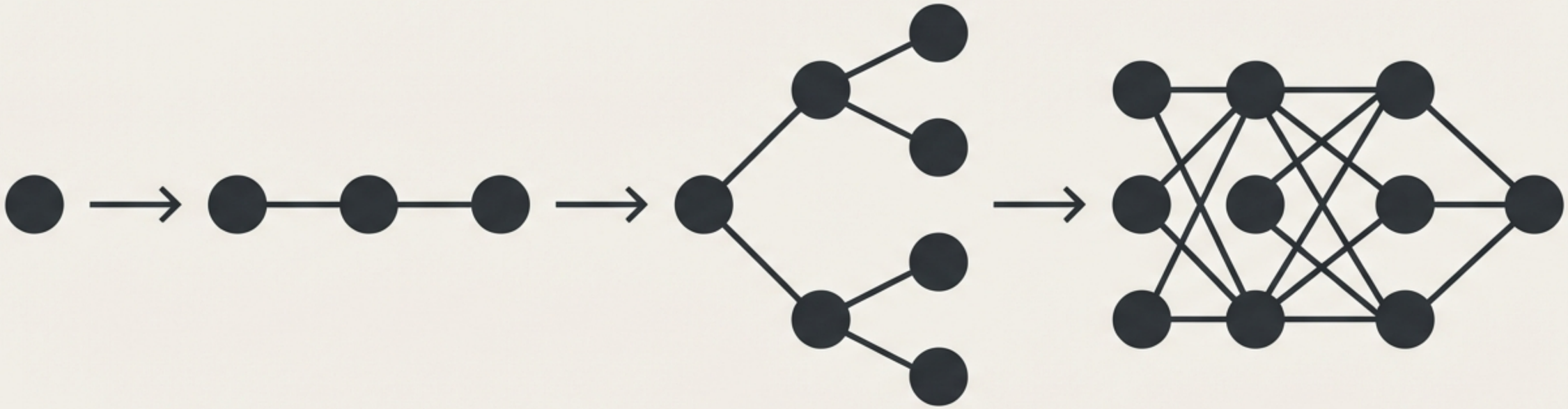
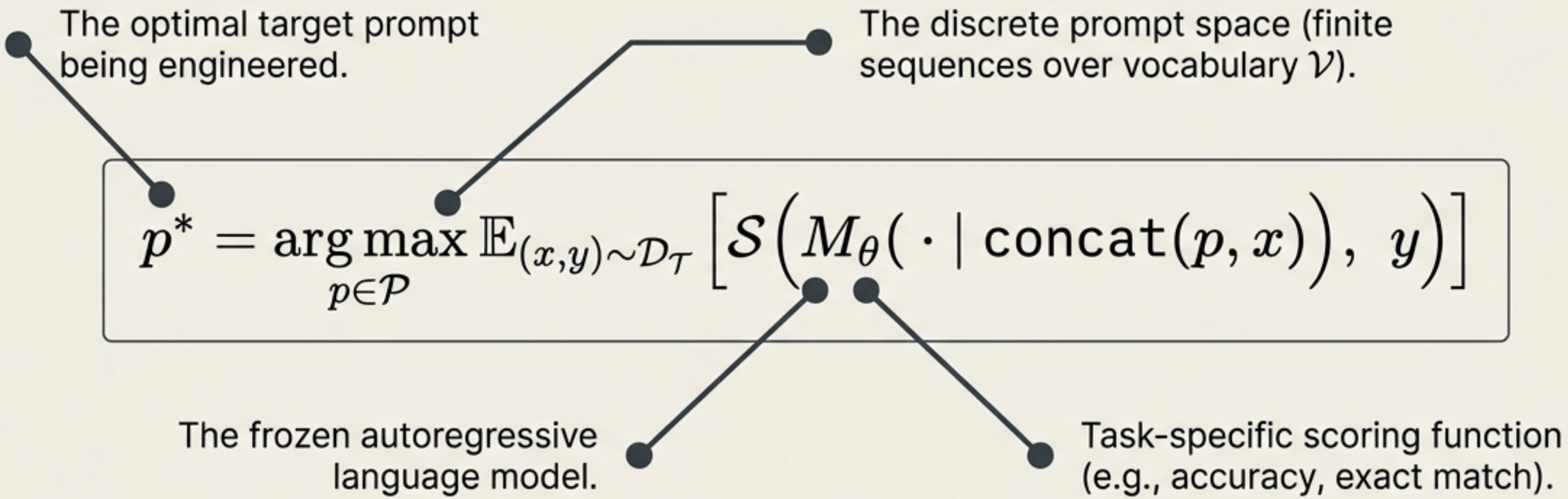


