Fleet of Agents: Coordinated Problem Solving with Large Language Models

Anonymous Author(s)
Affiliation
Address
email

Abstract

While numerous frameworks have been developed to enhance the reasoning abilities of large language models (LLMs), there is a scarcity of methods that effectively balance the trade-off between cost and quality. In this paper, we introduce FLEET OF AGENTS (FOA), a novel and intuitive yet principled framework utilizing LLMs as agents to navigate through dynamic tree searches, employing a *genetic-type particle filtering* approach. FOA spawns a multitude of agents, each exploring the search space autonomously, followed by a *selection phase* where resampling based on a heuristic value function optimizes the balance between exploration and exploitation. This mechanism enables *dynamic branching*, adapting the exploration strategy based on discovered solutions. We conduct extensive experiments on three benchmark tasks, "Game of 24", "Mini-Crosswords", and "WebShop", utilizing four different LLMs, "GPT-3.5", "GPT-4", "LLaMA3.2-11B", and "LLaMA3.2-90B". On average across all tasks and LLMs, FOA obtains a quality improvement of $\simeq 5\%$ while requiring only $\simeq 40\%$ of the cost of previous SOTA methods. Notably, our analyses reveal that (1) FOA achieves the best cost-quality trade-off among all benchmarked methods and (2) FOA + LLaMA3.2-11B surpasses the Llama3.2-90B model. FOA is publicly available at `https://anonymous.4open.science/r/FoA-4D83`.

1 Introduction

With strong reasoning and problem-solving abilities, large language models (LLMs) [5] such as GPT-4 [1], LLaMA [40, 41, 11], and PaLM [2], have sparked a new-found interest in building general-purpose autonomous agents. LLM-based agents have portrayed excellent performance on reasoning [7] and knowledge-intensive tasks [47], often requiring interactions with complex environments, such as playing complex video games [8], performing web navigation [50], or enabling tool-use [34].

Naturally, the rise of LLM-based agents has contributed to the prosperity of prompt-based reasoning frameworks [46, 44, 4, 35, 49, 51, 54, 37, 52] that further enhance the problem-solving and reasoning abilities of LLMs. Broadly, the reasoning frameworks can be categorized into two categories: (1) single-query reasoning and (2) multi-query reasoning. As the name implies, single-query methods [46, 44, 35, 30, 20] obtain an answer by querying the LLM only once, whereas, multi-query methods [51, 4, 54, 37, 52] perform multiple LLM queries to identify different plausible reasoning paths or to plan ahead. It is important to note that none of the two aforementioned paradigms is perfect.

On the one hand, despite being cost-effective by design, single-query methods require one or more of the following: intricate prompt engineering, high-quality demonstrations, or knowledge distilled from informative historical reasoning processes, to achieve competitive quality. More importantly, even

Submitted to the First Exploration in AI Today Workshop at ICML (EXAIT at ICML 2025). Do not distribute.

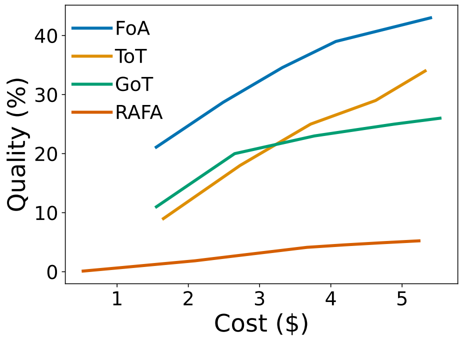

Figure 1: Analyzing the trade-off between cost and quality of representative SOTA methods with GPT-3.5 on the Game of 24 task. **FoA achieves the best cost-quality trade-off.**

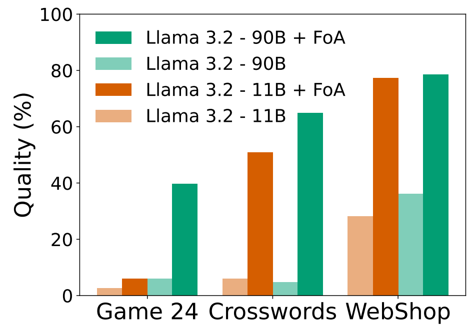

Figure 2: Evaluating the trade-off between model size and quality on benchmarked tasks with Llama3.2-11B and 90B. **FoA enables smaller models to achieve competitive quality.**

then, these methods are not well-suited for sequential decision-making tasks that require interactions with an environment, such as web navigation [50].

On the other hand, multi-query methods decompose a complex problem into a series of simpler sub-problems and search over all plausible reasoning paths. This allows them to obtain competitive quality but also renders them inefficient. With the objective of devising a reasoning framework applicable to both general problem-solving and sequential decision-making tasks, our focus in this paper is to improve the cost-efficiency of multi-query methods.

Present work. We introduce FLEET OF AGENTS (FoA), a novel and intuitive yet principled framework that brings the concept of genetic-type particle filtering [14] to dynamic tree searches. Fig. 3 provides an overview of our framework, while at the same time highlighting the conceptual differences between state-of-the-art (SOTA) tree-search-based methods [51, 4, 54, 13] and FoA.

FoA spawns a multitude of agents, each exploring the search space autonomously, followed by a *selection phase* where *resampling* based on a heuristic value function optimizes the balance between exploration and exploitation. If one of the agents has discovered a very promising solution approach indicated by states with a high value, the resampling mechanism can decide to create many copies of this agent. Conversely, if none of the agents is ahead of the others, or in other words, there are multiple promising states, the resampling mechanism can decide to keep all of them, thereby maintaining a fleet of high diversity. This mechanism enables *dynamic branching*, adapting the exploration strategy based on discovered solutions.

Cost-quality trade-off. The biggest advantage of FoA is its ability to strike a balance between exploration vs. exploitation. We provide early empirical evidence in Fig. 1, which compares the performance of tree-based SOTA methods with FoA for varying price points. We find that FoA substantially outperforms the existing SOTA methods at all possible price points, thereby achieving the best cost-quality trade-off among the benchmarked methods.

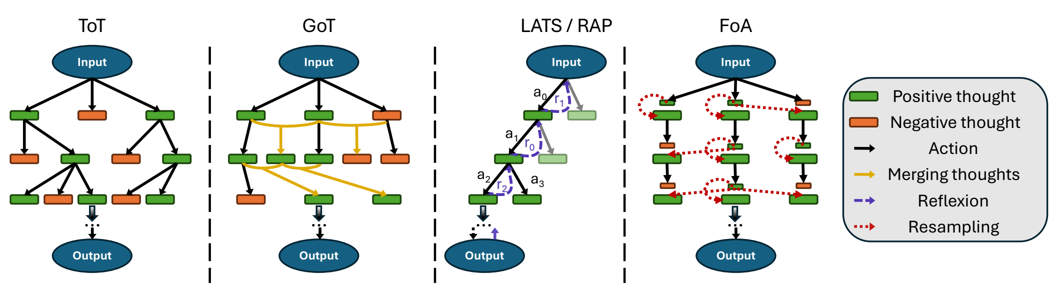

Figure 3: Comparison between SOTA tree-search-based reasoning [51, 4, 54, 13] and our FoA frameworks. FoA offers precise control over the tree width (n agents) and depth (t steps), leading to predictable latency and cost. However, by expanding the c most promising states at each step, tree-search methods offer no such control and their search trees might grow exponentially.

Contributions.

• We propose an intuitive yet principled framework Fleet of Agents (FoA) for improving the cost-quality trade-off of LLM-based reasoning (§ 2).

• Ours is the first work to explore the concept of genetic particle filtering in the context of AI agents (cf. § A for a detailed literature review).

• We conduct extensive experiments on three benchmark tasks using four LLMs as base models. On average across all tasks and LLMs, FoA obtains a quality improvement of $\simeq 5\%$ while requiring only $\simeq 40\%$ of the cost of previous SOTA methods (§ 3 and § 4).

2 Fleet of agents

2.1 Overview of FoA

Fig. 4 provides a pictorial overview of FoA. FoA spawns a fleet of n agents that collectively search for a solution. The genetic filtering search has a *mutation phase* during which each agent explores the search space autonomously. Specifically, it tracks the n agent states and allows k independent mutation steps before evaluating the values of the current states. During the *selection phase*, we resample, with replacement, the population of agents. The *resampling* mechanism is based on a heuristic value function and allows us to optimize the trade-off between exploration and exploitation. The FoA algorithm is comprehensively described in Algorithm 1 in the Appendix.

Preliminaries. The fleet consists of n agents. At time t agent i has the state $s_{i,t}$. When choosing an action, the agent samples from its policy $a_{i,t} \sim \pi_a(a|s_{i,t})$. The agent then transitions to a new state, following the dynamics of the environment $s_{i,t+1} \sim \mathbb{P}[s|a_{i,t}, s_{i,t}]$. With each agent searching for the solution, the fleet jointly discovers more and more of the state space. We describe the current set of states at timestep t as $S_t = \{s_{i,t}\}, i = 1..n$ and the set of all states visited so far as $\hat{S}_t = \bigcup_{\hat{t}} S_{\hat{t}}, \hat{t} = 1..t$.

We assume that we can identify solutions when they are found, i.e., we can decide whether a state s is a solution, $\mathbb{1}_{solution}(s)$. Depending on the task, we may also be able to identify invalid states, failed tasks, and other forms of dead-ends; we denote them as terminal states $\mathbb{1}_{terminal}(s)$.

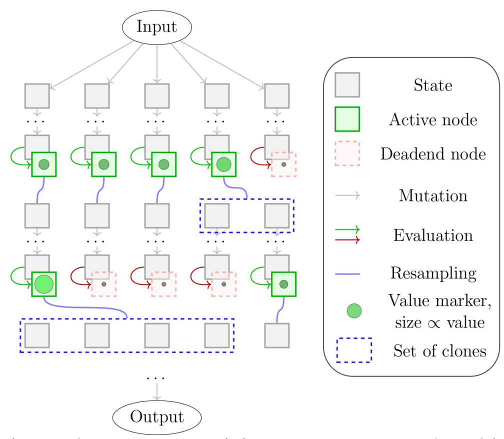

Figure 4: FoA comprising $n = 5$ agents that think autonomously for k steps and are then resampled to focus the search on promising regions.

For some tasks, we can stop as soon as a solution is found; for other tasks, we may instead collect a set of solution candidates and stop the search only if we run out of time or sampling budget. We assume we are given access to a value function $v(s)$ that can guide the search. For a solution state s, the value function represents the utility of the solution. For other states, s, the value function is a heuristic considering the uncertainty of eventually reaching a solution and its expected utility. Accordingly, the value of terminal, non-solution states is 0.

2.2 Genetic particle filtering

Each agent in the fleet of agents acts autonomously and tries to choose the best action given its current state. After k independent steps, we use the value function to resample the set of states and allocate more agents to high-value states. Our algorithm implements a genetic-type particle filter that captures the dynamics of a population of particles, i.e., agents, with a series of mutation and selection steps.

Mutation phase. During the mutation phase, each agent in FoA independently samples state transitions. To simplify the notation, we introduce a model π which captures both the stochasticity of the decision of the agent as well as the response of the environment, $s_{i,t+1} \sim \pi(s|s_{i,t})$ We use the same notation for multi-step state transitions by marginalizing over intermediate states, i.e., $s_{i,t+k} \sim \pi(s_{i,t+k}|s_{i,t})$. After each mutation step, we can check whether a solution has been found and decide to stop the search. Following the concept of genetic filtering, we apply two optimizations:

• **Enforced mutation**: The agent must mutate its state; it can not remain stationary.

• **Sudden death**: If we notice that an agent i has entered an invalid state $\mathbb{1}_{terminal}(s_{i,t})$, then we resample its state immediately from the set $s_{j,t}, j \neq i$. This resampling can be uniform to avoid expensive calls to the value function.

