ACTIVE SHORTCUT DISCOVERY IN SHARED STATE GRAPHS FOR FOUNDATION MODEL REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Foundation-model reasoning methods such as Tree of Thoughts and Graph of Thoughts improve over one-shot prompting by searching over intermediate states, but their trajectory-centric representations repeatedly explore equivalent subproblems and make errors costly. We study how such search can be reformulated around a shared AND-OR graph for tasks with reusable Markovian state structure. Equivalent intermediate states are merged within and across problem instances, proof status propagates through the graph, proof-relevant actions are verified before accepting solutions, and our novel Active Shortcut Discovery queries the generator to connect open states to already solved regions. On two challenging LLM reasoning benchmarks, this approach improves Macro-F1 over the strongest baselines by 25.9% on WickedGame24 and 19.6% on PrefixBooleanExpression while using 11.7 times fewer completion tokens. On retrosynthetic planning, it matches state-of-the-art success rates while producing the smallest search graphs. These results show that structured reasoning systems can reduce both redundant computation and hallucinated solutions by reusing verified intermediate states rather than restarting search for each trajectory or instance. We release data, code, and prompts for our method at this URL.

1 INTRODUCTION

Foundation models, such as Large Language Models (LLMs) or other (neural) generative models, have demonstrated impressive capabilities across a wide variety of domains. Yet, their probabilistic nature renders the generation process myopic and prone to errors. To address this, current research increasingly frames more complex problem-solving tasks as reasoning via a series of multi-prompt interactions. Methods such as Tree of Thoughts (ToT) (Yao et al., 2023), Graph of Thoughts (GoT) (Besta et al., 2024), or Buffer of Thoughts (BoT) (Yang et al., 2024), have been proposed as an external scaffolding to guide the generation process through a structured problem decomposition. Conditioned on the previously explored generation trajectory, possible transitions leading to new states are sampled recursively from a foundation model to close in on some target solution. In that sense, generation trajectories resemble individual paths in proof trees in logical reasoning. While these techniques have been shown to outperform single-step baselines in terms of success rates, they still face some shortcomings: (1) *effectiveness*: undetected errors may propagate through the generation trajectory and might thus invalidate large portions of the computation, (2) *efficiency*: forward-chaining intermediate steps needs to explore a high-entropy space of possible states, many of which may be valid, yet ultimately irrelevant to the target solution. The first problem may lead to unusable results, the second may seriously prolong computation times.

Since self-verification techniques have been shown to successfully mitigate the first limitation, this paper focuses on the problem of redundant explorations. We identify two main sources of redundancy in current multi-step generative approaches: intra- and inter-task-instance redundancy. Exploiting these redundancies in the sense of reusing intermediate results within and across problem instances promises to significantly boost efficiency and thus is the central focus of this work. We present a new framework specifically designed for tasks with reusable sub-structures. In particular, whenever a generation trajectory reaches a state that has already been reached in a previous generation attempt (within or across problem instances), three cases can occur: either (a) the previous trajectories did not lead to a solution and are fully exhausted, in which case the current trajectory can prune the state immediately, (b) no solutions have yet been found, but more exploration is still

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

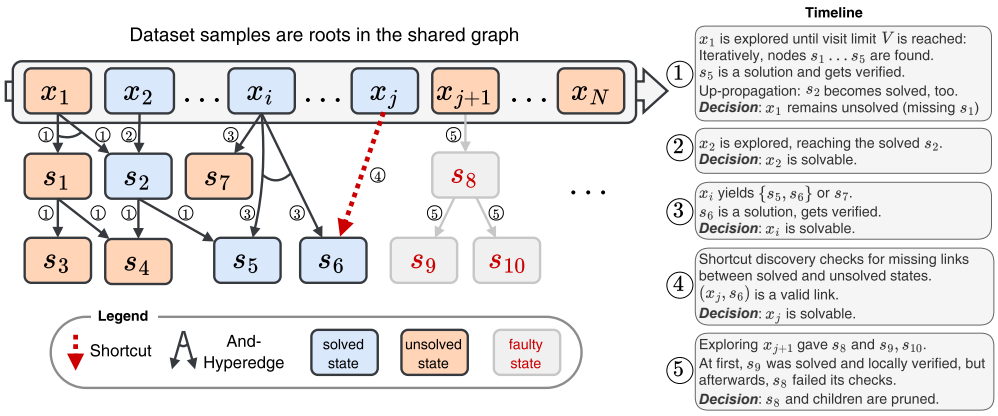

Figure 1: Overview of the shared AND-OR graph dynamics. Equivalent successor representations are merged into a single OR-node, action nodes implement AND semantics, solved truth values propagate bottom-up, and Active Shortcut Discovery ranks open–solved state pairs before asking the generator for a shortcut action that reaches a solved target while allowing additional successors.

possible, allowing to skip already explored states and actions, or (c) there is a known trajectory leading to a valid solution, which can directly be reused. Obviously, *state merging* approaches in cases (a) and (c) allow to conclude the reasoning process without incurring further costs, whereas in case (b) they avoid the pointless recomputation of already explored states.

This potential is largely ignored in current reasoning methods, because not all generative reasoning tasks can be expected to benefit greatly from the aforementioned reuse concepts. How probable is it that two LLM generations arrive at exactly the same state in their deliberations? We argue that for a large class of problems it is actually quite likely: in particular all problems with *overlapping subproblems*, where problem states additionally exhibit a certain *Markovian property*, i.e., states do not depend on the trajectories of previous deliberations. These properties are common among problems that are typically solved by dynamic programming, divide & conquer, or branch & bound. Moreover, processing structurally equivalent problem instances in batches generally creates overlapping subproblems between individual instances.

In brief, we can expect *state merging* to be beneficial for quite a share of real-world generation tasks. Yet, there is still the size and high entropy of the deliberation space to consider. To foster reuse even in these complex spaces, our approach additionally features a novel *active shortcut discovery* to actively infer connections between previously discovered, but otherwise disconnected states. This allows foundation models to propose direct transitions that bridge gaps between all relevant states.

In summary, our contributions are:

- We propose a new framework leveraging state merging and active shortcut discovery designed specifically for tasks with reusable sub-structures, severely improving the efficiency of multi-step generation tasks.
- We study our method's properties on a well-known theoretical problem and show its applicability and performance benefits on a practical real-world task in the chemical industry.
- Our experiments over three challenging datasets show significant gains in Macro-F1 over previous methods: 25.9% on WickedGame24, and 19.6% on PrefixBooleanExpression, while requiring **11.7 times fewer generated output tokens** than the runner-ups.

2 Background

Foundation models. Foundation models are machine learning models trained on vast datasets that can be adapted to a wide range of domains, catering for a variety of tasks. Examples include

Large Language Models for text, as well as generative models on other modalities, such as images, genes (Zvyagin et al., 2023), or materials (Merchant et al., 2023).

Early applications of foundation models usually followed a direct input-output interaction pattern. Modern, more complex applications demand multi-step interactions, evolving from simple Chain-of-Thought (CoT) (Wei et al., 2022) prompting into graph-based reasoning structures, e.g., Tree of Thoughts or Graph of Thoughts, that allow for solution space exploration. The underlying graphs reflect deliberations by expanding previous generations (called thoughts, states, or nodes) by new prompts (transitions, actions, or edges) into newly generated states. Applying this iteratively until a target is reached (or a compute budget is exhausted) yields individual trajectories (or paths) of visited states and transitions. It is an interesting observation that individual trajectories may intersect at certain states, and thus share common intermediate thoughts, opening up sets of alternatives to either arrive at or depart from the shared states. An active exchange on possible sub-trajectories within or across multiple deliberations could thus allow foundation models to tap into previously ignored optimization potential.