Prompt engineering optimizes discrete input sequences, not model parameters



Prompting is fundamentally an interface to latent knowledge, systematically searching the input space rather than altering the model's weight matrix.

Prompts act as soft constraints prone to non-Lipschitz instability

$$p^* = \arg\min_{p} D_{\mathrm{KL}}\left(P_{\text{target}} \,\|\, P_\theta(Y \mid p, X)\right)$$

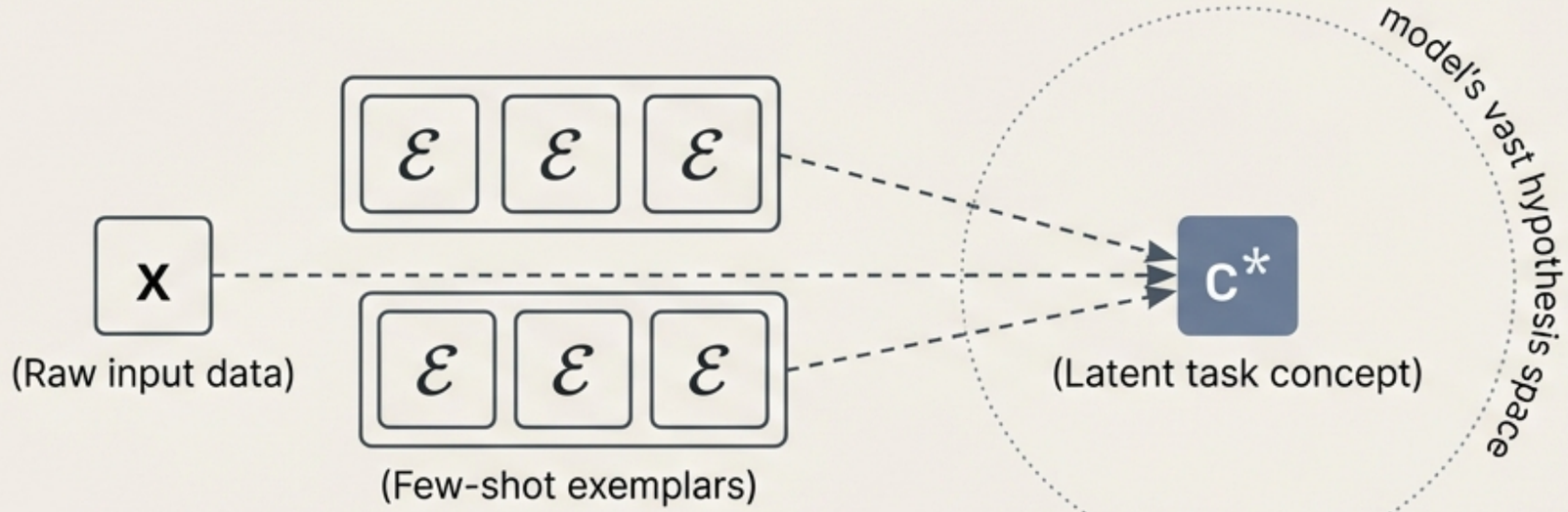
Prompts reshape the conditional output distribution, imposing soft syntactic and semantic constraints.

The Non-Lipschitz Problem

$$\|p_1 - p_2\|_{\text{edit}} = \epsilon \quad \not\Rightarrow \quad |\mathcal{S}_1 - \mathcal{S}_2| < \epsilon'$$

A fundamental mathematical challenge in discrete space. Minor surface-level perturbations (small edit distances) can trigger arbitrarily massive performance swings.

