction paradox: the same model that made the error must now detect it, but it lacks new information to distinguish correct from incorrect responses. □

The Self-Correction Paradox. Formally, let $\mathcal{M}(x)$ denote the model's belief distribution over answers given input x. Self-correction amounts to computing $\mathcal{M}(x|y_0)$ where $y_0 \sim \mathcal{M}(x)$. Without external information, conditioning on y_0 cannot systematically improve accuracy—it merely reshuffles probability mass according to the model's (potentially flawed) beliefs.

Proposition 6.9 (When Self-Correction Works)**.** *Self-correction can improve performance when:*

1. *External feedback is available (code execution, search results, tool outputs)*
2. *The model has asymmetric capabilities (better at verification than generation)*
3. *The task allows for "easy" verification (e.g., checking if code compiles)*

6.5 Verification Complexity

We now formalize the computational complexity of verification:

Definition 6.10 (Verification Problem)**.** Given problem x and candidate solution y, the *verification problem* is to determine whether y is correct for x.

For different problem classes:

Table 8: Verification Complexity by Problem Type

Problem Type	Verification Complexity	Oracle Available?
Arithmetic	$O(1)$	Yes (numerical check)
Code Generation	$O(T)$ for T test cases	Yes (execution)
Formal Proofs	$O(\|y\|)$ proof length	Yes (proof checkers)
Open QA	Undecidable in general	No
Creative Writing	Subjective	No

The Verification Bottleneck. For many real-world tasks, verification is as hard as generation—there is no oracle verifier. This creates a fundamental bottleneck for inference-time scaling:

Proposition 6.11 (Verification Bottleneck)**.** *If no oracle verifier exists and the LLM's verification accuracy is $p_{ver} < 1$, then:*

$$\lim_{N\to\infty} \textit{Accuracy}_{\textit{Best-of-N}} \leq p_{ver} \tag{40}$$

Scaling cannot exceed verifier quality.

6.6 Ensemble Verification

Recent work addresses verifier limitations through ensembling:

Definition 6.12 (Weak Supervision for Verification, Saad-Falcon et al. 2025)**.** Given K weak verifiers $\{V_1, \ldots, V_K\}$ with unknown accuracies, *Weaver* learns to combine their outputs:

$$V_{\text{ensemble}}(y) = \sum_{k=1}^{K} w_k \cdot V_k(y) \tag{41}$$

where weights w_k are estimated via weak supervision techniques without ground-truth labels.

Theorem 6.13 (Ensemble Improvement, Saad-Falcon et al. 2025)**.** *Weighted ensembles of weak verifiers significantly outperform individual verifiers and unweighted combinations. Weaver achieves o3-mini-level accuracy using Llama 3.3 70B as generator with an ensemble of 70B-or-smaller judge models.*

6.7 Theoretical Framework for Self-Improvement

The generation-verification gap enables a formal model of self-improvement:

Definition 6.14 (Solver-Verifier Dynamics)**.** Let $U_s(t)$ and $U_v(t)$ denote solver and verifier capabilities at training iteration t. The dynamics satisfy:

$$\frac{dU_s}{dt} = \alpha(U_v - U_s) \cdot g_s(U_s) \tag{42}$$

$$\frac{dU_v}{dt} = \beta \cdot g_v(U_v) \tag{43}$$

where $\alpha, \beta > 0$ and g_s, g_v are growth functions. Self-improvement occurs when $U_v > U_s$ (positive gap).

Theorem 6.15 (Self-Improvement Limit)**.** *Under the solver-verifier dynamics:*

1. *Accuracy exhibits exponential convergence toward a limit*
2. *The limit is determined by the asymptotic verifier capability*
3. *Cross-improvement (using external verifier data) achieves higher limits than pure self-improvement*

This framework explains why self-improvement plateaus: as solver capability approaches verifier capability, the gap $U_v - U_s \to 0$, and improvement stalls.

6.8 Summary: Verification as the Limiting Factor

The theoretical analysis reveals verification as the critical bottleneck for inference-time scaling:

1. **Generation-Verification Gap:** Verification is generally easier than generation, enabling inference-time scaling.
2. **Self-Correction Limits:** Without external feedback, LLMs cannot reliably improve their outputs through intrinsic self-correction.
3. **Verifier Quality Ceiling:** Scaling benefits are bounded by verifier accuracy; imperfect verifiers can cause performance degradation.
4. **Ensemble Solutions:** Combining multiple weak verifiers can partially address individual verifier limitations.

These insights inform practical system design: for tasks without oracle verifiers, significant effort should go toward improving verification quality, not just scaling generation.

7. Search-Theoretic Foundations

Inference-time scaling can be viewed through the lens of search theory: the LLM generates candidates, and a search algorithm explores the solution space. This section formalizes this perspective, analyzing branching factors, exploration-exploitation tradeoffs, and heuristic quality.

7.1 Search Space Formalization

Definition 7.1 (LLM Search Space)**.** The search space for an LLM reasoning problem is a tuple $\mathcal{S} = \langle S, s_0, A, T, G, c \rangle$ where:

- S is the set of partial solution states
- $s_0 \in S$ is the initial state (problem statement)
- $A(s)$ is the set of actions (next tokens/thoughts) available at state s
- $T : S \times A \to S$ is the transition function
- $G \subseteq S$ is the set of goal states (correct solutions)
- $c : S \times A \to \mathbb{R}^+$ is the cost function (compute per action)

Branching Factor. The effective branching factor b determines search complexity:

$$b = \mathbb{E}_{s \sim \rho}[|A(s)|] \tag{44}$$

For token-level generation, $b = |\Sigma|$ (vocabulary size, typically 32K-128K). For thought-level generation (as in ToT), b is much smaller (typically 3-10 candidate thoughts).

7.2 Exploration-Exploitation Tradeoff

Search algorithms must balance:

- **Exploitation:** Pursuing the currently most promising path
- **Exploration:** Investigating less-explored alternatives

Definition 7.2 (UCB for LLM Search)**.** The Upper Confidence Bound (UCB) selection rule at state s chooses action a maximizing:

$$\text{UCB}(s, a) = \underbrace{\hat{Q}(s, a)}_{\text{exploitation}} + \underbrace{c\sqrt{\frac{\ln N(s)}{N(s, a)}}}_{\text{exploration}} \tag{45}$$

where $\hat{Q}(s, a)$ is the estimated value of taking action a at state s, $N(s)$ is the visit count for state s, $N(s, a)$ is the visit count for action a at state s, and c is the exploration constant.

Theorem 7.3 (UCB Regret Bound)**.** *For a K-armed bandit with rewards in $[0, 1]$, UCB achieves expected regret:*

$$\mathbb{E}[R_n] = O\left(\sqrt{Kn \ln n}\right) \tag{46}$$

This is optimal up to logarithmic factors.

In the LLM setting, each "arm" corresponds to a reasoning path, and the "reward" is the probability of reaching a correct solution.

7.3 Heuristic Quality and Admissibility

Search efficiency depends critically on heuristic quality:

Definition 7.4 (Heuristic Function)**.** A heuristic $h : S \to \mathbb{R}$ estimates the cost-to-go (or value) from state s to a goal state. In LLM search, h is typically implemented by:

- LLM self-evaluation: "Is this solution on track?"
- Process reward model: $h(s) = R_{\text{PRM}}(s)$
- Monte Carlo rollout: $h(s) = \mathbb{E}_{\pi}[\text{outcome}|s]$

Definition 7.5 (Admissibility)**.** A heuristic h is *admissible* if it never overestimates the true cost-to-go: $h(s) \leq h^*(s)$ for all s. Admissibility guarantees optimality for A* search.

Proposition 7.6 (LLM Heuristics Are Not Admissible)**.** *LLM-based heuristics (self-evaluation, PRMs) are generally not admissible—they can overestimate solution quality, leading to suboptimal search paths.*

Despite non-admissibility, LLM heuristics provide useful guidance. The key metric is *correlation* with true value, not strict admissibility.

7.4 A* Search and Its LLM Variants

A* search is the canonical best-first search algorithm:

Algorithm 7 A* Search for LLM Reasoning

Require: Initial state s_0, heuristic h, cost function g
Initialize priority queue $Q \leftarrow \{(s_0, g(s_0) + h(s_0))\}$
Initialize closed set $C \leftarrow \emptyset$
while $Q \neq \emptyset$ **do**
  $(s, f) \leftarrow$ pop minimum from Q
  if $s \in G$ **then return** s
  end if
  $C \leftarrow C \cup \{s\}$
  for each action $a \in A(s)$ **do**
    $s' \leftarrow T(s, a)$
    if $s' \notin C$ **then**
      $Q \leftarrow Q \cup \{(s', g(s') + h(s'))\}$
    end if
  end for
end while
return failure

Theorem 7.7 (A* Optimality)**.** *If heuristic h is admissible and consistent (i.e., $h(s) \leq c(s, a) + h(s')$ for all transitions), then A* finds an optimal path to a goal state.*

Connection to Tree-of-Thoughts. ToT with BFS approximates A* search where:

- The heuristic h is the LLM's self-evaluation score
- The cost g is the number of reasoning steps
- The branching factor is controlled by sampling k thoughts per step

7.5 Beam Search Analysis

Beam search is the most practical search algorithm for LLM decoding:

Definition 7.8 (Beam Search)**.** Beam search with width b maintains the top-b partial sequences by cumulative log-probability at each step:

$$\text{score}(y_{1:t}) = \sum_{i=1}^{t} \log P(y_i | y_{1:i-1}, x) \tag{47}$$

At each step, expand all b beams, generate $b \cdot |V|$ candidates, and keep the top b.

