Autonomous Topological Reasoning with Large Language Models

Pedro Gimenes [1] **Zeyu Cao** [2] **Aaron Zhao** [1]

Abstract

Topological reasoning with Large Language Models (LLMs) emulates human-like thought processes by efficiently exploring various reasoning paths over thought trees or graphs. However, prior works rely on static, task-specific prompting schedules to decompose problems and synthesize solutions, which are subject to a hyperparameter set requiring extensive search for high query efficiency. Additionally, the task-specific requirement restricts generalization to novel problem domains and fails to adapt to varying problem complexities. In our work, we investigate the ability of LLMs to guide thought graph exploration in a multi-agent architecture with policy agents and reasoning agents. While reasoning agents solve decomposed subproblems, LLM policy agents maintain visibility of the reasoning trace, dynamically adaptating the problem-solving strategy. Through extensive controlled experiments, we observe that in problems with low decomposition depth, LLM-guided exploration can match or even outperform static schedules by up to $3.3\times$, without any search time required. At high decomposition depths, existing LLMs incur performance deterioration of up to $4.4\times$, highlighting the requirement for new solutions to enable more flexible and generalizable topological reasoning with LLMs.

1. Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities in a range of natural language processing tasks including language translation (Feng et al., 2020), question-answering (Yang et al., 2020) and sentiment analysis (Liu et al., 2021). Additionally, recent work has characterised the unpredictable emergence of abilities in LLMs as their parameter count grows (Wei et al., 2022). For instance, arithmetic reasoning abilities appear to manifest in GPT architectures beyond 13B parameters, suggesting the existence of capability thresholds. Despite growing evidence of these scaling trends, existing models continue to exhibit low performance in non-trivial reasoning tasks requiring extensive state-space exploration and strategic planning.

This performance gap can be understood by viewing LLM reasoning through the lens of cognitive science, which postulates that humans engage in two distinct modes of reasoning - a fast, intuitive mode (System 1) and a slow, deliberate mode (System 2). The autoregressive decoding procedure of LLMs can be viewed as operating in System 1, and recent research has focused on eliciting reasoning behaviour from LLMs by engaging models in System 2. Wei et al. 2023 pioneered the elicitation of step-by-step logical reasoning (CoT), with subsequent work by Wang et al. 2023 demonstrating improved performance through the sampling and arbitration along multiple reasoning sequences (CoT-SC). Yao et al. 2023a formulate concurrent exploration of multiple reasoning paths by scoring reasoning steps, leveraging tree search algorithms (ToT). Besta et al. 2024 generalize problem space exploration by formulating thoughts as a graph, enabling the use of arbitrary transformations such as node refinement and aggregation (GoT).

Despite the potential of structure-enhanced exploration for engaging LLMs in deeper levels of reasoning, prior works rely on predetermined traversal strategies parametrized by a discrete set of hyperparameters. This approach lacks generality, as these parameters must be tuned manually or through extensive bayesian search to achieve high query efficiency, due to the varying characteristics of each task. *We hypothesize that the generalization of artificial problem-solving towards (or beyond) human-like abilities in arbitrary domains requires a mechanism for autonomous exploration, falling outside the constrained scope of static schedules*. Such a mechanism would present a step towards general intelligent agents capable of leveraging existing world knowledge while adapting to out-of-distribution tasks.

Motivated by recent improvements in LLM planning and reasoning, we aim to investigate whether existing LLMs are capable of acting as autonomous reasoning agents by formulating thought graphs as interactive environments. We

[*]Equal contribution [1]Department of Electrical & Electronic Engineering, Imperial College London, London, United Kingdom [2]Department of Computer Science and Technology, University of Cambridge, Cambridge, England. Correspondence to: Pedro Gimenes <pg519@ic.ac.uk>.

