

IMPROVING PLANNING WITH LARGE LANGUAGE MODELS: A MODULAR AGENTIC ARCHITECTURE

Anonymous authors

Paper under double-blind review

ABSTRACT

Large language models (LLMs) demonstrate impressive performance on a wide variety of tasks, but they often struggle with tasks that require multi-step reasoning or goal-directed planning. Both cognitive neuroscience and reinforcement learning (RL) have proposed a number of interacting functional components that together implement search and evaluation in multi-step decision making. These components include conflict monitoring, state prediction, state evaluation, task decomposition, and orchestration. To improve planning with LLMs, we propose an agentic architecture, the Modular Agentic Planner (MAP), in which planning is accomplished via the recurrent interaction of the specialized modules mentioned above, each implemented using an LLM. MAP improves planning through the interaction of specialized modules that break down a larger problem into multiple brief automated calls to the LLM. We evaluate MAP on three challenging planning tasks – graph traversal, Tower of Hanoi, and the PlanBench benchmark – as well as an NLP task requiring multi-step reasoning (strategyQA). We find that MAP yields significant improvements over both standard LLM methods (zero-shot prompting, in-context learning) and competitive baselines (chain-of-thought, multi-agent debate, and tree-of-thought), can be effectively combined with smaller and more cost-efficient LLMs (Llama3-70B), and displays superior transfer across tasks. These results suggest the benefit of a modular and multi-agent approach to planning with LLMs.

1 INTRODUCTION

Large Language Models (LLMs) (Devlin et al., 2019; Brown et al., 2020) have become widely accepted as highly capable generalist systems with a surprising range of emergent capacities (Srivastava et al., 2022; Wei et al., 2022a; Webb et al., 2023). They have also sparked broad controversy, with some suggesting that they are approaching general intelligence (Bubeck et al., 2023), and others noting a number of significant deficiencies (Mahowald et al., 2023). A particularly notable shortcoming is their poor ability to plan or perform faithful multi-step reasoning (Valmeekam et al., 2023; Dziri et al., 2023). Recent work (Momennejad et al., 2023) has evaluated the extent to which LLMs might possess an emergent capacity for planning and exploiting *cognitive maps*, the relational structures that humans and other animals utilize to perform planning (Tolman, 1948; Tavares et al., 2015; Behrens et al., 2018). This work found that LLMs displayed systematic shortcomings in planning tasks that suggested an inability to reason about cognitive maps. Common failure modes included a tendency to ‘hallucinate’ (e.g., to use non-existent transitions and paths), and to fall into loops. This work raises the question of how LLMs might be improved so as to enable a capacity for planning, especially given the ubiquity of sequential decision making, reasoning, and planning problems across the wide application of generative AI and LLMs.

Here, we take a step toward improving planning with LLMs, by taking inspiration from both cognitive neuroscience and formal theories of decision-making and planning. In traditional theories of planning, such as those found in the field of reinforcement learning (RL) (Sutton & Barto, 2018), planning is carried out via the interaction of several specialized functions or modules, rather than through the activity of a single, monolithic system. For instance, many approaches involve distinct functions for action proposal, state evaluation, subgoal identification, or state prediction, many of which have also been related to the function of specific brain regions (see Section 6 for discussion). An interesting observation is that LLMs are often able to carry out these functions when probed

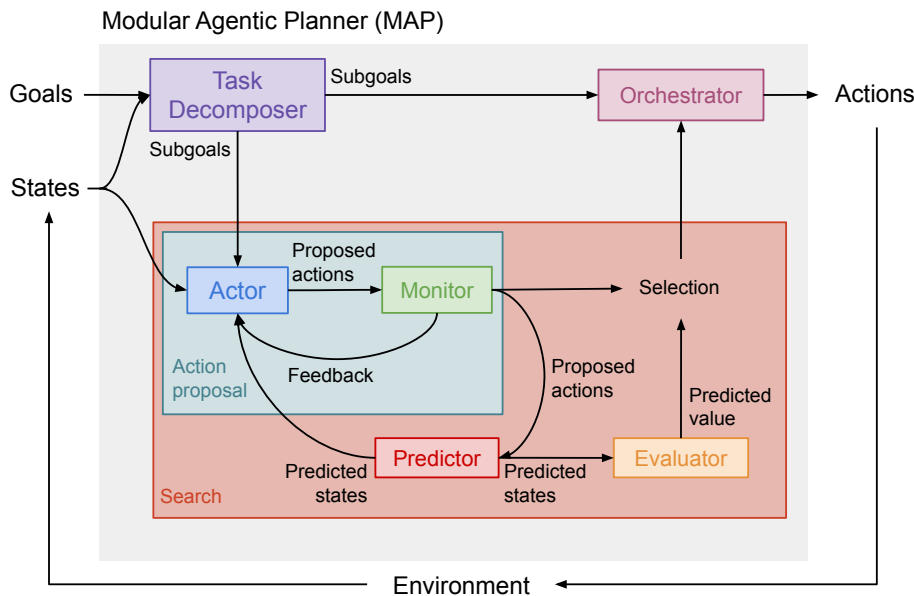


Figure 1: **Modular Agentic Planner (MAP)**. The agent receives states from the environment and high-level goals. These are processed by a set of specialized LLM modules. The TaskDecomposer receives high-level goals and generates a series of subgoals. The Actor generates proposed actions given a state and a subgoal. The Monitor gates these proposed actions based on whether they violate certain constraints (e.g., task rules) and provides feedback to the Actor. The Predictor predicts the next state given the current state and a proposed action. The Evaluator is used to estimate the value of a predicted state. The Predictor and Evaluator are used together to perform tree search. The Orchestrator determines when each subgoal has been achieved, and when the final goal has been achieved, at which point the plan is emitted to the environment as a series of actions.

in isolation, but are unable to reliably integrate and orchestrate these capacities in the service of a goal. For instance, Momennejad et al. (2023) noted that LLMs often attempt to traverse invalid or hallucinated paths in planning problems (e.g., to move between rooms that are not connected), even though they can correctly identify these paths as invalid when probed separately. This suggests the possibility of an agentic approach using LLMs, in which planning is carried out through the coordinated and recurrent interaction of multiple LLM modules, each of which is specialized to perform a distinct process.

With this goal in mind, we propose the Modular Agentic Planner (MAP) (Figure 1), an agentic architecture composed of modules that are specialized to perform specific functions within the planning process. Specifically, we have identified and implemented the following key modules: error monitoring, action proposal, state prediction, state evaluation, task decomposition, and task coordination. Action proposal, state prediction, and state evaluation are further combined to perform tree search. All modules are implemented using an LLM, which receives instructions describing the module’s role via prompting and few-shot in-context learning (ICL). The resulting MAP algorithm (Algorithm 2) is implemented via the recurrent interaction of these modules, combining the strengths of classical planning and search algorithms with the use of LLMs as general-purpose world models and planning functions.

We evaluate MAP on four challenging decision-making tasks that require planning and multi-step reasoning. First, we performed controlled experiments on a set of graph traversal tasks according to the CogEval protocol (Momennejad et al., 2023). These tasks require goal-directed navigation in novel environments (MDPs) described in natural language, of which we selected an environment that was most challenging for LLMs, including GPT-4. Second, we investigate Tower of Hanoi (ToH), a classic problem solving task that requires multi-step planning (Simon, 1975). Third, we investigate the two most challenging tasks in the PlanBench benchmark: mystery BlocksWorld and

Logistics (Valmeekam et al., 2023). Finally, we investigate a challenging NLP task that requires multi-step reasoning, StrategyQA (Geva et al., 2021). We find that, when implemented with GPT-4, MAP significantly improves performance on all four tasks (Figures 2 and 3, Tables 1 and 2), and that the approach can also be effectively implemented with a smaller and more cost-efficient LLM (Llama3-70B, Table 10). Transfer experiments further indicate that MAP displays an improved ability to generalize between tasks, and ablation experiments indicate that each of the individual modules plays an important role in the overall architecture’s performance (Figure 3). Taken together, these results indicate the potential of a modular agentic approach to improve the reasoning and planning capabilities of LLMs.