Proposition 7.9 (Beam Search Properties)**.** *1. **Non-optimality:** Beam search is not guaranteed to find the highest-probability sequence.*
*2. **Non-completeness:** The optimal sequence may be pruned early.*
*3. **Monotonicity:** With path-max enforcement, solution quality is non-decreasing in beam width.*
*4. **Convergence:** As $b \to \infty$, beam search approaches exhaustive search.*

Theorem 7.10 (Beam Search Complexity)**.** *For sequence length T, vocabulary size $|V|$, and beam width b:*

$$\textit{Time complexity:} \quad O(T \cdot b \cdot |V| \cdot \log(b \cdot |V|)) \tag{48}$$

$$\textit{Space complexity:} \quad O(b \cdot T) \tag{49}$$

The log *factor comes from maintaining the priority queue of candidates.*

Diverse Beam Search. Standard beam search often produces similar candidates. Diverse Beam Search partitions beams into G groups and penalizes within-group similarity:

$$\text{score}_g(y) = \log P(y|x) - \lambda \sum_{y' \in \text{Group}_g} \text{sim}(y, y') \tag{50}$$

This improves candidate diversity for downstream selection.

7.6 Search Algorithm Comparison

Table 9: Search Algorithm Properties for LLM Reasoning

Algorithm	Time	Space	Optimality
Greedy (CoT)	$O(d)$	$O(d)$	No guarantee
Beam Search	$O(b \cdot d \cdot \|V\|)$	$O(b \cdot d)$	No guarantee
A* (with admissible h)	$O(b^d)$ worst	$O(b^d)$	Optimal
MCTS (UCT)	Anytime	$O(N_{\text{iter}} \cdot d)$	Asymptotically

Here d is search depth, b is branching factor, and $|V|$ is vocabulary size.

7.7 Sample Complexity of Search

How many samples are needed to find a correct solution?

Theorem 7.11 (Sample Complexity for Correct Solution)**.** *If the probability of generating a correct solution in one attempt is $p > 0$, the expected number of samples to find at least one correct solution is:*

$$\mathbb{E}[N_{success}] = \frac{1}{1-(1-p)^N} \approx \frac{1}{p} \quad \textit{for small } p \tag{51}$$

For N parallel samples, $P(\textit{at least one correct}) = 1 - (1-p)^N$.

Corollary 7.12 (Scaling for Rare Solutions)**.** *When $p \ll 1$, achieving success probability $1 - \delta$ requires:*

$$N \geq \frac{\ln(1/\delta)}{\ln(1/(1-p))} \approx \frac{\ln(1/\delta)}{p} \tag{52}$$

samples. This scales as $O(1/p)$—difficult problems require proportionally more samples.

7.8 The Search-Verification Tradeoff

Given fixed compute budget C, how should it be allocated between search (generating candidates) and verification (selecting among them)?

Proposition 7.13 (Optimal Search-Verification Allocation)**.** *Let C_s be search compute and C_v be verification compute. For Best-of-N with verifier:*

$$N^* = \arg\max_N \textit{Accuracy}(N) \quad \textit{s.t.} \quad N \cdot C_{gen} + N \cdot C_{ver} \leq C \tag{53}$$

The optimal N^* depends on:

- *Generation success probability p_{gen}*
- *Verifier accuracy (TPR, FPR)*
- *Relative costs C_{gen}/C_{ver}*

When verification is cheap relative to generation (e.g., using a small reward model), generate many candidates. When verification is expensive (e.g., human evaluation), generate fewer, higher-quality candidates.

8. Training-Inference Tradeoffs

This section examines the relationship between training-time and inference-time compute, including distillation of reasoning capabilities, reinforcement learning for reasoning, and optimal compute allocation across the model lifecycle.

8.1 The Compute Allocation Problem

Given a total compute budget C_{total}, how should it be allocated between training (C_{train}) and inference (C_{inf})?

Definition 8.1 (Lifecycle Compute Optimization)**.** The optimal allocation solves:

$$\max_{C_{\text{train}}, C_{\text{inf}}} \text{Performance}(C_{\text{train}}, C_{\text{inf}}) \quad \text{s.t.} \quad C_{\text{train}} + Q \cdot C_{\text{inf}} = C_{\text{total}} \tag{54}$$

where Q is the expected number of queries over the model's lifetime.

The optimal tradeoff depends on deployment scale:

- **Low-volume deployment:** Favor larger models (amortize training cost over few queries)
- **High-volume deployment:** Favor smaller models with inference scaling (reduce per-query cost)

Theorem 8.2 (Inference-Adjusted Chinchilla, Sardana et al. 2023)**.** *When accounting for inference costs over Q queries, the compute-optimal model size N^* satisfies:*

$$N^* \propto \left(\frac{C_{total}}{1 + Q \cdot r} \right)^{0.5} \tag{55}$$

where r is the ratio of inference to training compute per token. For high-volume deployment ($Q \cdot r \gg 1$), optimal models are significantly smaller than Chinchilla-optimal.

8.2 Distillation of Reasoning

Can inference-time reasoning be distilled into a more efficient model?

Definition 8.3 (Chain-of-Thought Distillation)**.** Given a teacher model $\mathcal{T}$ that generates reasoning chains, *CoT distillation* trains a student model $\mathcal{S}$ on teacher-generated (problem, rationale, answer) triples:

$$\mathcal{L}_{\text{CoT}} = \mathbb{E}_{x \sim \mathcal{D}, (r,y) \sim \mathcal{T}(x)} \left[-\log P_{\mathcal{S}}(r, y|x) \right] \tag{56}$$

Theorem 8.4 (Structure Over Content, Li et al. 2025)**.** *When distilling reasoning from large models (e.g., DeepSeek-R1) to smaller models:*

1. *The* structure *of long CoT (reflection, backtracking, self-validation patterns) is more important than the* content *of individual steps*
2. *Introducing errors in reasoning steps (randomizing digits, removing keywords) has minimal impact ($< 5\%$ accuracy drop)*
3. *Even rationales with 100% wrong intermediate answers transfer reasoning structure effectively*

This surprising result suggests that distillation transfers *reasoning patterns* rather than *domain knowledge*.

Distillation Paradigms. Several approaches exist:

1. **Pre-thinking:** Student generates rationale before answer (standard CoT)
2. **Post-thinking:** Student generates answer first, then rationale as explanation
3. **Implicit reasoning:** Rationale is distilled into hidden states, not generated explicitly

Proposition 8.5 (Implicit CoT Distillation, Deng et al. 2024)**.** *Reasoning can be distilled "vertically" into hidden state computations rather than "horizontally" into token sequences. The student performs implicit reasoning across layers without generating intermediate tokens, achieving comparable accuracy with faster inference.*

8.3 Adaptive Distillation

Not all problems benefit equally from detailed reasoning:

Definition 8.6 (Adaptive CoT Distillation)**.** *ACoTD* (Adaptive Chain-of-Thought Distillation) classifies problems by student model difficulty and applies differentiated supervision:

$$\text{Supervision}(x) = \begin{cases} \text{Short CoT} & \text{if } x \text{ is Easy/Medium for student} \\ \text{Long detailed CoT} & \text{if } x \text{ is Hard for student} \end{cases} \tag{57}$$

Proposition 8.7 (Adaptive Distillation Improvement)**.** *ACoTD significantly outperforms uniform distillation by:*

- *Consolidating knowledge on easy problems (short CoT sufficient)*
- *Providing detailed reasoning for hard problems (long CoT needed)*
- *Improving training efficiency through adaptive sampling*

8.4 Reinforcement Learning for Reasoning

DeepSeek-R1 demonstrates that reasoning can emerge from pure RL without supervised CoT data:

Theorem 8.8 (RL-Induced Reasoning Emergence, DeepSeek 2025)**.** *Training with Group Relative Policy Optimization (GRPO) on outcome rewards induces:*

1. *Spontaneous emergence of chain-of-thought reasoning patterns*
2. *Self-verification behaviors ("let me check...")*
3. *Extended thinking ("aha moments" in reasoning traces)*
4. *Reflection and backtracking without explicit supervision*

without any supervised examples of reasoning chains.

Definition 8.9 (Group Relative Policy Optimization)**.** GRPO optimizes the policy by comparing outputs within a group:

$$\mathcal{L}_{\text{GRPO}} = -\mathbb{E}_{x,\{y_i\}_{i=1}^{G}} \left[\sum_{i=1}^{G} \frac{r(x, y_i) - \bar{r}}{\sigma_r} \log \pi_\theta(y_i|x) \right] \tag{58}$$

where $\bar{r}$ and σ_r are the mean and standard deviation of rewards within the group.

This suggests that reasoning may be a natural solution to the RL objective, not requiring explicit supervision.

8.5 Inference-Aware Training

Recent work proposes training methods that account for inference-time strategies:

Definition 8.10 (BoN-Aware Training)**.** Instead of maximizing single-sample performance, *BoN-aware training* optimizes:

$$\max_\theta \mathbb{E}_{x\sim\mathcal{D}} \left[\max_{i\in[N]} r(x, y_i) \right] \quad \text{where } y_i \sim \pi_\theta(\cdot|x) \tag{59}$$

This encourages the model to produce diverse, high-quality candidates rather than concentrating probability on a single response.

Proposition 8.11 (BoN-Aware Improvement)**.** *BoN-aware fine-tuning improves Bo32 performance on MATH benchmarks while maintaining greedy performance, by increasing response diversity without sacrificing average quality.*

8.6 The Distillation-RL Spectrum

Training approaches form a spectrum:

Table 10: Training Approaches for Reasoning

Approach	Supervision	Data Source	Emergent Reasoning
SFT on answers	Answer only	Human labels	No
CoT distillation	Rationale + answer	Teacher LLM	Transferred
RL from outcomes	Reward signal	Verifier/Oracle	Yes
RL from process	Step rewards	PRM	Yes (guided)

8.7 Theoretical Limits of Distillation

Proposition 8.12 (Distillation Capacity Bound)**.** *A student model with N_S parameters cannot perfectly distill a teacher with $N_T \gg N_S$ parameters. The approximation error satisfies:*

$$\textit{Error} \geq \Omega\left(\frac{H(\mathcal{T})}{N_S}\right) \tag{60}$$

where $H(\mathcal{T})$ is a measure of teacher complexity.