In-context learning executes implicit Bayesian inference



Latent task concept variable.

In-context exemplars.

$$P_\theta(\mathbf{y} \mid \mathbf{x}, \mathcal{E}) \approx \int P(\mathbf{y} \mid \mathbf{x}, \mathbf{c})\, P(\mathbf{c} \mid \mathcal{E})\, d\mathbf{c}$$

Latent task concept variable.

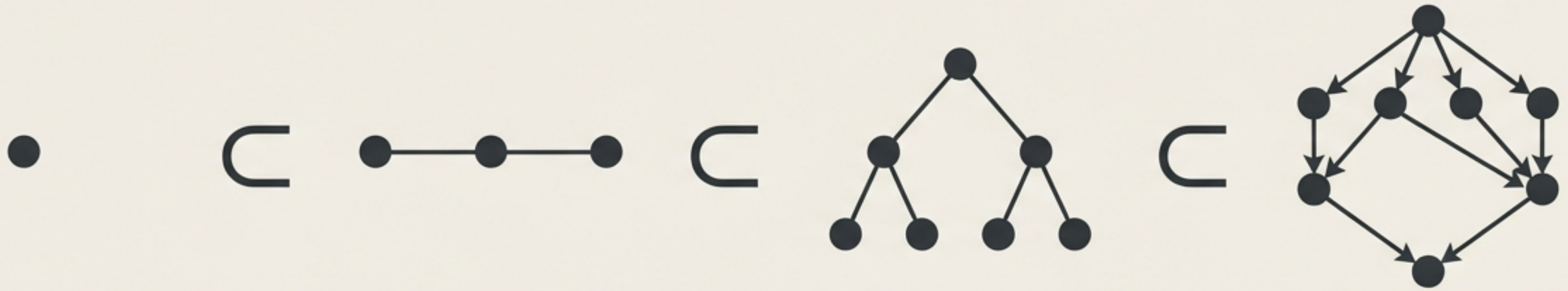
The posterior probability.

Exemplars do not teach the model new skills. They serve to concentrate the Bayesian posterior around the true task concept, functioning as a declarative program compiled at inference time.

Engineered prompts transform a frozen model into a polymorphic execution engine

Application Class	Prompt Mechanism	Output Space
Reasoning / QA	Decompose query into sub-questions → NL answer + justification chain	Natural language answer
Information Extraction	Schema-guided prompts specifying entities and relations	Structured records (JSON, tuples)
Code Generation	Functional spec + I/O examples + constraints	Executable source code
Tool Use / Agents	Action-observation-thought loops	API calls, environment commands

The Evolution of Reasoning Topology



Point
Direct IO Prompting
(x → y)

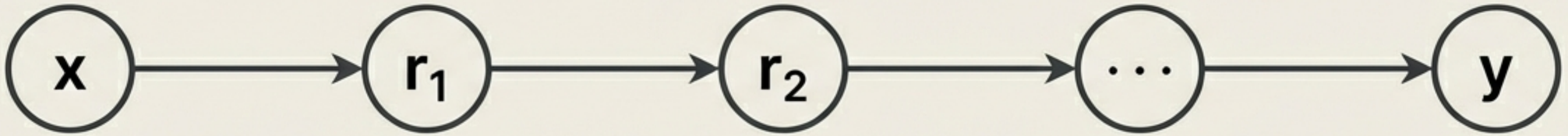
Path
Chain-of-Thought

Tree
Tree-of-Thought

DAG
Graph-of-Thought

Trading constraints for expressivity: Progressively relaxing topological constraints on the reasoning computation graph yields massive gains in error resilience and combinatorial power, at the cost of inference latency.