2 APPROACH

2.1 PROBLEM FORMULATION

We consider planning problems in environments that are both deterministic and fully observable. The environment and transition dependencies are described to the agent, along with a starting state x and a goal y , and the agent must generate a plan P (a series of actions) without directly interacting with the environment. Despite the relatively limited nature of the environments we consider (fully observable, deterministic), planning problems of this sort are nevertheless extremely challenging for LLMs (Valmeekam et al., 2023; Momennejad et al., 2023).

2.2 MODULES

MAP contains the following specialized modules, each constructed from a separate LLM instance through a combination of prompting and few-shot (≤ 3 examples) in-context learning (described in greater detail in section A.7):

- **TaskDecomposer.** The TaskDecomposer receives the current state x and a goal y and generates a set of subgoals Z that will allow the agent to gradually work toward its final goal. In the present work, the TaskDecomposer is only utilized to generate a single intermediate goal, though in future work we envision that it will be useful to generate a series of multiple subgoals.
- **Actor.** The Actor receives the current state x and a subgoal z and proposes B potential actions $A = a_{b=1} \dots a_{b=B}$. The Actor can also receive feedback ϵ from the Monitor about its proposed actions.
- **Monitor.** The Monitor gates the actions proposed by the Actor based on their validity (e.g., whether they violate the rules of a task). It emits an assessment of validity σ , and also feedback ϵ in the event the action is deemed invalid.
- **Predictor.** The Predictor receives the current state x and a proposed action a and predicts the resulting next state \tilde{x} .
- **Evaluator.** The Evaluator receives a next-state prediction \tilde{x} and produces an estimate of its value v in the context of goal y . This is accomplished by prompting the Evaluator (and demonstrating via a few in-context examples) to estimate the minimum number of steps required to reach the goal (or subgoal) from the current state.
- **Orchestrator.** The Orchestrator receives the current state x and a subgoal z and emits an assessment Ω of whether the subgoal has been achieved. When the Orchestrator determines that all subgoals (including the final goal) have been achieved, the plan is emitted to the environment as a series of actions.

2.3 ACTION PROPOSAL LOOP

This section describes MAP’s algorithms, the first of which is the action proposal loop. The Actor and Monitor interact via the ProposeAction function (Supplementary Algorithm 1). The Actor proposes a set of potential actions, which are then gated by the Monitor. If the Monitor determines that the actions are invalid (e.g., they violate the rules of a task), feedback is provided to the Actor, which then proposes an alternative action. The output of the ProposeAction function is a set of

potential actions, one of which will be selected as the action at the next time step (as described in the following section).

Algorithm 1: Action proposal loop. ProposeAction takes a state x and a goal y and generates B potential actions $A = a_{b=1} \dots a_{b=B}$. This is implemented via a loop, in which the Actor first proposes potential actions, and the Monitor then assesses those actions according to certain constraints (e.g., task rules), providing feedback if any of the actions are deemed to be invalid. This continues until the proposed actions are considered valid. See Sections A.7.2 and A.7.3 for more details.

```

170 Function ProposeAction ( $x, y, B$ ):
171    $\sigma \leftarrow \text{false}$  // Initialize validity
172    $E \leftarrow \{\}$  // Initialize feedback
173   while  $\sigma$  is false do
174      $A \leftarrow \text{Actor}(x, y, E, B)$  // Sample B actions
175      $\sigma, \epsilon \leftarrow \text{Monitor}(x, A)$  // Determine validity and provide feedback
176      $E \leftarrow E \cup \{\epsilon\}$  // Accumulate feedback
177   end
178   return  $A$ 

```

2.4 TREE SEARCH

ProposeAction is further embedded in a Search loop (Supplementary Algorithm 3). The actions emitted by ProposeAction are passed to the Predictor, which predicts the states that will result from these actions. A limited tree search is then performed, starting from the current state, and then exploring B branches recursively to a depth of L layers. Values are assigned to the terminal states of this search by the Evaluator, and the action leading to the most valuable predicted state is selected.

Algorithm 2: Modular Agentic Planner (MAP). MAP takes a state x and a goal y and generates a plan P , a series of actions with a maximum length of T . The TaskDecomposer first generates a set of subgoals Z . The agent then pursues each individual subgoal z in sequence, followed by the final goal y . At each time step, Search (Algorithm 3) is called to generate an action and a predicted next-state. Actions are added to the plan until the Orchestrator determines that the goal has been achieved, or the plan reaches the maximum length T .

```

193 Function MAP ( $x, y, T, L, B$ ):
194    $P \leftarrow []$  // Initialize plan
195    $Z \leftarrow \text{TaskDecomposer}(x, y)$  // Generate subgoals
196   for  $g$  in  $1 \dots \text{length}(Z) + 1$  do
197     if  $g \leq \text{length}(Z)$  then
198        $z \leftarrow Z_g$  // Update current subgoal
199     else
200        $z \leftarrow y$  // Final goal
201     end
202      $\Omega \leftarrow \text{Orchestrator}(x, z)$  // Initialize subgoal assessment
203     while  $\Omega$  is false and  $\text{length}(P) < T$  do
204        $a, x, v \leftarrow \text{Search}(l = 1, L, B, x, z)$  // Perform search
205        $P \leftarrow P.\text{append}(a)$  // Update plan
206        $\Omega \leftarrow \text{Orchestrator}(x, z)$  // Determine if subgoal is achieved
207     end
208   end
209   return  $P$ 

```

2.5 PLAN GENERATION

Algorithm 2 describes the complete MAP algorithm. To generate a plan, the TaskDecomposer component of MAP first generates a set of subgoals based on the final goal and current state. These subgoals guide the search and are internally pursued one at a time, utilizing the Search loop to generate actions until the Orchestrator determines that the subgoal has been achieved. The actions are accumulated in a plan buffer P until either the Orchestrator determines that the final goal has been reached, or the maximum allowable number of actions T are accumulated.

3 EXPERIMENTS

Experiment details are described in Section A.3. Code is available at: <https://github.com/MAPLLM/MAPICLR2025sub>.

3.1 TASKS

Graph Traversal. We performed experiments on four multi-step planning tasks based on graph traversal using the CogEval protocol (Momennejad et al., 2023). Natural language descriptions of a graph are provided with each node assigned to a room (e.g., ‘room 4 is connected to room 7’). The tasks included Valuepath, involving finding the shortest path between a given room and the largest of two possible rewards (but without going through the room with the smaller reward); Steppath, involving finding the shortest path between two rooms; Detour, in which the Valuepath task is first described, after which an edge is subsequently removed from the graph; and Reward Revaluation, in which the Valuepath task is first described, and the value associated with two reward locations is subsequently changed. Please see Section A.4 in the Appendix for more details.

Tower of Hanoi. We also investigated a classic multi-step planning task called the Tower of Hanoi (ToH) (Figure 5). In the original task, there are three pegs and a set of disks of different sizes. The disks must be moved into a particular goal configuration, while observing a set of constraints that prevent simple solutions. In our experiments, we designed an alternative (but isomorphic) formulation of this task in which the inputs are text-based rather than visual. This text-based formulation made it possible to evaluate language models on the task, but it also resulted in a task that does not share any surface features with the original task, making it unlikely that GPT-4 could rely on exposure to descriptions of ToH in its training data to solve the problem. Please see Section A.4 in the Appendix for more details.

PlanBench. To assess the generality and robustness of our approach, we also investigated a more extensive planning benchmark, PlanBench, consisting of synthetically generated problems in a number of distinct domains. We specifically investigated the Logistics domain, involving the transportation of goods between cities using airplanes and trucks, and the Mystery Blocksworld (deceptive) domain, which involves arbitrary names for entities and actions, and is the most challenging domain in the dataset (more details can be found in Valmeekam et al. (2023)).