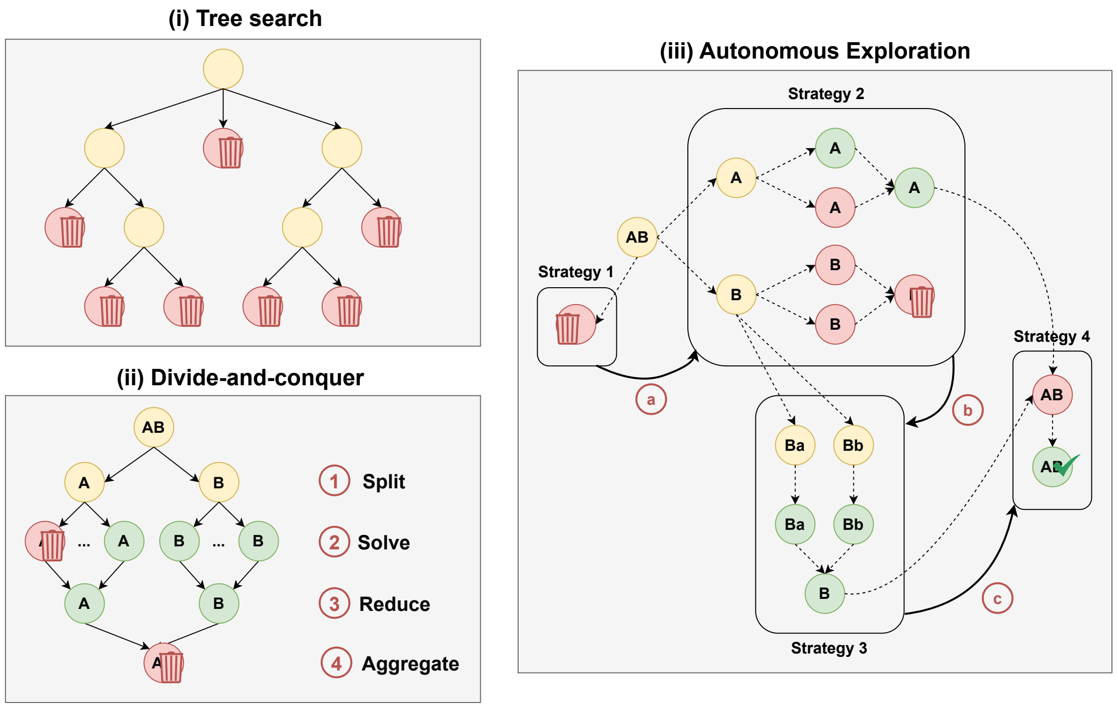


Figure 1. Example thought graph in solving a reasoning problem following (i) tree search, (ii) graph-based and (iii) autonomous exploration schedules. In (i), each node is an attempted solution, with the best-scoring nodes iteratively chosen and refined. In (ii), solutions to subproblems are merged to solve the original problem. In (iii) our approach, exploration is guided by an LLM policy agent. After a direct solution (Strategy 1) is unsuccessful, the agent attempts a divide-and-conquer approach (Strategy 2), displaying (a) **adaptive decomposition**. Subproblem B is incorrectly solved, leading the agent to (b) **self-correct** by decomposing it further before (c) aggregating.

equip LLM policy agents (i.e. LLM-based action planners) with the capacity to autonomously perform transformations on this graph such as thought proposal/evaluation, aggregation and refinement. As such, we consider the following research questions: **(1)** Can LLMs effectively adapt to feedback from thought graph environments to dynamically tune their exploration strategy? **(2)** can this approach match the performance of static exploration schedules extensively optimized for a given task? And finally, **(3)** what are the failure modes of existing LLMs in guiding thought graph exploration (i.e. factors affecting the ability to produce coherent exploration plans)?

Figure 1 presents an example of our findings regarding question **(1)**, by contrasting an extracted trace of our approach against static tree search and graph transformation schedules. In tree search, each node is an attempted solution, with the best-scoring nodes kept and refined at each tree level (Yao et al., 2023a). The graph transformation schedule involves decomposing the starting problem into smaller subproblems, solving them individually then merging the intermediate results (Besta et al., 2024). Both are subject to redundancy, as the exploration is constrained by the schedule parameters, impacting prompt efficiency - the system is unable to exit early when a solution is found, or adaptively allocate resources according to the complexity of subproblems. In our approach, we leverage the planning abilities of LLMs to choose a transformation action given the current state of the thought graph. We observe the policy agent displays **adaptive decomposition** and **self-correction**, meaning the depth of decomposition and query utilization are dynamically tuned according to the complexity of the task.

In summary, our contributions are as follows.

- We present an algorithmic formulation of topological reasoning schedules subject to a set of hyperparameters, which generalizes to most existing static approaches.
- We propose a framework for autonomous topological reasoning by formulating thought graphs as interactive environments with which LLM agents can interact by planning and executing graph transformations.
- We perform carefully controlled experiments, showing that LLM-guided thought graph exploration is highly sensitive to decomposition depth, with policy agents outperforming optimized static schedules at low depths by up to 3.3× (despite no search cost).

2. Related Work

LLMs as Planning Agents

Significant research has focused on leveraging LLMs for guiding action policies, such as in tasks requiring coordination of heterogeneous model ensembles (Shen et al., 2023). LLMs have also been deployed as action planners in interactive environments (i.e. where feedback is provided to the action scheduler), such as solving computer tasks (Kim et al., 2023) and online shopping (Yao et al., 2023b). However, some works have outlined the instability in obtaining action plans over long-range horizons, where LLMs have been shown to repeatedly generate invalid action plans (Xie et al., 2023). This limitation has been tackled by works such as (Shinn et al., 2023), which propose an episodic memory buffer of previous trials, prompting LLMs to learn from past experiences.

Topological Reasoning with LLMs

Following the introduction of topological reasoning with ToT (Yao et al., 2023a) and GoT (Besta et al., 2024), several works explored methods of improving their query efficiency, which suffer from high computational demand due to iterative LLM prompting. (Hu et al., 2023) proposed combining one-shot generation (i.e. LLM decoding without iterative prompting) with ToT search in visual reasoning benchmarks. (Sel et al., 2024) compresses thought graph exploration traces as in-context examples, prompting LLMs to reproduce the exploration in fewer queries, although this approach is not suitable for interactive environments . Finally, (Ding et al., 2024) combines pre-trained reinforcement learning with Monte-Carlo search on thought graphs to generate solution paths with minimal LLM intervention, although this requires extensive training and lacks task generality.

Despite improvements in query efficiency, few works have targeted the generality of topological reasoning by exploring dynamic exploration schedules. While (Yao et al., 2023a) leverage standard tree search algorithms, (Long, 2023) hypothesize that tree search can be enhanced through trained policy networks to guide node backtracking. However, this idea is not explored fully and their evaluation is focused on heuristics-based rules. As such, our work presents the first effort towards generalized topological reasoning through autonomous thought graph exploration.