Reusable states. Invoking action a in some state s transitions s into a (collection of) new state(s); the actual deliberation problem represented by s is broken down into subproblems. If states exhibit the *Markov property*, i.e. their actions and successor states depend only on the current state, and not on the trajectories leading to it, directly matching states identical in their respective representations enables safe reuse of already explored trajectories. This idea is not entirely new and has already been explored, e.g., in automata theory or state machines, so why do current works in foundation model reasoning rarely incorporate it? In a nutshell, the reason lies in the respective choices of state representations: ToT or GoT, which generalize many current reasoning topologies, always include the full trajectory in each state representation. Unfortunately, this all-encompassing representation severely hampers reuse. In contrast, the state representations in our framework are given by a limited number of tokens extracted from the last generation with respect to the selected action's expected result types. If two states have the same set of tokens as representation, we consider them equivalent, because the same actions can be applied on them.[1]

AND-OR graphs. Formalizing the above semantics of reusable states under the Markov property, we obtain an *AND-OR graph* structure. In this paradigm, a state acts as an OR-node: the reasoner can choose from a set of alternative action (e.g., attempting a mathematical proof by contradiction, or induction, or an algebraic approach). Conversely, when an action decomposes the deliberation into multiple interdependent sub-states, the transition acts as an AND-node (e.g., writing separate sections of a document that later need to be combined). Crucially, truth values representing solution viability propagate bottom-up through this topology: a state (OR-node) evaluates to true if *at least one* of its executed actions succeeds, whereas an action (AND-node) is considered valid *only if all* of its children are true.

Because traditional trajectory-based representations merge the entire history into each state, their reasoning structures unfold into exponentially growing AND-OR trees (Nilsson, 1982). In contrast, by merging equivalent subproblems from disparate trajectories the topology collapses to an AND-OR graph whose size is bounded by the cardinality of the unique state space. For example, computing the Fibonacci sequence without state merging has complexity $\mathcal{O}(1.618^n)$, which reduces to $\mathcal{O}(n)$ with merging.

Formally, we define AND-OR graphs as a directed bipartite graph $G = (S \cup A, E)$, partitioned into state vertices S and action vertices A. Generally, evolving such a graph involves iterating four procedures: 1) selecting an expand-worthy state $s \in S$ using some traversal algorithm, 2) proposing new actions by sampling, 3) applying these actions to obtain successors, and 4) evaluating the new states to assess their utility towards the solution (Nilsson, 1982). In our setting, foundation models instantiate the generative parts of this loop: they propose action nodes, produce successor states, and evaluate candidates. Let p_θ represent a foundation model with parameters θ that assigns a probability to a token sequence $x = (x^{(1)}, x^{(2)}, \ldots, x^{(N)})$ via the joint probability: $p_\theta(x) = \prod_{i=1}^{N} p_\theta(x^{(i)} \mid x^{(<i)})$. Sampling from p_θ conditioned on an input x, often framed within

[1] Of course, other representations are also possible. We only consider basic token-level equality so the observed effects can be cleanly attributed to our framework and not the representation.

task-specific instructions $\mathrm{prompt}_{\mathrm{M}}$, generates the next output: We write $y \sim p_\theta(\cdot \mid \mathrm{prompt}_{\mathrm{M}}(x))$, or simply $y \sim p_\theta^{\mathrm{M}}(\cdot \mid x)$ as a shorthand notation.

3 METHOD

We now describe how to turn the reusable-state AND-OR formalism from Section 2 into a search procedure. The method has two central components. First, we maintain a single shared graph over all input instances, so that equivalent intermediate states are represented once and can be reused across trajectories. Second, instead of waiting for forward sampling to accidentally rediscover useful states, we actively query the generator for actions that connect unsolved states to known useful regions of the graph. Figure 1 summarizes these dynamics.

Algorithm 1 Shared graph search

Require:
Samples X, workers w, shortcut interval κ, shortcut limit k, shortcut ranker c, visit limit V, IsSolution, p_θ, Repr
$S_0 \leftarrow X$, $A_0 \leftarrow \emptyset$, $G \leftarrow (S_0 \cup A_0, \emptyset)$
for w workers **in parallel do**
  while Expandable(G, V) **do**
    ScheduleNext(κ, k, G)
  end while
end for
return Finalize(G)

procedure ScheduleNext(κ, k, G)
  $\mathcal{E} \leftarrow \emptyset$
  if ExpandShortcut(κ) **then**
    $\hat{I} \leftarrow$ SelectShortcutTargets(G, k)
    $\mathcal{E} \leftarrow \{(s, \hat{a}) \mid (s, s') \in \hat{I},$
      $\hat{a} \sim p_\theta^{\mathrm{shortcut}}(\cdot \mid s, \{s'\})\}$
  else
    $s \leftarrow$ Select(G)
    $\mathcal{E} \leftarrow \{(s, \hat{a}) \mid \hat{a} \sim p_\theta^{\mathrm{propose}}(\cdot \mid s)\}$
  end if
  for each $(s, \hat{a}) \in \mathcal{E}$ **in parallel do**
    $\hat{y} \sim p_\theta^{\mathrm{apply}}(\cdot \mid s, \hat{a})$
    $\{\hat{s}_i\} \leftarrow$ Repr$(\hat{a}, \hat{y})$
    insert $\hat{a}$, edges, and novel successors into G
  end for
  UpdateTruth$(G,$ IsSolution$)$
  $\mathcal{E} \leftarrow$ VerifyWinning$(G, \mathcal{E})$
  UpdateTruth$(G,$ IsSolution$)$
  $Q \leftarrow$ Evaluate$(G, \mathcal{E})$
  UpdateStatistics$(G, \mathcal{E}, Q)$
end procedure

3.1 REUSE THROUGH A SHARED STATE GRAPH

Given a batch of inputs X, we initialize a shared AND-OR graph $G = (S \cup A, E)$ with one OR-node for each input state. OR-nodes are Markovian state representations, while AND-nodes represent actions whose application may produce one or more successor states. When an action is applied to a state s, a task-specific representation function Repr extracts successor states from the generated output. If a successor representation already exists in G, we reuse the existing OR-node rather than creating a duplicate. Thus, all trajectories reaching the same state share its outgoing actions, solved status, and proof information.

This is the main departure from trajectory-based search. In a tree, reaching the same intermediate state twice creates two independent subtrees. In our graph, both arrivals point to the same OR-node, so subsequent computation is shared. For example, in Game24, a state can be represented by the remaining multiset of numbers. Once the state "2 5 7" has been generated, later trajectories reaching the same multiset (e.g. from "1 1 5 7" or "2 2 5 5") can reuse any actions already explored from it. Truth values then propagate according to the AND-OR semantics from Section 2: an OR-node is solved if one outgoing action succeeds, while an AND-node succeeds only if all of its successor states are solved.