StrategyQA. Finally, to test the extent to which MAP can be applied to more real-world tasks, we investigated StrategyQA, an NLP task that requires multi-step reasoning, and has proven challenging for standard LLM methods (Geva et al., 2021). In this task, an unusual question is posed (e.g. ‘Did Aristotle own a laptop?’) that requires multi-step reasoning. The question must be decomposed into sub-questions which must be successively solved in order to arrive at a final answer (more details can be found in Geva et al. (2021))

3.2 BASELINES

We compared our model to several baseline methods. The first method involved asking GPT-4 (zero-shot) to provide the solution step by step. For the second method, in-context learning (ICL), we provided GPT-4 with a few in-context examples of a complete solution. We provided two examples for ToH, Valuepath, Detour, and Reward Reval, and three examples for Steppath (one each for 2, 3, and 4 steps) and PlanBench. The third method was chain-of-thought (CoT) (Wei et al., 2022b). For this method, the in-context examples were annotated with a series of intermediate computations that break down the planning process into multiple steps (see Sections A.7.7-A.7.9 for example baseline prompts). The fourth method was multi-agent debate (MAD), using the codebase from Du et al. (2023). In this approach, similar to MAP, a solution is generated through the interaction between multiple LLM instances (each instance was equivalent to the GPT-4 ICL baseline); however, unlike MAP, these instances are not specialized to perform specific functions. Finally, we investigated tree-of-thought (ToT), using the original codebase from Yao et al. (2023). Similar to MAP, ToT uses multiple LLM modules to perform tree search, although MAP incorporates additional modules and control processes (see Section A.2). To ensure that ToT was given the best chance of performing well on our tasks, we tested two versions, one with prompts that were similar to those in the original implementation (shown in the main results Section 4), and one that incorporated prompts from MAP (ToT-MAP, see Appendix Table 8). For ToT, multiple potential plans are generated for each

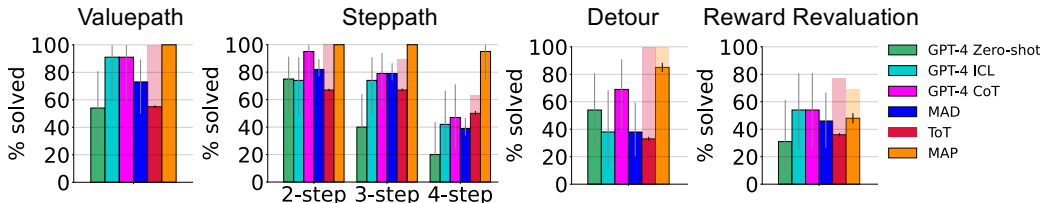
270 problem, and a method is required to select one of these plans (a problem-specific heuristic was used
 271 in the original work). To ensure a fair comparison, we evaluated using two metrics, 1) the best plan
 272 (according to a groundtruth evaluation) for each problem, 2) the average performance of all plans
 273 for each problem. We report both of these metrics for both ToT and MAP.

275 3.3 METRICS

276 Our primary metric is the percentage of problems that are solved for any given task. For non-
 277 deterministic methods (MAP, ToT), we consider both average percent solved (across multiple plans
 278 per problem) and percent solved for the best plan (according to a groundtruth evaluation). For
 279 Tower-of-Hanoi and Graph Traversal, we also consider the percentage of moves that are invalid
 280 (that violates the rules, or attempts to traverse a nonexistent edge).
 281

283 4 RESULTS

285 Figure 2 shows the results on the four graph traversal tasks (see Section A.5 for all results in tabu-
 286 lar form). On the Valuepath task, MAP solved 100% of problems, outperforming all baselines
 287 (outperforming the next-best baselines, GPT-4 CoT and ICL, by a margin of 9%). On the Steppath
 288 task, MAP displayed perfect performance for 2-step and 3-step paths, and near-perfect performance
 289 for 4-step paths, again outperforming all baselines (outperforming the next-best baseline, ToT, by a
 290 margin of 45%). MAP also outperformed all baselines on the Detour task (outperforming the next-
 291 best baseline, GPT-4 CoT, by a margin of 16%), and performed on par with the baselines on the
 292 Reward Revaluation task (while still outperforming GPT-4 zero-shot). This demonstrates that MAP
 293 can flexibly adjust to new circumstances when generating plans. Finally, the model did not propose
 294 any invalid actions in any of the four tasks (i.e., it did not hallucinate the presence of non-existent
 295 edges), due to the filtering of invalid actions by the Monitor (Figure 6).



304 Figure 2: **Graph traversal results.** ‘% solved’ indicates percentage of problems solved without
 305 proposing invalid actions (↑ better). GPT-4 Zero-shot, ICL, CoT, and MAD baselines are determin-
 306 istic, and therefore a single run was performed on all problems. Note that MAP did not employ tree
 307 search on the Steppath task, and did not employ task decomposition on any of the graph traversal
 308 tasks. Without tree search, MAP’s performance is deterministic, and therefore only a single run
 309 was performed on the Steppath task, whereas we performed 5 runs with ToT. Gray error bars reflect 95%
 310 binomial confidence intervals (for models evaluated on a single run). Dots reflect values of 0%. Dark
 311 bars indicate average performance over multiple plans/runs. Light bars indicate best performance.
 312 For Valuepath, Detour, and Reward Revaluation we performed 10, 10, and 5 runs respectively with
 313 MAP and ToT, and present average performance ± the standard error of the mean (black error bars).

314 Figure 3 shows the results on Tower of Hanoi (ToH). MAP demonstrated a significant improvement
 315 both in terms of the number of problems solved (left) and the number of invalid actions proposed
 316 (right). On 3-disk problems, MAP yielded a nearly seven-fold improvement in the number of prob-
 317 lems solved over zero-shot performance, and significantly outperformed standard in-context learning
 318 (ICL; by 28%), chain-of-thought (CoT; by 32%), multi-agent debate (MAD; by 49%), and tree-of-
 319 thought (ToT; by 68%). When considering the best plan (out of 5 runs) for each problem, MAP
 320 achieved a perfect score of 100%. MAP’s improved performance relative to MAD demonstrates the
 321 importance of interaction between *specialized* LLM instances (i.e., a modular approach), whereas
 322 MAD involves interactions between multiple LLM instances prompted to perform the same task.
 323 MAP’s superior performance relative to ToT demonstrates that tree search, though an important part
 of the approach, is not sufficient to explain MAP’s performance, and the other modules play an

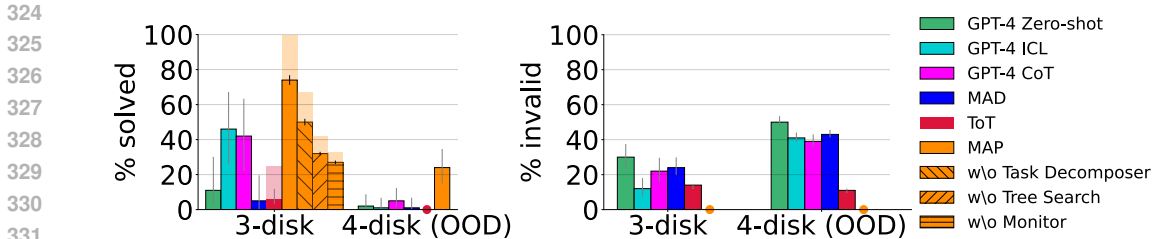


Figure 3: **Tower of Hanoi (ToH) results.** ‘% solved’ indicates percentage of problems solved without proposing invalid actions (\uparrow better). ‘% invalid’ indicates percentage of moves that are invalid (\downarrow better). Note that 4-disk problems are out-of-distribution (OOD). GPT-4 Zero-shot, ICL, CoT, and MAD baselines are deterministic and reflect a single run. Gray error bars reflect 95% binomial confidence intervals. Dots reflect values of 0%. Dark bars indicate average performance over multiple plans/runs. Light bars indicate best performance. MAP results for 3-disk problems reflect the average over 5 runs \pm the standard error of the mean (black error bars). MAP results for 4-disk problems reflect a single run, due to the high computational cost of multiple runs.

important role. For the problems that MAP solved, the average plan length (5.4) was close to the optimal number of moves (4.4). The model also demonstrated some ability to generalize out-of-distribution (OOD) to more complex 4-disk problems (not observed in any in-context examples), whereas the baseline models solved close to 0% of these problems. Notably, MAP did not propose any invalid actions, even on OOD 4-disk problems, whereas the baselines proposed a significant number of invalid actions.