3. Static Exploration Schedules

We consider a reasoning problem to be stated in language as an ordered tuple of tokens $p = (t_1, ..., t_m)$, where each token $t \in \mathcal{V}$ belongs to a vocabulary space $\mathcal{V}$. We define a thought $\tau = (t_1, ..., t_j)$ as a sequence of tokens sampled autoregressively from an LLM parametrized by θ following Equation 1. This consists of a language representation of an intermediate step towards the solution to the problem.

$$t_i \sim P(t_i \mid t_1, \ldots, t_{i-1}; \theta) \tag{1}$$

A thought sequence can be represented as an ordered tuple of thoughts $S = (\tau^1, \tau^2, ..., \tau^k)$ of length k, such that the final thought τ^k represents a candidate solution to the problem p. A thought tree T_r can be represented as $(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of thought vertices and $\mathcal{E}$ is a set of edges connecting them. The tree can be parametrized with a depth of d and a width of w, denoting the number of nodes per level. Additionally, each thought τ^{ij} (j-th thought at depth i) has a value $\lambda(\tau^{ij})$ corresponding to the probability of reaching a valid solution for the starting problem. Hence, tree-based thought exploration involves finding a path $\mathcal{T} \in \mathcal{V}$ that maximizes the cumulative value of thoughts, as follows.

$$T_r^* = \arg\max_{\mathcal{T}} \sum_{\tau \in \mathcal{T}} \lambda(\tau) \tag{2}$$

A thought graph G_t can also be represented via the tuple $(\mathcal{V}, \mathcal{E})$, with no imposed restriction on the arrangement of thoughts and edges. Thought graph exploration can be regarded as a sequence of m graph transformations as shown in Equation 3, where each $\phi_i : G_t^i \to G_t^{i+1}$ modifies the set of nodes and edges. The full set of considered transformations and their formulations are shown in Table 1.

$$G_t^* = \phi_1 \circ \phi_2 \circ ... \circ \phi_m(G_t^0) \tag{3}$$

A static thought graph exploration schedule can be composed as shown in Algorithm 1, and parametrized by the

Table 1. Thought graph transformations. Each transformation is defined as $\phi(G_t, m, S) = (V \cup V^+ \setminus V^-, E \cup E^+ \setminus E^-)$, where $G_t = (V, E)$ is a thought graph, $S \subset V$ is a subset of nodes, m is the multiplicity (number of attempts), and $\mathcal{E}$, $\mathcal{R}$, $\mathcal{A}$ represent arbitrary functions for node expansion, refinement and aggregation, respectively. The sets V^+, V^-, E^+, E^- are defined as follows.

Transformation	Symbol	$\mathbf{V}^+$	$\mathbf{V}^-$	$\mathbf{E}^+$	$\mathbf{E}^-$
Decompose	ϕ_{dec}	$\{\mathcal{E}(v) \mid v \in S\}$	$\emptyset$	$\{(u,v) \mid u \in S, v \in V^+\}$	$\emptyset$
Solve	ϕ_{sol}	$\{\mathcal{S}(v) \mid v \in S\}$	$\emptyset$	$\{(u,v) \mid u \in S, v \in V^+\}$	$\emptyset$
Refine	ϕ_{ref}	$\{\mathcal{R}(t) \mid t \in S\}$	$\emptyset$	$\{(u,v) \mid u \in S, v \in V^+\}$	$\emptyset$
Reduce	ϕ_{red}	$\emptyset$	S	$\emptyset$	$\{(u,v) \mid u \in S \vee v \in S\}$
Aggregate	ϕ_{agg}	$\mathcal{A}(S)$	$\emptyset$	$\{(u,v) \mid u \in S, v \in V^+\}$	$\emptyset$

tuple $(B, R_{ed}, R_{ef}, S^m, A^m, R_{ef}^m)$. $B \in \mathcal{N}$ represents the depth of decomposition of the original problem, such that ϕ_{dec} splits the original problem into nodes $M_1, ..., M_B$. $N_1, ..., N_B$ represent the solution nodes to each of the subproblems. A_m represents the multiplicity of aggregation, i.e. the number of aggregation attempts, such that O_k represents the kth aggregation attempt. $R_{ed}, R_{ef} \in \{0, 1\}$ indicate whether the ϕ_{red} and ϕ_{ref} transformations are applied after branch aggregation, respectively. R_{ef}^m represent the multiplicity of refinement, and the kth refined node is represented by P_k. Finally, Q represents the final solution node.

Algorithm 1 GoT Static Exploration Schedule

```
1: Require: Starting graph $G_t$, branch depth $B$, allow reduce $R_{ed}$, allow refine $R_{ef}$, solve, aggregate, and refine multiplicities $S^m, A^m, R_{ef}^m$
2: $M_1, ..., M_B, N_1, ..., N_B, O, P, Q \leftarrow \emptyset$
3: $M_1, ..., M_B \leftarrow \phi_{dec}(G_t, B)$
4: for $i = 1$ to $B$
5:     $N_i \leftarrow \phi_{sol}(G_t, S^m, M_i)$
6: end for
7: $O_1, ..., O_{A^m} \leftarrow \phi_{agg}(G_t, A^m, \{N_1, ..., N_B\})$
8: if $R_{ed}$
9:     $O \leftarrow \phi_{red}(G_t, \{O_1, ..., O_{A^m}\})$
10: else
11:     $O \leftarrow O_1 \cup ... \cup O_{A^m}$
12: end if
13: if $R_{ef}$
14:     $P_1, ..., P_{R_{ef}^m} \leftarrow \phi_{ref}(G_t, R_{ef}^m, O)$
15:     $Q \leftarrow \phi_{red}(G_t, A^m, \{P_1, ..., P_{R_{ef}^m}\})$
16: else
17:     $Q \leftarrow O$
18: end if
19: Return: $Q$
```

Algorithm 1 represents a standard divide-and-conquer strategy. The ϕ_{dec} transformation decomposes the starting problem into B subproblems, which are solved individually (ϕ_{sol}). The aggregation of the subproblem solutions is attempted A^m times, as the ϕ_{agg} transformation has a non-zero probability of failure. If $R_{ed} = 1$, a single aggregation attempt is kept, while others are removed from the graph. If $R_{ef} = 1$, the remaining aggregation attempts are then refined wth ϕ_{ref}, and the highest-scoring attempt is kept as the final solution.

