Caterpillar of Thoughts: The Optimal Test-Time Algorithm for Large Language Models

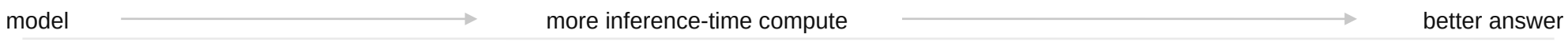
Amir Azarmehr · Soheil Behnezhad · Alma Ghafari

Azarmehr, Behnezhad, Ghafari. Caterpillar of Thoughts. arXiv:2603.22784, 2026.

Why inference-time scaling matters

Recent reasoning systems have shown that spending more computation during generation can substantially improve performance on hard math, code, and science tasks.

- Extra inference compute can mean longer reasoning traces, multiple sampled attempts, verifier calls, reranking, search, or backtracking.
- These methods often improve answers, but they consume very different amounts of compute and fail in different ways.
- So the question becomes algorithmic: given a compute budget, what should the inference-time procedure do?



From heuristics to a formal model

Many inference-time scaling strategies are plausible. This paper asks which ones are principled, and under what model.

1. Set up a formal model of inference-time search: states, transitions, targets, rewinding, and cost.
2. Derive the optimal algorithm within that model rather than choosing a search heuristic by intuition.
3. Translate the ideal algorithm into a practical method using estimated cost-to-go values.
4. Evaluate the resulting method on structured reasoning tasks to see whether the theory buys empirical gains.

Presentation overview

1. Start from the inference-time scaling question: how should extra compute be allocated?
2. Formalize inference-time search as a Markov chain with rewinding.
3. Understand the ideal Caterpillar of Thoughts (CaT) algorithm and the caterpillar theorem.
4. Bridge the theorem to practical CaT with noisy cost-to-go estimates.
5. Discuss experiments, caveats, and open questions.

One-slide thesis

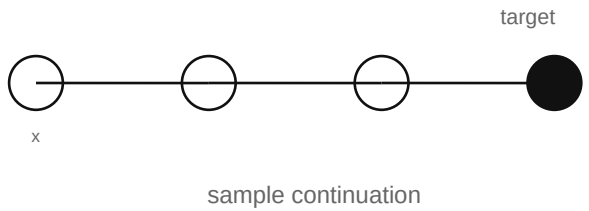
In the paper’s model, the optimal test-time search tree is always a caterpillar.

- The algorithm may rewind to any previously observed state.
- Nevertheless, arbitrary wide tree search is unnecessary in the ideal model.
- The optimal policy keeps a best-known state, samples from it, and promotes only improvements.
- The practical Caterpillar of Thoughts (CaT) algorithm approximates this with a softmax over noisy cost-to-go estimates.

1. Model

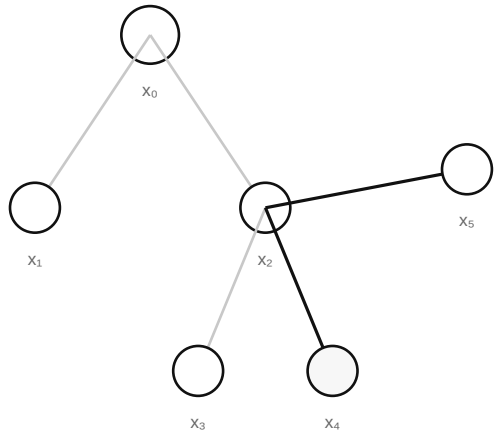
LLM inference as state-space exploration

- State x: a partial solution / token prefix / partially completed object.
- Transition P(x, ·): the LLM’s sampling distribution for one next step.
- Target t: a correct or sufficiently high-quality final state.
- Cost: total number of generated states before hitting t.