We also performed several experiments with ToH to better understand the MAP algorithm. First, we investigated how MAP’s performance varied as a function of the depth of tree search. We found that a depth of $L = 2$ provided the best tradeoff between performance and cost (Table 14). A depth of $L = 1$ resulted in worse performance, while a depth of $L = 3$ incurred greater cost without significantly improving performance. Second, we investigated the extent to which MAP’s performance on this task depended on being provided with an explicit strategy for task decomposition (the goal recursion strategy). To address this, we provided the GPT-4 zero-shot, ICL, and CoT baselines with a description of this strategy. We found that this strategy did improve baseline performance, but MAP still outperformed these baselines even when they were provided with the strategy (Table 12). Third, we investigated whether the computational costs of the MAP algorithm could be mitigated by using a smaller LLM. We found that a version of MAP that used Llama3-70B still outperformed baselines that used the same LLM, and even outperformed the best GPT-4 baseline, GPT-4 ICL (Table 10).

Table 1 shows the results for the PlanBench dataset, where MAP outperformed all of the baselines that we considered. Notably, due to the complexity of the problems in this dataset, it was very costly to perform tree search, so we evaluated a minimal version of MAP that did not involve tree search. For this same reason, we were unable to evaluate a ToT baseline on the full set of problems, but we include a comparison with ToT on a subset of problems in Table 13. Even without tree search, MAP outperformed ToT. These results demonstrate that, although MAP certainly benefits from the use of tree search, it can still provide significant performance benefits in domains where this is not feasible. We also investigated a zero-shot version of MAP (without in-context examples) on the most challenging PlanBench domain (mystery blocksworld), and found that it outperformed both GPT-4 zero-shot and GPT-4 ICL (Table 11), suggesting that MAP can be useful even in settings where in-context examples are not available.

Table 2 shows the results for the StrategyQA benchmark, where MAP outperformed both CoT and ToT, and performed on par with human participants. This demonstrates the potential of MAP to be beneficial in more real-world tasks, such as question-answering tasks that require multi-step reasoning.

378
379
380
381
382
383
384
385
386
387
388Table 1: PlanBench results.¹

Model	Logistics	Mystery BW
GPT-4 Zero-shot	7	0.2
GPT-4 ICL	12	7.8
MAD	16.2	7.3
GPT-4 CoT	17	10.6
MAP	24	27.4

Table 2: StrategyQA results.²

Model	Accuracy
ToT	81.7 ³ ± 1.2
GPT-4 CoT	84.7 ± 0.3
MAP	87.7 ± 0.7
Human ⁴	87.0

389
390
391
392
393
394
395
396

Finally, we performed transfer experiments to study whether few-shot in-context learning would support generalization to different planning tasks. Table 3 shows the results for these experiments, including results for transfer from planning on a smaller graph to planning on a larger graph (n7tree → n15star), transfer to a semantically distinct but structurally isomorphic task (blocksworld (BW) → mystery blocksworld (mystery BW)), and transfer between completely different tasks (ToH → Mystery BW). We found that MAP outperformed both GPT-4 ICL and CoT in each of these settings, indicating that MAP can improve the generalizability and robustness of planning in LLMs.

397

Table 3: Transfer between different planning tasks. Results reflect % solved problems.

398
399
400
401
402
403
404

Model	n7tree → n15star Valuepath	BW → Mystery BW	ToH → Mystery BW
GPT-4 ICL	51	0.2	0
GPT-4 CoT	65	1.4	0
MAP	80	12.2	6.6

405
406

4.1 ABLATION STUDY

407
408
409
410
411
412
413
414
415
416
417
418
419
420
421

We also carried out an ablation study to determine the relative importance of each of MAP’s major components, focusing on the 3-disk ToH problems. Figure 3 (left) shows the results. We found that the Monitor was the most important component, as ablating this module resulted in significantly fewer solved problems, due primarily to an increased tendency to propose invalid moves (31% invalid moves vs. 0% for other ablation models). This highlights the importance of having a separate, modularized monitoring process. Ablating the tree search and TaskDecomposer module also resulted in significantly fewer solved problems. The impaired performance following the ablation of the tree search indicates the benefit of considering the multi-step implications of proposed actions, rather than committing to a single action as is done in standard autoregressive methods such as chain-of-thought. The impaired performance following the ablation of the TaskDecomposer highlights the benefit of decomposing a task into subgoals, which allows MAP to factorize a complex task into a set of smaller, more manageable tasks. Overall, these results suggest that all major components played an important role in MAP’s performance. Moreover, the improved performance was not due entirely to the use of tree search (which is shared with tree-of-thoughts (Yao et al., 2023)), but also depended on the incorporation of other modules such as the TaskDecomposer and especially the Monitor.

422

423
424

5 RELATED WORK

425
426

Early work in AI formalized planning as a problem of search through a combinatorial state space, typically utilizing various heuristic methods to make this search tractable (Newell & Simon, 1956;

427
428
429
430
431

¹Results reflect % solved problems for a single run.

²Results reflect accuracy on a fixed random subset of 100 questions averaged over 3 runs (± standard error).

³Yao et al. (2023) reported performance of 83%, but since the subset of 100 questions they used for evaluation is unknown, we ran ToT using the publicly released code on a fixed subset of 100 questions for fair comparison with MAP.

⁴Geva et al. (2021) reported human performance on a random subset of 100 questions.

432 Newell et al., 1959). Problems such as ToH figured prominently in this early research (Simon, 1975),
433 as it affords the opportunity to explore ideas based on hierarchical or recursive planning (in which
434 a larger problem is decomposed into a set of smaller problems). Our proposed architecture adopts
435 some of the key ideas from this early work, including tree search and hierarchical planning.

436 A few recent studies have investigated planning and multi-step decision making in LLMs. These
437 studies suggest that, although LLMs can perform relatively simple planning tasks (Huang et al.,
438 2022), and can learn to make more complex plans given extensive domain-specific fine-tuning (Pal-
439 lagani et al., 2022; Wu et al., 2023), they struggle on tasks that require zero-shot or few-shot gen-
440 eration of multi-step plans (Valmeekam et al., 2023; Momennejad et al., 2023). These results also
441 align with studies that have found poor performance in tasks that involve other forms of extended
442 multi-step reasoning, such as arithmetic (Dziri et al., 2023). Our approach is in large part motivated
443 by the poor planning and reasoning performance exhibited by LLMs in these settings.

444 Some recent approaches have employed various forms of heuristic search to improve performance
445 in LLMs (Lu et al., 2021; Zhang et al., 2023b), but these approaches have generally involved search
446 at the level of individual tokens. Importantly, this is in contrast to our approach, in which search is
447 performed at the more abstract level of task states (described in natural language). Ours is similar
448 to other recently proposed black-box approaches in which ‘thoughts’ – meaningful chunks of nat-
449 ural language – are utilized as intermediate computations to solve more complex problems. These
450 approaches include scratchpads (Nye et al., 2021), chain-of-thought (Wei et al., 2022b), approaches
451 that combine LLMs with tree search (Yao et al., 2023; Zhao et al., 2024; Hao et al., 2023), re-
452 flexion (Shinn et al., 2023), agent-based and multi-agent approaches (Du et al., 2023; Zhang et al.,
453 2023a; Wang et al., 2023c; Li et al., 2023; Wang et al., 2023a;b; Zhou et al., 2023; Prasad et al.,
454 2023; Liu et al., 2023; Song et al., 2023), and methods for combining planning with external
455 tools (Ruan et al., 2023; Kong et al., 2023). All of these approaches can be viewed as imple-
456 menting a form of controlled, or ‘system 2’, processing (as contrasted with automatic, or ‘system
457 1’, processing) (Schneider & Shiffrin, 1977; Sloman, 1996; Kahneman, 2011). Our approach has a
458 similar high-level motivation, and shares some components with other black box approaches (e.g.,
459 tree search (Yao et al., 2023)), but also introduces a number of new components (error monitoring,
460 task decomposition, task coordination, state/action distinction), and combines these components in
461 a novel manner (see Section A.2 for further discussion).