3.2 EXPANDING THE SHARED GRAPH

Algorithm 1 outlines our approach. An expansion policy selects unsolved expandable states, while the generator proposes and applies actions to grow the shared graph. We intentionally leave the expansion policy abstract in the method description as it can be instantiated by any value-guided traversal or search policy. In a normal expansion step, Select chooses an unsolved state s. The generator samples candidate actions $\hat{a} \sim p_\theta^{\mathrm{propose}}(\cdot \mid s)$, each action is applied via $p_\theta^{\mathrm{apply}}$, and Repr maps the generated output to successor OR-nodes. The resulting action is inserted as an AND-node, and successor states are either added to the graph or merged with existing OR-nodes. An evaluation step scores the inserted candidates for future expansion.

3.3 ACTIVE SHORTCUT DISCOVERY

Forward expansion can exploit reuse once two trajectories reach the same state, but discovering such intersections by sampling alone can be unlikely. Thus, we propose to actively search for viable candidates among the previously discovered states. Unfortunately, even if we restrict attention to

only already discovered states, there are up to $2^{|S|} - 1$ possible successor sets for a fixed state s, before considering novel states or multiple actions that realize the same set. Materializing these is infeasible.

We therefore formulate shortcut discovery as a relaxed inverse query and aim to find an action among whose successors would be at least one already solved node. This query asks the generator to reach a useful subproblem, while allowing the action to produce additional successors that were not specified in the query. Thus, a shortcut prompt receives an unsolved parent state s and a single solved state s', and proposes an action $\hat{a} \sim p_\theta^{\text{shortcut}}(\cdot \mid s, \{s'\})$ intended to produce s' among its successors. If the accepted action produces only solved successors, truth can propagate immediately to s. If it produces s' together with additional unsolved states, the resulting AND-node is not yet solved, but it is anchored in an already solved region of the graph and therefore reuses part of the proof. However, even the singleton case can still create many possible shortcut queries. Let S_{open} denote unsolved states and S_{solved} solved states. The candidate space has size on the order of $|S_{\text{open}}| \times |S_{\text{solved}}|$, so exhaustively querying the generator may still be infeasible. We introduce a shortcut ranker $c : S \times S \to \mathbb{R}$ and query only the top k candidates every κ-th expand step. The ranker prioritizes targets that are expected to save future expansions and appear plausible to connect from s. In this sense, the graph is used to retrieve useful target states, while the generator synthesizes the action conditioned on those targets. Exact ranking heuristics are task-dependent and deferred to the experiments section.

3.4 Accepting Solutions

Because states and actions can be reused across many input instances, accepting an erroneous proof can affect more than one trajectory. After each expansion step, UpdateTruth propagates known solution literals through the AND-OR graph and identifies candidate proof subgraphs. Before returning a solution, VerifyWinning checks only actions and child states that are currently proof-relevant via self-verification (Li et al., 2023; Madaan et al., 2023). If verification fails, the invalid AND-node and dependent proof edges are pruned, and truth values are propagated again. This focuses verification on the part of the graph that would actually justify an answer, rather than spending calls on speculative branches.

Finally, Finalize extracts a proof DAG for each solved input and renders the corresponding answer.

3.5 Concerning Cycles

Although task decompositions may be naturally acyclic in principle, probabilistic generators can introduce cycles purely by accident. This introduces a series of challenges for the graph maintenance. First, cycles break the otherwise straightforward upwards propagation of truth state (whether a state is solved). A cycle that is considered true will remain true even if its outside support got retracted afterwards, since the participating states act as a circular support. Secondly, determining whether a node is "open", i.e., whether it has any descendant that can be further expanded, is circular for the same reason. The previously well-working propagation now results in a greatest fixed point of the respective monotone operator on the graph. The desirable behavior instead corresponds to the least fixed point, i.e., only solved terminals support a solved node, and only an expandable descendant supports an open node. In the appendix, we prove that iteration from the bottom on a cyclic AND-OR graph stabilizes, and that the assignment it reaches is the least fixed point of truth propagation (Theorem A.1). We also give an incremental worklist approach at cost $\sum_i O(|\Delta_i| + |K_i| + |E(K_i)|)$ over a series of graph edits, where Δ_i is the size of the i-th edit, K_i its affected upwards cone, and $E(K_i)$ the edges inside that cone (Theorem 2).

4 Experiments

To thoroughly analyze our method's potential we conduct experiments over three diverse tasks: (a) WickedGame24, a set of math puzzles where the objective is to combine four numbers into an arithmetic equation that equals 24, now including unsolvable instances, (b) PrefixBooleanExpression, a task we adapted from BIG-Bench Hard (Suzgun et al., 2023) where a boolean expression is given in prefix notation and must be evaluated, and (c) Retrosynthesis, a real world chemical synthesis task where a target molecule must be recursively decomposed into precursor molecules until reach-

Table 1: Comparing methods across challenging compositional reasoning tasks. Macro-F1 scores are averaged over five repetitions. Best scores are highlighted bold, second best are underlined.

Task	IO_{SC}	CoT_{SC}	ToT	GoT	BoT	AGoT	DC_{Cu}	$DC_{Cu+Tools}$	**Ours**
WickedGame24	9.6	10.6	43.8	<u>64.7</u>	14.2	34.5	10.7	34.8	**90.6**
PrefixBooleanExpression	18.1	18.2	42.0	<u>66.5</u>	8.8	31.1	33.5	55.7	**86.1**

ing commercially available building blocks. We study two types of foundation models within our framework, namely LLMs (for tasks a and b), as well as one-step retrosynthesis models for task (c). Throughout all experiments, we employ Monte-Carlo Graph Search (MCGS) as expansion policy and utilize the values returned by the EVALUATE step as Q-values estimates.

4.1 GAME24 AND PREFIXBOOLEANEXPRESSION

The Game24 task offers a well-defined setting to assess the reasoning capabilities of LLMs (Sel et al., 2024; Yao et al., 2023; Yang et al., 2024). However, the problem instances considered in previous work are all solvable and thus not representative of a realistic problem space. As such, we propose a new dataset, termed WickedGame24, that contains 1365 samples, out of which 301 have no solution. The unsolvable instances preclude proofs by construction from a single found solution and instead require the considerably harder evidence of absence. Since this demands more thorough search it makes the task particularly suited for stress-testing.

In the PrefixBooleanExpression task, we ask to reduce a boolean expression in prefix notation to a single boolean value. This task is adapted from the respective subtask of BIG-Bench Hard (Suzgun et al., 2023), but further tuned for difficulty by increasing the number of input constants and introducing malformed inputs that are unsolvable. Out of the 100 samples, 50 are malformed.

LLM Setup and Implementation We use Microsoft's Phi-4 (14B), Qwen's Qwen3-32B, and Meta's Llama-3.3 (70B). Prompts are run with a chat completion setup, sampling with a temperature of 0.7. Tool use, function calling and reasoning are disabled, unless otherwise noted.

In general, we run our method with $w = 100$ parallel workers on a visit limit of $V = 25$ per target root, and $k = 500$ shortcut candidates which are retrieved every 100-th expand step, some ablation experiments differ as indicated there. Note that $w > 1$ trades off reuse efficiency for higher GPU utilization and throughput.

ISSOLUTION is defined as follows: For Game24, it is satisfied if a state matches the string 24, for PrefixBooleanExpression, it is satisfied if the state evaluates to a single boolean value, and in retrosynthesis, it is satisfied if the state is molecule from our building blocks. We use a simple Levenshtein string edit distance to rank shortcut candidates.