Selection phase. The selection phase resamples the population of agents based on an importance sampling mechanism. We observe the value estimates $v(s_{i,t})$ for all current states and calculate a resampling weight $p_{i,t}$. This framework can capture many resampling schemes, such as linear, greedy, or exponential weighting of values:

$$
\begin{aligned}
p_{lin}(s_{i,t}) &= \alpha v(s_{i,t}) + \beta, \\
p_{exp}(s_{i,t}) &= \exp\left(\frac{v(s_{i,t})}{\beta}\right), \\
p_{greedy}(s_{i,t}) &= \begin{cases} 1, & \text{if } s_{i,t} = \arg\max\limits_{s_{j,t}, j=1..N} v(s_{j,t}) \\ 0, & \text{otherwise} \end{cases}.
\end{aligned}
$$

We then resample, with replacement, to select a new set of agent states:

$$
\begin{aligned}
p_t(s) &= \sum_{i=1}^{N} \frac{p_{i,t}}{\sum_{j=1} N p_{j,t}} \delta_{s_{i,t}}(s), \text{categorical resampling} \\
\hat{s}_{i,t} &\overset{iid}{\sim} p_t(s), \text{resampling with replacement} \\
s_{i,t} &= \hat{s}_{i,t}, i = 1..N
\end{aligned}
$$

Backtracking. Additionally, by keeping track of a history of states and the associated value function estimates, we extend the resampling process with an intuitive backtracking mechanism. Backtracking undoes localized mistakes and allows our fleet of agents to recover from a catastrophic scenario in which all agents might have made a wrong decision. Instead of resampling states from the set of current states S_t, we consider all previously visited states $\hat{S}_t$. To incentivize the fleet of agents to push forward and explore new regions of the state space, we introduce a discount factor γ. The value of a state that was visited t timesteps ago is discounted by γ^t. The mutation phase of our genetic particle filter comprises k steps of individual exploration, before evaluating and resampling in the selection phase. Therefore, we may not have a value estimate for all previously visited states $\hat{S}_t$. Depending on the task, and the corresponding cost of computing the value $v(s)$, we may limit the backtracking mechanism to a subset of $\hat{S}_t$ which only contains states with a known value estimate.

3 Experiments

We assess the effectiveness of our proposed FoA framework through comparisons with representative SOTA methods on a judicious mix of tasks that require a variety of reasoning, planning, and general problem-solving skills. Additional experimental details, e.g., task and method descriptions, hyperparameter tuning, additional results, etc. are present in Appx. C. The resources for reproducing our experiments are available at `https://anonymous.4open.science/r/FoA-4D83`.

Base model. Following convention in the literature, we use GPT-4[1] as the base model for the main results presented in this paper. To showcase the *generalizability* of our findings, we report results with other base models, namely, GPT-3.5, Llama3.2-11B, and Llama3.2-90B, in Appx. C.6. We use these models for the model and ablation analyses (§ 4 and § 5), primarily for practical cost-related reasons.

[1] Owing to the exorbitant cost of running GPT-4 (cf. App. C.4), we use GPT-3.5 instead for WebShop [50].

Table 1: Comparing FoA with previous methods using *success rate* (↑ better) and *cost* (↓ better) on the Game of 24 task (base model: GPT-4). The best performance is shown in **blue**, whereas the second best is shown in **orange**.

Method	Success Rate (%)	Cost (US$)
IO	6.0	**0.65**
CoT [46]	6.0	**6.98**
CoT-SC [44]	10.0	49.40
AoT [35]	49.0	20.98
ToT [51]	**74.0**	75.02
GoT [4]	63.0	70
FoA **(Present Work)**	**76.0**	62.93

Table 2: Comparing FoA with previous methods on (Left) Crosswords (base model: GPT-4) using *overlap* (↑ better) and *cost* (↓ better), and (Right) WebShop (base model: GPT-3.5) using *average score* (↑ better) and *cost* (↓ better). The best performance is highlighted in **blue**, and the second best in **orange**.

Method	Overlap (%)	Cost (US$)
IO	36.8	**0.51**
CoT [46]	39.4	**1.06**
CoT-SC [44]	39.4	2.82
ToT [51]	39.7	48.99
GoT [4]	**41.2**	30.28
FoA **(Present Work)**	**46.0**	12.94

Type	Method	Avg Score	Cost (US$)
Super-vised	IL [50]	59.9	NA
	IL+RL [50]	62.4	NA
	WebN-T5 [12]	**61.0**	NA
	WebGUM [10]	**67.5**	NA
In-context	Act [52]	58.1	**0.10**
	ReAct [52]	48.7	**0.17**
	Reflexion [37]	56.3	0.65
	LASER [25]	57.2	0.41
	LATS [54]	**66.1**	232.27
	FoA **(Present Work)**	**75.6**	1.68
	Hum. experts [50]	82.1	NA

Prompts. Unlike all existing works, we *do not craft custom prompts* for FoA, but instead, we reuse the prompts (cf. Appx. C.4 for details) provided by ToT [51] for the "Game of 24" and "Mini Crosswords" tasks and LATS [54] for the "WebShop" task. This ensures a *direct and fair comparison* of the reasoning abilities of the benchmarked frameworks and controls for the impact of prompt quality, which is a confounder.

Baselines. We only compare with methods that have made their code available for the tasks benchmarked in this study (cf. Appx. C.3 for details). Thus, we do not compare with LLMCompiler [19], TouT [28], RAP [13], and RecMind [45]. Moreover, we exclude BoT [49], where although the code is available, an important resource (the meta-buffer) to reproduce their results is unavailable.

Results. Across all benchmark tasks, Game of 24, Mini Crosswords, and WebShop, FoA consistently outperforms existing baselines, achieving the best overall quality and the most favorable cost-quality trade-offs. On Game of 24, FoA delivers a *70% quality improvement over GPT-4* and surpasses the next-best method, ToT, with a *2% higher success rate while reducing cost by 25%*. In the Mini Crosswords task, FoA achieves a *5% quality gain over GoT* and simultaneously cuts the cost by *60%*. On the WebShop benchmark, FoA even outperforms supervised fine-tuned models, delivering a *10% higher average score than LATS at just 1% of its cost*. Overall, FoA strikes an exceptional balance between exploration and exploitation, consistently advancing both quality and efficiency across diverse tasks.

4 Model analysis

FoA vs. SOTA. To better understand the improvements obtained by FoA, we compare it with the second most efficacious method (SOTA hereafter) on all three benchmark tasks. Specifically, ToT [51], GoT [4], and LATS [54] are the SOTA methods for the Game of 24, Crosswords, and WebShop tasks, respectively. As stated in § 3, we use GPT-4 as the base model for Game of 24 and Crosswords, and GPT-3.5 for WebShop. Fig. 13 shows that FoA not only achieves the best quality but also the lowest cost on all tasks. **On average, FoA obtains a quality improvement of $\simeq 5\%$ while requiring only $\simeq 40\%$ of the cost of SOTA methods.**

Trade-off between cost and quality. We analyze the trade-off between cost and quality for the top four methods, namely, ToT [51], GoT [4], RAFA [23], and FoA, in the Game of 24 task. For cost reasons, we use GPT-3.5 as the base model. Following RAFA [23], we allow each method to perform multiple trials (Δ) and consider them successful if they generate a valid solution in any of the Δ trials. To ensure fairness in the allocated resources, we allocate a fixed budget of 5$ per method, which governs the number Δ of trials for each method (5 each for ToT and GoT, 6 for FoA, and 10 for RAFA).

Fig. 14 (right) shows that FoA substantially outperforms the existing SOTA methods at all possible price points. Based on the portrayed trends, ToT might be able to close the quality gap or even surpass FoA with a higher budget, however, FoA is always favorable for resource-constrained settings. **Overall, FoA achieves the best cost-quality trade-off**.

Trade-off between model-size and quality. Fig. 14 (left) shows that on their own, both the 11B and 90B Llama3.2 models [11] achieve poor quality on the benchmarked tasks. However, FoA boosts the performance of both models by portraying significant (5x–6x) quality improvements. What's more, Llama3.2-11B + FoA surpasses the larger Llama3.2-90B model. Overall, **FoA enables smaller models to obtain comparable or even better performance than larger models**, thereby bridging the gap between their reasoning abilities.

5 Ablation analysis

In this section, we report results on Game of 24 with GPT-3.5 and Llama3.2 (11B&90B) as base models. We observed similar trends for the other two tasks, results in Appx. C.6.

Impact of the selection phase. We remove the selection phase by setting the resampling frequency $k = 0$. Thus, each agent constantly remains in its own mutation phase, and independently works towards its goal until a solution is found, a terminal state is found, or time runs out. As expected, FoA (without selection) obtains an extremely low success rate but is also cheaper (Fig. 10a). This is because, without the selection phase, the fleet does not evolve into a composition of more high-value states with each time step.

Impact of resampling. Instead of using a resampling strategy, we assign each agent to the highest-value state during the selection phase. If there are multiple such states, the first one is randomly chosen by all agents. When there are multiple promising states, a meaningful resampling strategy may decide to explore all of them in the next time step, however, retaining only the highest-value state reduces the fleet diversity. Thus, FoA (no/max resampling) obtains a slightly lower success rate but also incurs a lower cost (Fig. 10b).

Impact of the backtracking mechanism. We vary the discount factor γ to study two aspects of the backtracking mechanism: (1) $\gamma = 0$ (no backtracking) and (2) $\gamma = 1$ (no decay). When $\gamma = 0$, resampling from a past state is not permitted, and thus, FoA cannot undo localized mistakes, resulting in a lower success rate but also lower cost (Fig. 10c). With $\gamma = 1$, all past states are considered during resampling, thereby increasing the spectrum of states to explore but at the risk of overwhelming the resampling mechanism with noisy value estimates of (relatively older) past states. This explains the reduction in success rate and a slight increase in cost owing to potentially futile explorations (Fig. 10c).

Impact of caching. Similar to ToT [51] and LATS [54], FoA utilizes a caching mechanism (details in Appx. C.4) to enhance its cost efficiency, which ensures that a given state is evaluated only once by an LLM. As expected, disabling the cache leads FoA to incur a higher cost, but interestingly, it also leads to a lower success rate (Fig. 10d). We hypothesize that frequent re-evaluations of the same state while solving a puzzle might introduce inconsistencies further disrupting the flow of reasoning structures, thereby leading to a reduction in quality.

Impact of batching. Similar to ToT [51] and LATS [54], FoA utilizes a batching mechanism (details in Appx. C.4) to enhance its cost efficiency, which groups many individual LLM calls into a single combined call (one input prompt leading to many output responses). Disabling batching leads FoA to incur a higher cost with no noticeable impact on quality (Fig. 10e).

In sum, Fig. 10 shows that each component of the FoA framework has an overall positive impact on its performance.

6 Discussion and concluding insights

6.1 Summary of findings

FoA achieves better quality than all existing methods. Based on the results presented in § 3 and Appx. C.6, FoA consistently outperforms all existing SOTA methods across all benchmarked tasks and base models.

FoA achieves a better cost-quality trade-off. Our results show that FoA is cost-efficient, much more so than other SOTA reasoning frameworks. Moreover, our analysis in § 4 reveals that FoA substantially outperforms all SOTA methods at all possible price points.

Other advantages. Beyond better performance, FoA offers many important practical advantages. First and most importantly, FoA is not a prompting scheme but a runtime. Unlike existing prompt-based frameworks, e.g. ToT [51], GoT [4], etc., FoA does not require custom-crafted prompts, but, instead, can be used in combination with any existing agent or prompting strategy. Next, FoA offers precise control over the tree's width (n agents) and depth (t steps), leading to predictable latency and cost. In particular, it is possible to tune the size n of the fleet to the available resources, such as an optimal batch size for a given hardware configuration. Finally, as evidenced by a smaller standard error (Appx. C.6), FoA generates consistent responses across multiple runs and is relatively more stable than other methods.

6.2 Implications and Broader Impact

Throughout an LLM's lifecycle, the majority of costs are incurred during inference rather than training [9]. Thus, it is crucial to benchmark the cost incurred by various SOTA reasoning frameworks. Our work is the first to report, analyze, and include a discussion on the trade-off between cost and quality of a variety of SOTA reasoning frameworks for LLMs. We hope that this work will spark further discussions on the cost-efficiency of reasoning frameworks and eventually lead to the development of practically feasible and sustainable reasoning frameworks for LLMs.

6.3 Limitations and future work

Constant fleet size. Currently, we assign a fixed number n of agents to each task. However, it may be advantageous to allocate more agents to more difficult tasks to enhance sample efficiency. In the future, we would like to explore the possibility of an adaptive fleet size.

Resampling mechanisms. One avenue for improvement is the design of more intelligent value functions, for example, pooling information from neighboring states and smoothing predictions. Building on precise value estimates, we can investigate different resampling mechanisms, to tune FoA towards either more risk-seeking or careful behavior, i.e., different tradeoffs between exploration and exploitation.