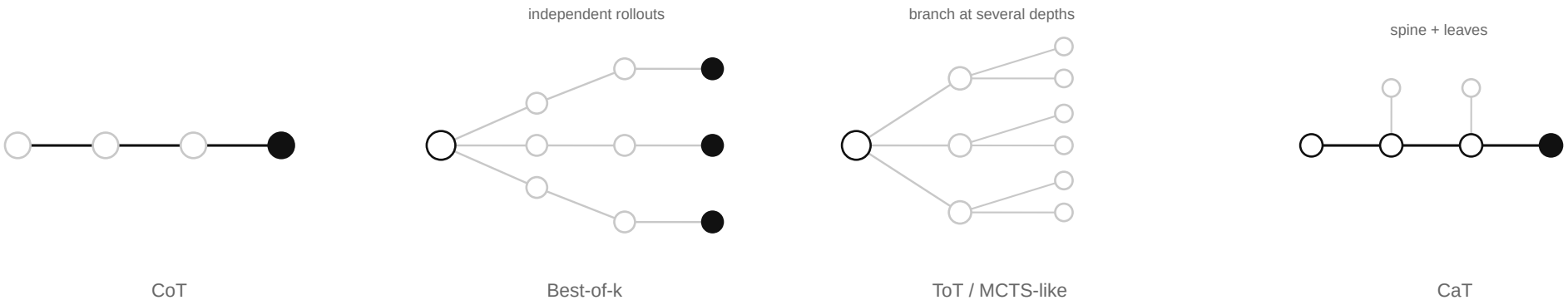
What rewinding adds

- At time k, the algorithm has observed states x_0, …, x_k.
- It chooses a parent p_k from the observed set.
- It then samples x_{k+1} ~ P(p_k, ·).
- The observed computation history is a tree, not merely a trajectory.



rewind to x_2, then sample again

Common test-time methods become tree shapes



- Best-of-k samples complete answers independently from the root.
- ToT / MCTS-like methods branch at several depths.
- CaT keeps one promoted spine; unsuccessful attempts become leaves.

Optimization objective

$$\mathcal{O}(x) = \text{minimum expected number of generated states needed to hit } t, \text{ starting from } x$$

- $\mathcal{O}$(t) = 0.
- States with smaller $\mathcal{O}$ are better partial solutions.
- The ideal algorithm assumes oracle access to $\mathcal{O}$(x).
- The practical algorithm must estimate $\mathcal{O}$(x) from model/evaluator scores.