Metrics We consider three main metrics, namely f1-macro to assess the accuracy of the methods, as well as the sum of completion tokens assess their efficiency. Additionally, we look at the three types of errors that occur and measure their rates across the answers: 1) false rejection, e.g., a solvable instance is rejected as unsolvable, 2) wrong solution, e.g., a solvable instance is correctly deemed solvable but the solution is incorrect, and 3) hallucination, when an unsolvable instance gets a solution proposed.

Baselines We use standard direct input-output generation (IO) and CoT as single-query baselines with few-shot examples and a self-consistency extension over 25 tries (Wang et al., 2023). Representing static multi-query methods, we include ToT (Yao et al., 2023) and GoT (Besta et al., 2024), which employ iterative search procedures, self-verification and evaluation to converge on a final solutation. Adaptive-GoT (Pandey et al., 2025) is a dynamic multi-query approach that recursively decomposes a given problem and focuses analysis on subproblems were additional work is needed. We include two recent reuse works, Dynamic-Cheatsheet (DC) (Suzgun et al., 2026) and Buffer of Thoughts (BoT) (Yang et al., 2024), which maintain a memory of previous search, either as an evolving notebook of strategies, snippets or lessons (DC), or as a meta-buffer of suitable thought templates (Bot).

4.1.1 RESULTS

Macro-F1 Improvements As per Table 1, our method consistently achieves significantly higher Macro-F1 scores than all previous methods. Compared to the best-performing methods, we improve upon the second best by 25.9% and 19.6% for WickedGame24, and PrefixBooleanExpression, respectively. The robustness of our method is particularly evident in the strong improvement on the WickedGame24 task, where others frequently hallucinate solutions that do not exist. Current reuse methods fail differently. Both BoT and DC_{Cu} depend strongly on Python code execution to carry strategies from one instance to the next. Across WickedGame24's 1365 inputs, Dynamic Cheatsheet accumulates those strategies strictly in sequential order and manifests incorrect ones that then dominate later instances. Many runs had the majority of their correct answers within the first 100 attempted puzzles. Only afterwards did the accuracy deteriorate and only sporadically recover. The same pattern remains with tool use enabled: $DC_{Cu+Tools}$ still reaches only 34.8 Macro-F1.

Efficiency Improvements Compared to the runner-up method GoT, we require 11.7 times fewer completion tokens on average. In terms of token efficiency, BoT is the cheapest method, requiring only a total of 4.32M/0.39M tokens for WickedGame24/PrefixBool. Simultaneously, this efficiency advantage seems to heavily impacts BoT's general ability to produce correct solutions: Its Macro-F1 score is often on par with random guessing.

4.1.2 ANALYSIS

Error Analysis Taking WickedGame24, we analyze the types of errors that typically occur, as shown in Table 2. As we can see, the IO and CoT methods collapse to almost entirely predicting solvable samples as unsolvable. Another extreme can be found in the BoT approach, which, on the one hand, less frequently disregards solvable instances, but on the other, hallucinates solutions to problems that do not have any. ToT also suffers from many such hallucinations. GoT is able to prevent these cases the best among the baselines, but is still improved on by our method, which is able to balance the three rates at comparatively low levels. AGoT and Dynamic Cheatsheet show high rates across the board, with notably low false rejection rates in some cases, signaling that identifying true positives works comparatively well and finding the correct answer is rather the bottleneck.

Table 2: Error rates on WickedGame24 by method. We report *False rejection*: a solvable instance is rejected as unsolvable, *wrong solution*: a solvable instance is answered incorrectly, and *hallucination*: an unsolvable instance receives a spurious solution.

Method	False Reject.	Wrong Solution	Halluc.
IO_{SC}	0.97	<u>0.02</u>	0.57
CoT_{SC}	0.95	**0.01**	0.65
ToT	0.29	0.12	0.70
GoT	<u>0.17</u>	0.14	<u>0.28</u>
BoT	0.55	0.26	0.95
AGoT	0.21	0.46	0.53
DC_{Cu}	0.44	0.54	0.50
$DC_{Cu+Tools}$	0.18	0.52	0.42
Ours	**0.07**	**0.01**	**0.06**

Component Ablation Table 3 varies verification and shortcut discovery for Phi-4. Verification is either off, restricted to solved subgraphs (*true only*), or applied to every generated element (*all*). The results highlight a strong dependency of shortcuts on verified elements in the graph. Without verification, the rather noisy Phi-4 suffers from poor macro-f1, and although enabling shortcuts will spend fewer tokens, their quality is low, as indicated by the f1-score. Always verifying brings substantial gains in accuracy, but at the same time incurs the highest token consumption. Here, shortcut discovery can save a lot of that work while retaining high f1. Still, verifying only once a full proof subgraph is assembled arguably saves even more, sacrificing four pp. f1-score. Adding shortcuts then improves that score and achieves a new best token efficiency.

Table 3: Combinatorial analysis of the verification and shortcut components in our framework using Phi-4.

Verify	Shortcut	#M Tokens	Macro-F1
none	✗	118M	28.3
none	✓	42M	21.9
all	✗	476M	83.1
all	✓	122M	84.4
true only	✗	107M	80.1
true only	✓	96M	84.2

Table 4: Results of the retrosynthesis task. We report success rates over six iteration budgets, the average number of iterations, as well as the mean reaction and molecule node counts. The latter three correspond to the search budget with 500 iterations.

Algorithm	Success rate [%] ↑						#Iterations ↓	# Reaction Nodes ↓	# Molecule Nodes ↓
	50	100	200	300	400	500			
Greedy DFS	-	38.42	40.53	44.21	45.26	46.84	300.56	-	-
Retro*	40.00	55.79	70.53	76.84	82.11	85.79	158.81	2632.84	3685.31
Retro*+-0	56.84	67.37	83.16	92.11	94.74	96.32	97.95	1444.52	2139.3
EG-MCTS	81.58	87.89	91.05	94.21	94.21	94.21	55.43	828.78	1091.65
RetroGraph	69.47	88.42	**97.89**	98.95	**99.47**	**99.47**	45.13	674.23	500.44
DreamRetro	-	93.16	96.32	97.37	97.89	98.42	-	574.42	480.12
PDVN	93.16	**96.84**	**97.89**	**99.47**	**99.47**	**99.47**	20.24	486.87	417.54
Ours	**93.68**	95.79	96.84	98.95	**99.47**	**99.47**	**19.12**	**317.01**	**246.55**

4.2 RETROSYNTHESIS

Retrosynthetic planning is a real world strategy of designing a synthesis by working backward from some target molecule to simpler, readily available precursors. This involves several tasks: identifying strategic bond disconnections, representing fragments as idealized synthons, mapping them to real reagents and reactions, and iterating until a feasible forward route is obtained. In the literature, such setups always require (a) a test set of target molecules, (b) a library of template reactions, and (c) a set of basic building blocks. Since the latter two are usually fixed across individual targets, reusing previously discovered molecules across different targets can save search costs, given that overlapping routes exist.