However, for specific tasks, much smaller students can match larger teachers:

Proposition 8.13 (Task-Specific Distillation)**.** *On mathematical reasoning tasks:*

- *DeepSeek-R1-Distill-Qwen-7B matches QwQ-32B-Preview*
- *Distilled models retain most reasoning capability with 4-5× fewer parameters*
- *Inference-time scaling still benefits distilled models*

The key insight: distillation transfers *task-specific reasoning patterns*, not general model capability.

9. Open Problems

We conclude with a hierarchy of open theoretical problems, ranked by tractability and potential impact. These problems represent the frontier of our understanding of inference-time compute scaling.

9.1 High Tractability, High Impact

These problems are likely solvable with current techniques and would significantly advance the field.

Open Problem 1 (Tight Complexity Bounds for Log-CoT). Determine whether $\mathbf{CoT}(O(\log n)) = \mathbf{L}$ or if strict containment holds. Current bounds: $\mathbf{TIME}(\log n) \subseteq \mathbf{CoT}(\log n) \subseteq \mathbf{L}$. A separation would clarify the power of limited sequential reasoning.

Approach: Identify a problem in $\mathbf{L}$ that requires $\omega(\log n)$ CoT steps, or prove that log-space simulation is possible with logarithmic CoT.

Open Problem 2 (Optimal Verifier Design). Given a fixed compute budget for verifier training, what architecture and training objective maximizes verification accuracy? Current PRMs are trained on human-annotated step labels; can self-supervised objectives achieve comparable quality?

Approach: Formalize the verifier design problem as a PAC learning problem; derive sample complexity bounds for different training objectives.

Open Problem 3 (Compute-Optimal Inference Allocation). Derive closed-form expressions for optimal allocation between generation, verification, and search given problem difficulty distribution and verifier quality. Current results are empirical; theoretical characterization would enable principled system design.

Approach: Model the problem as constrained optimization; use convexity analysis to derive closed-form solutions for special cases.

9.2 Medium Tractability, High Impact

These problems require new techniques but appear within reach.

Open Problem 4 (Faithfulness of Chain-of-Thought). To what extent does the generated chain-of-thought reflect the model's actual reasoning process? If CoT is merely post-hoc rationalization, its theoretical guarantees may not apply to practical systems.

Approach: Develop interpretability methods to compare internal representations with generated rationales; design experiments that distinguish faithful reasoning from confabulation.

Open Problem 5 (Positive Globality Conjecture). Abbe et al. prove that high-globality problems cannot be learned. The converse: do constant-globality problems admit efficient learning? This would characterize which problems benefit from CoT training.

Approach: Identify structural properties of low-globality problems; show that these properties enable polynomial-sample learning.

Open Problem 6 (Self-Improvement Ceiling). What is the theoretical limit of self-improvement without external data? The solver-verifier framework suggests a ceiling when $U_s \to U_v$, but precise characterization of this limit remains open.

Approach: Analyze the fixed points of the solver-verifier dynamics; characterize when the gap closes completely.

Open Problem 7 (Verifier Ensemble Theory). How do multiple weak verifiers combine? Under what conditions does an ensemble of imperfect verifiers approach oracle verification quality? Current weak supervision methods lack theoretical guarantees for the verification setting.

Approach: Extend weak supervision theory to the ranking/selection setting; derive conditions for consistent ensemble verification.

9.3 Low Tractability, High Impact

These are ambitious problems that may require fundamental new insights.

Open Problem 8 (Mechanism of RL-Induced Reasoning). DeepSeek-R1 demonstrates reasoning emergence from pure RL. What properties of the reward landscape and model architecture enable this? A mechanistic understanding could guide more efficient training.

Challenge: The emergence appears to require large scale; understanding the phase transition is difficult without expensive experiments.

Open Problem 9 (Lower Bounds for Inference Scaling). Are there problems where inference-time scaling provably cannot help—even with unlimited compute? Characterizing such problems would bound the applicability of test-time scaling.

Challenge: Proving lower bounds for adaptive computation is notoriously difficult; may require new proof techniques.

Open Problem 10 (Unified Theory of Scaling). Develop a unified framework relating pretraining scaling (Chinchilla), post-training scaling, and inference scaling. Current treatments are separate; a unified theory would enable global compute optimization across the model lifecycle.

Challenge: The three phases involve different optimization landscapes and success metrics; unification requires bridging these disparate formalisms.

9.4 Foundational Questions

These questions probe the deepest aspects of reasoning in language models.

Open Problem 11 (Expressivity vs. Learnability Bridge). The globality barrier shows expressivity $\neq$ learnability. Can we characterize the full space of problems that are both expressible (with sufficient CoT) and learnable (from polynomial data)?

Significance: This would provide a complete theory of which problems benefit from CoT, guiding both algorithm design and benchmark selection.

Open Problem 12 (Verification Complexity of Natural Language). For tasks without oracle verifiers (open QA, creative writing), what is the complexity of verification? Is there a fundamental limit to how well such tasks can be verified, and thus how much they can benefit from inference scaling?

Significance: Many practical applications lack oracle verifiers; understanding their limits would inform deployment decisions.

Open Problem 13 (Reasoning vs. Retrieval). To what extent is LLM "reasoning" genuine sequential computation versus sophisticated pattern retrieval? This question underlies whether theoretical complexity results apply to practical systems.

Significance: If LLMs primarily retrieve pre-computed patterns, complexity-theoretic analysis may be inapplicable; if they genuinely compute, the analysis is directly relevant.

9.5 Research Directions

Based on these open problems, we identify promising research directions:

Theoretical Directions.

1. Develop tighter bounds on transformer expressivity with bounded CoT
2. Characterize the sample complexity of learning different CoT patterns
3. Prove lower bounds for inference scaling on specific problem classes
4. Formalize the relationship between verifier quality and scaling limits

Empirical Directions.

1. Systematic study of CoT faithfulness across model families
2. Scaling law experiments across diverse inference strategies
3. Ablation studies on verifier components and training objectives
4. Investigation of emergent reasoning in RL-trained models

Applied Directions.

1. Develop adaptive inference allocation based on problem difficulty
2. Design efficient verification ensembles for real-world deployment
3. Build systems that combine multiple inference strategies optimally
4. Create benchmarks that probe specific theoretical predictions

10. Conclusion

This survey has presented a theoretical framework for understanding inference-time compute scaling in large language models. We conclude by synthesizing the key insights and their implications.

10.1 Summary of Theoretical Contributions

Computational Foundations. We established that transformers with $O(1)$ layers compute $\mathbf{TC}^{0}$ functions, fundamentally limiting single-pass capabilities. The addition of chain-of-thought reasoning with T intermediate steps expands expressivity to $\mathbf{CoT}(T)$, a hierarchy between $\mathbf{TC}^{0}$ and $\mathbf{P}$. This formalization provides the theoretical grounding for why inference-time scaling can unlock capabilities inaccessible to single-pass inference.

Complexity Barriers. The Merrill-Sabharwal characterization theorem precisely delineates when transformers succeed or fail: bounded intermediate values admit log-precision solutions, while unbounded growth requires extended computation. The globality barrier of Abbe et al. further constrains *learnability*—even expressible functions may be unlearnable from polynomial samples. These barriers explain fundamental limits that no amount of scaling can overcome.

Algorithmic Taxonomy. We analyzed the major inference-time algorithms—chain-of-thought, self-consistency, best-of-N, tree-of-thoughts, and MCTS—characterizing their complexity, success conditions, and failure modes. The common thread: all methods trade additional compute for improved accuracy, but their effectiveness depends critically on problem structure and verifier quality.

Scaling Laws. Empirical scaling laws reveal that optimal inference allocation can substitute for $14\times$ model parameters. Provable scaling laws establish exponential error decay under specific conditions (knockout and league algorithms), while also identifying failure modes (majority voting with wrong modal answer, best-of-N with high false positive verifiers).

Verification Theory. The generation-verification gap explains why inference scaling works: verification is systematically easier than generation. However, self-correction without external feedback is provably limited—LLMs cannot reliably improve their own outputs through intrinsic critique. Verification quality is the fundamental ceiling on scaling benefits.

10.2 Practical Implications

For practitioners deploying LLM systems:

1. **Match method to problem.** Easy problems benefit from simple Best-of-N; hard problems require structured search with strong verifiers.
2. **Invest in verification.** Verifier quality bounds scaling benefits. For tasks without oracle verifiers, ensemble methods and process reward models can partially close the gap.
3. **Don't trust self-correction.** Without external feedback (code execution, search results, formal verification), self-correction is unreliable and can degrade performance.
4. **Consider the full lifecycle.** Optimal compute allocation depends on deployment volume. High-volume applications favor smaller models with aggressive inference scaling.

5. **Recognize fundamental limits.** Some problems are provably beyond single-pass capability (high globality), while others cannot benefit from CoT (problems requiring true parallelism). Scaling cannot overcome architectural limitations.

10.3 The Road Ahead

Inference-time scaling represents a paradigm shift in how we think about LLM capabilities. Rather than viewing model capacity as fixed at training time, we can now trade compute for capability at inference time—a more flexible and often more efficient approach.

The theoretical foundations presented here are just the beginning. Key open questions remain: What are the tight complexity bounds for limited CoT? How do we design optimal verifiers? Can we develop a unified theory of scaling that encompasses pretraining, post-training, and inference?