4. Autonomous Thought Graph Exploration

Beyond the fixed schedule shown in Algorithm 1, the exploration schedule of a thought graph can be generalized as a Markov Decision Process $(\mathcal{S}, \mathcal{A}, \mathcal{P}_a)$, where:

- **State** $s_t \in \mathcal{S}$: represents an arrangement of nodes and edges in the thought graph, with the associated value of each node, i.e. $s_t = (\mathcal{V}, \mathcal{E}, \{\lambda(v) | v \in \mathcal{V}\})$.
- **Action** $a \in \mathcal{A}$: indicates which transformation to perform on the thought graph, and which nodes to perform it on, i.e. $\mathcal{A} = \{(\mathcal{V}_s, \phi) \mid \mathcal{V}_s \subset \mathcal{V}, \phi \in \Omega\}$, where Ω is the set of transformations (Table 1).
- **Transition probability** $\mathcal{P}_a(s, s')$: represents the probability of transitioning to a new state s' assuming an action a was performed in state s.

It should be noted that the transition between two states is a random process governed by the token distribution parametrized by the LLM (Equation 1). Given a starting thought graph $G_t = (\{v\}, \emptyset)$ corresponding to starting state $s_0 = (\{v\}, \emptyset, \{\lambda(v)\})$, where v represents the starting problem, the action ϕ_{sol} yields a solution state s' with probability $P(s' \mid s_0, \phi_{sol})$. A solution state is any state with at least one node v' that has $\lambda(v') = 1$, e.g. $s' = (\{v, v'\}, \{(v, v')\}, \{\lambda(v), 1\})$. The optimal transformation sequence Φ is then defined as the sequence of actions that maximize the conditional probability of reaching a solution state s^+, i.e. $\Phi = (\phi_0, ..., \phi_n)$ that solves the following optimization problem.

$$\begin{aligned} \max_{\Phi} \quad & P(s^+ \mid s^0, \Phi) \\ \text{s.t.} \quad & |\Phi| < \epsilon \end{aligned}$$

We bound the number of queries by the constant ϵ, as in the limit $|\Phi| \to \infty$, $P(s^+ | s^0, \Phi) \to 1$.

4.1. Multi-Agent Reasoning

Given the infeasibility of characterizing the distribution for the full set of state/action pairs for arbitrary tasks, we hypothesize that LLMs can approximate a solution to the stated optimization problem by acting as policy agents. We develop an interactive framework consisting of a policy agent and a reasoning agent, as shown in Figure 2. In each iteration, (1) the policy agent selects an action from the action space, outlining its rationale. (2) The policy agent directs the reasoning agent to perform the selected action, through futher LLM calls or otherwise. (3) The reasoning agent updates the thought graph. The process is repeated until a solution is found or a maximum number of iterations is reached, at which point the trial history is updated.

The policy agent is invoked using the prompt template shown in Figure 2. (i) The system prompt outlines the problem setting, input format and expected behaviour from the policy agent. (ii) A task-specific list of actions, describing the preconditions and effects of each transformation, provides a semantic understanding of the action space. (iii) The current state of the graph is provided in a textual format, enumerating all nodes and edges. (iv) The action history in

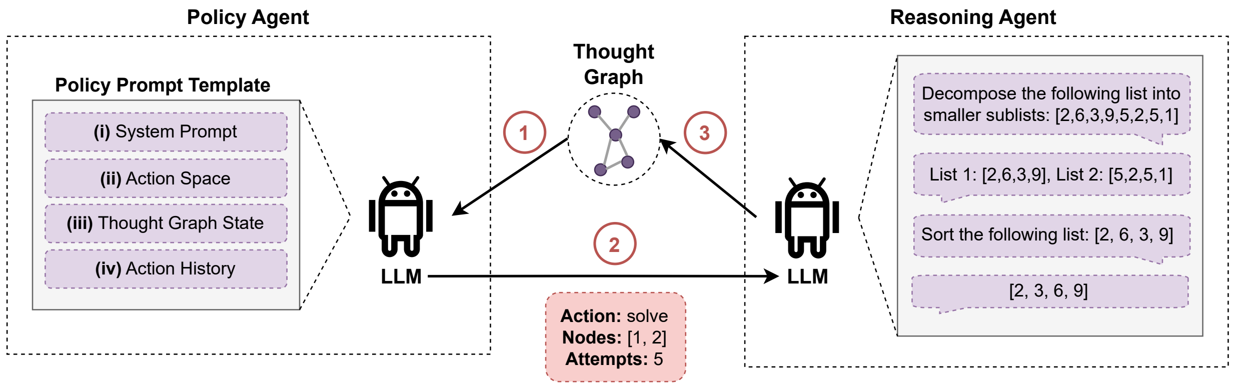


Figure 2. Interaction between the policy and reasoning agents. Each iteration involves (1) action samling by the policy agent, (2) message passing among agents, and (3) updating the thought graph. The prompt template for the policy agent includes (i) general instructions, (ii-iv) an overview of the exploration state, and (v) feedback from previous trials. After analyzing the state of the thought graph, the agent selects an action, a subset of nodes and the number of attempts for the next action.

the current trial is included, promoting continuity in strategies outlined in previous steps. Finally, (v) a number of examples of previous trials are included to guide the agent to tune its exploration strategy based on past experiences.