Fleet organization. Currently, we consider a homogenous fleet of identical agents. In the future, we would like to introduce further coordination between the individual agents with a hierarchical organizational structure. For instance, what if agents could spawn other agents in a nested particle filtering framework? Finally, as a runtime that embraces modular compatibility with any agent, FoA will also enable research into more complex fleet compositions.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2024.

[2] Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

[3] Akhil Arora, Martin Gerlach, Tiziano Piccardi, Alberto García-Durán, and Robert West. Wikipedia reader navigation: When synthetic data is enough. In *WSDM*, page 16–26, 2022.

[4] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *AAAI*, volume 38, pages 17682–17690, 2024.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *NeurIPS*, volume 33, pages 1877–1901, 2020.

[6] Harrison Chase. Langchain. `https://github.com/hwchase17/langchain`, 2022.

[7] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[8] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *NeurIPS: Datasets and Benchmarks Track*, 2022.

[9] Zhenxiao Fu, Fan Chen, Shan Zhou, Haitong Li, and Lei Jiang. Llmco2: Advancing accurate carbon footprint prediction for llm inferences. *arXiv preprint arXiv:2410.02950*, 2024.

[10] Hiroki Furuta, Kuang-Huei Lee, Ofir Nachum, Yutaka Matsuo, Aleksandra Faust, Shixiang Shane Gu, and Izzeddin Gur. Multimodal web navigation with instruction-finetuned foundation models. In *ICLR*, 2024.

[11] Aaron Grattafiori et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[12] Izzeddin Gur, Ofir Nachum, Yingjie Miao, Mustafa Safdari, Austin V Huang, Aakanksha Chowdhery, Sharan Narang, Noah Fiedel, and Aleksandra Faust. Understanding HTML with large language models. In *EMNLP*, 2023.

[13] Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *EMNLP*, 2023.

[14] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[15] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, and Chenglin Wu. Metagpt: Meta programming for multi-agent collaborative framework. *CoRR*, abs/2308.00352, 2023.

[16] Martin Josifoski, Lars Klein, Maxime Peyrard, Yifei Li, Saibo Geng, Julian Paul Schnitzler, Yuxing Yao, Jiheng Wei, Debjit Paul, and Robert West. Flows: Building blocks of reasoning and collaborating ai. *arXiv preprint arXiv:2308.01285*, 2023.

[17] Tomoyuki Kagaya, Thong Jing Yuan, Yuxuan Lou, Jayashree Karlekar, Sugiri Pranata, Akira Kinose, Koki Oguri, Felix Wick, and Yang You. Rap: Retrieval-augmented planning with contextual memory for multimodal llm agents. *arXiv preprint arXiv:2402.03610*, 2024.

[18] 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 *ICLR*, 2023.

[19] Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM compiler for parallel function calling. In *ICML*, 2024.

[20] Takeshi Kojima, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *NeurIPS*, pages 22199–22213, 2022.

[21] Bo Li, Wei Tian, Chufan Zhang, Fangfang Hua, Guangyu Cui, and Yufei Li. Positioning error compensation of an industrial robot using neural networks and experimental study. *Chinese Journal of Aeronautics*, 35(2):346–360, 2022.

[22] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large scale language model society. *arXiv preprint arXiv:2303.17760*, 2023.

[23] Zhihan Liu, Hao Hu, Shenao Zhang, Hongyi Guo, Shuqi Ke, Boyi Liu, and Zhaoran Wang. Reason for future, act for now: A principled architecture for autonomous LLM agents. In *ICML*, 2024.

[24] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. *ArXiv*, abs/2304.09842, 2023.

[25] Kaixin Ma, Hongming Zhang, Hongwei Wang, Xiaoman Pan, and Dong Yu. LASER: LLM agent with state-space exploration for web navigation. In *NeurIPS Workshops*, 2023.

[26] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

[27] Yannis Marinakis and Magdalene Marinaki. A hybrid genetic - particle swarm optimization algorithm for the vehicle routing problem. *Expert Systems with Applications*, 37(2):1446–1455, 2010.

[28] Shentong Mo and Miao Xin. Tree of uncertain thoughts reasoning for large language models. In *ICASSP*, pages 12742–12746, 2024.

[29] Yohei Nakajima. BabyAGI. `https://github.com/yoheinakajima/babyagi`, 2023.

[30] Maxwell I. Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models. *CoRR*, abs/2112.00114, 2021.

[31] Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. Refiner: Reasoning feedback on intermediate representations. *arXiv preprint arXiv:2304.01904*, 2023.

[32] Toran Bruce Richards. AutoGPT. `https://github.com/Significant-Gravitas/Auto-GPT`, 2023.

[33] Jose L. Salmeron and Wojciech Froelich. Dynamic optimization of fuzzy cognitive maps for time series forecasting. *Knowledge-Based Systems*, 105:29–37, 2016.

[34] Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*, 2023.

[35] Bilgehan Sel, Ahmad Tawaha, Vanshaj Khattar, Ruoxi Jia, and Ming Jin. Algorithm of thoughts: Enhancing exploration of ideas in large language models. In *ICML*, 2024.

[36] Yongliang Shen, Kaitao Song, Xu Tan, Dong Sheng Li, Weiming Lu, and Yue Ting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *ArXiv*, abs/2303.17580, 2023.

[37] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *NeurIPS*, pages 8634–8652, 2023.

[38] Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. In *NeurIPS*, volume 36, pages 58202–58245, 2023.

[39] Mirac Suzgun and Adam Tauman Kalai. Meta-prompting: Enhancing language models with task-agnostic scaffolding. *arXiv preprint arXiv:2401.12954*, 2024.

[40] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *ArXiv*, abs/2302.13971, 2023.

[41] Hugo Touvron, Louis Martin, Kevin R. Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[42] Xiaoxuan Wang, Ziniu Hu, Pan Lu, Yanqiao Zhu, Jieyu Zhang, Satyen Subramaniam, Arjun R. Loomba, Shichang Zhang, Yizhou Sun, and Wei Wang. Scibench: Evaluating college-level scientific problem-solving abilities of large language models, 2024.

[43] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

[44] 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 *ICLR*, 2023.

[45] Yancheng Wang, Ziyan Jiang, Zheng Chen, Fan Yang, Yingxue Zhou, Eunah Cho, Xing Fan, Yanbin Lu, Xiaojiang Huang, and Yingzhen Yang. Recmind: Large language model powered agent for recommendation. In *NAACL-HLT (Findings)*, pages 4351–4364, 2024.

[46] 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 *NeurIPS*, pages 24824–24837, 2022.

[47] Robert West, Joelle Pineau, and Doina Precup. Wikispeedia: An online game for inferring semantic distances between concepts. In *IJCAI*, page 1598–1603, 2009.

[48] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversation framework. *CoRR*, abs/2308.08155, 2023.

[49] 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 *NeurIPS*, 2024.

[50] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In *NeurIPS*, pages 20744–20757, 2022.

[51] 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. *NeurIPS*, 36, 2024.

[52] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *ICLR*, 2023.

[53] Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. ReST-MCTS*: LLM self-training via process reward guided tree search. In *NeurIPS*, 2024.

[54] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *ICML*, 2024.

[55] 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 *ICLR*, 2023.

A Related Work

In this section, we review works that overlap closely with our study.

Prompt-based reasoning. Recent research focuses on developing strategies to enhance the reasoning capabilities of LLMs. Few-shot prompting employs demonstrations of high-quality input/output samples to exploit the eagerness of the LLMs to imitate patterns seen in their context window [5]. Algorithm of thoughts (AoT) [35], goes a step further by including algorithmic examples within the prompt to propel the LLM through algorithmic reasoning pathways. Chain-of-Thought (CoT) prompting [30, 46, 20] as well as other variants such as Decomposed Prompting [18] and Least-to-Most [55] guide LLMs to decompose a complex question into a sequence of thoughts and then synthesize an answer by resolving them methodically. It has been shown that Self-Consistency (CoT-SC) [43] can be used to augment such methods by generating multiple thought sequences and then selecting the most accurate answer through majority voting. Recent meta-prompting techniques [39] employ a uniform, task-independent prompting framework across multiple tasks, enabling a single LLM to iteratively refine its responses and dynamically adapt to diverse input queries. The Buffer of Thoughts (BoT) [49] framework extracts task-specific information, uses it to retrieve relevant thought templates from its meta-buffer, and then instantiates them with more task-specific reasoning structures before continuing with the reasoning process.

Refinement. Closed-loop approaches that allow an LLM to interact with an external environment can help in choosing and potentially revising an action. Notable examples are ReAct [52], REFINER [31] and Self-Refine [26]. Reflexion [37] provides further linguistic feedback based on previous attempts while AdaPlanner [38] also incorporates positive and negative feedback of an individual trajectory. Reason for future, act for now (RAFA) [23] develops further by planning a trajectory, gathering feedback for the potential planned actions, and then revising the trajectory based on the feedback.

Tree search. Thoughts are individual ideas or steps in reasoning, and when connected together, they can be modeled as a tree data structure. Tree search algorithms can then be used to explore the tree of thoughts and optimize the search for a final answer. In "Tree of Thoughts" (ToT), the authors utilize a value function that compares different branches to describe both DFS and BFS flavors of a guided tree-search [51]. The closely related "Graph of Thoughts" (GoT) approach relaxes the assumption of a strict tree structure [4]. Reasoning via Planning (RAP) [13] augments LLMs with a world model and employs Monte Carlo Tree Search (MCTS)-based planning to reduce the search complexity. Language Agent Tree Search (LATS) [54] extends this concept by leveraging environment interactions, thereby eliminating the need for a world model. ReST-MCTS* [53] builds on this line by integrating process-level rewards into MCTS, by identifying high-quality reasoning traces.

Particle filtering. Genetic particle filters have been used successfully across a large variety of domains, ranging from vehicle routing [27], to fuzzy cognitive maps in time series forecasting [33], to the positioning of industrial robots [21]. Near-universally, the efficacy of a genetic particle optimization approach has, in these contexts, been demonstrated on standard artificial neural networks, however, to the best of our knowledge, it has not been employed on LLMs.

Key differences. FoA exhibits fundamental differences from all aforementioned methods. First, it does not require extensive or intricate prompt engineering, unlike AoT [35] and meta prompting [39].

Additionally, it is well-suited for sequential decision-making tasks that require interactions with an environment, such as web navigation [50, 3, 47], which can be challenging for methods such as CoT [46] and BoT [49]. Furthermore, FoA distinguishes itself from approaches like Reflexion [37] and RAFA [23] by utilizing a refinement strategy based on a genetic-type particle filter, instead of linguistic feedback. Lastly, unlike tree-search-based methods [51, 54, 4], FoA employs a more principled approach to search tree exploration, achieving a better balance between exploration and exploitation. This structured approach also grants FoA precise control over the tree's width and depth, leading to predictable latency and cost.

AI agents. AI agents extend the capabilities of LLMs by integrating external tools or orchestrating collaborations between multiple LLMs. To solve a task, such an AI agent may take many individual steps, usually one depending on the other, e.g., querying a database, searching with a web browser, running computations in a code interpreter, etc. There exists a diverse and quite fragmented ecosystem of frameworks and libraries that motivate specific interaction patterns or are designed to offer the flexibility to implement new forms of collaboration. A well-established library is LangChain [6], but many practitioners chose to accompany their research with a custom solution. Cameleon [24], Camel [22], HuggingGPT [36], AutoGPT [32], BabyAGI [29], MetaGPT [15], Flows [16], and AutoGen [48], all these works present some form of blueprint or framework for building AI agents.

Building on these works, creating highly sophisticated AI agents is possible. Nevertheless, even if an action is only taken after many steps of deliberation and careful consideration, the agent must ultimately follow a sequential chain of actions on its way toward a solution. This is fundamentally similar to structured reasoning, i.e., the agent approaches a solution in smaller discrete steps, each step following logically from the previous steps. In many scenarios, multiple actions may be promising; the agent is faced with uncertainty and a large state space to explore. In this setting, the perspective of a single agent is necessarily a myopic and localized worldview. We overcome this limitation by instantiating a fleet of individual agents and coordinating a joint search process between them. This research is orthogonal to existing work on optimizing AI agents: Instead of a novel way to build or improve a specific AI agent, we implement a runtime that can readily accommodate any existing agent and multiply its capabilities.

B Fleet of agents: additional details

Algorithm 1 Fleet of Agents: A genetic particle filter.