As models continue to scale and inference-time methods mature, the interplay between theory and practice will be essential. Theory guides the design of more efficient algorithms; practice reveals the empirical phenomena that theory must explain. This survey aims to accelerate that productive cycle by providing a rigorous foundation for the rapidly evolving field of inference-time compute scaling.

A. Detailed Proofs

This appendix provides complete proofs for the main theorems stated in the survey.

A.1 Proof of the CoT Hierarchy Theorem

Proof. We prove that $\mathbf{TC}^{0^0} \subsetneq \mathbf{CoT}(O(\log n)) \subseteq \mathbf{CoT}((n)) = \mathbf{P}$ by establishing each containment.

Step 1: $\mathbf{TC}^{0^0} \subseteq \mathbf{CoT}(1)$. A constant-depth, polynomial-size threshold circuit can be simulated by a transformer in a single pass. Each layer of the circuit corresponds to a transformer layer computing threshold functions via attention aggregation. Since $\mathbf{TC}^{0^0}$ circuits have $O(1)$ depth, a constant number of transformer layers suffices.

Step 2: $\mathbf{CoT}(O(\log n)) \supsetneq \mathbf{TC}^{0^0}$. Consider the problem ITERATED-ADDITION: given n numbers $x_1, \ldots, x_n$ each of $O(\log n)$ bits, compute their sum. This requires $\Omega(\log n / \log \log n)$ depth in $\mathbf{TC}^{0^0}$ circuits (Ajtai 1983). However, a transformer with $O(\log n)$ CoT steps can compute this by iteratively adding pairs and maintaining running totals, simulating a balanced binary tree of additions.

Step 3: $\mathbf{CoT}((n)) \subseteq \mathbf{P}$. Each CoT step involves:

- Reading the previous token (constant time)
- Computing attention over $O(n + T)$ positions where T is the CoT length
- Applying FFN layers

With $T = (n)$ steps and polynomial-size attention, the total computation is polynomial in n. Since the transformer can simulate any Turing machine step-by-step (with appropriate encoding), $\mathbf{CoT}((n)) = \mathbf{P}$.

Step 4: $\mathbf{CoT}((n)) = \mathbf{P}$. The reverse direction follows from the universality of transformers: given a $\mathbf{P}$ algorithm running in time $T(n) = (n)$, we can construct a transformer that simulates it step-by-step, outputting the computation trace as the chain of thought. □

A.2 Proof of the Merrill-Sabharwal Characterization (Theorem 3.4)

Proof. We prove both directions of the characterization.

Direction 1: Bounded intermediate values ⇒ log-precision solution. Suppose problem P has a solution algorithm where all intermediate values are bounded by (n). Each such value can be represented in $O(\log n)$ bits. A transformer with $O(\log n)$-precision arithmetic can:

- Store these values in its residual stream
- Perform arithmetic operations via attention and FFN layers
- Propagate values across positions via attention

The key insight is that log-precision is sufficient when values don't grow beyond polynomial bounds.

Direction 2: Unbounded growth ⇒ extended computation required. Consider a problem where intermediate values grow as 2^n (e.g., computing 2^n exactly). With $O(\log n)$ precision, representing such values requires:

$$\frac{n}{\log n} = \omega(1) \tag{61}$$

separate "chunks" of precision. Propagating and combining these chunks requires multiple sequential operations, manifesting as CoT steps in the transformer output.

The formalization uses the concept of *precision hierarchy*: problems are stratified by the precision required for their intermediate computations, and this hierarchy corresponds to the CoT length hierarchy. □

A.3 Proof of Theorem 5.8: Knockout Exponential Scaling

Proof. Let N be the number of initial candidates and K be the number of comparisons per pair.

Step 1: Probability of correct candidate surviving one round. Consider a correct candidate c^+ facing an incorrect candidate c^-. In each comparison, c^+ wins with probability $p_{\text{win}} > 0.5$ by assumption. Over K comparisons with majority vote:

$$P(c^+ \text{ wins round}) = \sum_{k=\lceil K/2 \rceil}^{K} \binom{K}{k} p_{\text{win}}^k (1 - p_{\text{win}})^{K-k} \tag{62}$$

By Hoeffding's inequality:

$$P(c^+ \text{ loses round}) \leq \exp(-2K(p_{\text{win}} - 0.5)^2) \tag{63}$$

Step 2: Probability of correct candidate surviving all rounds. The knockout tournament has $\lceil \log_2 N \rceil$ rounds. A correct candidate must win all rounds:

$$P(c^+ \text{ survives}) \geq (1 - \exp(-2K(p_{\text{win}} - 0.5)^2))^{\log_2 N} \tag{64}$$

Step 3: Probability of at least one correct survivor. The expected number of correct candidates initially is $N \cdot p_{\text{gen}}$. Using a union bound and the analysis above:

$$P(\text{no correct survivor}) \leq (1 - p_{\text{gen}} \cdot (1 - e^{-\Omega(K)}))^N \leq e^{-\Omega(N \cdot p_{\text{gen}})} \tag{65}$$

when K is sufficiently large.

Step 4: Combining the bounds. The failure probability is bounded by:

$$P(\text{failure}) \leq \exp\left(-\Omega(\min\{N \cdot p_{\text{gen}}, K \cdot (p_{\text{win}} - 0.5)^2\})\right) \tag{66}$$

This is exponential in the minimum of the two scaling parameters. □

A.4 Proof of Proposition 4.3: Self-Consistency Success Probability

Proof. Let $p > 0.5$ be the probability that the model generates the correct answer.

Step 1: Majority vote formulation. With N samples, let $X_i = 1$ if sample i is correct. The majority vote succeeds if $\sum_{i=1}^{N} X_i > N/2$.

Step 2: Apply Hoeffding's inequality. Since X_i are i.i.d. Bernoulli(p):

$$P\left(\sum_{i=1}^{N} X_i \leq N/2\right) = P\left(\frac{1}{N}\sum_{i=1}^{N} X_i - p \leq \frac{1}{2} - p\right) \tag{67}$$

By Hoeffding:

$$P(\text{failure}) \leq \exp(-2N(p - 0.5)^2) \tag{68}$$

Step 3: Success probability. Therefore:

$$P(\text{success}) \geq 1 - \exp(-2N(p - 0.5)^2) \tag{69}$$

which converges to 1 exponentially in N when $p > 0.5$. □

A.5 Proof of Theorem 4.6: Best-of-N with Imperfect Verifier

Proof. Let p_{gen} be the probability of generating a correct solution, TPR be the true positive rate, and FPR be the false positive rate of the verifier.

Step 1: Probability that a correct solution is top-ranked. Among N samples, let $N_c \sim \text{Binomial}(N, p_{\text{gen}})$ be correct and $N_i = N - N_c$ be incorrect.

For a correct solution to be selected, it must:

1. Be verified as correct (probability TPR)
2. Have the highest score among all verified solutions

Step 2: Limiting behavior. As $N \to \infty$, we have approximately $N \cdot p_{\text{gen}}$ correct and $N \cdot (1 - p_{\text{gen}})$ incorrect solutions. The number verified as correct:

$$\text{True positives:} \quad N \cdot p_{\text{gen}} \cdot \text{TPR} \tag{70}$$

$$\text{False positives:} \quad N \cdot (1 - p_{\text{gen}}) \cdot \text{FPR} \tag{71}$$

Step 3: Selection probability. With uniform random selection among verified solutions:

$$P(\text{correct selected}) = \frac{p_{\text{gen}} \cdot \text{TPR}}{p_{\text{gen}} \cdot \text{TPR} + (1 - p_{\text{gen}}) \cdot \text{FPR}} \tag{72}$$

Step 4: Failure condition. If FPR $>$ TPR, then incorrect solutions are more likely to be verified than correct ones, and:

$$\lim_{N \to \infty} P(\text{correct selected}) = 0 \tag{73}$$

The verifier's false positives eventually dominate. □

B. Extended Complexity Analysis

B.1 Detailed CoT Complexity Bounds

Proposition B.1 (Tight Bounds for Specific Problems)**.** *1.* ***Integer Addition:*** *n-bit addition requires $\Theta(\log n)$ CoT steps for a bounded-precision transformer.*

2. ***Integer Multiplication:*** *n-bit multiplication requires $\Theta(n)$ CoT steps using grade-school algorithm simulation.*
3. ***Sorting:*** *Sorting n elements requires $\Theta(n \log n)$ CoT steps (comparison-based lower bound applies).*
4. ***Matrix Multiplication:*** *Multiplying $n \times n$ matrices requires $\Theta(n^2)$ CoT steps for naive algorithm, $\Theta(n^\omega)$ for optimal algorithms where $\omega \approx 2.373$.*

B.2 Attention Complexity

Proposition B.2 (Attention Computation Cost)**.** *For sequence length L (including CoT tokens), each transformer layer requires:*

- *$O(L^2 \cdot d)$ operations for attention (standard)*
- *$O(L \cdot d^2)$ operations for FFN*

where d is the model dimension. Total forward pass: $O(n_{layers} \cdot L^2 \cdot d + n_{layers} \cdot L \cdot d^2)$.

For very long CoT ($L \gg d$), attention dominates. For short CoT ($L \ll d$), FFN dominates.