Setup and Implementation We select proven foundation models from the literature to generate actions and states. A state represents a molecule encoded as a SMILES string (Weininger, 1988) (e.g., L-Alanine, `N[C@@H](C)C(=O)O`), while an action is a reaction template encoded in SMARTS format (e.g., `[C:1](=[O:2])O>>[C:1](=[O:2])Cl`). We use the single-step model from (Chen et al., 2020) as action and state generator. For state evaluation, we adopt the training routine from (Hong et al., 2023) (EG-MCTS) to obtain a lightweight value network.

Several SOTA methods serve as baselines, in particular EG-MCTS (Hong et al., 2023), Retro* (Chen et al., 2020), Retro*-0 (Kim et al., 2021), RetroGraph (Xie et al., 2022), PDVN (Liu et al., 2023), and DreamRetro (Zhang et al., 2025b).

Metrics and Data We quantify effectiveness by the ratio of solved to unsolved targets (success rate). Efficiency is measured through node counts in the final search graphs. We follow common practice in the literature and use the same set of commercially available building blocks (approx. 23M) from eMolecules[2] for all approaches. For the test data, we adopt the 190 challenging target molecules (Retro-190) from Chen et al. (2020), as it is widely used in previous work.

4.3 RETROSYNTHESIS RESULTS

We match the state-of-the-art success rates while being the most efficient planner (Table 4). It solves 93.68% of targets at 50 iterations (best), 95.79% at 100 (second to PDVN), and achieves 99.47% at 500 iterations. More importantly, the graph size is substantially smaller: 317 reaction nodes and 247 molecule nodes on average, representing roughly 35% and 41% fewer nodes than PDVN, and over 50% fewer than RetroGraph and EG-MCTS. Overall, our method achieves SOTA success rates with the fewest iterations and smallest search graphs, validating reuse with multi-target planning as an effective paradigm for retrosynthesis.

[2] `https://downloads.emolecules.com/free/`

5 Related Work

Structured test-time reasoning. Chain-of-Thought prompting (Wei et al., 2022; Kojima et al., 2022) elicits a sequential reasoning trace for each input, while Self-Consistency (Wang et al., 2023) samples multiple traces and aggregates their answers. Tree of Thoughts (Yao et al., 2023), Graph of Thoughts (Besta et al., 2024), and recent adaptive variants (Pandey et al., 2025; Chi et al., 2025) generalize this view into explicit search over intermediate thoughts and richer reasoning topologies. These methods improve exploration within an instance, but their states are typically tied to the current trajectory or input rather than shared across equivalent subproblems.

Reusable reasoning procedures and programs. Least-to-Most prompting (Zhou et al., 2023) and Decomposed Prompting (Khot et al., 2023) decompose tasks into subquestions, while Program-of-Thoughts (Chen et al., 2023) and PIPS (Stein et al., 2025) use executable programs to separate symbolic computation from natural-language reasoning. SELF-DISCOVER (Zhou et al., 2024) composes task-specific reasoning structures from reusable modules, and Chain of Methodologies (Liu et al., 2025) guides extended reasoning through selected methodological prompts. These approaches reuse procedures, modules, or programs, whereas our framework reuses the intermediate states produced during search.

Experience-aware and memory-augmented reasoning. Memory-of-Thought and Thought Propagation reuse prior thought chains as in-context guidance (Li & Qiu, 2023; Yu et al., 2024), and retrieval-augmented reasoning methods retrieve relevant context or thought structures for long-horizon generation (Wang et al., 2024; Zhang et al., 2025a). Other systems store templates, strategies, code snippets, or distilled schemas from past instances (Yang et al., 2024; Suzgun et al., 2026; Zhou et al., 2026), organize prior steps into thought graphs (Ahmed et al., 2026), or distill strategies from agent trajectories (Ouyang et al., 2026). These systems make inference experience-aware, but the retrieved objects usually guide a new generation rather than serving as already-verified states shared by multiple proof trajectories.

Efficiency through state reuse. Efficiency-oriented methods reduce the cost of reasoning through shorter traces (Xu et al., 2025), parallel generation (Ning et al., 2024), or better allocation of test-time compute (Snell et al., 2025; Muennighoff et al., 2025). Our approach is orthogonal: instead of only making each trajectory cheaper, it changes the search object from isolated trajectories to a shared AND-OR graph. Equivalent Markovian states are merged across attempts and across problem instances, proof status propagates through the graph, and active shortcut discovery asks the generator to connect unsolved states to already useful regions of the graph. Prior work largely reuses knowledge about how to reason; our framework reuses reached reasoning states themselves.

6 Conclusion

Structured search powered by imperfect probabilistic generators can be beneficially enhanced to reuse its own progress in related problem instances by switching to a more suitable state representation. Our framework allows to employ global search strategies over all samples of tasks with reusable sub-structures. We introduce one such strategy in the form of Active Shortcut Discovery, a novel technique that actively infers connections between acquired intermediate solutions and rapidly closes gaps in the search space. We show that this reuse technique paired with focused self-verification significantly improves Macro-F1 scores and at the same time reduce resource consumption. By introducing novel unsolvable problem instances into our datasets, we are able to study a new aspect concerning the evaluation of these systems. Surprisingly, many of the previous methods tested by us have a much stronger tendency to produce a positive answer, usually by means of hallucinating illegal transitions. In contrast, our proposed method is able to maintain high Macro-F1 scores even when the shared reasoning graph grows to hundreds of thousands of states.

AI use statement

In this work, we used generative AI tools for assistance with the language of the paper and LaTeX formatting. In particular, language help only included author-verified and -edited suggestions for reformulating and polishing manually written text as well as checking for grammar and spelling errors. Also, we have used generative AI to assist with setting up GPU cluster allocation, and job

scheduling of the author-defined experiments. We have not used generative AI tools for literature search, automatically drafting sections of the paper, the generation of our datasets or the mathematical proofs. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Ammar Ahmed, Azal Ahmad Khan, Ayaan Ahmad, Sheng Di, Zirui Liu, and Ali Anwar. Retrieval-of-thought: Efficient reasoning via reusing thoughts. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, pp. 99141–99164, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/a10da26f47120217c1b7c2aeb2979048-Abstract-Conference.html.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan (eds.), *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17682–17690. AAAI Press, 2024. doi: 10.1609/aaai.v38i16.29720. URL https://ojs.aaai.org/index.php/AAAI/article/view/29720.

Binghong Chen, Chengtao Li, Hanjun Dai, and Le Song. Retro*: Learning retrosynthetic planning with neural guided A* search. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1608–1616. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/chen20k.html.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=YfZ4ZPt8zd.

Yizhou Chi, Kevin Yang, and Dan Klein. ThoughtSculpt: Reasoning with intermediate revision and search. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 7700–7726, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.428. URL https://aclanthology.org/2025.findings-naacl.428/.

Siqi Hong, Hankz Hankui Zhuo, Kebing Jin, Guang Shao, and Zhanwen Zhou. Retrosynthetic planning with experience-guided Monte Carlo tree search. *Communications Chemistry*, 6(1):120, 2023. ISSN 2399-3669. doi: 10.1038/s42004-023-00911-8. URL https://www.nature.com/articles/s42004-023-00911-8.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=_nGgzQjzaRy.

Junsu Kim, Sungsoo Ahn, Hankook Lee, and Jinwoo Shin. Self-improved retrosynthetic planning. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 5486–5495. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/kim21b.html.