2. Ideal CaT

Algorithm 1: ideal Caterpillar of Thoughts

Ideal CaT

$$\textbf{Input: } s_0,\ t,\ \mathcal{O}(\cdot)$$

$$m \leftarrow s_0$$

$$\textbf{while } m \neq t \textbf{ do}$$

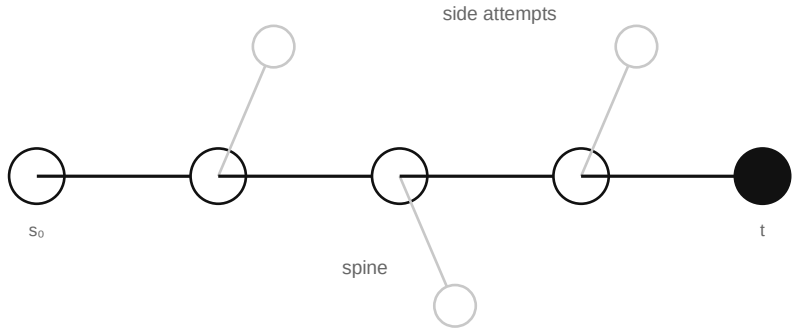
$$\quad y \sim P(\cdot \mid m)$$

$$\quad \textbf{if } \mathcal{O}(y) < \mathcal{O}(m) \textbf{ then}$$

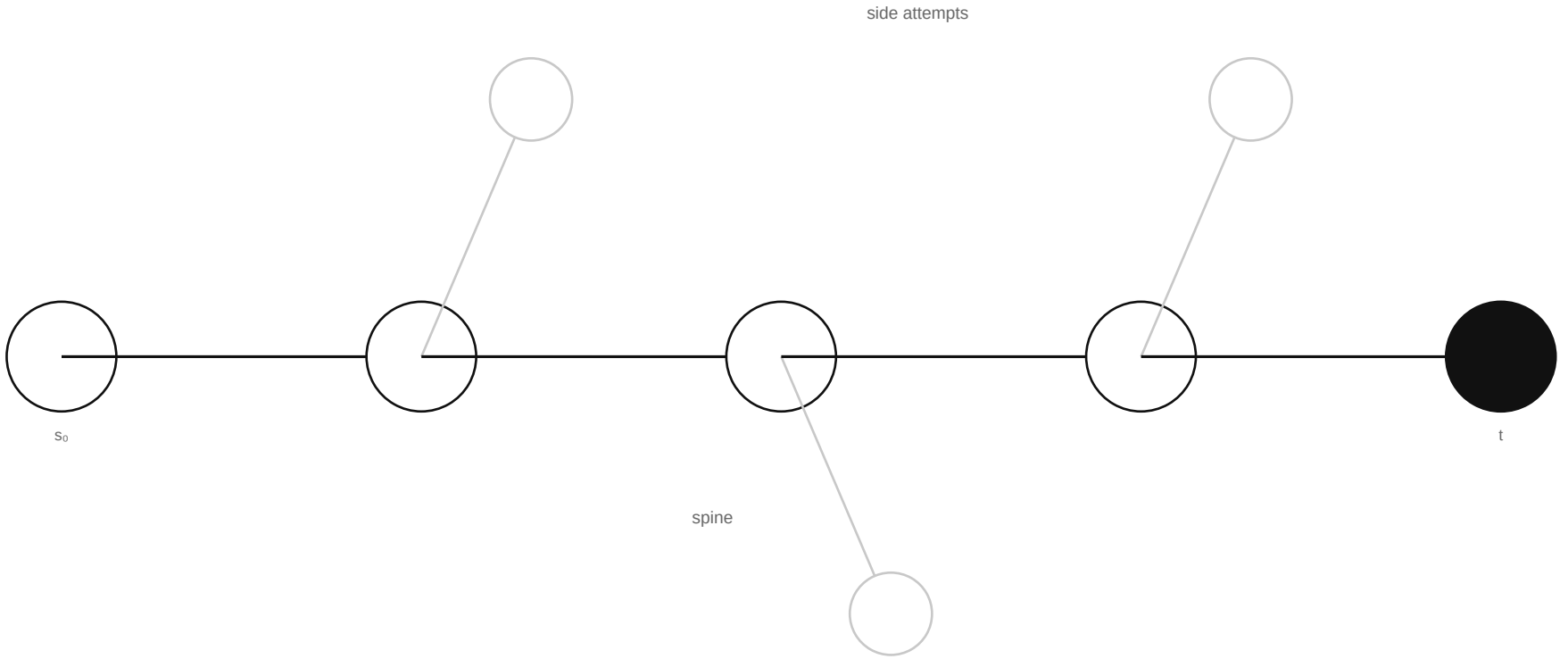
$$\quad\quad m \leftarrow y$$

$$\textbf{return } t$$

The only persistent state is the best observed state m. Side samples that fail to improve m become leaves.



Why the tree is a caterpillar



Definition: a tree is a caterpillar if deleting all leaves leaves a path.

Main theorem

For any fully observable finite Markov chain with a target state, ideal CaT reaches the target in the minimum possible expected number of generated states.

$$\mathbb{E}[\text{steps from } s_0 \text{ under Algorithm 1}] = \mathcal{O}(s_0)$$

- Equivalent reading: there exists an optimal policy whose computation tree is a caterpillar.
- Arbitrary multi-level branching gives no additional benefit when $\mathcal{O}$ is known exactly.

Proof skeleton

1. History compression: among optimal policies, the exact explored tree can be ignored; only the observed set X matters.

2. Best-state principle: once X is observed, the optimal continuation is to resume from the best state in X.

3. Promotion rule: sample from that best state and switch only when a sample has strictly lower cost-to-go.

$$V(X) = \min_{x \in X} \mathcal{O}(x)$$

The recursive equation behind the algorithm

$$B_x = \{y : \mathcal{O}(y) < \mathcal{O}(x)\}, \qquad p_x = \sum_{y \in B_x} P(x, y)$$

$$\mathcal{O}(x) = \frac{1}{p_x} + \sum_{y \in B_x} \frac{P(x, y)}{p_x} \mathcal{O}(y)$$

- p_x is the probability that one draw from x produces a strictly better child.
- The first term is the expected number of draws needed to get such a child.
- The second term is the expected remaining cost after conditioning on the successful child.
- This is the recurrence that makes the Dijkstra-like computation possible.