4.2. In-Context Action Selection

Prior work has shown that reasoning abilities of LLMs are enhanced when prompted to output a verbose sequence of steps before the solution (Wei et al., 2023; Wang et al., 2023). This mechanism can be seen as enabling in-context task learning from some extracted innate world knowledge. A such, our policy agent is instructed generate a detailed analysis on the state of the thought graph and and exploration history before sampling the action space. The analysis includes the following components.

1. Describe the action history and how each action relates to an exploration strategy.
2. Describe the state of the thought graph, including how each node corresponds to a previous action.
3. Discuss the outlined strategy, stating whether it is successful, unsucessful, or pending further actions.
4. Outline a number of options for the next action, detailing the expected outcome of each, and how it relates to the current state of the thought graph.

4.3. Policy Agent Ensembles

Given the stochastic nature of token prediction as outlined in Equation 1, we observe high variability in the chosen action over several invocations of a policy agent under the same thought graph state. Given the preconditions and effects of each action are represented via text rather than any rigorous formulation, actions selected by the policy agent can display flawed understanding of the problem constraints, leading to ineffective exploration of the thought graph. To overcome this limitation, we democratize action selection over an ensemble of agents, meaning a parametrizable number of LLM queries are performed concurrently at every iteration. The selected action is takes as the most frequent proposal among the ensemble.

5. Experiments

We evaluate Llama-3.1-70b, and Llama-3.1-405b as policy and reasoning agents with a temperature of 1. Llama-3.1-70B was hosted using SGLang on a machine with 8 A6000 GPUs. Llama-3.1-405B was hosted using 16 H100 GPUs distributed over 4 nodes.

5.1. Benchmarks

We choose two popular tasks for topological reasoning with LLMs, which are amenable to a divide-and-conquer strategy (i.e. decomposition, solving subproblems and merging): list sorting and set intersection. Despite their simplicity, prior works have shown that LLMs display low performance on these tasks in naive autoregressive decoding (Besta et al., 2024). We evaluate sorting and set intersection at various levels of difficulty, quantified by the size of the lists and sets, respectively. For each task, we report the mean value of the score function $\mathcal{E}$ over a set of trials, which we find to be the most indicative metric for evaluating performance across various methods.

Sorting: involves sorting a list of numbers between 0 and 9 in ascending order. The score function $\mathcal{E} = X + Y$ has its subterms defined in Equation 4, where a is the input list and b is a candidate solution. X corresponds to the number of incorrectly sorted pairs, while Y corresponds to the frequency difference between a and b for each digit.

$$X = \sum_{i=1}^{m-1} \text{sign}(\max(b_i - b_{i+1}, 0)) \tag{4}$$

$$Y = \sum_{i=0}^{9} ||b_p : b_p = i| - |a_q : a_q = i||$$

Set Intersection: involves finding the intersection of sets A and B. The score function is defined in Equation 5, where C is the candidate solution. The first and second terms correspond to missing and extra elements, respectively.

$$\mathcal{E} = |(A \cap B) \setminus C| + |C \setminus (A \cap B)| \tag{5}$$

5.2. Static Schedule Parameter Search

As described in Section 3, a static exploration schedule can be characterized using a set of discrete parameters. We ran bayesian search using using Tree-structured Parzen Estimator (TPE) sampling to determine each parameter, establishing strong baselines for each task.

Table 2. Search space for each parameter characterizing a static exploration schedule.

	Parameter	Search Space
R_{ed}	Allow reduction	$\{0, 1\}$
R_{ef}	Allow refinement	$\{0, 1\}$
S^m	Solve multiplicity	$\{1, 5, 10, 15, 20\}$
A^m	Aggregate multiplicity	$\{1, 5, 10, 15, 20\}$
R^m_{ef}	Refine multiplicity	$\{1, 5, 10, 15, 20\}$

The search space is shown in Table 2. We run multi-objective search to concurrently minimize the task-specific error function $\mathcal{E}$ (Section 5.1) and associated cost, measured as $|\Phi(\omega)|$ where $\Phi(\omega) = (\phi_0, ..., \phi_m)$ is a tuple enumerating thought graph transformations, as a function of the schedule parameters $\omega \in \Omega$, where Ω is the search space. Note that $|\Phi(\omega)|$ correlates with the number of LLM queries, meaning this formulation aims to minimize eploration cost.

In selecting parameter configurations, we use the cost function in Equation 6, such that the objectives of cost and error minimization are balanced through the scalar constant $\alpha \in (0, 1)$. We aim to assign equal importance to the cost and error objectives by tuning α independently for each task such that the mean value of the first term matches the second term, i.e. $\alpha E[\mathcal{E}] = (1 - \alpha)E[|\Phi(\omega)|]$, or equivalently $\alpha = \frac{E[|\Phi(\omega)|]}{E[\mathcal{E} + |\Phi(\omega)|]}$ where E denotes the expected value. The expectations are obtained with random sampling.

$$\min_{\omega} [\alpha\mathcal{E} + (1 - \alpha)|\Phi(\omega)|] \tag{6}$$

Search was conducted separately on Llama-3.1-70b and Llama-3.1-405b. Configurations obtained from the 405b model were reused on Claude-3.5-Haiku and GPT-4o without further search, as these models are deemed to have similar reasoning abilities. For sorting and set intersection tasks, search is conducted separately for each difficulty level, ensuring the chosen parameters are adapted to the task. Note that we present three search checkpoints GoT$_n$ for $n \in \{25, 50, 100\}$, where n corresponds to the percentage of trials until convergence. We define the convergeance point as the first iteration where a rolling window J of size 20 matches the condition $J^k = J^{k-1}$. This enables comparing our proposed LLM-guided approach to optimized search schedules at various search budgets.