C. Additional Algorithms

C.1 Weighted Majority with Confidence

Algorithm 8 Confidence-Weighted Self-Consistency

Require: LLM $\mathcal{M}$, problem x, samples N
for $i = 1$ to N **do**
  $(y_i, c_i) \leftarrow$ sample answer and confidence from $\mathcal{M}(x)$
end for
Group answers: $\{y\} \leftarrow$ unique answers, $S_y \leftarrow \{i : y_i = y\}$
for each unique answer y **do**
  score$[y] \leftarrow \sum_{i \in S_y} c_i$ ▷ Sum of confidences
end for
return $\arg\max_y$ score$[y]$

C.2 Iterative Refinement with Verification

Algorithm 9 Verified Iterative Refinement

Require: LLM $\mathcal{M}$, verifier V, problem x, max iterations T
$y_0 \leftarrow \mathcal{M}(x)$ ▷ Initial solution
for $t = 1$ to T **do**
  $v_t \leftarrow V(x, y_{t-1})$ ▷ Verify current solution
  if $v_t =$ correct **then**
    return y_{t-1}
  end if
  $y_t \leftarrow \mathcal{M}(x, y_{t-1}, v_t)$ ▷ Refine with feedback
end for
return y_T

D. Empirical Scaling Data

D.1 Benchmark Results Summary

Table 11: Inference Scaling Results on MATH Benchmark

Method	N=1	N=8	N=32	N=128
Greedy (GPT-4)	42.2%	—	—	—
Self-Consistency (GPT-4)	—	58.1%	63.4%	66.2%
Best-of-N + ORM	—	54.3%	60.1%	63.8%
Best-of-N + PRM	—	61.2%	68.4%	72.1%
MCTS + PRM	—	63.5%	71.2%	76.8%

Table 12: Compute-Performance Tradeoffs

Configuration	Accuracy	Relative Compute
Llama-7B, Greedy	12.3%	1×
Llama-7B, Bo32 + PRM	28.6%	32×
Llama-7B, MCTS-64	34.2%	64×
Llama-34B, Greedy	25.8%	4.9×
Llama-34B, Bo8 + PRM	38.4%	39×

Note: These results illustrate the tradeoff between model size and inference compute. A smaller model with aggressive inference scaling can match or exceed a larger model with naive inference.

E. Worked Examples

This section provides detailed worked examples illustrating the key concepts.

E.1 Example: Self-Consistency on Arithmetic

Consider the problem: "What is 23×47?"

Setup. We sample $N = 5$ responses from an LLM with temperature $T = 0.7$:

1. Response 1: "$23 \times 47 = 23 \times 50 - 23 \times 3 = 1150 - 69 = 1081$" ✓
2. Response 2: "$23 \times 47 = 20 \times 47 + 3 \times 47 = 940 + 141 = 1081$" ✓
3. Response 3: "$23 \times 47 = 1081$" (direct) ✓
4. Response 4: "$23 \times 47 = 23 \times 40 + 23 \times 7 = 920 + 161 = 1081$" ✓
5. Response 5: "$23 \times 47 = 1071$" (error) ×

Self-Consistency Vote.

- Answer 1081: 4 votes
- Answer 1071: 1 vote

Majority selects 1081 (correct).

Analysis. Even with one incorrect response (20% error rate), self-consistency recovers the correct answer. The probability of failure with N samples when $p = 0.8$ is:

$$P(\text{failure}) = P(\text{incorrect majority}) = \sum_{k>N/2} \binom{N}{k} (0.2)^k (0.8)^{N-k} \tag{74}$$

For $N = 5$: $P(\text{failure}) \approx 0.0064$ (less than 1%).

E.2 Example: Best-of-N with Verifier

Consider a code generation task where the model has $p_{\text{gen}} = 0.3$ (30% chance of correct code per attempt).

Verifier Characteristics.

- TPR = 0.9 (90% chance of accepting correct code)
- FPR = 0.1 (10% chance of accepting incorrect code)

Expected Performance with $N = 10$ Samples. Expected correct samples: $10 \times 0.3 = 3$
Expected incorrect samples: $10 \times 0.7 = 7$
Among verified as correct:

- True positives: $3 \times 0.9 = 2.7$
- False positives: $7 \times 0.1 = 0.7$

Probability of selecting correct answer:

$$P(\text{correct}) = \frac{2.7}{2.7 + 0.7} = \frac{2.7}{3.4} \approx 0.79 \tag{75}$$

This is substantially better than the base $p_{\text{gen}} = 0.3$, demonstrating the power of verification-guided selection.

E.3 Example: Knockout Tournament

Consider $N = 8$ candidates with $p_{\text{gen}} = 0.5$ and $p_{\text{win}} = 0.7$.

Tournament Structure.

```
Round 1: 8 -> 4 candidates
Round 2: 4 -> 2 candidates
Round 3: 2 -> 1 winner
```

Probability Analysis. Expected correct candidates initially: $8 \times 0.5 = 4$

For a correct candidate to win:

- Must beat opponent in each round
- If opponent is incorrect: win probability = 0.7
- If opponent is correct: win probability = 0.5 (random)

Lower bound on correct winning (assuming all matchups are against incorrect):

$$P(\text{correct wins}) \geq 1 - (1 - 0.5 \times 0.7^3)^4 \approx 0.87 \tag{76}$$

The knockout tournament amplifies the advantage of correct solutions through successive rounds.

E.4 Example: MCTS for Mathematical Reasoning

Consider solving: "Find the sum of primes less than 20."

MCTS Tree Structure.

```
Root: "Find sum of primes < 20"
+-- [Thought 1] "List primes: 2, 3, 5, 7, 11, 13, 17, 19"
|   +-- [Thought 1.1] "Sum = 2+3+5+7+11+13+17+19"
|   |   +-- [Answer] "= 77" (correct)
|   +-- [Thought 1.2] "Count: 8 primes"
|       +-- (continues exploring)
+-- [Thought 2] "Primes are numbers with exactly 2 factors"
|   +-- (explores definition-based approach)
+-- [Thought 3] "2 is the only even prime"
    +-- (explores parity-based approach)
```

UCT Selection. At each node, UCT selects the child maximizing:

$$\text{UCT}(n) = \frac{W_n}{N_n} + c\sqrt{\frac{\ln N_{\text{parent}}}{N_n}} \tag{77}$$

After several iterations:

- Thought 1 has high win rate (correct listing leads to correct sum)
- UCT balances exploiting Thought 1 with exploring Thoughts 2, 3
- Eventually, path to "77" dominates

F. Implementation Considerations

F.1 Efficient Parallel Sampling

For Best-of-N and Self-Consistency, parallel sampling is crucial:

Algorithm 10 Parallel Best-of-N Sampling

Require: LLM $\mathcal{M}$, verifier V, problem x, samples N, batch size B
candidates $\leftarrow []$
for $i = 1$ to $\lceil N/B \rceil$ **do**
  batch $\leftarrow$ parallel sample $\min(B, N - (i-1)B)$ from $\mathcal{M}(x)$
  candidates.extend(batch)
end for
return $\arg\max_{y \in \text{candidates}} V(x, y)$

Batching Considerations.

- GPU memory limits batch size
- KV-cache can be shared across samples for same prompt
- Speculative decoding can accelerate sampling

F.2 Caching Strategies for Tree Search

For ToT and MCTS, prefix caching is essential:

- **KV-Cache Sharing:** All branches from a node share the same prefix; cache once, reuse for all children.
- **Evaluation Caching:** Store verifier scores for visited nodes to avoid recomputation during backpropagation.
- **Pruning:** Remove low-scoring branches early to save memory.

F.3 Verifier Deployment

Process reward models add latency. Strategies to mitigate:

1. **Batch Verification:** Score multiple steps/candidates simultaneously
2. **Speculative Verification:** Generate ahead, verify asynchronously
3. **Distilled Verifiers:** Use smaller, faster verifier for early pruning, full verifier for final selection

A. Detailed Proofs

This appendix provides complete proofs of the main theorems stated in the body of the survey.

A.1 Proof of Theorem ?? (Merrill-Sabharwal Characterization)

Theorem (Merrill-Sabharwal Characterization, Restated)**.** *A function $f : \Sigma^* \to \{0, 1\}$ is computable by a constant-depth log-precision transformer if and only if there exists a solution g to f such that:*

1. *g depends on $O(\log n)$ bits from the input*
2. *The intermediate values in computing g remain bounded by poly(n)*

Proof. We prove both directions.

($\Rightarrow$) Transformer $\implies$ Bounded Computation. Let $\mathcal{T}$ be a constant-depth transformer with L layers and precision $p = O(\log n)$.

Step 1: Precision Constraint. Each attention weight α_{ij} is computed via softmax and represented with p bits. The number of distinct attention patterns per head is at most $2^{p \cdot n} = n^{O(n)}$, but the effective information per position is $O(p) = O(\log n)$ bits.

Step 2: Information Flow. At each layer, position i receives information from all positions via attention:

$$h_i^{(\ell+1)} = \text{FFN}\left(\sum_{j=1}^{n} \alpha_{ij}^{(\ell)} v_j^{(\ell)}\right) \tag{78}$$

Since α_{ij} has p-bit precision, the weighted sum effectively extracts $O(\log n)$ bits from the sequence per attention head.

Step 3: Depth Amplification. With $L = O(1)$ layers and H heads per layer, total information at any position is $O(L \cdot H \cdot \log n) = O(\log n)$ bits (treating L, H as constants).

Step 4: Intermediate Value Bound. FFN computations involve polynomially many additions and multiplications of p-bit numbers. By induction on layers:

$$|h_i^{(\ell)}| \leq \text{poly}(n) \cdot \max_j |x_j| \tag{79}$$

Since inputs are bounded and depth is constant, intermediate values remain $\text{poly}(n)$.

($\Leftarrow$) Bounded Computation $\implies$ Transformer. Suppose f has a solution g depending on $O(\log n)$ input bits with $\text{poly}(n)$-bounded intermediates.

Step 1: Attention Selection. Construct attention patterns that select the $O(\log n)$ relevant positions. Hard attention can select specific positions; soft attention approximates this with $O(\log n)$ precision.

Step 2: Computation Simulation. The FFN layers can simulate any polynomial-time computation on $O(\log n)$ bits of input, as FFNs are universal function approximators on bounded domains.