Initialize
$t \leftarrow 0$
$s_{i,t} \leftarrow \mathbf{x}_0, i = 1..N$ {setting initial state}
loop
  Mutation phase
  $s_{i,t+k} \leftarrow$ Call Algorithm 2 with $s_{i,t}$, $i = 1..N$ and S_t

  Selection phase
  $s_{i,t+k} \leftarrow$ Call Algorithm 3 with S_{t+k}

  $t \leftarrow t + k$
end loop

B.1 Comparison of FoA with a standard tree-search algorithm

We posit that any multi-step structured reasoning prompt will rely on two flavors of prompts: *mutation steps*, in which the current state of the reasoning process is advanced, and *evaluation steps*, in which different states are judged and compared. The key difference between structured reasoning algorithms lies in how these steps are orchestrated. In Fig. 6, we show a standard tree search algorithm, based on one of the Tree-of-Thoughts (ToT) algorithms, side by side with Fleet of Agents.

At every step during the search for a solution, ToT keeps track of b thoughts, generates c follow-up thoughts for each, estimates their value, and from the cb candidate thoughts, selects the b best. It is important to note:

Algorithm 2 Mutation Phase

Input: List of current states $s_{i,t}$, $i = 1..N$ and states S_t at time t
Output: List of states $s_{i,t+k}$, $i = 1..N$ at time $t + k$
for $j = 1$ to k **do**
  Each agent independently mutates its state
  $s_{i,t+1} \sim \pi(s|s_{i,t}), i = 1..N$
  $t \leftarrow t + 1$
  Check for solution
  if $\exists s \in S_t : \mathbb{1}_{solution}(s)$ **then**
    We have found a solution: early exit
  end if
  Identify and prune terminal states,
  resample across all viable states S_t discovered so far
  $B \leftarrow \{s_{i,t}|\mathbb{1}_{terminal}(s_{i,t})\}$
  $G \leftarrow \{s|s \in S, s \notin B\}$
  for $i \in B$ **do**
    $\hat{s}_i \sim \text{Uniform}(G)$
    $s_{i,t} \leftarrow \hat{s}_i$
  end for
end for

Algorithm 3 Selection Phase

Input: Set of states S_t at time t
Output: Resampled list of states $s_{i,t}$, $i = 1..N$ at time t
Evaluate heuristic value function
$v_s \leftarrow v(s), s \in S_t$
Calculate the resampling distribution
$p_s = \exp(1\beta v_s), s \in S_t$, {resampling weight}
$p(s) = \frac{1}{\sum_{\hat{s}\in S} p_{\hat{s}}} \sum_{s\in S} p_s \delta_{s_{i,t}(s)}$ {categorical resampling distribution}
$\hat{s}_i \overset{iid}{\sim} p_t(s), i = 1..N$ {resampling with replacement}
$s_{i,t} = \hat{s}_i, i = 1..N$

- Mutation and evaluation prompts are called in lockstep; each new thought is immediately evaluated.
- At every step, $cb - b$ candidate thoughts are discarded. Depending on the parametrization, the majority of mutation and value calls correspond to thoughts that immediately become dead ends.
- The number of promising states selected and the number of follow-up thoughts are fixed, resulting in a constant branching factor.

This ToT algorithm closely resembles beam search using a value function as a heuristic. By comparison, FLEET OF AGENTS follows a different design philosophy. Instead of modeling the multi-step reasoning process as a graph traversal or heuristic tree search, we take inspiration from the concept of individual AI agents exploring a complex search space on their own. We believe that for many real-world applications, it is best to trust the individual agent to make local decisions based on their best judgment. Rather than exploring c branches at each step, we allow our fleet of N agents to make a series of educated guesses.

We evaluate the states of individual agents only every k steps. Instead of selecting the b best states, we then resample with replacement. If one of the agents has found an extremely promising state, it is duplicated and the search focuses on the region that agent is exploring. This corresponds to a decision to exploit the findings of this agent. Conversely, if all agents have a similar value, none of them may be duplicated during the resampling process, keeping maximum diversity in the fleet of agents. This corresponds to a decision to explore further.

Refer to the first resampling stage in Fig. 6, which shows that one agent is duplicated (replacing an agent of very low value) while three others are kept. In practice, this means that we allocate a branching factor of 2 to the duplicated agent and a branching factor of 1 to the rest of the fleet. This

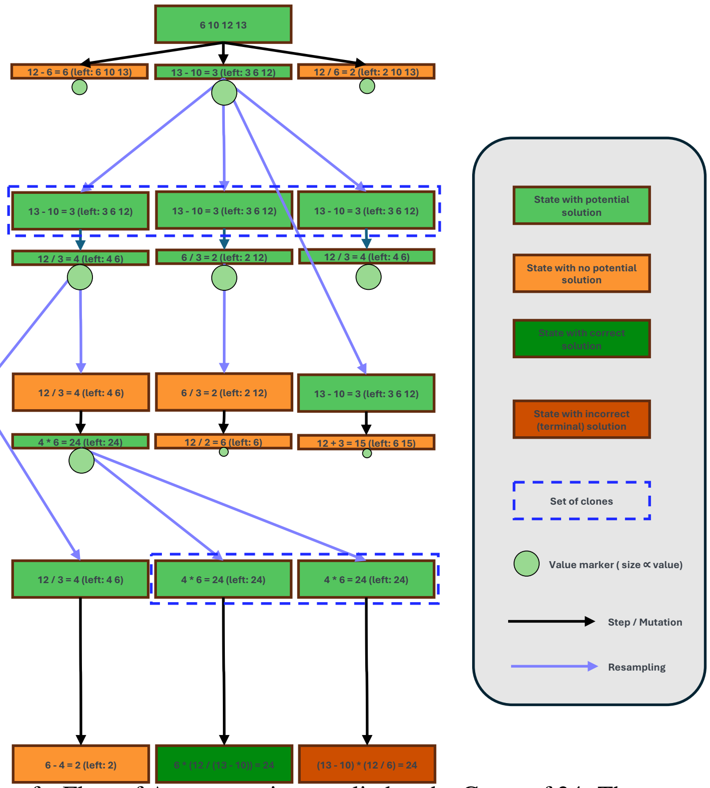

Figure 5: Example of a Fleet of Agents runtime applied to the Game of 24. Three agents are deployed, and the runtime undergoes the selection phase every $k = 1$ steps. A positive discount factor $\gamma > 0$ is retained to allow backtracking to previous states. During the resampling in the mutation phase, agents that have taken incorrect actions are corrected by replacing their state with one that has a potential solution.

is a reasonable choice: When we have yet to observe a distinct difference in thought value, then it is better to explore further before concentrating the search. In the second resampling stage, we note that one thought has a much higher value than the others. The resampling creates four copies of this thought but also retains one medium-value state. In this toy scenario, ToT always explores the dynamic search tree with a fixed branching factor of 3, whereas FLEET OF AGENTS selects branching factors dynamically, ranging from 1 to 4.

Notable properties of FLEET OF AGENTS include:

• Individual decisions are made according to each agent's localized best judgment. This enables quick and efficient spread across the search space.

• Resampling (i.e., discarding some agents) occurs only every k steps. Ideally, only thoughts clearly worse than their competitors are discarded.

• The resampling process provides an intuitive trade-off between exploitation and exploration. In practice, FLEET OF AGENTS can dynamically choose a branching factor between 1 (all agents are retained, maximizing exploration) and N (one agent finds a highly promising state and the search is focused accordingly).

C Additional Experimental Details

C.1 Main experiments

Number of runs. For cost reasons, experiments with GPT-4 were run only once. However, for other base models (results in Appx. C.6), we run each experiment 5 times and report both the mean and standard error of the evaluation metrics.

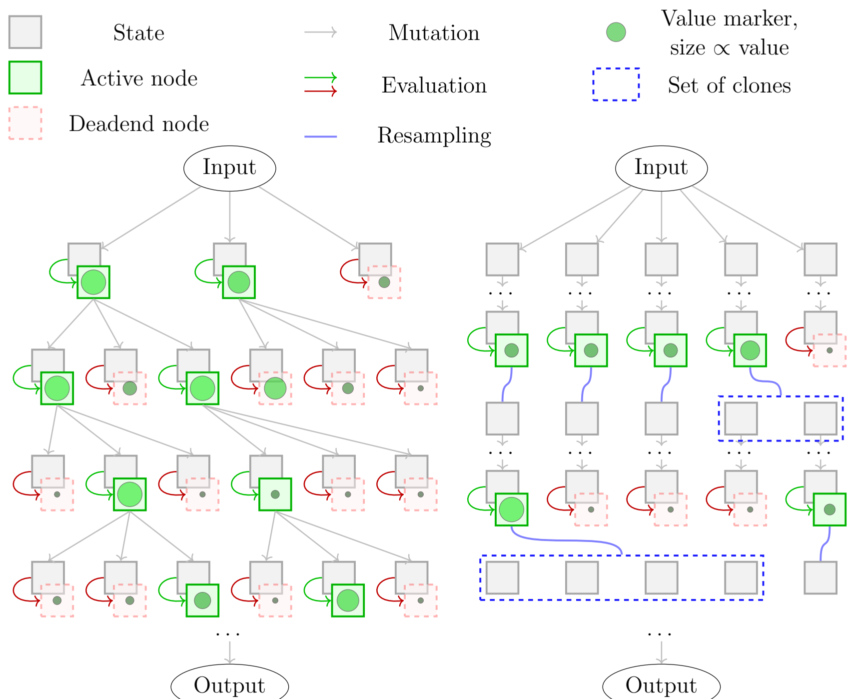

Figure 6: (Left) Search-tree for Tree-of-thoughts (ToT) that generates $c = 3$ candidates for the next thought step and maintains a set of $b = 2$ most promising states at each step. (Right) Fleet-of-Agents (FoA) comprising $N = 5$ agents that think autonomously for k steps and are then resampled to focus the search on promising regions.

C.1.1 Game of 24

Task and data. Game of 24 is a mathematical puzzle, where four numbers are given, and the objective is to form an arithmetic expression that equals 24 using each number exactly once. The benchmark data consists of 1362 puzzles. Following ToT [51], we use the puzzles indexed 901-1000 as the test set (cf. Appx. C.2 for details).

Evaluation metrics. We use *success rate*, i.e., the percentage of solved puzzles, to evaluate the quality of the benchmarked methods. For efficiency, we use cost (in US$).

Baselines. We compare FoA with: (1) Standard IO prompting, (2) CoT [46], (3) CoT-SC [44], (4) AoT [35], (5) ToT [51], (6) GoT [4], and (7) RAFA [23]. Owing to the unavailability of their code for Game of 24, we do not compare with LATS [54].

Results. Table 1 shows that FoA outperforms all existing baselines and achieves the best quality. Taking GPT-4 as a baseline, FoA achieves a whopping 70% improvement in quality. On the one hand, IO and CoT [46] are the most cost-effective methods, their success rate is extremely low at just 6%. On the other hand, sophisticated reasoning frameworks like ToT [51] achieve a high success rate (74%) but also incur a high cost (75$). Striking a good balance between exploration vs. exploitation, our FoA obtains a *2% improvement in quality over the second best method, ToT, simultaneously lowering the cost requirement by 25%.*

C.1.2 Mini Crosswords

Task and data. Mini Crosswords is a puzzle, where, given 5 vertical and 5 horizontal clues, the objective is to use the clues to identify answers and place them on a 5×5 crossword board. The benchmark data consists of 156 puzzles. Following ToT [51], we use the puzzles 0, 5, . . . , 90, and 95 as the test set (cf. Appx. C.2 for details).

Evaluation metrics. For quality, we use *overlap*, i.e., the percentage of correct letters in the proposed solution. For efficiency, we compute the cost (in US$).

Baselines. We compare FoA with: (1) Standard IO prompting, (2) CoT [46], (3) CoT-SC [44], (4) ToT [51], and (5) GoT [4]. Owing to the unavailability of their code for Mini Crosswords, we do not compare with AoT [35].

Results. Table 2 (Left) shows that FoA outperforms all existing baselines and achieves the best quality. Once again, IO and CoT [46] are the most cost-effective methods. Moreover, they also obtain good performance with their quality being comparable to ToT [51] and GoT [4] at just a fraction (2.5%) of the cost. Notably, our FoA reports the best cost-quality trade-off among all the benchmarked methods. We obtain a *5% improvement in quality over the second best method, GoT, simultaneously lowering its cost requirement by 60%*.