Chain-of-Thought linearizes reasoning to expand computational depth



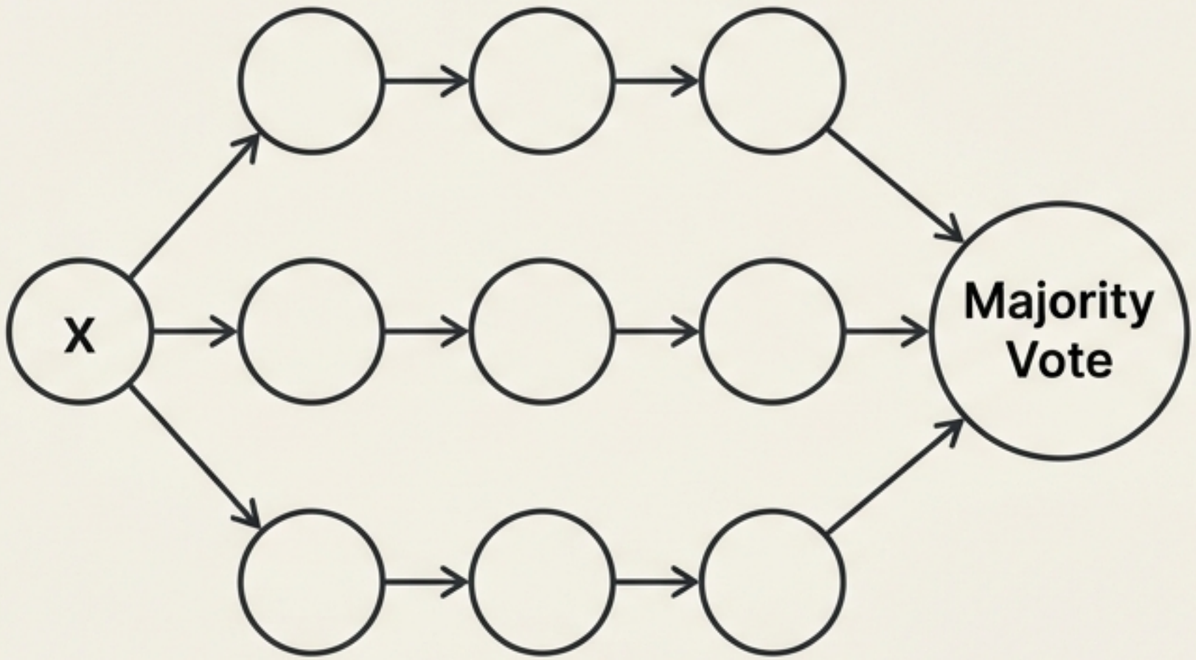
$$P_\theta(y \mid x) = \sum_{\mathbf{r}} P_\theta(y \mid x, \mathbf{r}) \cdot P_\theta(\mathbf{r} \mid x)$$

(Marginalizing over the latent reasoning chain **r**)

Theoretical Justification (Feng et al., 2023)

- A standard constant-depth transformer single forward pass is limited to $\mathbf{TC^0}$ (constant-depth threshold circuits).
- Forcing the LLM to **serialize** intermediate computations (T sequential passes) expands its expressivity to **P** (polynomial-time decidable).
- Expressivity(Transformer + CoT_T) $\supseteq$ DTIME(T).