Step 3: Precision Sufficiency. With $\text{poly}(n)$ intermediate values and $O(\log n)$ precision, all computations can be represented exactly (up to the precision needed for correct output).

This completes the characterization. □

A.2 Proof of Theorem ?? (CoT Complexity Hierarchy)

Theorem (CoT Hierarchy, Restated)**.** *The chain-of-thought complexity classes form a strict hierarchy:*

$$\mathbf{TC}^{0^0} \subsetneq \mathbf{CoT}(O(1)) \subsetneq \mathbf{CoT}(O(\log n)) \subseteq \mathbf{CoT}(O(n)) \subseteq \mathbf{CoT}(\textit{poly}(n)) = \mathbf{P} \tag{80}$$

Proof. We prove each containment and separation.

$\mathbf{TC}^{0^0} \subseteq \mathbf{CoT}(O(1))$: With $O(1)$ CoT steps, we can simulate $O(1)$ sequential transformer passes. Since transformers compute $\mathbf{TC}^{0^0}$ in one pass, $O(1)$ passes compute $\mathbf{TC}^{0^0}$.

$\mathbf{TC}^{0^0} \subsetneq \mathbf{CoT}(O(1))$: Consider string equality: $\text{EQ}(x, y) = 1$ iff $x = y$ for $|x| = |y| = n$.

Claim: $\text{EQ} \notin \mathbf{TC}^{0^0}$.

Proof: By communication complexity lower bounds, any $\mathbf{AC}^{0^0}$ circuit for EQ requires $\Omega(n)$ wires crossing any balanced partition. Since $\mathbf{TC}^{0^0} \subseteq \mathbf{AC}^{0^0}$ for this problem, $\text{EQ} \notin \mathbf{TC}^{0^0}$.

Claim: $\text{EQ} \in \mathbf{CoT}(O(1))$.

Proof: With 2 CoT steps: (1) compute $h = \text{hash}(x)$, (2) compare h with $\text{hash}(y)$. Layer-norm provides a hash function (Theorem **??**).

$\mathbf{CoT}(O(1)) \subsetneq \mathbf{CoT}(O(\log n))$: Consider iterated composition: $f^{(k)}(x) = f(f(\cdots f(x)))$ for $k = \log n$ iterations.

$O(1)$ CoT steps cannot simulate $\log n$ sequential compositions (each step adds constant depth).

$O(\log n)$ CoT steps can simulate by writing intermediate $f^{(i)}(x)$ values.

$\mathbf{CoT}(O(\log n)) \subseteq \mathbf{L}$: Each CoT step produces $O(\log n)$ bits (bounded precision). $O(\log n)$ steps yield $O(\log^2 n)$ bits total, computable in logspace.

$\mathbf{CoT}(\textbf{poly}(n)) = \mathbf{P}$: $\subseteq$*:* $\text{poly}(n)$ CoT steps, each polynomial-time, is polynomial time total.

$\supseteq$*:* Any polynomial-time TM can be simulated: write TM configuration at each step (polynomial configurations, polynomial steps). □

A.3 Proof of Theorem ?? (Self-Consistency Success Probability)

Theorem (Self-Consistency Success, Restated)**.** *For self-consistency with N samples, if the model produces the correct answer with probability $p > 0.5$:*

$$P(\textit{majority vote correct}) \geq 1 - \exp\left(-2N(p-0.5)^2\right) \tag{81}$$

Proof. Let $X_i \in \{0, 1\}$ indicate whether sample i produces the correct answer. Then X_i are i.i.d. Bernoulli(p).

The majority vote is correct iff $\sum_{i=1}^{N} X_i > N/2$.

By Hoeffding's inequality, for any $t > 0$:

$$P\left(\frac{1}{N}\sum_{i=1}^{N} X_i - p \leq -t\right) \leq \exp(-2Nt^2) \tag{82}$$

Setting $t = p - 0.5$ (valid since $p > 0.5$):

$$P\left(\frac{1}{N}\sum_{i=1}^{N} X_i \leq 0.5\right) = P\left(\frac{1}{N}\sum_{i=1}^{N} X_i - p \leq 0.5 - p\right) \tag{83}$$

$$\leq \exp(-2N(p-0.5)^2) \tag{84}$$

Therefore:

$$P(\text{majority correct}) = P\left(\sum_{i=1}^{N} X_i > N/2\right) \geq 1 - \exp(-2N(p-0.5)^2) \tag{85}$$

□

A.4 Proof of Theorem ?? (Best-of-N with Imperfect Verifier)

Theorem (Best-of-N ROC Characterization, Restated)**.** *For Best-of-N with a verifier having true positive rate TPR and false positive rate FPR:*

$$P(\textit{correct selected}) = \frac{p_{\textit{gen}} \cdot (1 - (1 - \textit{TPR})^N)}{p_{\textit{gen}} \cdot (1 - (1 - \textit{TPR})^N) + (1 - p_{\textit{gen}}) \cdot (1 - (1 - \textit{FPR})^N)} \tag{86}$$

If FPR > TPR, then $\lim_{N\to\infty} P(\textit{correct}) = 0$.

Proof. Consider the selection process:

Step 1: Highest-Scoring Candidate. The verifier assigns scores. Best-of-N selects the candidate with the highest score.

Step 2: Probability of Correct Being Highest. Let C be the event that at least one correct solution exists and has the highest score.

For the correct solution to be selected:

- At least one correct solution must be generated (probability $1 - (1 - p_{\text{gen}})^N$)
- That correct solution must score highest

Step 3: Simplified Binary Model. Assume the verifier outputs binary accept/reject with TPR and FPR. A candidate is "accepted" if it passes the verifier.

$P(\text{correct accepted}) = \text{TPR}$ $P(\text{incorrect accepted}) = \text{FPR}$

Among accepted candidates, select uniformly. The probability that a correct candidate exists and is selected:

$$P(\text{correct selected}) = \frac{E[\#\text{ correct accepted}]}{E[\#\text{ total accepted}]} \tag{87}$$

Step 4: Expected Counts. $E[\#\text{ correct accepted}] = N \cdot p_{\text{gen}} \cdot \text{TPR}$ $E[\#\text{ incorrect accepted}] = N \cdot (1 - p_{\text{gen}}) \cdot \text{FPR}$

Step 5: Selection Probability.

$$P(\text{correct}) \approx \frac{p_{\text{gen}} \cdot \text{TPR}}{p_{\text{gen}} \cdot \text{TPR} + (1 - p_{\text{gen}}) \cdot \text{FPR}} \tag{88}$$

Step 6: Limit Behavior. As $N \to \infty$:

- If TPR $>$ FPR: correct solutions are more likely to be accepted, $P(\text{correct}) \to 1$
- If TPR $<$ FPR: incorrect solutions are more likely to be accepted, $P(\text{correct}) \to 0$
- If TPR $=$ FPR: no discrimination, $P(\text{correct}) = p_{\text{gen}}$

The exact formula in the theorem accounts for the probability that at least one solution of each type is accepted. □

A.5 Proof of Theorem 5.8 (Knockout Exponential Scaling)

Theorem (Knockout Scaling, Restated)**.** *Under Assumptions 5.6 and 5.7, the knockout algorithm's failure probability satisfies:*

$$P(\textit{incorrect}) \leq \exp\left(-\Omega(\min\{N \cdot p_{\textit{gen}}, K \cdot (p_{\textit{win}} - 0.5)^2\})\right) \tag{89}$$

Proof. The proof proceeds in two parts: (1) a correct solution must exist among candidates, (2) it must survive all rounds.

Part 1: Existence of Correct Candidate. The probability that no correct solution exists among N candidates:

$$P(\text{no correct}) = (1 - p_{\text{gen}})^N \leq \exp(-N \cdot p_{\text{gen}}) \tag{90}$$

by the inequality $1 - x \leq e^{-x}$.

Part 2: Survival Through Tournament. Consider a correct solution s^+ facing an incorrect solution s^- in a round. Each of K comparisons independently favors s^+ with probability $p_{\text{win}} > 0.5$.

By majority vote over K comparisons, s^+ wins if $> K/2$ comparisons favor it. By Hoeffding:

$$P(s^+ \text{ loses}) \leq \exp(-2K(p_{\text{win}} - 0.5)^2) \tag{91}$$

Part 3: Rounds in Tournament. A knockout tournament with N candidates has $\lceil \log_2 N \rceil$ rounds. A correct solution must win all rounds it participates in.

By union bound over $\log_2 N$ rounds:

$$P(s^+ \text{ eliminated}) \leq \log_2 N \cdot \exp(-2K(p_{\text{win}} - 0.5)^2) \tag{92}$$

Part 4: Combined Bound. The algorithm fails if either:

1. No correct solution exists, OR
2. All correct solutions are eliminated

Using union bound and the above:

$$P(\text{fail}) \leq \exp(-N \cdot p_{\text{gen}}) + \log_2 N \cdot \exp(-2K(p_{\text{win}} - 0.5)^2) \tag{93}$$

$$\leq \exp\left(-\Omega(\min\{N \cdot p_{\text{gen}}, K \cdot (p_{\text{win}} - 0.5)^2\})\right) \tag{94}$$

for sufficiently large N and K. □

A.6 Proof of Theorem 6.8 (Self-Correction Impossibility)

Theorem (Self-Correction Limits, Restated)**.** *For reasoning tasks, LLMs cannot reliably improve via intrinsic self-correction, and performance can degrade.*

Proof. We provide a formal argument for why intrinsic self-correction cannot systematically improve performance.

Setup. Let $\mathcal{M}$ be an LLM with distribution $P_{\mathcal{M}}(y|x)$ over answers y given problem x. Let y^* be the correct answer.