C.1.3 WebShop

Task and data. WebShop [50] is a simulated e-commerce website environment, where, given a textual instruction specifying a product and its properties, the objective is to find the product by navigating webpages using a variety of actions and purchase it. The benchmark data consists of 12,087 subtasks. Following [52, 54, 37], we use 50 randomly sampled subtasks as the test set (cf. Appx. C.2 for details).

Evaluation metrics. The quality of a purchase is assessed using an environment-generated reward, which measures the percent overlap between the purchased product and the user-specified attributes. We use *average score*, i.e., the average of subtask rewards, to evaluate the quality of the benchmarked methods. For efficiency, we use cost (in US$).

Baselines. We compare FoA with: (1) Act [52], (2) ReAct [52], (3) Reflexion [37], (4) LASER [25], (5) LATS [54], and (6) multiple fine-tuned models from **(author?)** [50]. We also use the performance of human experts as an upper bound for quality. Owing to the unavailability of their code for WebShop, we do not compare with RAP [17].

Results. Table 2 (Right) shows that FoA outperforms all existing baselines, even the supervised fine-tuned models [50, 12, 10], and achieves the best quality. While Act [52] is by far the cheapest method (0.1$), it obtains a moderate average score (58.1%). LATS [54], on the other hand, obtains a better average score (66.1%), it suffers from an exorbitant cost footprint (232$). Yet again, our FoA achieves the best cost-quality trade-off: obtaining a *10% improvement in quality over the second-best method, LATS, requiring only 1% of its cost*.

C.2 Detailed Task Descriptions

C.2.1 Game of 24

Game of 24 is a mathematical puzzle where the participants are presented with four numbers, and their objective is to find a combination of arithmetic operations (+-*/) to construct an arithmetic expression that uses each given number exactly once to obtain a final total of 24.

The benchmark consists of 1362 such puzzles scraped from 4nums.com, which are sorted in increasing order of their difficulty. The input data in each puzzle are the four initial numbers and the expected output is an equation that equals 24. Following ToT [51], we use the puzzles indexed 901–1000 as the test set. We also created a validation set from the puzzles indexed 876–900 and 1001–1025. The validation set is constructed such that, in expectation, its overall difficulty is similar to the test set.

C.2.2 Mini Crosswords

Mini Crosswords is a puzzle where participants are given 5 vertical and 5 horizontal clues. Each clue leads to a 5-letter word, and the objective is to use the clues to identify answers and place them on a 5×5 crossword board. We use the percentage of correct letters to measure the quality of a proposed crossword solution. For a letter to be correct, it has to match both the letter and its position on the ground truth board.

The benchmark constitutes 156 such puzzles scraped from GooBix. Following ToT [51], we use the puzzles $0, 5, \ldots, 90$, and 95 as the test set. We also created a validation set from the puzzles indexed $3, 8, \ldots, 93$, and 98.

C.2.3 WebShop

WebShop [50] is a simulated e-commerce website environment comprising 1.18 million real-world products and 12,087 crowd-sourced textual instructions. The participants are provided with textual instructions specifying a product and its properties, and their objective is to find and purchase the product by navigating webpages using a variety of actions.

The benchmark data consists of 12,087 subtasks. We noticed that the website environment randomizes the set of subtasks upon every initialization. Thus, to fix the test set across all the experiments and methods benchmarked in this study, we add a fixed random seed to the website environment. Following [52, 54, 37], we use 50 subtasks to construct the test set. Specifically, we use the subtasks indexed 5–54 as the test set. We also created a validation set from subtasks indexed 55–69.

C.3 Detailed Baseline Descriptions

C.3.1 Game of 24

- Input-Output (IO) prompting uses the LLM to directly generate an output, with no intermediate steps.
- Chain-of-Thought (CoT) [46] solves the problem step by step by decomposing it into a sequence of thoughts.
- Chain-of-Thought [46] with Self-Consistency [43] CoT-SC, generates multiple responses for the same CoT prompt and then selects the best one based on majority voting.
- Algorithm-of-Thoughts (AoT) [35] (AoT), guides the reasoning through algorithmic pathways by including such examples in its prompt.
- Tree-of-Thoughts (ToT) [51]. decomposes the problem into multiple chain of thoughts, organized in a tree structure. Thought evaluation and search traversal algorithms are utilized to solve the problem.
- Graph of Thoughts (GoT) [4] allows the organization of thoughts in a graph structure. It introduces arbitrary graph-based thought transformations such as thought aggregation and thought refinement.
- Reason for futre, act for now (RAFA) [23] structures its reasoning by initially implementing a potential plan for a trajectory. Then, feedback is gathered for actions included in the plan. Finally, a new plan is generated with the gathered feedback in context. Even though we compared with RAFA using models GPT3.5 and Llama 3.2 11B, we did not compare with GPT4 or Llama 3.2 90b as its cost would be prohibitive.
- Reasoning via Planning (RAP) [13] augments LLMs with a world model and employs Monte Carlo Tree Search (MCTS)-based planning to generate and traverse its thought process. Language Agent Tree Search (LATS) [54] extends this concept by leveraging environment interactions, thereby eliminating the need for a world model. We did not compare with any of these two methods as code for the game of 24 task was not available.
- The Buffer of Thoughts (BoT) [49] framework extracts task-specific information, uses it to retrieve relevant thought templates from its meta-buffer, and then instantiates them with more task-specific reasoning structures before continuing with the reasoning process. For BoT [49] we were unable to reproduce the results reported in their paper as a component of their method (meta-buffer) is not available and we have no detailed instructions on how to recreate it ourselves.
- The LLMCompiler [19] is an LLM compiler that optimizes the parallel function calling performance of LLMs. There was no code available for the task of Game of 24, so we do not compare against it.
- Tree of uncertain Thoughts (TouT) [28] leverages Monte Carlo Dropout to quantify uncertainty scores associated with LLMs' diverse local responses at intermediate thoughts. This local uncertainty quantification is then united with global search algorithms. There was no code available so we do not compare against TouT.
- ReST-MCTS* [53] introduces a self-training framework that uses a modified Monte Carlo Tree Search (MCTS*) guided by process-level rewards. Instead of relying solely on final answers, it infers per-step rewards to identify and reinforce high-quality reasoning traces,

improving the LLM's reasoning ability over successive iterations. In our experiments, we only tested the MCTS* component for inference-time reasoning, without the full iterative self-training loop, to ensure a fair comparison, as all other baselines were also evaluated purely in the inference-time setting.

C.3.2 Mini Crosswords

- Input-Output (IO) prompting uses the LLM to directly generate an output, with no intermediate steps.
- Chain-of-Thought (CoT) [46] solves the problem step by step by decomposing it into a sequence of thoughts.
- Chain-of-Thought [46] with Self-Conistency [43] CoT-SC, generates multiple responses for the same CoT prompt and then selects the best one based on majority voting.
- Algorithm-of-Thoughts (AoT) [35] (AoT), guides the reasoning through algorithmic pathways by including such examples in its prompt. For its Mini Crosswords implementation, AoT utilizes 2 prompts that need to be run sequentialy and provides both of them. However, in between the two prompts a necessary step is performed which extracts the word combination of the highest "compatibility". No further details are found about this step and since it can be interpreted in a number of ways we chose not to move on with it.
- Tree-of-Thoughts (ToT) [51]. decomposes the problem into multiple chain of thoughts, organized in a tree structure. Thought evaluation and search traversal algorithms are utilized to solve the problem.
- Graph of Thoughts (GoT) [4] allows the organization of thoughts in a graph structure. It introduces arbitrary graph-based thought transformations such as thought aggregation and thought refinement.

C.3.3 WebShop

- Act [52] simply prompts the framework to perform an action within a closed loop.
- ReAct [52] integrates reasoning into Act by allowing the model to think instead of explicitly performing an action to the environment.
- Reflexion [37] generates linguistic feedback that is utilized during subsequent runs.
- Agent with State-Space ExploRation (LASER) [25] models environment interactive tasks as state-space exploration. This is achieved by allowing the LLM agent to transition among a pre-defined set of states by performing actions to complete the task.
- Language Agent Tree Search (LATS) [54] employs Monte Carlo Tree Search (MCTS)-based planning to generate and traverse its thought process, leveraging environment interactions. Even though we compare with LATS using GPT3.5, we do not repeat the experiment for any other model as it is prohibitively expensive.
- Multiple deep learning approaches[50]. We also compare with a variety of deep learning approaches such as supervised, reinforcement and imitation learning. We report their average score as given in their published paper.
- Human Experts: A human annotators have been recruited by the Webshop authors [50] to study their trajectories. Based on their results, thirteen of them are recruited and trained further. Finally, the top 7 performers are selected as experts.[50]
- Retrieval-Augmented Planning (RAP) [17] dynamically leverage past experiences corresponding to the current situation and context in both textual and multimodal environments.
- The LLMCompiler [19] is an LLM compiler that optimizes the parallel function calling performance of LLMs. There was no code available for the task of WebShop, so we do not compare against it.

C.4 Implementation Details

Platforms. GPT models were were accessed through the OpenAI API while the utilization of the Llama models was facilitated by the TogetherAI API.

Model checkpoints and prices. To compute the costs of our experiments we used the current model prices indicated OpenAI and Together AI, accordingly to the model. The specific models snapshot we used, along with their respective prices are presented in 3

	US$ per 1m prompt tokens	US$ Per 1m completion tokens
gpt-3.5-turbo-0125	0.5	1.5
gpt-4-0613	30.0	60.0
Llama-3.2-90B-Vision-Instruct-Turbo	1.2	0.06
Llama-3.2-11B-Vision-Instruct-Turbo	0.18	0.18

Table 3: **Model snapshot prices.** OpenAI and TogetherAI prices for each model used, during the implementation of the project.

Model configurations. Generation parameters specified when making calls to any of the models used throughout this project. These parameters were not defined by us, but by the implementation where the respective prompts where introduced. Specifically, Game of 24 and Mini Crosswords parameters were used from [51], WebShop step request parameters was taken from [52] and WebShop evaluate request parameters from [54]. Configurations presented in Table 4

	max_tokens	temperature	top_p	stop
Game of 24	100	0.7	1	null
Mini Crosswords	1000	0.7	1	null
WebShop (step)	100	1	1	["\n"]
WebShop (eval)	100	1	1	null

Table 4: **Generation parameters.** Generation parameters specified when making requests to any model.

Base model selection strategy. We selected GPT4 to be our base model as it was the one for which the prompts we used were originally designed for. Excepions were made for the RAFA [23] and LATS [54] baselines as their cost was prohibitive for us to run using GPT4. Finally, for the task of WebShop, we didn't repeat any baseline for GPT4. That was because firstly, the price was extremely steep for some of the baselines and secondly because some of the baselines had already achieved near-human level of performance.