Self-Consistency marginalizes over reasoning paths via Monte Carlo estimation

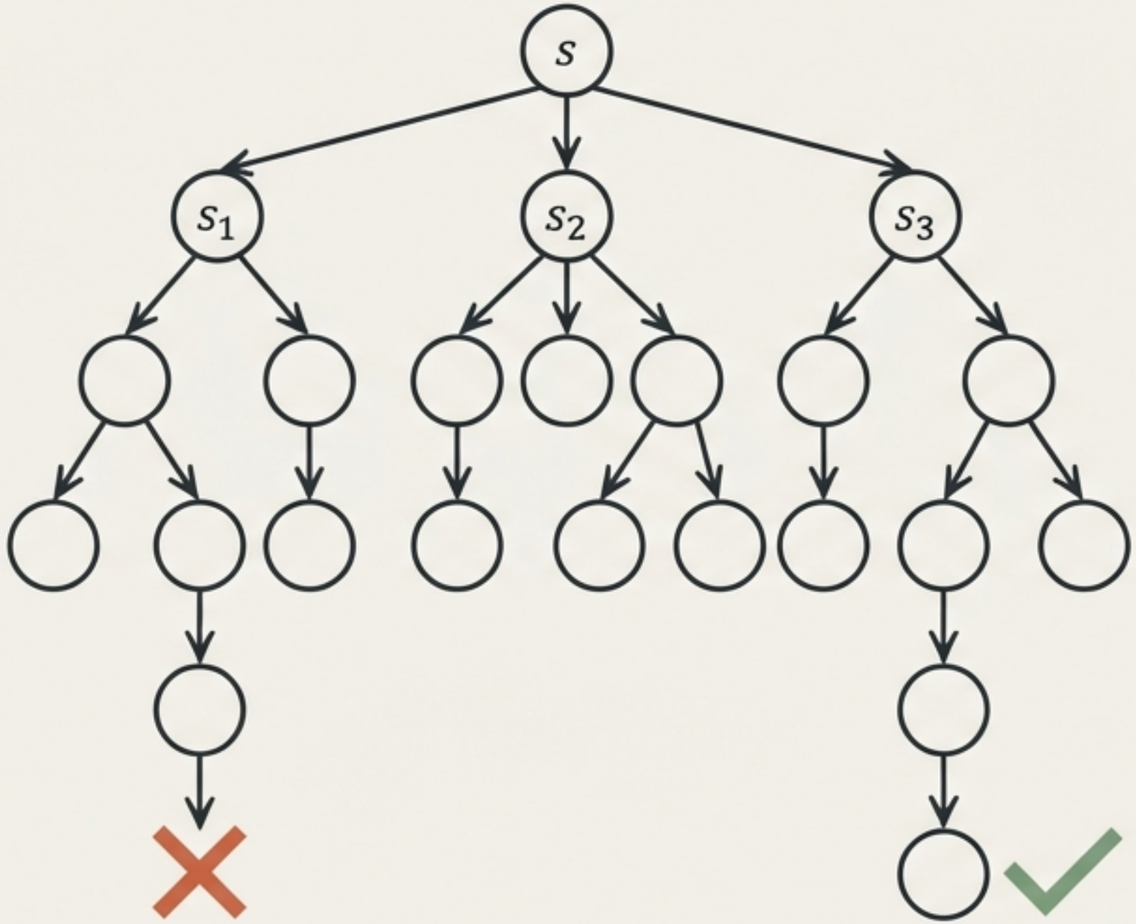


$$a^* = \arg\max_a \sum_{i=1}^{K} \mathbf{1}\left[\texttt{Extract}(\mathbf{r}_i) = a\right]$$

The Problem: Standard CoT greedily decodes a single chain (O(1) calls per step), offering zero error recovery.

The Solution: Self-Consistency samples K independent chains with temperature $\tau > 0$, approximating true mathematical marginalization over all possible reasoning paths via voting.

Tree-of-Thought operationalizes systematic search and backtracking



ToT breaks away from greedy decoding, utilizing the LLM in three simultaneous roles:

1. **Thought Generator**
   Proposes candidate next steps ($G_\theta(s) \to \{t_1, \ldots, t_b\}$).

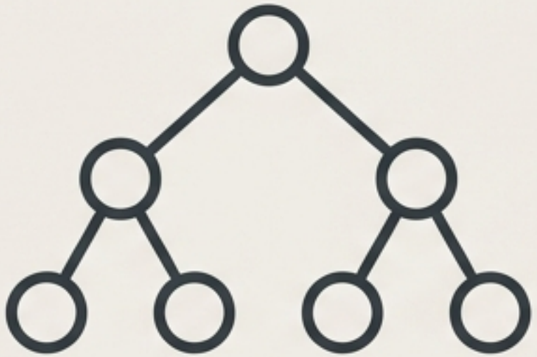
2. **State Evaluator**
   Assesses node viability via value-based scalars or vote-based continuations ($V_\theta(s) \to \mathbb{R}$).

3. **Search Controller**
   Executes classical algorithms (BFS/DFS) to expand or prune.

Cost Warning: Worst-case LLM calls scale exponentially $O(b^d)$ for depth d.