Initial Response. The initial response $y_0 \sim P_{\mathcal{M}}(y|x)$ is correct with probability $p = P_{\mathcal{M}}(y^*|x)$.

Self-Correction Step. Intrinsic self-correction generates a critique c and revised answer y_1:

$$c \sim P_{\mathcal{M}}(c|x, y_0) \tag{95}$$

$$y_1 \sim P_{\mathcal{M}}(y|x, y_0, c) \tag{96}$$

Information-Theoretic Argument. The model has access only to (x, y_0, c) when generating y_1. Crucially:

- x is the same as before
- y_0 was generated by the same model
- c is generated by the same model from (x, y_0)

No new information about y^* is introduced. The model cannot distinguish whether $y_0 = y^*$ or $y_0 \neq y^*$ better than its original uncertainty.

Symmetry Argument. Let A be the event "y_0 is correct" and B be the event "y_1 is correct."
If the model could improve, we'd need $P(B|A^c) > P(A)$ (corrections fix errors).
But by symmetry of the model's uncertainty:

$$P(\text{model changes correct } y_0 \text{ to incorrect } y_1) \approx P(\text{model changes incorrect } y_0 \text{ to correct } y_1) \tag{97}$$

The model is equally likely to "correct" a right answer to wrong as vice versa.

Degradation Mechanism. In practice, self-correction can degrade performance because:

1. The critique prompt biases the model toward changing its answer
2. If the original answer was correct, any change is wrong
3. The model may be overconfident in its critique

Formal Condition for Improvement. Self-correction improves iff:

$$P(y_1 = y^*|y_0 \neq y^*) \cdot P(y_0 \neq y^*) > P(y_1 \neq y^*|y_0 = y^*) \cdot P(y_0 = y^*) \tag{98}$$

This requires asymmetric error detection, which the model lacks without external feedback. □

B. Extended Algorithm Descriptions

B.1 Complete MCTS for LLM Reasoning

Algorithm 11 Complete MCTS with LLM (MCTSr)

Require: LLM $\mathcal{M}$, PRM R, problem x, iterations N_{iter}, exploration constant c, rollout depth $d_{\max}$

Initialize root node $s_0 \leftarrow x$
Children$(s_0) \leftarrow \emptyset$, $N(s_0) \leftarrow 0$, $Q(s_0) \leftarrow 0$
for $i = 1$ to N_{iter} **do**
    // Selection
    $s \leftarrow s_0$
    while s is not terminal and s is fully expanded **do**
        $s \leftarrow \arg\max_{s' \in \text{Children}(s)} \left[\frac{Q(s')}{N(s')} + c\sqrt{\frac{\ln N(s)}{N(s')}} \right]$
    end while

    // Expansion
    if s is not terminal and s is not fully expanded **then**
        Generate new thought $t \sim \mathcal{M}(\cdot|s)$
        $s' \leftarrow \text{Append}(s, t)$
        Add s' to Children(s)
        $N(s') \leftarrow 0$, $Q(s') \leftarrow 0$
        $s \leftarrow s'$
    end if

    // Evaluation (Rollout or Value Estimate)
    if using rollouts **then**
        Complete solution $y \sim \mathcal{M}(\cdot|s)$ greedily up to $d_{\max}$ steps
        $v \leftarrow R(x, y)$ ▷ PRM score of complete solution
    else
        $v \leftarrow R(x, s)$ ▷ PRM score of partial solution
    end if

    // Backpropagation
    while $s \neq$ null **do**
        $N(s) \leftarrow N(s) + 1$
        $Q(s) \leftarrow Q(s) + v$
        $s \leftarrow \text{Parent}(s)$
    end while
end for

// Final Selection
$s^* \leftarrow \arg\max_{s \in \text{Children}(s_0)} N(s)$ ▷ Most visited child
return path from s_0 to terminal via highest-N children

B.2 Diverse Beam Search

Algorithm 12 Diverse Beam Search for LLM Generation

Require: LLM $\mathcal{M}$, prompt x, beam width b, groups G, diversity penalty λ, max length T

Initialize $B_g \leftarrow \{x\}$ for each group $g \in [G]$ ▷ b/G beams per group
for $t = 1$ to T **do**
  for $g = 1$ to G **do**
    Candidates$_g \leftarrow \emptyset$
    for each beam $y \in B_g$ **do**
      for each token w in top-k by $P_{\mathcal{M}}(w|y)$ **do**
        $y' \leftarrow \text{Append}(y, w)$
        $\text{score}(y') \leftarrow \log P_{\mathcal{M}}(y') - \lambda \sum_{y'' \in B_g} \text{sim}(y', y'')$
        Add $(y', \text{score}(y'))$ to Candidates$_g$
      end for
    end for
    $B_g \leftarrow$ top b/G candidates from Candidates$_g$ by score
  end for
end for
return $\bigcup_{g=1}^{G} B_g$

C. Complexity Class Definitions

For completeness, we provide formal definitions of complexity classes used in this survey.

Definition C.1 ($\mathbf{TC}^0$)**.** $\mathbf{TC}^0$ is the class of languages decidable by constant-depth, polynomial-size circuits with AND, OR, NOT, and MAJORITY gates. Equivalently, $\mathbf{TC}^0$ captures problems solvable by threshold circuits of constant depth.

Definition C.2 ($\mathbf{NC}^1$)**.** $\mathbf{NC}^1$ is the class of languages decidable by $O(\log n)$-depth, polynomial-size circuits with AND, OR, NOT gates of fan-in 2. $\mathbf{NC}^1$ contains problems like formula evaluation and is contained in $\mathbf{L}$.

Definition C.3 ($\mathbf{L}$ (Log-Space))**.** $\mathbf{L}$ is the class of languages decidable by a deterministic Turing machine using $O(\log n)$ bits of work tape (in addition to read-only input and write-only output).

Definition C.4 ($\mathbf{P}$ (Polynomial Time))**.** $\mathbf{P}$ is the class of languages decidable by a deterministic Turing machine in polynomial time.

Definition C.5 ($\mathbf{NP}$ (Nondeterministic Polynomial Time))**.** $\mathbf{NP}$ is the class of languages for which membership can be verified in polynomial time given a polynomial-size certificate.

Definition C.6 ($\mathbf{PSPACE}$)**.** $\mathbf{PSPACE}$ is the class of languages decidable by a deterministic Turing machine using polynomial space.

The standard inclusions are:

$$\mathbf{TC}^0 \subseteq \mathbf{NC}^1 \subseteq \mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \tag{99}$$

All inclusions except $\mathbf{L} \subseteq \mathbf{P}$ are believed to be strict, but separations remain open.

Acknowledgments

This survey benefited from the theoretical foundations established by Will Merrill, Ashish Sabharwal, Zhiyuan Li, Emmanuel Abbe, and their collaborators. Their rigorous complexity-theoretic analysis of transformers provides the backbone for understanding inference-time compute scaling.

References

[1] W. Merrill and A. Sabharwal. The Expressive Power of Transformers with Chain of Thought. In *Proceedings of ICLR*, 2024.

[2] Z. Li, H. Hong, S. Du, and J. Lee. Chain of Thought Empowers Transformers to Solve Inherently Serial Problems. In *Proceedings of ICLR*, 2024.

[3] W. Merrill and A. Sabharwal. The Parallelism Tradeoff: Limitations of Log-Precision Transformers. *Transactions of the Association for Computational Linguistics*, 2023.

[4] E. Abbe, S. Bengio, A. Lotfi, and K. Rizk. How Far Can Transformers Reason? The Globality Barrier and Inductive Scratchpad. In *Proceedings of NeurIPS*, 2024.

[5] M. Hahn. Theoretical Limitations of Self-Attention in Neural Sequence Models. *Transactions of the Association for Computational Linguistics*, 2020.

[6] J. Pérez, J. Barceló, and J. Marinkovic. Attention is Turing Complete. *Journal of Machine Learning Research*, 22(75):1–35, 2021.

[7] D. Angluin, D. Chiang, and A. Yang. Masked Hard-Attention Transformers and Boolean RASP Recognize Exactly the Star-Free Languages. In *Proceedings of ACL*, 2024.

[8] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Proceedings of NeurIPS*, 2022.

[9] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large Language Models are Zero-Shot Reasoners. In *Proceedings of NeurIPS*, 2022.

[10] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *Proceedings of ICLR*, 2023.

[11] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *Proceedings of NeurIPS*, 2023.

[12] M. Besta, N. Blach, A. Kubicek, R. Gerstenberger, M. Podstawski, L. Gianinazzi, J. Gajda, T. Lehmann, H. Niewiadomski, P. Nyczyk, and T. Hoefler. Graph of Thoughts: Solving Elaborate Problems with Large Language Models. In *Proceedings of AAAI*, 2024.

[13] C. Snell, J. Lee, K. Xu, and A. Kumar. Scaling LLM Test-Time Compute Optimally Can Be More Effective than Scaling Parameters. In *Proceedings of ICLR*, 2025.

[14] Y. Wu, Z. Sun, S. Li, S. Welleck, and Y. Yang. Inference Scaling Laws: An Empirical Analysis of Compute-Optimal Inference for Problem-Solving with Language Models. In *Proceedings of ICLR*, 2025.

[15] Y. Chen, X. Pan, Y. Li, B. Ding, and J. Zhou. Provable Scaling Laws for the Test-Time Compute of Large Language Models. In *Proceedings of NeurIPS*, 2025.

[16] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, et al. Training Compute-Optimal Large Language Models. In *Proceedings of NeurIPS*, 2022.

[17] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling Laws for Neural Language Models. *arXiv preprint arXiv:2001.08361*, 2020.

[18] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*, 2021.

[19] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe. Let's Verify Step by Step. In *Proceedings of ICLR*, 2024.

[20] A. Setlur, et al. Rewarding Progress: Scaling Automated Process Verifiers for LLM Reasoning. In *Proceedings of ICLR*, 2025.