Prompts. This section provides all the prompts used for the models evaluated in our experiments. We include the exact phrasing and formatting of each prompt to ensure reproducibility and allow for detailed examination of how the tasks were presented to the models.

```
Input: 2 8 8 14
Possible next steps:
2 + 8 = 10 (left: 8 10 14)
8 / 2 = 4 (left: 4 8 14)
14 + 2 = 16 (left: 8 8 16)
2 * 8 = 16 (left: 8 14 16)
8 - 2 = 6 (left: 6 8 14)
14 - 8 = 6 (left: 2 6 8)
14 /  2 = 7 (left: 7 8 8)
14 - 2 = 12 (left: 8 8 12)
Input: {input}
Possible next steps:
```

Prompt 1: **Game of 24 - Step prompt** The prompt used to generate candidate new states. Taken from [51].

```
Use numbers and basic arithmetic operations (+ - * /) to obtain 24.
    Each step, you are only allowed to choose two of the remaining
    numbers to obtain a new number.
Input: 4 4 6 8
Steps:
4 + 8 = 12 (left: 4 6 12)
```

```
6 - 4 = 2 (left: 2 12)
2 * 12 = 24 (left: 24)
Answer: (6 - 4) * (4 + 8) = 24
Input: 2 9 10 12
Steps:
12 * 2 = 24 (left: 9 10 24)
10 - 9 = 1 (left: 1 24)
24 * 1 = 24 (left: 24)
Answer: (12 * 2) * (10 - 9) = 24
Input: 4 9 10 13
Steps:
13 - 10 = 3 (left: 3 4 9)
9 - 3 = 6 (left: 4 6)
4 * 6 = 24 (left: 24)
Answer: 4 * (9 - (13 - 10)) = 24
Input: 1 4 8 8
Steps:
8 / 4 = 2 (left: 1 2 8)
1 + 2 = 3 (left: 3 8)
3 * 8 = 24 (left: 24)
Answer: (1 + 8 / 4) * 8 = 24
Input: 5 5 5 9
Steps:
5 + 5 = 10 (left: 5 9 10)
10 + 5 = 15 (left: 9 15)
15 + 9 = 24 (left: 24)
Answer: ((5 + 5) + 5) + 9 = 24
Input: {input}
```

Prompt 2: **Game of 24 - Last step prompt** In the game of 24, once all initial numbers were combined, if the resulting number is 24, then the following chain of thought prompt was used to summarized the operations that have taken place to get there [51].

```
Evaluate if given numbers can reach 24 (sure/likely/impossible)
10 14
10 + 14 = 24
sure
11 12
11 + 12 = 23
12 - 11 = 1
11 * 12 = 132
11 / 12 = 0.91
impossible
4 4 10
4 + 4 + 10 = 8 + 10 = 18
4 * 10 - 4 = 40 - 4 = 36
(10 - 4) * 4 = 6 * 4 = 24
sure
4 9 11
9 + 11 + 4 = 20 + 4 = 24
sure
5 7 8
5 + 7 + 8 = 12 + 8 = 20
(8 - 5) * 7 = 3 * 7 = 21
I cannot obtain 24 now, but numbers are within a reasonable range
likely
5 6 6
5 + 6 + 6 = 17
(6 - 5) * 6 = 1 * 6 = 6
I cannot obtain 24 now, but numbers are within a reasonable range
likely
10 10 11
10 + 10 + 11 = 31
(11 - 10) * 10 = 10
10 10 10 are all too big
```

```
impossible
1 3 3
1 * 3 * 3 = 9
(1 + 3) * 3 = 12
1 3 3 are all too small
impossible
{input}
```

Prompt 3: **Game of 24 - Value prompt** The prompt used to evaluate a state [51].

```
Let's play a 5 x 5 mini crossword, where each word should have exactly
     5 letters.

{input}

Given the current status, list all possible answers for unfilled or
    changed words, and your confidence levels (certain/high/medium/low
    ), using the format "h1. apple (medium)". Use "certain" cautiously
     and only when you are 100% sure this is the correct word. You can
     list more then one possible answer for each word.
```

Prompt 4: **Mini Crosswords - Step prompt** The prompt used to generate candidate new states [51].

```
Evaluate if there exists a five letter word of some meaning that fit
    some letter constraints (sure/maybe/impossible).

Incorrect; to injure: w _ o _ g
The letter constraint is: 5 letters, letter 1 is w, letter 3 is o,
    letter 5 is g.
Some possible words that mean "Incorrect; to injure":
wrong (w r o n g): 5 letters, letter 1 is w, letter 3 is o, letter 5
    is g. fit!
sure

A person with an all-consuming enthusiasm, such as for computers or
    anime: _ _ _ _ u
The letter constraint is: 5 letters, letter 5 is u.
Some possible words that mean "A person with an all-consuming
    enthusiasm, such as for computers or anime":
geek (g e e k): 4 letters, not 5
otaku (o t a k u): 5 letters, letter 5 is u
sure

Dewy; roscid: r _ _ _ l
The letter constraint is: 5 letters, letter 1 is r, letter 5 is l.
Some possible words that mean "Dewy; roscid":
moist (m o i s t): 5 letters, letter 1 is m, not r
humid (h u m i d): 5 letters, letter 1 is h, not r
I cannot think of any words now. Only 2 letters are constrained, it is
     still likely
maybe

A woodland: _ l _ d e
The letter constraint is: 5 letters, letter 2 is l, letter 4 is d,
    letter 5 is e.
Some possible words that mean "A woodland":
forest (f o r e s t): 6 letters, not 5
woods (w o o d s): 5 letters, letter 2 is o, not l
grove (g r o v e): 5 letters, letter 2 is r, not l
I cannot think of any words now. 3 letters are constrained, and _ l _
    d e seems a common pattern
maybe

An inn: _ d _ w f
```

```
The letter constraint is: 5 letters, letter 2 is d, letter 4 is w,
    letter 5 is f.
Some possible words that mean "An inn":
hotel (h o t e l): 5 letters, letter 2 is o, not d
lodge (l o d g e): 5 letters, letter 2 is o, not d
I cannot think of any words now. 3 letters are constrained, and it is
    extremely unlikely to have a word with pattern _ d _ w f to mean "
    An inn"
impossible

Chance; a parasitic worm; a fish: w r a k _
The letter constraint is: 5 letters, letter 1 is w, letter 2 is r,
    letter 3 is a, letter 4 is k.
Some possible words that mean "Chance; a parasitic worm; a fish":
fluke (f l u k e): 5 letters, letter 1 is f, not w
I cannot think of any words now. 4 letters are constrained, and it is
    extremely unlikely to have a word with pattern w r a k _ to mean "
    Chance; a parasitic worm; a fish"
impossible

{input}
```

Prompt 5: **Mini Crosswords - Value prompt** The prompt used to evaluate a state. Specifically, this prompt evaluates a potential solution of 1 out of the 10 questions of the Crossword. To get the value of the overall state this prompt was called for each row/column of the crosswords board with a potential solution [51].

```
Webshop
Instruction:
i would like a 3 ounce bottle of bright citrus deodorant for sensitive
     skin, and price lower than 50.00 dollars
[Search]

Action: search[3 ounce bright citrus deodorant sensitive skin]
Observation:
[Back to Search]
Page 1 (Total results: 50)
[Next >]
[B078GWRC1J]
Bright Citrus Deodorant by Earth Mama | Natural and Safe for Sensitive
     Skin, Pregnancy and Breastfeeding, Contains Organic Calendula 3-
    Ounce
$10.99
[B078GTKVXY]
Ginger Fresh Deodorant by Earth Mama | Natural and Safe for Sensitive
    Skin, Pregnancy and Breastfeeding, Contains Organic Calendula 3-
    Ounce
$10.99
[B08KBVJ4XN]
Barrel and Oak - Aluminum-Free Deodorant, Deodorant for Men, Essential
     Oil-Based Scent, 24-Hour Odor Protection, Cedar & Patchouli Blend
    , Gentle on Sensitive Skin (Mountain Sage, 2.7 oz, 2-Pack)
$15.95

Action: think[B078GWRC1J and B078GTKVXY are bright citrus deodorant
    less then 50 dollars. I can check B078GWRC1J first.]
Observation: OK.

Action: click[B078GWRC1J]
Observation:
[Back to Search]
[< Prev]
scent [assorted scents][bright citrus][calming lavender][ginger fresh
    ][simply non-scents]
size [travel set (4-pack)][3 ounce (pack of 1)][3-ounce (2-pack)]
```

```
Bright Citrus Deodorant by Earth Mama | Natural and Safe for Sensitive
     Skin, Pregnancy and Breastfeeding, Contains Organic Calendula 3-
    Ounce
Price: $10.99
Rating: N.A.
[Description]
[Features]
[Reviews]
[Buy Now]

Action: think[For 3 ounce bottle of bright citrus deodorant for
    sensitive skin, the item has options 'bright citrus' and '3 ounce
    (pack of 1)' and seems good to buy.]
Observation: OK.

Action: click[bright citrus]
Observation: You have clicked bright citrus.

Action: click[3 ounce (pack of 1)]
Observation: You have clicked 3 ounce (pack of 1).

Action: click[Buy Now]
```

Prompt 6: **WebShop - Step prompt** The prompt used to generate the next action so that after performing the action a new state would be achieved. The observation and action history of the agent would be appended at the end of this prompt. Taken from [52].

```
Given an item to purchase and a trajectory that aims to buy an item
    that exactly matches the specification, which corresponds to the
    ideal score of 1.0, analyze the following trajectory, then at the
    last line conclude "Thus the correctness score is {s}", where s is
     an integer from 1 to 10.

Here are some examples

Webshop
Instruction:
i am looking for dairy free and apple variety pack of chips, and price
     lower than 30.00 dollars
[Search]

Action: search[dairy free and apple variety pack of chips]
Observation:
[Back to Search]
Page 1 (Total results: 50)
[Next >]
[B07HRFJWP8]
Enjoy Life Foods Soft Baked Ovals, Breakfast Bars, Nut Free Bars, Soy
    Free, Dairy Free, Non GMO, Gluten Free, Vegan, Variety Pack, 4
    Boxes (20 Bars Total)
$100.0
[B01KMHY5PG]
Enjoy Life Soft Baked Chewy Bars, Variety Pack, Nut Free Bars, Soy
    Free, Dairy Free, Gluten Free, 6 Boxes (30 Total Bars)
$21.49
[B008D2X8C4]
Enjoy Life Lentil Chips Variety Pack, Dairy Free Chips, Soy Free, Nut
    Free, Non GMO, Vegan, Gluten Free, 24 Bags (0.8 oz)
$100.0

Action: think[The first two items are not what I want. The third item
    is the dairy free and apple variety pack of chips I want.]
Observation: OK.

Action: click[B008D2X8C4]
```

```
Observation:
[Back to Search]
[< Prev]
flavor name [dill and sour cream][garlic & parmesan][light sea salt][
   margherita pizza][thai chili lime][variety pack]
size [0.8 ounce (pack of 24)][4 ounce (pack of 12)]
Price: $100.0
Rating: N.A.
[Description]
[Features]
[Reviews]
[Buy Now]

Action: think[The item has options 'variety pack' and '0.8 ounce (pack
    of 24)' and seems good to buy.]
Observation: OK.

Action: click[variety pack]
Observation: You have clicked variety pack.

Action: click[0.8 ounce (pack of 24)]
Observation: You have clicked 0.8 ounce (pack of 24).

Action: click[Buy Now]

Reflection: In this attempt, I was unsuccessful. I accidentally bought
    a product that was $100, which is more than my budget of $30.
   Either way, the initial search results were not good. Next time, I
    will do search["variety pack of chips"] and then check if the
   results meet the dairy free and the $30 budget constraints. I will
    continue to refine my searches so that I can find more products.
Thus the correctness score is 5

Webshop
Instruction:
i would like a 3 ounce bottle of bright citrus deodorant for sensitive
    skin, and price lower than 50.00 dollars
[Search]

Action: search[3 ounce bright citrus deodorant sensitive skin]
Observation:
[Back to Search]
Page 1 (Total results: 50)
[Next >]
[B078GWRC1J]
Bright Citrus Deodorant by Earth Mama | Natural and Safe for Sensitive
    Skin, Pregnancy and Breastfeeding, Contains Organic Calendula 3-
   Ounce
$10.99
[B078GTKVXY]
Ginger Fresh Deodorant by Earth Mama | Natural and Safe for Sensitive
   Skin, Pregnancy and Breastfeeding, Contains Organic Calendula 3-
   Ounce
$10.99
[B08KBVJ4XN]
Barrel and Oak - Aluminum-Free Deodorant, Deodorant for Men, Essential
    Oil-Based Scent, 24-Hour Odor Protection, Cedar & Patchouli Blend
   , Gentle on Sensitive Skin (Mountain Sage, 2.7 oz, 2-Pack)
$15.95

Action: think[B078GWRC1J and B078GTKVXY are bright citrus deodorant
   less then 50 dollars. I can check B078GWRC1J first.]
Observation: OK.
```

```
Reflection: Although the task is not yet complete, the first search is
     correct and returns relevant results. The thought is also logical
     and has potential to lead to the correct selection.
Thus the correctness score is 9
{input}
```

Prompt 7: **WebShop - Value prompt** The prompt used to evaluate a state. The observation and action history of the agent would be appended at the end of this prompt [54].

```
Given a science problem, you need to answer the problem based on your
    existing knowledge. The input may include some existing steps to
    solve the question and you should continue to complete the
    solution based on these existing steps.

If the input does not provide any existing steps, you need give the
    first step in solving or calculating the problem. If partial
    solution steps are provided, you need to output the next step
    along the lines of the existing steps.
The output format is limited to: "Next step: ..." where ... indicates
    omitted output information, which is the next step in the answer
    that you should give. Your output must be a complete step, which
    may include detailed calculations, reasoning, choosing answers,
    etc. but no reasoning.

If the existing steps are already sufficient, you can output "The
    final answer is: $...$" where ... indicates the final answer to
    the question.

Please provide MULTIPLE alternative next steps. Use the following
    format:
"Next step: $...$
Next step: $...$
Next step: $...$".

Below is the input, please follow the specified format for your output
    .

Problem: {problem}
Existing steps:
{existing_steps}
Output:
```