462 There have also been a number of proposals for incorporating modularity into deep learning sys-
463 tems, including neural module networks (Andreas et al., 2016), and recurrent independent mech-
464 anisms (Goyal et al., 2019). Ours is distinguished from these approaches by the use of black-box
465 modules that perform specific high-level functions (many of which are inspired by formal theories
466 of decision-making, as discussed below), rather than merely incorporating a general bias toward
467 modularity.

468 Our approach is inspired by formal theories of decision-making and planning, and has a particu-
469 larly close connection to reinforcement learning (RL) (Sutton & Barto, 2018). In particular, many
470 of the modules in the MAP algorithm are closely related to aspects of traditional RL algorithms.
471 Specifically, the Actor and Evaluator modules bear some resemblance to the actor and the critic
472 in the popular actor-critic framework (Barto et al., 1983), and the TaskDecomposer is related to
473 hierarchical RL (Sutton et al., 1999; Dietterich, 2000; Bacon et al., 2017), in which temporal ab-
474 stractions are learned to achieve subgoals. The Predictor is also closely related to the world model
475 that can substitute for direct interaction with the environment in model-based RL (Sutton & Barto,
476 2018; Daw, 2012). An important difference in each of these cases is that the modules in MAP only
477 receive a task description and a couple of examples, relying on the general-purpose knowledge of the
478 LLM to effectively perform the task, rather than being trained through RL. This also distinguishes
479 the approach from other recent efforts to combine LLMs and RL (Carta et al., 2023; Zhou et al.,
480 2024; Zhai et al., 2024), which involve training with RL through direct interaction with an external
481 environment, whereas MAP generates plans internally. Finally, there are also some modules that
482 have no obvious analog in previous RL algorithms, but which were necessitated by weaknesses that
483 LLMs display in the planning domain. These include the Monitor, which was necessitated by the
484 tendency of LLMs to hallucinate or violate task constraints, and the Orchestrator, which allows
485 MAP to autonomously determine when a goal has been achieved (without groundtruth evaluation)
and thus terminate planning.

6 CONCLUSION AND FUTURE DIRECTIONS

In this work, we have proposed the MAP architecture, a modular agentic approach aimed at improving planning with LLMs. In experiments on four challenging domains, we found that MAP significantly improved multi-step planning and decision-making performance over other LLM methods (e.g., Chain of Thought, Multi-Agent Debate, Tree of Thought). While these results represent a significant step forward, there is still room for improvement. In particular, while improving performance significantly, in this work we only considered problems in which the environment is fully observable and deterministic. Future work should investigate how to extend the proposed approach to more complex open-ended environments, especially by incorporating memory mechanisms for storing knowledge about the environment as it is accumulated. Additionally, the model still has less than optimal performance on Tower of Hanoi, the Reward Revaluation graph traversal task, and the PlanBench benchmark (Valmeekam et al., 2023) (see Section A.6 for discussion of failure modes). This may be due in part to the inherent limitations of prompting and in-context learning as methods for the specialization of MAP’s modules. A promising avenue for further improvement may be to jointly fine-tune smaller open-source LLMs to serve as modules across a range of diverse tasks, rather than relying only on black box methods (as with GPT-4). This approach would also eliminate the need for task-specific prompts, and may further improve zero-shot planning on novel tasks.

An additional limitation of the current implementation is the computational cost incurred by the model (see Section A.8). Although this aligns well with the deliberative nature of controlled (i.e., ‘system 2’) processes (Kahneman, 2011), it would nevertheless be desirable to find ways to reduce these costs. In Section A.8, we present results from a version of MAP that achieves significantly improved efficiency, while retaining the same level of performance, by caching and re-using the outputs of some modules. We also found that MAP is effective when used with a smaller model (Llama3-70B), though performance was not as strong as the version that used GPT-4. Further improvements may result from fine-tuning smaller models to perform the specialized roles of each module.

Finally, it is interesting to consider the close parallels between our proposed approach and the neural basis of human planning and decision-making. In the human brain, planning is generally thought to depend on the prefrontal cortex (PFC) (Owen, 1997; Momennejad et al., 2018; Momennejad, 2020; Russin et al., 2020; Brunec & Momennejad, 2022; Mattar & Lengyel, 2022), a region in the frontal lobe that is notably most developed in humans and is broadly involved in executive function, decision-making, and reasoning (Miller & Cohen, 2001). Research in cognitive neuroscience has revealed the involvement of several subregions or modules within the PFC that appear to be specialized to perform certain functions, many of which are closely aligned with some of the modules in our proposed approach. These include the Anterior Cingulate Cortex, which is known to play a role in conflict monitoring (Botvinick et al., 1999), similar to our Monitor module; the Orbitofrontal Cortex, which plays a role in state prediction and state evaluation (Wallis, 2007; Schuck et al., 2016), similar to our Predictor and Evaluator modules; and the Anterior PFC, which plays a role in task decomposition and coordination, similar to our TaskDecomposer and Orchestrator modules. Human planning then emerges through the coordinated and recurrent interactions among these specialized subregions, and, similar to our approach, the algorithms implemented via these interactions are closely related to RL (O’doherly, 2004; Daw et al., 2005; Valentin et al., 2007; Takahashi et al., 2011; Silvetti et al., 2014; Brunec & Momennejad, 2022; Wang et al., 2018; Botvinick et al., 2019). An exciting direction for future work is to consider how the present approach might further contribute to understanding the brain basis of planning and decision-making.

REFERENCES

- Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 39–48, 2016.
- Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.
- Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

- 540 Timothy EJ Behrens, Timothy H Muller, James CR Whittington, Shirley Mark, Alon B Baram,
541 Kimberly L Stachenfeld, and Zeb Kurth-Nelson. What is a cognitive map? organizing knowledge
542 for flexible behavior. *Neuron*, 100(2):490–509, 2018.
- 543
544 Matthew Botvinick, Leigh E Nystrom, Kate Fissell, Cameron S Carter, and Jonathan D Cohen.
545 Conflict monitoring versus selection-for-action in anterior cingulate cortex. *Nature*, 402(6758):
546 179–181, 1999.
- 547 Matthew Botvinick, Sam Ritter, Jane X Wang, Zeb Kurth-Nelson, Charles Blundell, and Demis
548 Hassabis. Reinforcement learning, fast and slow. *Trends in cognitive sciences*, 23(5):408–422,
549 2019.
- 550 Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal,
551 Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are
552 few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- 553
554 Iva K Brunec and Ida Momennejad. Predictive representations in hippocampal and prefrontal hier-
555 archies. *Journal of Neuroscience*, 42(2):299–312, 2022.
- 556 Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Ka-
557 mar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general
558 intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.
- 559
560 Patricia A Carpenter, Marcel A Just, and Peter Shell. What one intelligence test measures: a theo-
561 retical account of the processing in the raven progressive matrices test. *Psychological review*, 97
562 (3):404, 1990.
- 563 Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves
564 Oudeyer. Grounding large language models in interactive environments with online reinforcement
565 learning. In *International Conference on Machine Learning*, pp. 3676–3713. PMLR, 2023.
- 566
567 Nathaniel D Daw. Model-based reinforcement learning as cognitive search: neurocomputational
568 theories. 2012.
- 569 Nathaniel D Daw, Yael Niv, and Peter Dayan. Uncertainty-based competition between prefrontal
570 and dorsolateral striatal systems for behavioral control. *Nature neuroscience*, 8(12):1704–1711,
571 2005.
- 572 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep
573 bidirectional transformers for language understanding. *Proceedings of NAACL-HLT*, 17:4171–
574 4186, 2019.
- 575
576 Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decompo-
577 sition. *Journal of artificial intelligence research*, 13:227–303, 2000.
- 578 Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improv-
579 ing factuality and reasoning in language models through multiagent debate. *arXiv preprint*
580 *arXiv:2305.14325*, 2023.
- 581
582 Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jian, Bill Yuchen Lin, Peter West,
583 Chandra Bhagavatula, Ronan Le Bras, Jena D Hwang, et al. Faith and fate: Limits of transformers
584 on compositionality. *arXiv preprint arXiv:2305.18654*, 2023.
- 585 Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. Did aristotle
586 use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of*
587 *the Association for Computational Linguistics*, 9:346–361, 2021.
- 588
589 Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio,
590 and Bernhard Schölkopf. Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*,
591 2019.
- 592 Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu.
593 Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*,
2023.