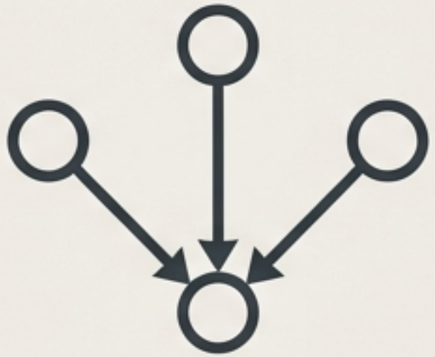
Graph-of-Thought introduces multi-premise reasoning and branch merging

Tree Topology: Strict Divergence



$$v_j = f_\theta(v_{\text{parent}(j)})$$

DAG Topology: Branch Merging

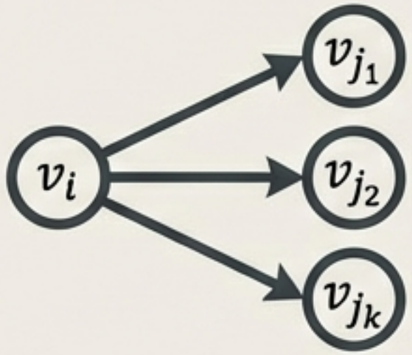


$$v_j = f_\theta(v_{i_1}, \ldots, v_{i_m})$$

Insight Badge

Why Trees Are Insufficient: In a tree topology, a thought can mathematically only depend on exactly one preceding thought. This fundamentally breaks on tasks requiring multi-premise synthesis, where evidence from independent reasoning branches must be merged.

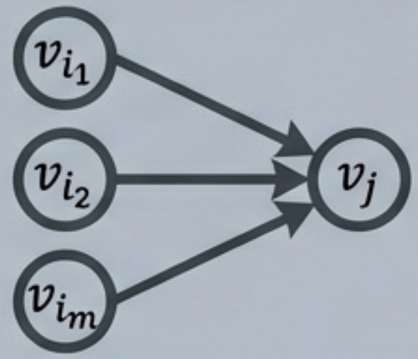
Four primitive operations govern arbitrary reasoning DAGs



1. Generate (GEN)

$v_i \rightarrow \{v_{j_1}, \ldots, v_{j_k}\}$

Branching into new candidate thoughts.



2. Aggregate (AGG) — Unique to GoT

$\{v_{i_1}, \ldots, v_{i_m}\} \rightarrow v_j$

Creates a vertex with an in-degree > 1.