Takeshi Kojima, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 22199–22213. Curran Associates, Inc., 2022. doi: 10.52202/068431-1613. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/8bb0d291acd4acf06ef112099c16f326-Abstract-Conference.html.

Xiaonan Li and Xipeng Qiu. MoT: Memory-of-thought enables ChatGPT to self-improve. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 6354–6374, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.392. URL https://aclanthology.org/2023.emnlp-main.392/.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. Making language models better reasoners with step-aware verifier. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5315–5333, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.291. URL https://aclanthology.org/2023.acl-long.291/.

Cong Liu, Jie Wu, Weigang Wu, Xu Chen, Liang Lin, and Wei-Shi Zheng. Chain of methodologies: Scaling test time computation without training. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 5298–5312, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.276. URL https://aclanthology.org/2025.findings-acl.276/.

Guoqing Liu, Di Xue, Shufang Xie, Yingce Xia, Austin Tripp, Krzysztof Maziarz, Marwin Segler, Tao Qin, Zongzhang Zhang, and Tie-Yan Liu. Retrosynthetic planning with dual value networks. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 22266–22276. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/liu23as.html.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. doi: 10.52202/075280-2019. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Katerina Margatina, Timo Schick, Nikolaos Aletras, and Jane Dwivedi-Yu. Active learning principles for in-context learning with large language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5011–5034, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.334. URL https://aclanthology.org/2023.findings-emnlp.334/.

Amil Merchant, Simon Batzner, Samuel S. Schoenholz, Muratahan Aykol, Gowoon Cheon, and Ekin Dogus Cubuk. Scaling deep learning for materials discovery. *Nature*, 624(7990):80–85, 2023. doi: 10.1038/s41586-023-06735-9. URL https://www.nature.com/articles/s41586-023-06735-9.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20275–20321, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1025. URL https://aclanthology.org/2025.emnlp-main.1025/.

Nils J. Nilsson. *Principles of Artificial Intelligence*. Symbolic computation. Springer Berlin, Heidelberg, 1982. ISBN 978-3-540-11340-9. URL https://link.springer.com/book/9783540113409.

Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. Skeleton-of-thought: Prompting LLMs for efficient parallel generation. In

B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, pp. 917–967, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/03d7e13f0092405804f3a381ade8f3f0-Abstract-Conference.html.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, pp. 94327–94354, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/980ea04d23d1f6908964eba2a74afe45-Abstract-Conference.html.

Tushar Pandey, Ara Ghukasyan, Oktay Goktas, and Santosh Kumar Radha. Adaptive graph of thoughts: Test-time adaptive reasoning unifying chain, tree, and graph structures. arXiv preprint arXiv:2502.05078, 2025. URL https://arxiv.org/abs/2502.05078.

Bilgehan Sel, Ahmad Tawaha, Vanshaj Khattar, Ruoxi Jia, and Ming Jin. Algorithm of thoughts: Enhancing exploration of ideas in large language models. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 44136–44189. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/sel24a.html.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, pp. 10131–10165, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/1b623663fd9b874366f3ce019fdfdd44-Abstract-Conference.html.

Adam Stein, Neelay Velingker, Mayur Naik, and Eric Wong. Once upon an input: Reasoning via per-instance program synthesis. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 136940–136973. Curran Associates, Inc., 2025. doi: 10.52202/085713-4571. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/c828f33af3c3f669690c2e28ae7af5e2-Abstract-Conference.html.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. Challenging BIG-Bench tasks and whether chain-of-thought can solve them. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.824. URL https://aclanthology.org/2023.findings-acl.824/.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In Vera Demberg, Kentaro Inui, and Lluís Marquez (eds.), *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106, Rabat, Morocco, March 2026. Association for Computational Linguistics. ISBN 979-8-89176-380-7. doi: 10.18653/v1/2026.eacl-long.333. URL https://aclanthology.org/2026.eacl-long.333/.

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

Zihao Wang, Anji Liu, Haowei Lin, Jiaqi Li, Xiaojian Ma, and Yitao Liang. RAT: Retrieval augmented thoughts elicit context-aware reasoning in long-horizon generation. arXiv preprint arXiv:2403.05313, 2024. URL https://arxiv.org/abs/2403.05313.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022. doi: 10.52202/068431-1800. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36, 02 1988. ISSN 0095-2338. doi: 10.1021/ci00057a005. URL https://pubs.acs.org/doi/10.1021/ci00057a005.

Shufang Xie, Rui Yan, Peng Han, Yingce Xia, Lijun Wu, Chenjuan Guo, Bin Yang, and Tao Qin. RetroGraph: retrosynthetic planning with graph search. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2120–2129. Association for Computing Machinery, 2022. doi: 10.1145/3534678.3539446. URL https://dl.acm.org/doi/10.1145/3534678.3539446.

Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. Chain of draft: Thinking faster by writing less. arXiv preprint arXiv:2502.18600, 2025. URL https://arxiv.org/abs/2502.18600.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E. Gonzalez, and Bin Cui. Buffer of thoughts: Thought-augmented reasoning with large language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 113519–113544. Curran Associates, Inc., 2024. doi: 10.52202/079017-3607. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/cde328b7bf6358f5ebb91fe9c539745e-Abstract-Conference.html.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822. Curran Associates, Inc., 2023. doi: 10.52202/075280-0517. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html.

Junchi Yu, Ran He, and Rex Ying. Thought propagation: an analogical approach to complex reasoning with large language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, pp. 10989–11015, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/2dae7d1ccf1edf76f8ce7c282bdf4730-Abstract-Conference.html.

Jinghan Zhang, Xiting Wang, Weijieying Ren, Lu Jiang, Dongjie Wang, and Kunpeng Liu. RATT: A thought structure for coherent and correct LLM reasoning. In Toby Walsh, Julie Shah, and Zico Kolter (eds.), *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 26733–26741. AAAI Press, 2025a. doi: 10.1609/aaai.v39i25.34876. URL https://ojs.aaai.org/index.php/AAAI/article/view/34876.

Xuefeng Zhang, Haowei Lin, Muhan Zhang, Yuan Zhou, and Jianzhu Ma. A data-driven group retrosynthesis planning model inspired by neurosymbolic programming. *Nature Communications*, 16(1):192, 2025b. doi: 10.1038/s41467-024-55374-9. URL https://www.nature.com/articles/s41467-024-55374-9.

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

Mingjun Zhou, Weixin Zeng, and Xiang Zhao. Experience is the best teacher: Augmenting LLM reasoning with knowledge learned from the past. In *Proceedings of the ACM Web Conference 2026*, pp. 2361–2372. Association for Computing Machinery, 2026. doi: 10.1145/3774904.3792630. URL https://dl.acm.org/doi/10.1145/3774904.3792630.

Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, Denny Zhou, Swaroop Mishra, and Huaixiu Steven Zheng. SELF-DISCOVER: Large language models self-compose reasoning structures. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 126032–126058. Curran Associates, Inc., 2024. doi: 10.52202/079017-4004. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/e41efb03e20ca3c231940a3c6917ef6f-Abstract-Conference.html.