[21] A. Saad-Falcon, et al. Shrinking the Generation-Verification Gap with Weak Verifiers. *arXiv preprint arXiv:2506.18203*, 2025.

[22] J. Huang, X. Chen, S. Mishra, H. S. Zheng, A. W. Yu, X. Song, and D. Zhou. Large Language Models Cannot Self-Correct Reasoning Yet. In *Proceedings of ICLR*, 2024.

[23] Y. Song, H. Zhang, C. Eisenach, S. Kakade, D. Foster, and U. Ghai. Mind the Gap: Examining the Self-Improvement Capabilities of Large Language Models. In *Proceedings of ICLR*, 2025.

[24] Z. Gou, Z. Shao, Y. Gong, Y. Shen, Y. Yang, N. Duan, and W. Chen. CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing. In *Proceedings of ICLR*, 2024.

[25] R. Kamoi, et al. When Can LLMs Actually Correct Their Own Mistakes? A Critical Survey of Self-Correction of LLMs. *Transactions of the Association for Computational Linguistics*, 2024.

[26] L. Kocsis and C. Szepesvári. Bandit Based Monte-Carlo Planning. In *Proceedings of ECML*, 2006.

[27] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529(7587):484–489, 2016.

[28] D. Zhang, S. Wu, Y. Peng, and K. Q. Weinberger. Accessing GPT-4 Level Mathematical Olympiad Solutions via Monte Carlo Tree Self-Refine with LLaMA-3 8B. *arXiv preprint arXiv:2406.07394*, 2024.

[29] S. Hao, Y. Gu, H. Ma, J. J. Hong, Z. Wang, D. Z. Wang, and Z. Hu. Reasoning with Language Model is Planning with World Model. In *Proceedings of EMNLP*, 2023.

[30] DeepSeek-AI. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948*, 2025.

[31] OpenAI. Learning to Reason with LLMs. OpenAI Blog, 2024.

[32] Qwen Team. QwQ: Reflect Deeply on the Boundaries of the Unknown. Qwen Blog, 2024.

[33] G. Hinton, O. Vinyals, and J. Dean. Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531*, 2015.

[34] Y. Li, et al. LLMs Can Easily Learn to Reason from Demonstration. *arXiv preprint*, 2025.

[35] Y. Deng, et al. Implicit Chain of Thought Reasoning via Knowledge Distillation. In *Proceedings of ICLR*, 2024.

[36] N. Ho, L. Schmid, and S. Yun. Large Language Models Are Reasoning Teachers. In *Proceedings of ACL*, 2023.

[37] L. C. Magister, J. Mallinson, J. Adamek, E. Malmi, and A. Severyn. Teaching Small Language Models to Reason. In *Proceedings of ACL*, 2023.

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention Is All You Need. In *Proceedings of NeurIPS*, 2017.

[39] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language Models are Few-Shot Learners. In *Proceedings of NeurIPS*, 2020.

[40] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-Time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2):235–256, 2002.

[41] W. Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[42] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, et al. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*, 2021.

[43] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring Mathematical Problem Solving with the MATH Dataset. In *Proceedings of NeurIPS*, 2021.

[44] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374*, 2021.

[45] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Proceedings of NeurIPS*, 2023.

[46] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, et al. Self-Refine: Iterative Refinement with Self-Feedback. In *Proceedings of NeurIPS*, 2023.

[47] S. Welleck, X. Lu, P. West, F. Brahman, T. Shen, D. Khashabi, and Y. Choi. Generating Sequences by Learning to Self-Correct. In *Proceedings of ICLR*, 2023.

[48] E. Zelikman, Y. Wu, J. Mu, and N. Goodman. STaR: Bootstrapping Reasoning With Reasoning. In *Proceedings of NeurIPS*, 2022.

[49] S. Polu, J. M. Han, K. Zheng, M. Baksys, I. Babuschkin, and I. Sutskever. Formal Mathematics Statement Curriculum Learning. In *Proceedings of ICLR*, 2023.

[50] N. Sardana, J. Frankle, and S. Bubeck. Beyond Chinchilla-Optimal: Accounting for Inference in Language Model Scaling Laws. *arXiv preprint arXiv:2401.00448*, 2023.

[51] G. Feng, B. Zhang, Y. Gu, H. Ye, D. He, and L. Wang. Towards Revealing the Mystery behind Chain of Thought: A Theoretical Perspective. In *Proceedings of NeurIPS*, 2023.

[52] C. Sanford, D. Hsu, and M. Telgarsky. Transformers, Parallel Computation, and Logarithmic Depth. In *Proceedings of ICML*, 2024.

[53] L. Strobl, W. Merrill, G. Weiss, D. Chiang, and D. Angluin. What Formal Languages Can Transformers Express? A Survey. *Transactions of the Association for Computational Linguistics*, 2024.

[54] C. Yun, S. Bhojanapalli, A. S. Rawat, S. J. Reddi, and S. Kumar. Are Transformers Universal Approximators of Sequence-to-Sequence Functions? In *Proceedings of ICLR*, 2020.

[55] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, et al. Emergent Abilities of Large Language Models. *Transactions on Machine Learning Research*, 2022.

[56] R. Schaeffer, B. Miranda, and S. Koyejo. Are Emergent Abilities of Large Language Models a Mirage? In *Proceedings of NeurIPS*, 2023.

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

[58] R. Coulom. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Proceedings of Computers and Games*, 2006.

[59] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[60] E. Jones. Scaling Scaling Laws with Board Games. *arXiv preprint arXiv:2104.03113*, 2021.

[61] J. Uesato, N. Kushman, R. Kumar, F. Song, N. Siegel, L. Wang, A. Creswell, G. Irving, and I. Higgins. Solving Math Word Problems with Process- and Outcome-Based Feedback. *arXiv preprint arXiv:2211.14275*, 2022.

[62] P. Wang, L. Li, Z. Shao, R. Xu, D. Dai, Y. Li, D. Chen, Y. Wu, and Z. Sui. Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations. In *Proceedings of ACL*, 2024.

[63] L. Luo, Z. Liu, Z. Huang, B. Zhang, J. Ma, and Y. Tian. Improve Mathematical Reasoning in Language Models by Automated Process Supervision. *arXiv preprint arXiv:2406.06592*, 2024.

[64] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training Language Models to Follow Instructions with Human Feedback. In *Proceedings of NeurIPS*, 2022.

[65] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In *Proceedings of NeurIPS*, 2023.

[66] A. Havrilla, Y. Du, S. C. Raparthy, C. Nalmpantis, J. Dwivedi-Yu, M. Zhuravinskyi, E. Hambro, S. Sukhbaatar, and R. Raileanu. Teaching Large Language Models to Reason with Reinforcement Learning. *arXiv preprint arXiv:2403.04642*, 2024.

[67] A. Kumar, V. Zhuang, R. Agarwal, Y. Su, J. Flick, D. Zhou, G. Gong, A. Dubey, et al. Training Language Models to Self-Correct via Reinforcement Learning. *arXiv preprint arXiv:2409.12917*, 2024.

[68] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. The Curious Case of Neural Text Degeneration. In *Proceedings of ICLR*, 2020.

[69] C. Meister, T. Vieira, and R. Cotterell. If Beam Search is the Answer, What was the Question? In *Proceedings of EMNLP*, 2020.

[70] A. K. Vijayakumar, M. Cogswell, R. R. Selvaraju, Q. Sun, S. Lee, D. Crandall, and D. Batra. Diverse Beam Search: Decoding Diverse Solutions from Neural Sequence Models. In *Proceedings of AAAI*, 2018.

[71] Y. Xie, K. Kawaguchi, Y. Zhao, J. Zhao, M. Kan, J. He, and Q. Xie. Self-Evaluation Guided Beam Search for Reasoning. In *Proceedings of NeurIPS*, 2024.

[72] A. Lewkowycz, A. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. Ramasesh, A. Slone, C. Anil, I. Schlag, T. Gutman-Solo, et al. Solving Quantitative Reasoning Problems with Language Models. In *Proceedings of NeurIPS*, 2022.

[73] Z. Azerbayev, H. Schoelkopf, K. Paster, M. Dos Santos, S. McAleer, A. Q. Jiang, J. Deng, S. Biderman, and S. Welleck. Llemma: An Open Language Model for Mathematics. In *Proceedings of ICLR*, 2024.

[74] T. Trinh, Y. Wu, Q. Le, H. He, and T. Luong. Solving Olympiad Geometry without Human Demonstrations. *Nature*, 625(7995):476–482, 2024.

[75] B. Gao, et al. OmniMATH: A Universal Olympiad Level Mathematic Benchmark For Large Language Models. *arXiv preprint arXiv:2402.01118*, 2024.

[76] X. Chen, M. Lin, N. Schärli, and D. Zhou. Teaching Large Language Models to Self-Debug. In *Proceedings of ICLR*, 2024.

[77] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago, et al. Competition-Level Code Generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022.

[78] T. Ridnik, et al. Code Generation with AlphaCodium: From Prompt Engineering to Flow Engineering. *arXiv preprint arXiv:2401.08500*, 2024.

[79] K. Yang, A. Swope, A. Gu, R. Chalapathy, P. Song, S. Zhai, S. Liber, A. Anand, et al. LeanDojo: Theorem Proving with Retrieval-Augmented Language Models. In *Proceedings of NeurIPS*, 2024.

[80] T. Trinh and T. Luong. AlphaGeometry: Solving Olympiad Geometry Without Human Demonstrations. *Nature*, 2024.

[81] T. Lattimore and C. Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020.

[82] S. Bubeck and N. Cesa-Bianchi. Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems. *Foundations and Trends in Machine Learning*, 5(1):1–122, 2012.