Computing the oracle in small chains

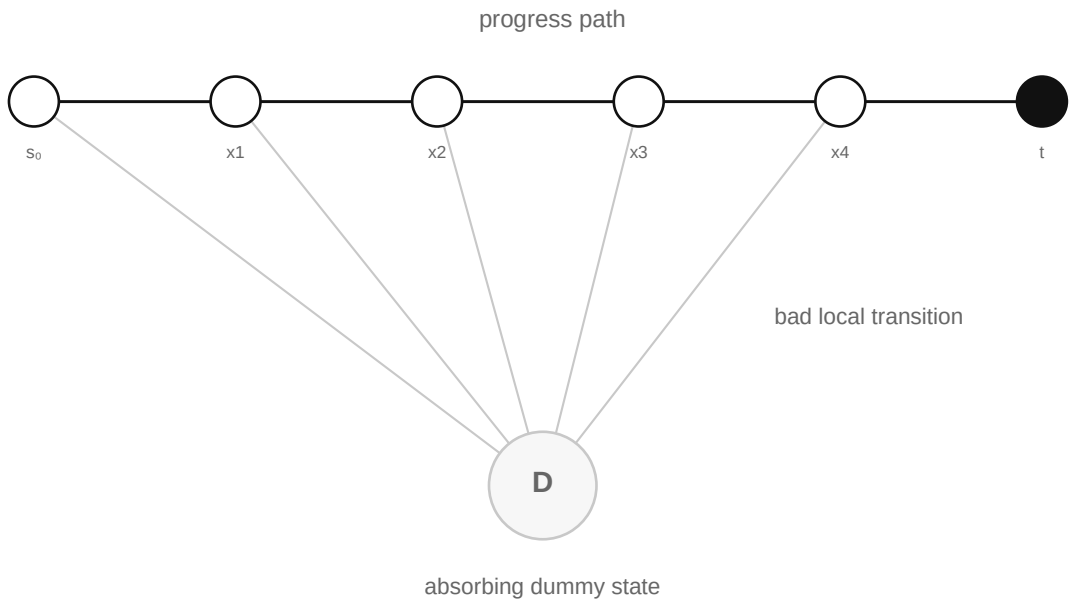
1. Start with the finalized set G = {t}.
2. For each outside state, compute its candidate cost.
3. Finalize the outside state with the smallest candidate cost.
4. Repeat until the initial state is finalized.

$$C_G(x) = \frac{1 + \sum_{y \in G} P(x, y)\, \mathcal{O}(y)}{\sum_{y \in G} P(x, y)}, \qquad x^\star = \arg\min_{x \notin G} C_G(x)$$

For LLMs, this is not an implementation recipe. It explains what the estimated cost-to-go is trying to approximate.

3. Why rewinding matters

The dummy-trap example



Passive rollout: one bad local transition sends the whole trajectory to D, where success is impossible.

Rewinding: keep the furthest successful prefix and retry the next local transition.

Same Markov chain; exponentially different expected search cost.

Interpretation for LLM reasoning

- Long reasoning chains compound local errors.
- Independent rollouts restart from scratch after every failure.
- Tree search often spends compute maintaining many mediocre branches.
- CaT says: identify the best prefix and retry from there until you find a better prefix.

4. Massive chains

Exact values are unavailable in LLMs

Ideal analysis	**LLM setting**
Explicit finite Markov chain	Enormous implicit state space
True cost-to-go $\mathscr{C}(x)$	Noisy estimate of remaining difficulty
Greedy expansion of the best state	Soft parent selection to hedge against score error

Key point: the theorem identifies the policy class; the empirical method succeeds only if the estimator gives useful relative rankings of partial states.