Prompt 8: **SciBench - Step prompt** The prompt used to generate candidate new states. Taken from [53]

```
Given a math problem and its corresponding solution, your task is to
    extract the final answer obtained in the solution.
You should summarize the answer using the format: "The final answer is
     $...$". Replace "..." with the answer obtained in the solution.
Problem: {problem}
Solution: {existing_steps}
Extracted answer:
```

Prompt 9: **SciBench - Summary prompt** This prompt is applied after a solution is found to adjust the output into the expected format. Taken from [53]

```
Your task is to assess whether the provided solution steps can
    successfully solve the given science/mathematics problem and
    output a score.
The score should be a decimal between 0 and 1. If all the provided
    steps are incorrect (every step is wrong), the score should be 0.
    If all steps are correct and the final answer is successfully
    calculated, the score should be 1. The more errors there are in
    the steps, the closer the score should be to 0. The closer the
    steps are to the final correct answer, the closer the score should
     be to 1.
```

```
Steps that only contain verbal descriptions without any mathematical
    expressions should generally receive a low score. A score equal to
     or greater than 0.9 can only be given if the answer has already
    been calculated to a specific numerical value. If the thought
    process is complete but the answer is not computed, or only the
    mathematical expression is written without solving it, the score
    must be below 0.9.

First provide an analysis, then the score. Your analysis and scoring
    should be entirely based on the given steps. Do not continue
    solving the problem. Please study the following examples.

{examples}

Below is a problem and the existing steps, with analysis and scoring.
    Be careful not to output the next steps in the analysis, and the
    scoring should be based entirely on the steps given in the input.
The output format is limited to: "Analysis:...\nScore:...", where ...
    indicates omitted output content, which is the part you need to
    fill in.

Input:
Problem: {problem}
Existing steps:
{existing_steps}
Output:
```

Prompt 10: **SciBench - Value prompt** This prompt used to evaluate a state. The original prompt was taken from [53] but was translated from Chinese to English using Google Translate.

Practical extensions in the FoA framework.

- **Caching:** The caching mechanism is utilized during the evaluation phase of our method to enhance its efficiency. It operates by ensuring that a given state is evaluated only once by the language model. This is achieved through a temporary state-to-value map maintained for the duration of a single run of the algorithm. Consequently, only when an agent encounters a previously unseen state, the LLM evaluates it and stores it in the cache. However, if a different agent (or the same agent at a later step) revisits that state, the LLM does not re-evaluate it; instead, the value is retrieved from the state-to-value cache. In comparison to other baselines such as ToT [51] and LATS [54], no additional caching is being performed.
- **Batching:** During the mutation or selection phase (depending on the task) prompts are callected by all agents. Once that happens, if duplicated prompts occur, instead of making several individual requests for the same prompt, we make a single request and ask for multiple outputs. Employing batching in this way, ensures competitive fairness to methods such as ToT which utilize the this mechanic in the same way. This approach enhances efficiency and resource management by reducing network latency, server requests, on top of lowering the costs, as the user pays for the input tokens only once. Additionally, batching ensures consistency, as slight changes in the model's state or data processing on the provider's side, can affect individual requests differently.

Task-specific modifications.

- For the Scibench task [42], we used the prompts provided in the ReST-MCTS* paper [53]. However, from our understanding, the evaluation prompt used in that paper was written in Chinese, and no official English version was available. To ensure better control over the evaluation process and to align the task with the predominantly English-language setup of our experiments, we translated the original Chinese prompt into English. This allowed us to directly inspect, adjust, and verify the evaluation inputs, ensuring consistency and clarity across all baselines.

C.5 Hyperparameter Tuning

We perform a hyperparameter grid-search on the validation set, to explore trade-offs between success rate and cost. For this search, we use GPT-3.5-turbo, since GPT-4 would be prohibitively expensive. The hyperparameters we consider are the number of agents, the total number of steps each agent is allowed to perform, the discount factor γ, the resampling frequency k and the resampling method.

The grid search is implemented in two steps. Initially, a broader, more general grid search is conducted to obtain an approximate understanding of where the optimal configurations are located. Subsequently, a more precise grid search is performed based on the findings from the initial step. The results of the second grid search for Game of 24 are presented in Figure 7, for Mini Crosswords in Figure 8 and for WebShop in Figure 9.

C.5.1 Game of 24

The strategy for selecting the optimal configuration for the Game of 24 involves choosing the configuration that yields the best performance at the lowest cost. Following this approach, it is evident that the optimal number of agents and steps is achieved when the either hyperparameter is set to 9 or 12. However, since the cost is lowest at 9, this value is chosen for both the number of agents and the number of steps. Regarding the resampling frequency, resampling after every step (i.e., $k = 1$) results in significantly better performance and is therefore selected. For the discount factor γ, no notable differences in performance or cost are observed. Thus, $\gamma = 0.5$ is chosen as it represents a balanced choice between not allowing backtracking ($\gamma = 0$) and maximally encouraging backtracking by rendering the discount factor inconsequential ($\gamma = 1$). Finally, the linear filtered resampling method provides similar results to the linear method but at a significantly lower cost, making it the preferred choice.

The linear filtered resampling method is essentially the same as linear resampling, but it only considers states whose values are equal to or greater than the value of the current best-evaluated state. Across the different tasks, we found that this method is advantageous only when multiple states with sparse values are taken into consideration during resampling.

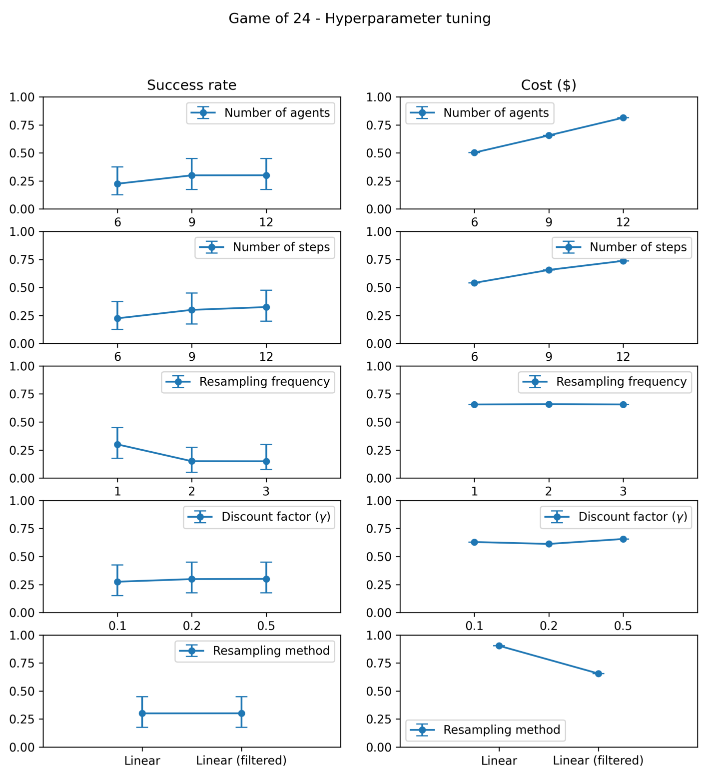

Figure 7: **Results of the final grid-search for Game of 24.** Each subplot illustrates the performance (left) and cost (right) as a function of a varying hyperparameter. The values of the remaining hyperparameters are set to those of the final, optimal configuration.

C.5.2 Mini Crosswords

For the Mini Crosswords task, in our final grids-search we observed that performance plateaued at approximately 0.4 overlap percentage, with minimal variation. Consequently, our primary strategy for this task was to minimize cost. The overlap remained similar when tuning the number of agents, number of steps, and the resampling frequency. However, in each case, there was always a specific value that significantly minimized cost, and this value was selected. Thus, we opted for 2 agents, running for 6 steps each, and resampling every $k = 3$ steps. For the discount factor and the resampling method, there was no clear advantage in terms of performance or cost. Therefore, we selected the most moderate options in each category: a discount factor of $\gamma = 0.5$ and the linear resampling method.

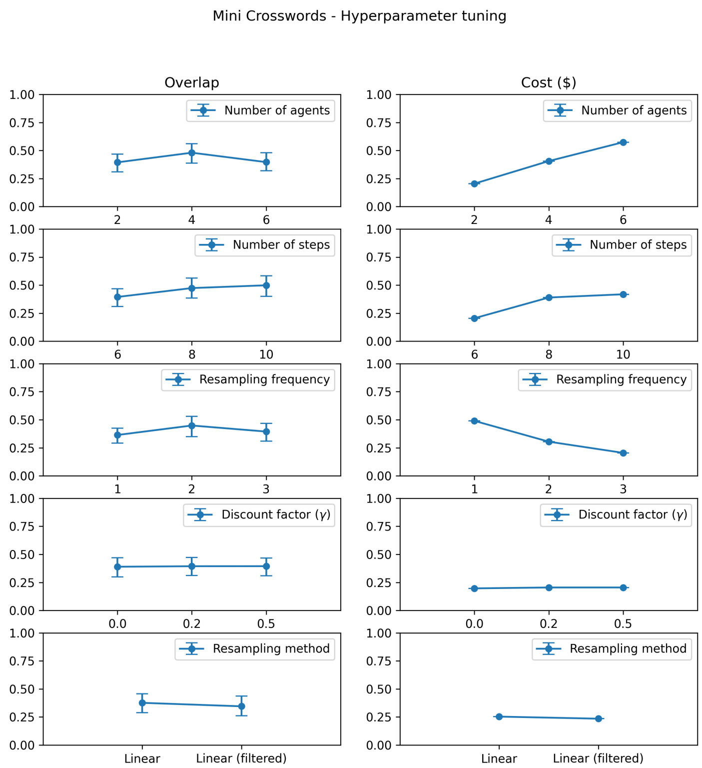

Figure 8: **Results of the final grid-search for Mini Crosswords.** Each subplot illustrates the performance (left) and cost (right) as a function of a varying hyperparameter. The values of the remaining hyperparameters are set to those of the final, optimal configuration.

C.5.3 WebShop

Finally, for the WebShop task, we reverted to our original strategy: achieving the best performance at the lowest possible cost. It is noteworthy that, due to the complexity of the WebShop task, more agents and steps were required for optimal performance. Consequently, we tested a broader range of values for both resampling frequency and discount factor compared to the previous tasks. The results indicated that the best average scores at the lowest cost were achieved with 15 agents running for 10 steps each. For the resampling frequency, the best scores were obtained with $\gamma \in \{2, 4, 5\}$, with 4 and 5 being the most cost-effective. Since there was no significant cost difference between 4 and 5, we selected 4 to allow for more frequent resampling. Finally, we omitted filtering during resampling as it provided no additional advantage.

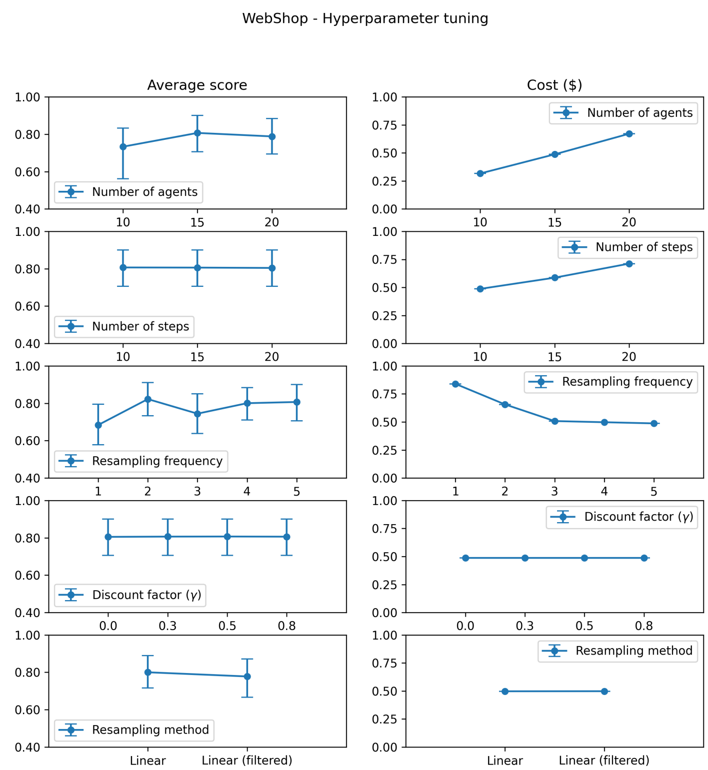

Figure 9: **Results of the final grid-search for WebShop.** Each subplot illustrates the performance (left) and cost (right) as a function of a varying hyperparameter. The values of the remaining hyperparameters are set to those of the final, optimal configuration.