Maxim Zvyagin, Alexander Brace, Kyle Hippe, Yuntian Deng, Bin Zhang, Cindy Orozco Bohorquez, Austin Clyde, Bharat Kale, Danilo Perez-Rivera, Heng Ma, Carla M. Mann, Michael Irvin, Defne G. Ozgulbas, Natalia Vassilieva, James Gregory Pauloski, Logan Ward, Valerie Hayot-Sasson, Murali Emani, Sam Foreman, Zhen Xie, Diangen Lin, Maulik Shukla, Weili Nie, Josh Romero, Christian Dallago, Arash Vahdat, Chaowei Xiao, Thomas Gibbs, Ian Foster, James J. Davis, Michael E. Papka, Thomas Brettin, Rick Stevens, Anima Anandkumar, Venkatram Vishwanath, and Arvind Ramanathan. GenSLMs: genome-scale language models reveal SARS-CoV-2 evolutionary dynamics. *The International Journal of High Performance Computing Applications*, 37(6):683–705, 2023. doi: 10.1177/10943420231201154. URL https://journals.sagepub.com/doi/10.1177/10943420231201154.

A Technical Appendices and Supplementary Material

A.1 Proofs

Truth propagation in cyclic AND-OR graphs In the following, we use Knaster-Tarski and Kleene fixed point theorems to show that there always exists a least fixed point that can be reached with synchronous iteration. We start with our graph definition as presented in Section 2. We have a directed bipartite graph $G = (S \cup A, E)$ with state vertices S, action vertices A, and edges $E \subseteq (S \times A) \cup (A \times S)$. Let $V = S \cup A$. We partition V into nonterminal vertices B and terminal vertices, which we further differentiate by unsolved (T_0) and solved (T_1) status. We count the number of nonterminal nodes as $n = \mid B \mid$. For $x \in \{0,1\}^n$, we define its extension $\mathrm{ext}(x) \in \{0,1\}^{|V|}$ by

$$\mathrm{ext}(x)_i = \begin{cases} x_i, & i \in B, \\ 0, & i \in T_0, \\ 1, & i \in T_1 \end{cases}$$

For each $j \in B$, let g_j be the Boolean expression induced by the bipartite AND-OR semantics:

$$g_j(z) = \begin{cases} \bigvee_{(j,a)\in E} z_a, & j \in S, \\ \bigwedge_{(j,s)\in E} z_s, & j \in A. \end{cases}$$

Empty neighborhoods are represented by the corresponding constants (false for empty disjunctions, true for empty conjunctions), so each g_j is built only from $\wedge, \vee$, and constants. Further, define

$$F_j(x) = g_j\big(\mathrm{ext}(x)\big), F : \{0,1\}^n \to \{0,1\}^n$$

and let $\leq$ be the coordinatewise partial order on $\{0,1\}^n$: for $x, y \in \{0,1\}^n$ we write $x \leq y$ iff $x_i \leq y_i, \forall i \in B$.

Lemma 1 (Monotonicity with arbitrary constants)**.** *F is monotone with respect to the coordinatewise order on $\{0,1\}^n$.*

Proof. The map ext is monotone in x, and $\wedge, \vee$ are monotone in each argument. Constants in g_j (0 or 1) do not affect monotonicity. Hence $x \leq y \Rightarrow F(x) \leq F(y)$. □

Theorem 1 (Synchronous iteration from the bottom converges to the least fixed point)**.** *Consider the sequence $x^0 = \mathbf{0}$ (the all-zero vector on B) and $x^{k+1} = F(x^k), k \geq 0$. Then:*

1. *The sequence is nondecreasing:* $x^0 \leq x^1 \leq x^2 \leq \cdots$.
2. *It stabilizes in at most* n *steps:* $\exists m \leq n : x^{m+1} = x^m$, x^m *is a fixed point of* F.
3. x^m *is the least fixed point of* F *with respect to* $\leq$.

Proof. (1) Since $\mathbf{0} \leq F(\mathbf{0})$, we have $x^0 \leq x^1$. If $x^k \leq x^{k+1}$, then by monotonicity $F(x^k) \leq F(x^{k+1})$, i.e. $x^{k+1} \leq x^{k+2}$.

(2) Since the sequence is non-decreasing, no coordinate ever flips from 1 to 0. Thus, there can be at most n strict increases. So, for some $m \leq n$ we must have $x^{m+1} = x^m$ (fixed point).

(3) Let y be any fixed point. Since $0 \leq y$ and F is monotone, for all k we have $F^k(0) \leq F^k(y) = y$. With $x^k = F^k(0)$, we get $x^k \leq y$ for all k, hence $x^m \leq y$. □

Remark 1 (Arbitrary seeds can oscillate under synchronous iteration)**.** *Although not applicable to our scenario, there exist 2-cycles for arbitrary seeds, even with monotone* F*. E.g. two* `Ox` *gates feeding into each other starting from the seed* $(0, 1)$ *will swap to* $(1, 0)$*, then back to* $(0, 1)$ *and so on.*

Incremental maintenance across graph transactions. Consider a sequence of graph transactions $(T_0, T_1, \ldots)$, let $G_i = (V_i, E_i)$ denote the live reasoning graph after transaction T_i, let

$$B_i = V_i \setminus (T_{0,i} \cup T_{1,i})$$

be its nonterminal vertices, and let

$$F_i : \{0,1\}^{|B_i|} \to \{0,1\}^{|B_i|}$$

be its truth operator. We write $x^{(i)}$ for the least fixed point of F_i.

An expansion may create states, create an action node (representing a hyperedge), and insert the edges connecting the action to its parent and child states. A retraction removes an action from the live graph together with its incident edges. A transaction may change the terminal class of a state.

For the transition from G_i to G_{i+1}, we consider the set of seed states $D_i \subseteq V_{i+1}$ to contain:

1. every new state,
2. every surviving state whose terminal class changes,
3. every surviving nonterminal state whose outgoing neighborhood changes.

Thus, the source of every inserted or deleted edge is a seed whenever that source survives the transaction. Deleted vertices themselves are not coordinates of F_{i+1}. However, their remaining parents are seeds because their outgoing neighborhoods change.

Let $K_i \subseteq V_{i+1}$ be the reversed dependency cone of D_i:

$$K_i = \{u \in V_{i+1} : \text{there exists a directed path from } u \text{ to some } d \in D_i\}.$$

Seeds are included. Since an edge (u, v) means that u depends on v, K_i contains exactly the states whose values may be affected by the transaction.

If $u \notin K_i$, then u is not a seed and cannot possibly be affected by a state in K_i. Otherwise u would itself reach a seed. Consequently, its local Boolean expression is unchanged and depends only on other vertices outside K_i.

Next, we consider the starting truth values for next least fixed point computation after one graph edit. Define an assignment y^0 on B_{i+1} by

$$y^0_j = \begin{cases} 0, & j \in K_i \cap B_{i+1}, \\ x^{(i)}_j, & j \in B_{i+1} \setminus K_i. \end{cases}$$

All states in the affected cone of the edit are set to false, otherwise we keep the value from the iteration before.

Starting from y^0, define

$$y^{k+1} = F_{i+1}(y^k).$$

The reset inside K_i is necessary after deletions. A deletion may remove a justification, so the previous fixed point need not be below its image under F_{i+1}.