- 594 Hosein Hasanbeig, Hiteshi Sharma, Leo Betthausen, Felipe Vieira Frujeri, and Ida Momennejad. Al-
595 lure: A systematic protocol for auditing and improving llm-based evaluation of text using iterative
596 in-context-learning. *arXiv preprint arXiv:2309.13701*, 2023.
- 597
- 598 Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot
599 planners: Extracting actionable knowledge for embodied agents. In *International Conference on*
600 *Machine Learning*, pp. 9118–9147. PMLR, 2022.
- 601 Daniel Kahneman. *Thinking, fast and slow*. macmillan, 2011.
- 602
- 603 Yilun Kong, Jingqing Ruan, Yihong Chen, Bin Zhang, Tianpeng Bao, Shiwei Shi, Guoqing Du,
604 Xiaoru Hu, Hangyu Mao, Ziyue Li, et al. Tptu-v2: Boosting task planning and tool usage of large
605 language model-based agents in real-world systems. *arXiv preprint arXiv:2311.11315*, 2023.
- 606
- 607 Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Com-
608 municative agents for” mind” exploration of large language model society. *Advances in Neural*
609 *Information Processing Systems*, 36:51991–52008, 2023.
- 610 Zhiwei Liu, Weiran Yao, Jianguo Zhang, Le Xue, Shelby Heinecke, Rithesh Murthy, Yihao Feng,
611 Zeyuan Chen, Juan Carlos Niebles, Devansh Arpit, et al. Bolaa: Benchmarking and orchestrating
612 llm-augmented autonomous agents. *arXiv preprint arXiv:2308.05960*, 2023.
- 613
- 614 Ximing Lu, Sean Welleck, Peter West, Liwei Jiang, Jungo Kasai, Daniel Khashabi, Ronan Le Bras,
615 Lianhui Qin, Youngjae Yu, Rowan Zellers, et al. Neurologic a* esque decoding: Constrained text
616 generation with lookahead heuristics. *arXiv preprint arXiv:2112.08726*, 2021.
- 617 Kyle Mahowald, Anna A Ivanova, Idan A Blank, Nancy Kanwisher, Joshua B Tenenbaum, and
618 Evelina Fedorenko. Dissociating language and thought in large language models: a cognitive
619 perspective. *arXiv preprint arXiv:2301.06627*, 2023.
- 620
- 621 Marcelo G Mattar and Máté Lengyel. Planning in the brain. *Neuron*, 110(6):914–934, 2022.
- 622
- 623 Earl K Miller and Jonathan D Cohen. An integrative theory of prefrontal cortex function. *Annual*
624 *review of neuroscience*, 24(1):167–202, 2001.
- 625 I Momennejad, A R Otto, N D Daw, and K A Norman. Offline replay supports planning in human
626 reinforcement learning. *Elife*, 2018.
- 627
- 628 Ida Momennejad. Learning structures: Predictive representations, replay, and generalization. *Cur-*
629 *rent Opinion in Behavioral Sciences*, 32:155–166, April 2020.
- 630
- 631 Ida Momennejad, Hosein Hasanbeig, Felipe Vieira Frujeri, Hiteshi Sharma, Robert Osazuwa Ness,
632 Nebojsa Jojic, Hamid Palangi, and Jonathan Larson. Evaluating cognitive maps in large language
633 models with cogeal: No emergent planning. In *Advances in neural information processing*
634 *systems*, volume 37, 2023. URL <https://arxiv.org/abs/2309.15129>.
- 635
- 636 Allen Newell and Herbert Simon. The logic theory machine—a complex information processing
637 system. *IRE Transactions on information theory*, 2(3):61–79, 1956.
- 638
- 639 Allen Newell, John C Shaw, and Herbert A Simon. Report on a general problem solving program.
640 In *IFIP congress*, volume 256, pp. 64. Pittsburgh, PA, 1959.
- 641
- 642 Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin,
643 David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show
644 your work: Scratchpads for intermediate computation with language models. *arXiv preprint*
645 *arXiv:2112.00114*, 2021.
- 646
- 647 Adrian M Owen. Cognitive planning in humans: neuropsychological, neuroanatomical and neu-
648 ropharmacological perspectives. *Progress in neurobiology*, 53(4):431–450, 1997.
- 649
- 650 John P O’doherly. Reward representations and reward-related learning in the human brain: insights
651 from neuroimaging. *Current opinion in neurobiology*, 14(6):769–776, 2004.

- 648 Vishal Pallagani, Bharath Muppasani, Keerthiram Murugesan, Francesca Rossi, Lior Horesh, Biplav
649 Srivastava, Francesco Fabiano, and Andrea Loreggia. Plansformer: Generating symbolic plans
650 using transformers. *arXiv preprint arXiv:2212.08681*, 2022.
- 651 Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit
652 Bansal, and Tushar Khot. Adapt: As-needed decomposition and planning with language mod-
653 els. *arXiv preprint arXiv:2311.05772*, 2023.
- 654
655 Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi,
656 Hangyu Mao, Xingyu Zeng, and Rui Zhao. Tptu: Task planning and tool usage of large language
657 model-based ai agents. *arXiv preprint arXiv:2308.03427*, 2023.
- 658
659 Jacob Russin, Randall C O’Reilly, and Yoshua Bengio. Deep learning needs a prefrontal cortex.
660 *Work Bridging AI Cogn Sci*, 107(603-616):1, 2020.
- 661
662 Anna C Schapiro, Timothy T Rogers, Natalia I Cordova, Nicholas B Turk-Browne, and Matthew M
663 Botvinick. Neural representations of events arise from temporal community structure. *Nature*
664 *neuroscience*, 16(4):486–492, 2013.
- 665
666 Walter Schneider and Richard M Shiffrin. Controlled and automatic human information processing:
667 I. detection, search, and attention. *Psychological review*, 84(1):1, 1977.
- 668
669 Nicolas W Schuck, Ming Bo Cai, Robert C Wilson, and Yael Niv. Human orbitofrontal cortex
670 represents a cognitive map of state space. *Neuron*, 91(6):1402–1412, 2016.
- 671
672 Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic
673 memory and self-reflection. *arXiv preprint arXiv:2303.11366*, 2023.
- 674
675 Massimo Silvetti, William Alexander, Tom Verguts, and Joshua W Brown. From conflict man-
676 agement to reward-based decision making: actors and critics in primate medial frontal cortex.
677 *Neuroscience & Biobehavioral Reviews*, 46:44–57, 2014.
- 678
679 Herbert A Simon. The functional equivalence of problem solving skills. *Cognitive psychology*, 7
680 (2):268–288, 1975.
- 681
682 Steven A Sloman. The empirical case for two systems of reasoning. *Psychological bulletin*, 119(1):
683 3, 1996.
- 684
685 Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su.
686 Llm-planner: Few-shot grounded planning for embodied agents with large language models. In
687 *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2998–3009,
688 2023.
- 689
690 Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam
691 Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the
692 imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint*
693 *arXiv:2206.04615*, 34:1877—1901, 2022.
- 694
695 Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- 696
697 Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A frame-
698 work for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–
699 211, 1999.
- 700
701 Yuji K Takahashi, Matthew R Roesch, Robert C Wilson, Kathy Toreson, Patricio O’donnell, Yael
Niv, and Geoffrey Schoenbaum. Expectancy-related changes in firing of dopamine neurons de-
pend on orbitofrontal cortex. *Nature neuroscience*, 14(12):1590–1597, 2011.
- Rita Morais Tavares, Avi Mendelsohn, Yael Grossman, Christian Hamilton Williams, Matthew
Shapiro, Yaacov Trope, and Daniela Schiller. A map for social navigation in the human brain.
Neuron, 87(1):231–243, 2015.
- Edward C Tolman. Cognitive maps in rats and men. *Psychological review*, 55(4):189, 1948.