Table 3. Results from GoT static schedule parameter search on Llama-3.1-405b.

Task	Alpha (α)	GoT$_{25}$	GoT$_{50}$	GoT$_{100}$
sorting32	0.99	0.38	0.38	0.37
sorting64	0.96	4.85	4.49	3.84
sorting128	0.84	28.76	25.76	24.36
set32	0.99	0.16	0.16	0.12
set64	0.99	0.71	0.51	0.31
set128	0.98	3.51	3.51	2.99

The complete search results for Llama-3.1-405b are shown in Table 3, while results for Llama-3.1-70b are shown in Appendix B. It can be seen that tasks with higher decomposition depth incur lower values of α due to the higher magnitude of the error function. sorting64, sorting128 and set-intersection64 show a smooth decline in the cost function, while the remaining tasks remain at local minima until close to the end of the search. The non-convexity of the search space highlights the cost associated to optimize the parameter set associated with static exploration schedules.

5.3. Results

The parameters obtained from bayesian search (Section 4) were used to define the static schedule parameters for GoT baselines at various search budgets. In Table 5.2, these are compared against the LLM-guided thought graph exploration approach outlined in Section 3. We make the following observations based on the results.

Expectedly, the obtained error values are lower for all tasks with Llama-405b relative to Llama-70b, due to the improved reasoning performance of models with larger parameter counts. In fact, The LLM-guided graph explo-

Table 4. Core results for topological reasoning across all tasks and models. We show the mean value of the score function $\mathcal{E}$ (↓), which is defined for each task in Section 5. GoT_{100}, GoT_{50}, GoT_{25} represent the obtained values from static schedule parameters obtained at convergeance, 50% and 25% of convergeance trials, respectively.

Task	**Llama-70b**				**Llama-405b**			
	GoT_{25}	GoT_{50}	GoT_{100}	GoT_{LLM}	GoT_{25}	GoT_{50}	GoT_{100}	GoT_{LLM}
sorting32	0.82	0.95	**0.73**	1.29	0.74	0.82	0.28	**0.22**
sorting64	4.73	4.73	**4.64**	10.04	**2.22**	2.74	3.46	9.15
sorting128	16.18	**13.86**	16.07	31.79	13.96	**12.65**	18.65	32.74
set-intersection32	0.41	**0.0**	0.37	1.22	0.07	0.0	0.09	**0.03**
set-intersection64	3.40	2.66	**1.27**	7.34	0.67	**0.64**	0.72	1.08
set-intersection128	13.23	12.92	**12.73**	22.98	**1.07**	0	2.54	4.62

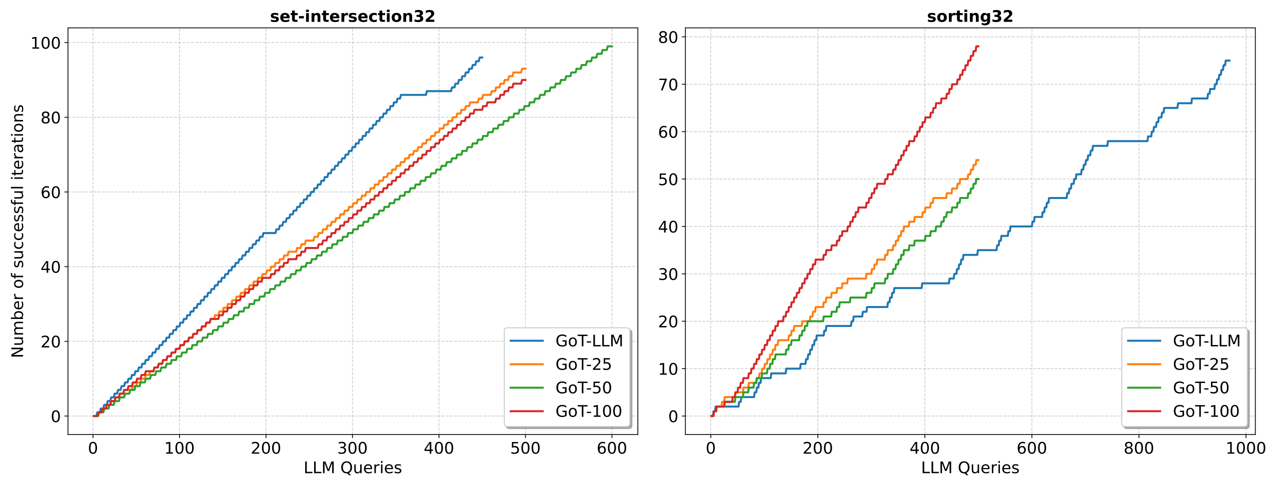
ration with Llama-70b leads to no improvement relative to static schedules obtained from bayesian search across all tasks, outlining the poor planning abilities of smaller models. Additionally, we observe that further search trials at times cause an increase in error values, i.e. the condition $\text{GoT}_{100} < \text{GoT}_{50} < \text{GoT}_{25}$ is not met. This is understood due to the multi-objective search formulation, causing the optimizer to prioritize configurations with improved query efficiency, despite incurring an error cost.