Lemma 2 (Correctness of cone-local recomputation)**.** *The sequence $(y^k)_{k\geq 0}$:*

1. *is nondecreasing,*
2. *never changes a state outside K_i,*
3. *and stabilizes at $x^{(i+1)}$, the least fixed point of F_{i+1}.*

Proof. For every $j \in K_i \cap B_{i+1}$, we have $y^0_j = 0$, and hence

$$y^0_j \leq F_{i+1}(y^0)_j.$$

Now consider $j \notin K_i$. Its local expression did not change and does not depend on K_i. Its children therefore have the same value under y^0 as under $x^{(i)}$. Since $x^{(i)}$ is a fixed point of F_i,

$$F_{i+1}(y^0)_j = F_i(x^{(i)})_j = x^{(i)}_j = y^0_j.$$

Therefore $y^0 \leq F_{i+1}(y^0)$ and monotonicity gives $y^0 \leq y^1 \leq y^2 \leq \cdots$.

Moreover, the preceding equality shows that values outside K_i do not change in the first iteration. Since such values only depend on other states outside K_i, the same argument applies inductively at every iteration. Therefore, K_i contains all values that can possibly change.

Next, we show that the new fixed point is the same as a full restart from a fully zero-initialized y^0 would produce. Every value equal to 1 in $x^{(i)}$ has a finite derivation from solved terminals because $x^{(i)}$ is obtained by iteration from the bottom. Let $j \notin K_i$ with $x^{(i)}_j = 1$. No state in such a derivation can belong to K_i: if j reached a vertex in K_i, and that vertex reached a seed, then j would also belong to K_i. Therefore the complete derivation of j lies outside K_i, where all local expressions and terminal classes are unchanged. That derivation remains in G_{i+1}, so $y^0 \leq x^{(i+1)}$ and by monotonicity, $y^k \leq x^{(i+1)}$ for every k.

Because the graph is finite and the sequence is nondecreasing, it stabilizes at some assignment z. This assignment is a fixed point of F_{i+1}. Since $x^{(i+1)}$ is the least fixed point, $x^{(i+1)} \leq z$. On the other hand, the preceding induction gives $z \leq x^{(i+1)}$ and therefore $z = x^{(i+1)}$. □

Thus, resetting and recomputing only the affected dependency cone of a transaction results in exactly the same truth assignment as recomputing the entire graph from the bottom.

Maintenance complexity. For K_i, define

$$E(K_i) = \{(u, v) \in E_{i+1} : u \in K_i\}.$$

These are the edges whose source lies in the affected cone. Let $|\Delta_i|$ denote the number of vertices, edges, and terminal classifications changed by transaction T_{i+1}.

Theorem 2 (Affected-cone maintenance cost)**.** *Assume that the graph maintains forward and reverse adjacency indices together with, for every action, its number of false premises and, for every state, its number of true supporting actions. Then:*

1. *recomputing the least fixed point from scratch costs*

$$O(|V_{i+1}| + |E_{i+1}|),$$

2. *incrementally maintaining the least fixed point costs*

$$O(|\Delta_i| + |K_i| + |E(K_i)|)\,.$$

Proof. For the full recomputation, initialize all nonterminal values to false. For each action, store the number of false child states and for each state, store the number of true outgoing actions. Counters are initialized by a full state and edge scan once.

Whenever a state changes from false to true, we update the counters of its parents. An action becomes true when its false-premise count reaches zero, and a state becomes true when its true-support count becomes positive. Each state changes from false to true at most once, and each edge causes at most one constant-time counter update. The total cost is therefore

$$O(|V_{i+1}| + |E_{i+1}|).$$

For incremental maintenance, we first process the transaction and identify its seeds, costing $O(|\Delta_i|)$. Then, we construct K_i by traversing edges upwards from the seeds. Every such parent belongs to K_i, so traversal visits only vertices in K_i and edges in $E(K_i)$.

Then, we reset the states' values in K_i, rebuild their local counters, and propagate truth from unchanged inputs outside K_i and solved terminals. Again, every affected state changes from false to true at most once, and every edge in $E(K_i)$ causes at most a constant number of counter operations. The resulting cost is

$$O(|\Delta_i| + |K_i| + |E(K_i)|)\,.$$

Correctness follows from Lemma 2. □

For comparison, direct synchronous iteration from the bottom may require $|B_{i+1}|$ rounds, each scanning the complete graph, and therefore has the weaker bound

$$O\big(|B_{i+1}|\big(|V_{i+1}| + |E_{i+1}|\big)\big)\,.$$

This is a bound for naïve synchronous evaluation, not for worklist-based full recomputation.

Finally, over a series of transactions $T_1, \ldots, T_t$, full worklist recomputation costs

$$\sum_{i=0}^{t-1} O(|V_{i+1}| + |E_{i+1}|)\,,$$

whereas incremental maintenance costs

$$\sum_{i=0}^{t-1} O(|\Delta_i| + |K_i| + |E(K_i)|)\,.$$

These collapse to the same bound when every transaction affects the entire graph. For example, this can happen when changing the solved terminal at the end of a directed chain (e.g. CoT-like reasoning traces), as every state is in the affected cone, and any correct algorithm must update $\Theta(|V|)$ truth values. When a transaction is confined to one branch of a wider graph, however, incremental maintenance avoids revisiting the unaffected parts.

B LIMITATIONS

State identity. Our current notion of reuse relies on exact equality of the state representations returned by REPR. This choice makes state merging simple and keeps the effect of the shared graph easy to attribute, but it also means that semantically equivalent states are not reused unless they are rendered identically. Future versions could replace this strict token-level identity with canonicalization, learned equivalence functions, or lifted templates that map instance-specific states to parameterized representations while preserving the AND-OR semantics.

Markovian state assumptions. The soundness of reuse depends on the state representation containing all information relevant for subsequent actions, evaluation, and solution extraction. This Markovian assumption is natural for many decomposable search problems, but it can be difficult to satisfy in domains with implicit constraints, long histories, or large external context. In such cases, states may need to carry richer information, which can reduce merge rates and increase the cost of comparing or expanding states.

When reuse helps. The framework is intended for tasks with overlapping subproblems, reusable intermediate states, or structurally similar batches of inputs. For one-off problems whose useful trajectories rarely intersect, maintaining a shared graph may offer little benefit over ordinary trajectory-based search. The method therefore requires some prior expectation that repeated substructure exists and can be exposed by the chosen state representation.

Task-specific instantiations. Several components of our method remain task-specific. In particular, a new domain requires a suitable REPR, an ISSOLUTION predicate, instructions for the generator, and a shortcut ranker. Poor choices for any of these components can limit reuse or spend generator calls on states and shortcuts that are unlikely to help.

Shortcut discovery. Active shortcut discovery only approximates the space of useful connections. Even when restricted to single solved targets, the number of candidate queries scales with the product of open and solved states, so the method must rank and query only a small subset. Its benefit therefore depends on both the quality of this ranking and the generator's ability to synthesize valid actions that actually connect the proposed states. The evolving graph may provide useful supervision for improving such rankers, for example through active learning (Margatina et al., 2023).

Graph overhead. The shared graph introduces bookkeeping that tree-style prompting does not require. Our method must maintain shared state and action nodes, propagate truth values, schedule expansions, evaluate new candidates, and periodically select shortcut targets. We expect this overhead to pay off when reuse is frequent enough to amortize it, but it may be unnecessary for small instances or domains with little overlap.

Verification and error propagation. Because solved states and proof fragments can be reused across many trajectories, an incorrectly accepted action can affect more than a single local branch. Our method mitigates this risk by verifying proof-relevant actions before returning solutions, but ultimately this depends on the reliability limits of the verifier.