3. Refine (REF)

$v_i \rightarrow v_i'$

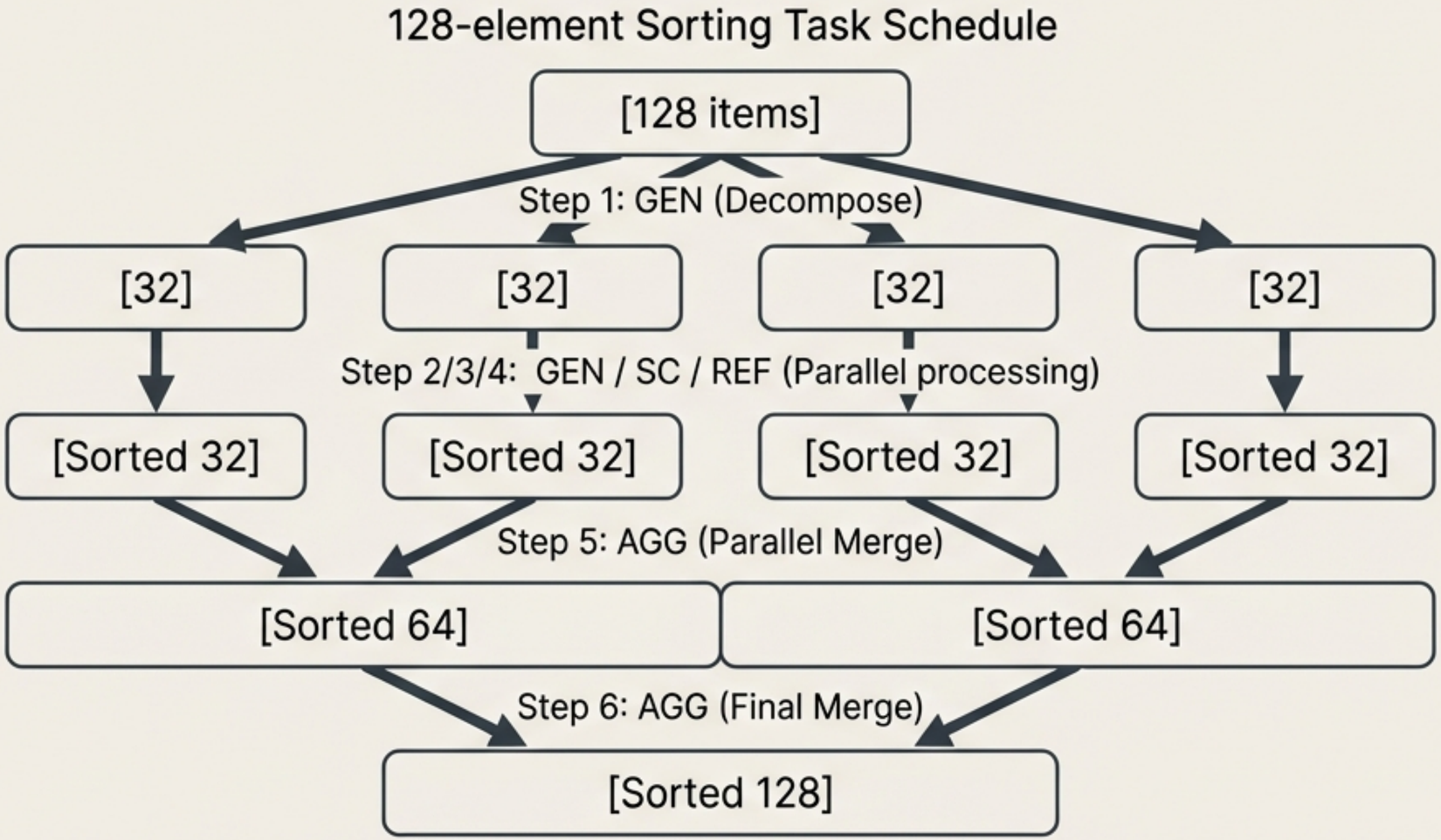
Iterative self-correction in-place.

4. Score (SC)

Quantitative evaluation without modifying the graph.

Graph Metrics: Volume (|E|) | Latency (longest path) | Parallelism ({|E| / \Lat})

DAG topologies drastically reduce latency through parallel execution



By divorcing latency from total reasoning volume, GoT reduces latency to $O(\log m)$ compared to $O(n)$ in linear CoT, executing non-dependent sub-problems concurrently.

Topological structures define the bounds of expressivity and latency

Method	Graph Structure	In-degree	Out-degree	Aggregation	Latency
IO (Direct)	Single edge	$0 \to 1$	$1 \to 0$	—	$O(1)$
CoT	Linear chain	≤ 1	≤ 1	No	$O(n)$
ToT	Tree	≤ 1	$\leq b$	No	$O(d)$
GoT	DAG	$\leq m$ (arbitrary)	$\leq k$ (arbitrary)	Yes	O(longest path)

GoT strictly subsumes prior methods. CoT is simply a DAG constrained to a path graph. ToT is a DAG constrained to an in-degree of 1.

Method selection depends **strictly** on task task combinatorial structure

Single-step, low complexity	→	**Deploy IO**
Sequential reasoning, no branching	→	**Deploy CoT**
Sequential + requires robustness	→	**Deploy CoT-SC** (if compute budget allows K samples)
Combinatorial search, backtracking required	→	**Deploy ToT**
Decomposable sub-problems requiring re-synthesis	→	**Deploy GoT**

Do not default to the most complex graph. Match the topological expressivity precisely to the intrinsic complexity class of the target task.

Relaxing topological constraints trades **inference cost** for expressivity

Advanced prompting is not a collection of fragmented hacks. It is a unified topological hierarchy. By systematically loosening structural constraints on the computational graph, we unlock multi-premise synthesis, algorithmic search, and robust error recovery directly within frozen LLMs.