Despite the stated observations, the LLM-guided thought graph exploration leads to improvements in the sorting32 and set-intersection32 tasks, which have lower decomposition depth. In sorting32, GoT_{LLM} leads to a $3.3\times$ improvement relative to GoT_{25}, as well as a $1.3\times$ improvement relative to the configuration obtained at convergeance (GoT_{100}). In set-intersection32, we observe a $2.33\times$ improvement relative to GoT_{25}, although search achieves zero error after 50% of convergeance. These gains are obtained despite no search cost, outlining the potential of LLM-guided thought graph exploration. On the other hand, the LLM-guided thought graph exploration leads to performance deterioration in tasks with higher decomposition depth. In sorting62, sorting128, set-intersection64 and set-intersection128, the performance penalties relative to GoT_{25} are $4.15\times$, $2.34\times$, $1.6\times$ and $4.31\times$, respectively.

5.4. Query Efficiency Analysis

In addition to investigating the achievable performance on reasoning tasks, we are interested in the query efficiency, i.e. the ratio between achieved acurracy and required number of LLM queries, for each thought graph exploration methodology. In Figure 5.2, we show the accuracy growth with respect to the number of reasoning LLM agent queries. We compare GoT_{LLM} to GoT_{100}, as the latter represents a near-optimal trade-off in error and cost obtained with a significant search cost.

Figure 3. Growth plot for set-intersection32 and sorting32 task accuracy (↑) with the number of reasoning agent LLM queries. Llama-3.1-405b was used for both reasoning and planning agents across all GoT baselines and the LLM-guided approach. Note that policy agent queries are not considered, i.e. the queries required to analyze the thought graph and sample the action space.



We observe variability across the tasks, due to the adaptiveness of the LLM guided exploration strategies. In the set-intersection32 task, the query efficiency for GoT_{LLM} is 29.3% higher than GoT_{100}, meaning the policy agent is able to leverage the task simplicity to explore direct solutions that fall beyond the scope of static schedules. Note that sorting32 is a comparably more difficult task, which is further reflected in the error function values shown in Table 5.2. In this case, the policy agent adapts to increased complexity with a significantly higher number of reasoning agent queries, as new strategies are explored and refined. This leads to 50% worse query efficiency, although the accuracy is comparable to the search convergeance point (only 3% lower) and 26% higher than the GoT_{50} baseline. It should be noted that despite the loss in test-time query efficiency, there is no search cost associated with the LLM-guided approach.

6. Discussion and Future Work

The LLM-guided thought graph exploration approach shows promise in tasks with low decomposition depth, where the policy agent can leverage the task simplicity to explore direct solutions that fall beyond the scope of static schedules. However, the approach leads to performance deterioration in tasks with high decomposition depth, where the policy agent is unable to identify the sequence of transformations required to merge solutions of subproblems back into a complete solution. We consider the following avenues of future work to improve the query efficiency and generalization of LLM-guided thought graph exploration.

Strategy Feedback: When solving the same problem repeatedly with different data, a policy agent can learn from past experiences to improve its reasoning performance. Efficient strategies are rewarded with positive feedback from successful trials, while inefficient strategies can be avoided with negative feedback from unsuccessful trials. In particular, we observe that in tasks with high decomposition depth, policy agents fail to identify the sequence of transformations required to merge solutions of subproblems back into a complete solution, which can be overcome through improved exemplars of such action sequences. However, considering the context length constraint in LLMs, trial feedback requires an efficient sampling method to select a limited number of examples that maximize the information gain.

Action Scheduling for Enhanced Query Efficiency: Despite the gains in generalization and performance at low decomposition depths, we observe a loss in query efficiency in LLM-guided thought graph exploration. This problem is inherent to this approach, as further queries are required to sample the action space before guiding the reasoning agent. We consider multi-step scheduling as a potential solution to alleviate this problem, whereby the policy agent outlines a sequence of n actions at a time, rather than a single action. This would lead to a significant reduction in the number of queries, despite potentially impacting performance when incorrect action sequences are proposed. As such, the n hyperparameter must be tuned to balance efficiency gains with accuracy loss.

Heterogeneous Multi-Agent Reasoning: In this work, we make the simplification of using the same model across all agents in our experiments. However, policy and reasoning agents present a diverse set of requirements - while reasoning agents require extensive world knowledge, policy agents require long-horizon planning capabilities. As such, further benefits can be obtained through a heterogeneous approach where each agent in the system is optimized for its specific task. This can take the form of fine-tuning for better alignment of policy agents, or deploying smaller models for solving subproblems with lower complexity. The adaptiveness of LLM policy agents can extend to dynamically allocating workloads to models of varying sizes, reducing cost by leveraging smaller models when appropriate.

Beyond Iterative Multi-Agent Collaboration: In our framework, the interaction between policy and reasoning agents follows a simple iterative schedule. In each iteration, the policy agent is prompted to outline its thought process in sampling the action space, which is analogous to CoT (or CoT-SC in the case of policy agent ensembles). However, sampling the thought graph action space is in itself a complex reasoning problem, due to the nuance of formulating a correct strategy according to the thought graph state. As such, the policy agent could benefit from a ToT strategy to sample and evaluate multiple reasoning strategies in parallel, rather than a single action. This could lead to more coherent strategies, potentially improving task outcomes. The effectiveness of this approach is subject to the effective evaluation of action subsequences.

7. Conclusion

In this paper, we present a novel approach to topological reasoning with LLMs, where a policy agent guides a reasoning agent in exploring thought graphs. We show that LLMs can be leveraged to approximate a solution to the optimization problem of finding a sequence of actions that maximize the probability of reaching a solution state. We evaluate our approach on three popular tasks for topological reasoning, showing that LLM-guided thought graph exploration leads to improvements in tasks with low decomposition depth, while deteriorating performance in tasks with high decomposition depth. We outline avenues of future work to improve the query efficiency and generalization of LLM-guided thought graph exploration.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Besta, M., Blach, N., Kubicek, A., Gerstenberger, R., Podstawski, M., Gianinazzi, L., Gajda, J., Lehmann, T., Niewiadomski, H., Nyczyk, P., and Hoefler, T. Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690, March 2024. ISSN 2159-5399. doi: 10.1609/aaai.v38i16.29720. URL http://dx.doi.org/10.1609/aaai.v38i16.29720.