C.6 Additional Results

C.6.1 Generalizability of the findings with other base models.

In the following section, we present our results for various models and demonstrate that the findings generalize across different settings. You can find the results for the task of Game of 24 in Table 5, Mini Crosswords in Table 6 and WebShop in Table 7.

Note on the performance of Act and ReAct. Act and ReAct have essentially the same architecture in the sense that an initial prompt is repeatedly being given to the LLM while it's being updated by the actions that have been chosen and the resulting environment observations. The only difference is that the ReAct prompt introduces the possibility of a new action for the LLM : "Think". When this action is chosen the LLM does not interact with the environment, it simply states its thoughts and aims to come up with a strategy to solve the problem. However, these prompts came up years ago where much less powerful models were used. As a result, more contemporary models interact differently with them.

C.6.2 Ablation analysis for the Crosswords and WebShop tasks

In the following section, we present the remaining ablation studies we performed and display that our findings generalize across different settings. You can find the results for the Game of 24 ablation in Figure 10, the Mini Crosswords in Figure 11 and for WebShop in Figure 12.

Table 5: Comparing FoA with previous methods using *success rate* (↑ better) and *cost* (↓ better) on the Game of 24 task. Owing to its exorbitant cost ($\simeq$ 600 US$), we could not run RAFA (shown as DNR).

Task : Game of 24			
model	method	success rate	cost
GPT-3.5	IO	0.068	0.048
	CoT	0.036	0.122
	CoT-SC	0.052	1.404
	AoT	0.010	0.322
	ToT	0.136	1.711
	GoT	0.112	1.603
	RAFA	0.080	10.47
	FoA	**0.251**	1.547
GPT-4	IO	0.060	0.652
	CoT	0.060	6.98
	CoT-SC	0.10	49.395
	AoT	0.490	20.984
	ToT	0.740	75.02
	GoT	0.630	70
	RAFA	DNR	DNR
	FoA	**0.760**	62.93
Llama 3.2-11B	IO	0.026	0.004
	CoT	0.036	0.008
	CoT-SC	0.048	0.049
	AoT	0.051	0.121
	ToT	0.027	0.37
	GoT	0.014	0.305
	RAFA	0.000	23.102
	FoA	**0.060**	0.32
Llama 3.2-90B	IO	0.060	0.027
	CoT	0.068	0.054
	CoT-SC	0.080	0.334
	AoT	0.368	0.813
	ToT	0.355	2.51
	GoT	0.301	2.11
	RAFA	DNR	DNR
	FoA	**0.397**	2.05

C.7 Details on RAFA Results and Metric Differences

In our evaluation of RAFA [23] for the Game of 24 task, we used the official implementation provided by the authors. However, the results we report differ from those presented in the original RAFA paper. This difference stems from a variation in the definition of the success rate evaluation metric.

Specifically, the RAFA paper reports results using a relaxed version of the success rate metric (see Footnote 1, page 50, ICML'24 camera-ready), which differs from the stricter formulation used in other benchmarks. To ensure consistency and fairness across all methods evaluated in our study, we applied the success rate implementation as defined in the original ToT [51] paper uniformly across all baselines.

As shown in Table 8, when we apply the relaxed success rate metric used in the original RAFA paper, our results align closely with those reported by its authors. This adjustment ensures that readers can understand the basis of any apparent discrepancies and interpret our comparisons across methods accurately.

Table 6: Comparing FoA with previous methods using *overlap* (↑ better) and *cost* (↓ better) on the Crosswords task .

Task : Mini Crosswords			
model	method	overlap	cost
GPT-3.5	IO	0.312	0.008
	CoT	0.331	0.019
	CoT-SC	0.331	0.063
	ToT	0.333	0.479
	GoT	0.345	0.398
	FoA	**0.362**	0.246
GPT-4	IO	0.368	0.511
	CoT	0.394	1.064
	CoT-SC	0.394	2.822
	ToT	0.397	48.988
	GoT	0.412	30.281
	FoA	**0.460**	12.938
Llama 3.2-11B	IO	0.062	0.006
	CoT	0.210	0.008
	CoT-SC	0.210	0.037
	ToT	0.465	0.440
	GoT	0.415	0.567
	FoA	**0.509**	0.160
Llama 3.2-90B	IO	0.050	0.038
	CoT	0.306	0.044
	CoT-SC	0.306	0.129
	ToT	0.628	6.010
	GoT	0.625	4.715
	FoA	**0.649**	1.550

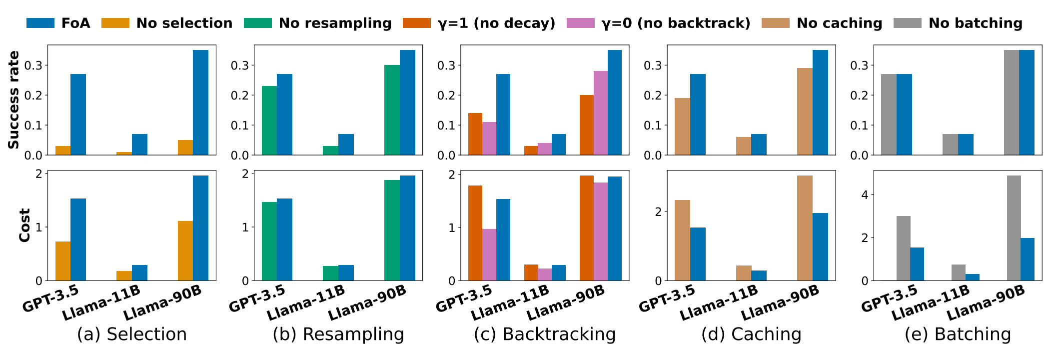

Figure 10: Ablation analysis to study the impact of (a) Selection phase, (b) Resampling, (c) Backtracking, (d) Caching, and (e) Batching on the performance of FoA using the Game of 24 task with GPT-3.5, Llama3.2-11B, and 90B as base models.

Table 7: Comparing FoA with previous methods using *average score* (↑ better) and *cost* (↓ better) on the WebShop task.

Task : WebShop			
model	method	average score	cost
GPT-3.5	Act	0.581	0.095
	ReAct	0.487	0.17
	Reflexion	0.563	0.652
	LASER	0.572	0.405
	LATS	0.661	232.27
	FoA	**0.756**	1.68
Llama 3.2-11B	Act	0.282	0.096
	ReAct	0.167	0.116
	Reflexion	0.248	0.493
	LASER	0.54	0.75
	LATS	-	-
	FoA	**0.772**	2.6
DL	IL [50]	0.599	-
	IL+RL [50]	0.624	-
	WebN-T5 [12]	0.610	-
	WebGUM [10]	**0.675**	-
Human experts [50]		0.821	-

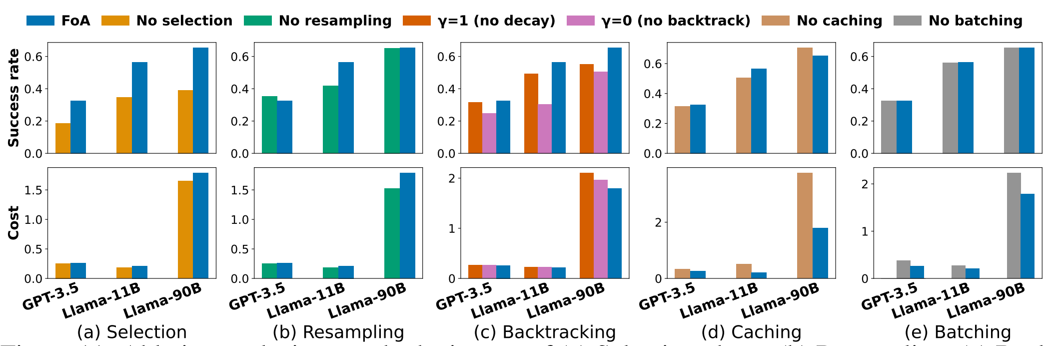

Figure 11: Ablation analysis to study the impact of (a) Selection phase, (b) Resampling, (c) Backtracking, (d) Caching, and (e) Batching on the performance of FoA using the Mini Crosswords task with GPT-3.5, Llama3.2-11B, and 90B as base models.

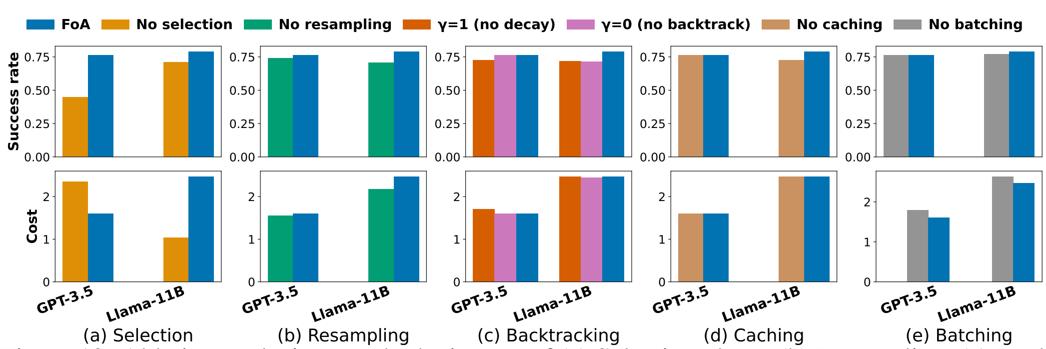

Figure 12: Ablation analysis to study the impact of (a) Selection phase, (b) Resampling, (c) Backtracking, (d) Caching, and (e) Batching on the performance of FoA using the WebShop task with GPT-3.5 and Llama3.2-11B base models.

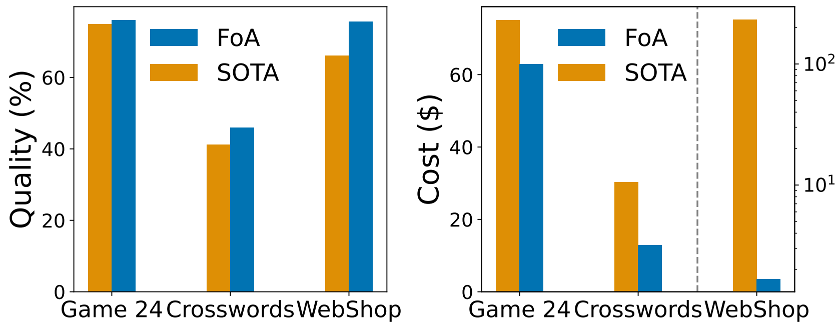

Figure 13: Comparing (Left) quality and (Right) cost of FoA with the second most efficacious method (labeled SOTA) on each benchmark task.

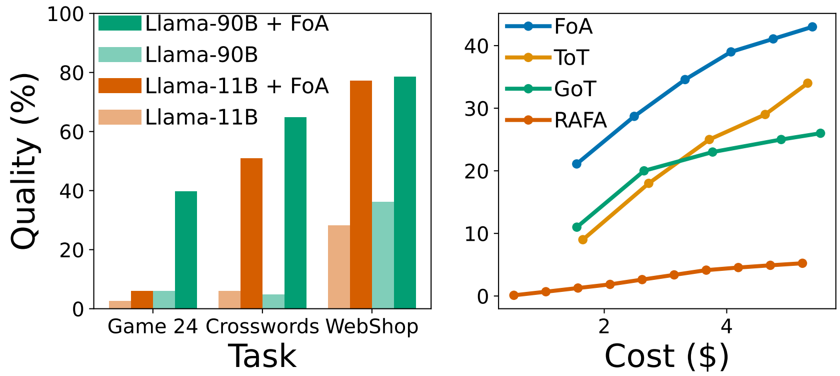

Figure 14: Evaluating the trade-off between (Left) model size and quality on benchmarked tasks with Llama3.2-11B and 90B, and (Right) cost and quality of SOTA methods with GPT-3.5 on Game of 24.

RAFA results (Game-of-24)	Accuracy (%)	Low interval	High interval
Our run with RAFA metric	26	23	28
Our run with ToT metric	8	6	9
RAFA run with RAFA metric	29	–	–

Table 8: Comparison of RAFA results across different runs and evaluation metrics.