Noise model: stochastic vs. adversarial

Approximation model	Meaning	Consequence
Independent noise	Each estimate is randomly perturbed; repeated estimates can be averaged or compared robustly.	Best-state search can still simulate large improvements.
Adversarial multiplicative noise	Each reported value is within a factor, but errors may be chosen coherently to mislead the policy.	Some chains become impossible to solve efficiently.

Message: value-guided search is only as reliable as the structure of the value errors. Random errors can be mitigated; systematically biased rankings can destroy the caterpillar logic.

Practical CaT: softmax parent selection

Empirical CaT

Input: $\widehat{\mathcal{O}}(\cdot),\ \tau$

repeat until budget exhausted or target found

choose parent x from observed states

with probability $\propto \exp\big(-\widehat{\mathcal{O}}(x)/\tau\big)$

$y \sim P(\cdot \mid x)$

add y to the observed set

Softmax parent selection keeps the best-state bias while leaving probability mass for uncertain alternatives.

- small τ: concentrate on the current best estimate
- large τ: explore more uncertain states
- useful when estimates are noisy or near-tied

5. Experiments

Game of 24 setup

- Input: four numbers; use each exactly once with arithmetic operations to make 24.
- State: operations so far plus the multiset of remaining values.
- Transition: sample a valid next operation and operands.
- Target: a complete expression evaluating to 24.

$$\{4, 6, 8, 8\} \xrightarrow{4+8=12} \{6, 8, 12\}$$

Game of 24 results

Method	Success rate	Avg. tokens
Tree-of-Thoughts	74%	19.2k
CaT, 15 steps	81%	15.3k
CaT, 10 steps	78%	14.2k

Interpretation: CaT searches less broadly but concentrates compute on promising partial states.

Crossword setup and results

- State: a partially filled crossword grid.
- Action: choose an unfilled clue and propose a candidate word.
- Constraint: proposed words must agree at intersecting letters.
- Target: a complete grid whose entries satisfy the clues.

Method	Word	Letter	Solved	Tokens
Truncated ToT	39.5%	64.8%	5%	73.4k
CaT	50.0%	68.6%	15%	66.8k

Why this task fits CaT: a wrong word can poison many crossings, so retrying from a strong partial grid is valuable.

6. Discussion and takeaways

What is conceptually strong

- A simple formal model covers many inference-time search strategies.
- The main theorem is structural: optimal search need not expand many branches at many depths.
- The proof explains when “best-prefix retry” is sufficient.
- The empirical algorithm is a plausible relaxation under noisy values.

Main caveats

- Exact $\mathscr{O}$-values are not available for real LLM systems.
- The model treats generation as a Markov chain over partial states; actual prompting can change the transition process.
- With estimated values, the exact theorem no longer directly guarantees the empirical policy.
- The experiments are on structured search tasks, not a broad suite of reasoning benchmarks.

Main takeaway

CaT says: spend inference-time compute by repeatedly retrying from the best known partial state, and only advance the main line when a sample looks genuinely closer to success.

- Theory: this is exactly optimal with true hitting-time values.
- Practice: use estimated values and soften the greedy choice.
- Design lesson: backtracking is valuable; unconstrained wide tree expansion is not automatically necessary.

Discussion questions

1. What is the right empirical proxy for $\mathcal{O}(x)$ in open-ended reasoning tasks?

2. When does width help despite the theorem—only estimator noise, or also objective mismatch?

3. Can we learn a cost-to-go estimator specifically optimized for rewinding policies?

Reference

Azarmehr, A.; Behnezhad, S.; Ghafari, A. “Caterpillar of Thoughts: The Optimal Test-Time Algorithm for Large Language Models.” arXiv:2603.22784, 2026.

OpenAI. “Learning to reason with LLMs.” 2024. Used for high-level motivation around inference-time compute in reasoning models.

Guo, D. et al. “DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning.” Nature, 2025. Used for high-level motivation around reasoning models and inference-time compute.

The deck paraphrases definitions, theorem statements, algorithms, and experimental numbers from the Caterpillar of Thoughts paper.