Ding, R., Zhang, C., Wang, L., Xu, Y., Ma, M., Zhang, W., Qin, S., Rajmohan, S., Lin, Q., and Zhang, D. Everything of thoughts: Defying the law of penrose triangle for thought generation, 2024. URL https://arxiv.org/abs/2311.04254.

Feng, F., Yang, Y., Cer, D., Arivazhagan, N., and Wang, W. Language-agnostic bert sentence embedding. *arXiv preprint arXiv:2007.01852*, 2020.

Hu, P., Qi, J., Li, X., Li, H., Wang, X., Quan, B., Wang, R., and Zhou, Y. Tree-of-mixed-thought: Combining fast and slow thinking for multi-hop visual reasoning, 2023. URL https://arxiv.org/abs/2308.09658.

Kim, G., Baldi, P., and McAleer, S. Language models can solve computer tasks, 2023. URL https://arxiv.org/abs/2303.17491.

Liu, J., Shen, D., Zhang, Y., Dolan, B., Carin, L., and Chen, W. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*, 2021.

Long, J. Large language model guided tree-of-thought, 2023. URL https://arxiv.org/abs/2305.08291.

Sel, B., Al-Tawaha, A., Khattar, V., Jia, R., and Jin, M. Algorithm of thoughts: Enhancing exploration of ideas in large language models, 2024. URL https://arxiv.org/abs/2308.10379.

Shen, Y., Song, K., Tan, X., Li, D., Lu, W., and Zhuang, Y. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face, 2023. URL https://arxiv.org/abs/2303.17580.

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models, 2023. URL https://arxiv.org/abs/2203.11171.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., and Fedus, W. Emergent abilities of large language models, 2022. URL https://arxiv.org/abs/2206.07682.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Xie, Y., Yu, C., Zhu, T., Bai, J., Gong, Z., and Soh, H. Translating natural language to planning goals with large-language models, 2023. URL https://arxiv.org/abs/2302.05128.

Yang, Z., Garcia, N., Chu, C., Otani, M., Nakashima, Y., and Takemura, H. Bert representations for video question answering. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1556–1565, 2020.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models, 2023a. URL https://arxiv.org/abs/2305.10601.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. React: Synergizing reasoning and acting in language models, 2023b. URL https://arxiv.org/abs/2210.03629.

A. Exploration Schedules

```
Require: Starting prompt $x$
  $O \leftarrow sort(x)$
Return: $O$
```

(a) Input/Output (IO)

```
Require: Starting prompt $x$
Require: Width $w$
  $\mathcal{G} \leftarrow \varnothing$
for $b \in [0, w)$
  $\mathcal{G} \leftarrow sort(x)$
end for
$\mathcal{G} \leftarrow score(\mathcal{G})$
$O \leftarrow keepbest(\mathcal{G})$
Return: $O$
```

(b) Chain-of-Thoughts with Self Consistency (CoT-SC)

```
Require: Starting prompt $x$
Require: Width $w$
Require: Depth $d$
  $\mathcal{G} \leftarrow \varnothing$
for $b \in [0, w)$
  $\mathcal{G} \leftarrow sort(x)$
end for
$\mathcal{G} \leftarrow score(\mathcal{G})$
$O \leftarrow keepbest(\mathcal{G})$
for $l \in [0, d-1)$
  $\mathcal{G} \leftarrow refine(B)$
$\mathcal{G} \leftarrow score(\mathcal{G})$
$O \leftarrow keepbest(\mathcal{G})$
end for
Return: $O$
```

(c) Tree-of-Thoughts

```
Require: Starting prompt $x$
Require: Width $w$
Require: Depth $d$
  $\mathcal{G} \leftarrow \varnothing$
for $b \in [0, w)$
  $\mathcal{G} \leftarrow sort(x)$
end for
$\mathcal{G} \leftarrow score(\mathcal{G})$
$O \leftarrow keepbest(\mathcal{G})$
for $l \in [0, d-1)$
  $\mathcal{G} \leftarrow sort(B)$
$\mathcal{G} \leftarrow score(\mathcal{G})$
$O \leftarrow keepbest(\mathcal{G})$
end for
Return: $O$
```

(d) Graph-of-Thoughts

Figure 4. Exploration schedules for solving the number sorting problem with various prompting strategies: (a) Input/Output (IO) Prompting, (b) Chain-of-Thought with Self-Consistency (CoT-SC), (c) Tree-of-Thoughts (ToT), and (d) Graph-of-Thoughts (GoT).

B. Llama-3.1-70b Search Results

Table 5. Results from GoT static schedule parameter search for Llama-3.1-70b. The **GoT$_5$**, **GoT$_{25}$** and **GoT$_{50}$** columns show the optimal values of the cost function (Equation 6) after 5, 25 and 50 trials, respectively.

Task	**Alpha** (α)	**GoT$_5$**	**GoT$_{25}$**	**GoT$_{50}$**
sorting32	0.97	1.38	1.13	1.01
sorting64	0.95	5.19	5.19	4.85
sorting128	0.87	22.64	19.51	17.58
set32	0.99	0.51	0.14	0.04
set64	0.97	3.21	2.43	1.62
set128	0.91	18.34	17.12	13.51