- 702 Vivian V Valentin, Anthony Dickinson, and John P O’Doherty. Determining the neural substrates
703 of goal-directed learning in the human brain. *Journal of Neuroscience*, 27(15):4019–4026, 2007.
704
- 705 Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On
706 the planning abilities of large language models—a critical investigation. *arXiv preprint*
707 *arXiv:2305.15771*, 2023.
- 708 Jonathan D Wallis. Orbitofrontal cortex and its contribution to decision-making. *Annu. Rev. Neu-*
709 *rosci.*, 30:31–56, 2007.
710
- 711 Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan,
712 and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models.
713 *arXiv preprint arXiv:2305.16291*, 2023a.
714
- 715 Jane X Wang, Zeb Kurth-Nelson, Dharshan Kumaran, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo,
716 Demis Hassabis, and Matthew Botvinick. Prefrontal cortex as a meta-reinforcement learning
717 system. *Nature neuroscience*, 21(6):860–868, 2018.
- 718 Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin,
719 Zhaofeng He, Zilong Zheng, Yaodong Yang, et al. Jarvis-1: Open-world multi-task agents with
720 memory-augmented multimodal language models. *arXiv preprint arXiv:2311.05997*, 2023b.
721
- 722 Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and
723 select: Interactive planning with large language models enables open-world multi-task agents.
724 *arXiv preprint arXiv:2302.01560*, 2023c.
- 725 Taylor Webb, Keith J Holyoak, and Hongjing Lu. Emergent analogical reasoning in large language
726 models. *Nature Human Behaviour*, 7:1526—1541, 2023. URL [https://doi.org/10.](https://doi.org/10.1038/s41562-023-01659-w)
727 [1038/s41562-023-01659-w](https://doi.org/10.1038/s41562-023-01659-w).
728
- 729 Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani
730 Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto,
731 Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large lan-
732 guage models. *Transactions on Machine Learning Research*, 2022a. ISSN 2835-8856. URL
733 <https://openreview.net/forum?id=yzkSU5zdwD>. Survey Certification.
- 734 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny
735 Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in*
736 *Neural Information Processing Systems*, 35:24824–24837, 2022b.
737
- 738 Zhenyu Wu, Ziwei Wang, Xiuwei Xu, Jiwen Lu, and Haibin Yan. Embodied task planning with
739 large language models. *arXiv preprint arXiv:2307.01848*, 2023.
740
- 741 Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik
742 Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv*
743 *preprint arXiv:2305.10601*, 2023.
- 744 Yuexiang Zhai, Hao Bai, Zipeng Lin, Jiayi Pan, Shengbang Tong, Yifei Zhou, Alane Suhr, Saining
745 Xie, Yann LeCun, Yi Ma, et al. Fine-tuning large vision-language models as decision-making
746 agents via reinforcement learning. *arXiv preprint arXiv:2405.10292*, 2024.
747
- 748 Bin Zhang, Hangyu Mao, Jingqing Ruan, Ying Wen, Yang Li, Shao Zhang, Zhiwei Xu, Dapeng
749 Li, Ziyue Li, Rui Zhao, et al. Controlling large language model-based agents for large-scale
750 decision-making: An actor-critic approach. *arXiv preprint arXiv:2311.13884*, 2023a.
- 751 Shun Zhang, Zhenfang Chen, Yikang Shen, Mingyu Ding, Joshua B Tenenbaum, and Chuang Gan.
752 Planning with large language models for code generation. *arXiv preprint arXiv:2303.05510*,
753 2023b.
754
- 755 Zirui Zhao, Wee Sun Lee, and David Hsu. Large language models as commonsense knowledge for
large-scale task planning. *Advances in Neural Information Processing Systems*, 36, 2024.

756 Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Lan-
757 guage agent tree search unifies reasoning acting and planning in language models. *arXiv preprint*
758 *arXiv:2310.04406*, 2023.

759 Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. Archer: Training language
760 model agents via hierarchical multi-turn rl. *arXiv preprint arXiv:2402.19446*, 2024.

761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809

A APPENDIX

A.1 SUPPLEMENTARY ALGORITHMS

Algorithm 3: Search loop. Tree search with a depth of L layers, with B branches at each layer l . For each branch, a proposed action is sampled, and the Predictor predicts the next state \tilde{x} . This process continues recursively until the terminal layer L , at which point the value $v_{l=L}$ of the terminal states is estimated by the Evaluator. The values are backpropagated to their parent states in the first layer, and the action that leads to the most valuable state is selected. In our implementation, we accelerate this process by caching the actions and predicted states from deeper search layers and then reusing them in subsequent searches. We also employ the Orchestrator to prematurely terminate search if the goal state is achieved.

```

810
811
812
813
814
815 Function Search ( $l, L, B, x, y$ ):
816    $V_l \leftarrow \{\}$  // Initialize value record
817    $\tilde{X}_l \leftarrow \{\}$  // Initialize next-state record
818    $A_l \leftarrow \text{ProposeAction}(x, y, B)$  // Propose B actions
819   for  $b$  in  $1 \dots B$  do
820      $\tilde{x}_{lb} \leftarrow \text{Predictor}(x, A_{lb})$  // Predict next state
821      $\tilde{X}_l \leftarrow \tilde{X}_l \cup \{\tilde{x}_{lb}\}$  // Update next-state record
822      $\Omega \leftarrow \text{Orchestrator}(\tilde{x}_{lb}, y)$  // Terminate search if goal achieved
823     if  $l < L$  and  $\Omega$  is false then
824        $a_{l+1}, \tilde{x}_{l+1}, v_{l+1} \leftarrow \text{Search}(l+1, L, B, \tilde{x}_{lb}, y)$  // Advance search depth
825        $V_l \leftarrow V_l \cup \{v_{l+1}\}$  // Update value record
826     else
827        $v_{lb} \leftarrow \text{Evaluator}(\tilde{x}_{lb}, y)$  // Evaluate predicted state
828        $V_l \leftarrow V_l \cup \{v_{lb}\}$  // Update value record
829     end
830   end
831    $v_l \leftarrow \max(V_l)$  // Maximum value (randomly sample if equal value)
832    $a_l \leftarrow A_l \text{ argmax}(V_l)$  // Select action
833    $\tilde{x}_l \leftarrow \tilde{X}_l \text{ argmax}(V_l)$  // Predicted next-state
834 return  $a_l, \tilde{x}_l, v_l$ 
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863

```

864 A.2 EXTENDED RELATED WORK

865
866 In this section, we consider in more detail how MAP relates to existing black-box and agentic LLM
867 approaches:

- 868 • Similar to MAP, both scratchpad (Nye et al., 2021) and chain-of-thought (CoT) (Wei et al.,
869 2022b) decompose a problem into intermediate computations. However, unlike MAP, nei-
870 ther scratchpad nor CoT factorize these intermediate computations into specialized mod-
871 ules.
- 872 • Tree-of-thought (ToT) (Yao et al., 2023) introduces some degree of factorization, but the
873 factorization is not as extensive as in MAP. The ‘generator’ module in ToT carries out
874 a combination of the functions carried out by both the Actor (action proposal) and the
875 Predictor (prediction of the states that will result from these actions) in MAP. The ‘eval-
876 uator’ module in ToT carries out a combination of the functions carried out by both the
877 Monitor (error detection) and the Evaluator (prediction of state value) in MAP. ToT does
878 not contain any component that carries out the functions of the TaskDecomposer (subgoal
879 proposal) and the Orchestrator (autonomously determining when a goal or subgoal has
880 been achieved). LLM-MCTS (Zhao et al., 2024) and RAP (Hao et al., 2023) also propose
881 similar approaches involving the combination of LLMs with tree search.
- 882 • Multi-agent debate (i.e., Society of Mind) (Du et al., 2023) involves the interaction of multi-
883 ple LLM instances; but, unlike MAP, these model instances are not specialized to perform
884 specific functions. Similarly, the Large Language Model-based Actor-Critic (LLaMAC)
885 approach (Zhang et al., 2023a) involves interaction between many LLM agents, and incor-
886 porates more specialization (there is a ‘critic’ module that coordinates the decision-making
887 process across many ‘actors’), though there is less module specialization than there is in
888 MAP. Similarly, CAMEL (Li et al., 2023) involves interaction between two role-playing
889 modules (a user and an assistant).
- 890 • Similar to MAP, reflexion (Shinn et al., 2023) involves an element of self evaluation of
891 proposed policies, but this depends on interaction with the external environment to deter-
892 mine the outcome of each policy (whereas in MAP, this self evaluation process is entirely
893 internal to the agent). The dependence on interaction with the external environment makes
894 this approach unsuitable for the planning domain (planning is, by definition, performed
895 internally).
- 896 • Describe-Explain-Plan-Select (Wang et al., 2023c) involves the coordination of multiple
897 modules, but the approach is specific to settings involving an agent that is spatially embed-
898 ded in a 2D environment. For instance, the method utilizes the spatial proximity of objects
899 to the agent for prioritization of subgoals. This approach cannot be directly applied to the
900 tasks that we consider in the present work.
- 901 • JARVIS-1 (Wang et al., 2023b), Voyager (Wang et al., 2023a), BOLAA (Liu et al., 2023),
902 and LLM-Planner (Song et al., 2023) all involve agent-based approaches featuring multiple
903 LLM modules, but these approaches require interaction with an external environment to
904 iteratively solve a problem, rather than developing a plan internally as in our approach.
- 905 • Adapt (Prasad et al., 2023) is another modular agent-based approach that, similar to our
906 approach, involves task decomposition, and LATS (Zhou et al., 2023) is another agent-
907 based approach that employs tree search. Both of these approaches require interaction with
908 an external environment to develop plans, whereas plans are developed internally in our
909 approach. Furthermore, Adapt doesn’t use any tree search, and LATS doesn’t decompose
910 a task into subgoals, both of which are present in MAP.
- 911 • Some recent work has aimed to combine external tool use with LLM planning agents,
912 including Task Planning and Tool Usage (TPTU) (Ruan et al., 2023; Kong et al., 2023).
913 Because this work evaluates on tasks involving external tool use, it is not directly compar-
914 able with our approach, but we plan to extend our approach to incorporate tool use in future.
915 We expect that MAP’s modular approach will afford improved performance in this domain,
916 relative to the more minimal planning approaches that have previously been employed (i.e.,
917 involving planning in a single LLM agent).

A.3 EXPERIMENT DETAILS

We implemented each of the modules using a separate GPT-4 (32K context, ‘2023-03-15-preview’ model index for ToH and cogeval tasks, and 128K context, ‘0125-preview’ model index for strategyQA and planbench tasks from Microsoft Azure openAI service) instance through a combination of prompting and few-shot in-context examples. We set Top-p to 0 and temperature to 0, except for the Actor (as detailed in section A.7.2). The Search loop explored $B = 2$ branches recursively for a depth $L = 2$.

For ToH, we used two randomly selected in-context examples of three-disk problems, and a description of the problem in the prompts for all the modules. For the graph traversal tasks, we used two in-context examples for all modules, except for the Actor and Evaluator in the Steppath task, where we used three in-context examples, one each for 2-, 3-, and 4-step paths. For strategyQA, we didn’t use any in-context examples. For the logistics task from Planbench, we used two incontext examples for all modules except for Actor which used three in-context examples. For the mystery blocksworld (deceptive) task from Planbench, we used two incontext examples for all modules except for Actor and Predictor which used three in-context examples. For both the tasks from Planbench, we extracted the goal from the initial state conditions, and the state and the goal was separately fed as input to the modules as required. The prompt also described the specific task that was to be performed by each module (e.g., monitoring, task decomposition). For more details about the prompts and specific procedures used for each module, see Section A.7.

For three-disk problems, we allowed a maximum of $T = 10$ actions per problem, and evaluated on 24 out of 26 possible problems (leaving out the two problems that were used as in-context examples for the Actor). We also evaluated on four-disk problems, for which we allowed a maximum of $T = 20$ actions per problem. The same three-disk problems were used as in-context examples, meaning that the four-disk problems tested for out-of-distribution (OOD) generalization. For the graph traversal tasks, we allowed a maximum of $T = 6$ actions per problem. For strategyQA we allowed $T = 1$ action per problem. For the Planbench tasks we allowed a maximum of $T = 4 +$ number of actions in the ground truth plan.

We didn’t use a separate Predictor for the graph traversal tasks, since the action proposed by the Actor gives the next state. We also did not include the TaskDecomposer for these tasks, and did not use the Search loop for the Steppath task, as the model’s performance was already at ceiling without the use of these components. For strategyQA we didn’t use the Evaluator or the Orchestrator. For the Planbench tasks we didn’t use tree search, and for mystery blocksworld task we didn’t use the TaskDecomposer.

A.4 ADDITIONAL DESCRIPTION OF PLANNING TASKS

Graph Traversal. We focused on a particular type of graph (Figure 4) with community structure (Schapiro et al., 2013) previously found to be challenging for a wide variety of LLMs. The first task, Valuepath, involves finding the shortest path from a given room to the room with the largest reward, while avoiding the room that has a smaller reward. A smaller reward and a larger reward are located at two different positions in the graph. We fixed the two reward locations, and created 13 problems based on different starting locations. The second task, Steppath, involves finding the shortest path between a pair of nodes. We evaluated problems with an optimal shortest path of 2, 3, or 4 steps. We generated 20 problems for each of these conditions by sampling different starting and target locations.

The other two tasks, Detour and Reward Revaluation, involve modifications to the Valuepath task that test for flexibility in planning. In these tasks, the problem description and in-context examples for the Valuepath task are presented, and a single Valuepath problem is solved as in the original task. The task is then modified in-context in one of two ways. In the Detour task, an edge is removed from the graph and replaced with a new edge (e.g., ‘the door from room 1 to room 11 is locked and now room 13 is connected to room 11’). In the Reward Revaluation task, the value associated with the two reward locations is changed (e.g., ‘the reward of the chest in room 8 has been changed to 12 and the reward of the chest in room 15 has been changed to 48’). As with the Valuepath task, the Detour and Reward Revaluation tasks each involved 13 problems based on different starting locations.

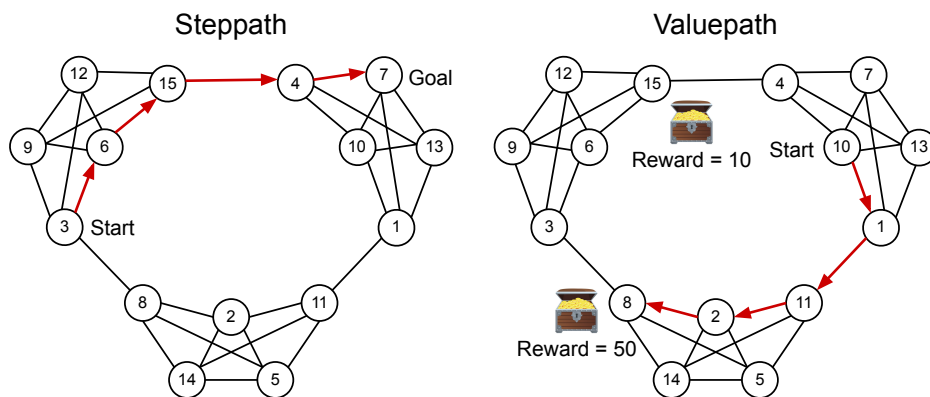
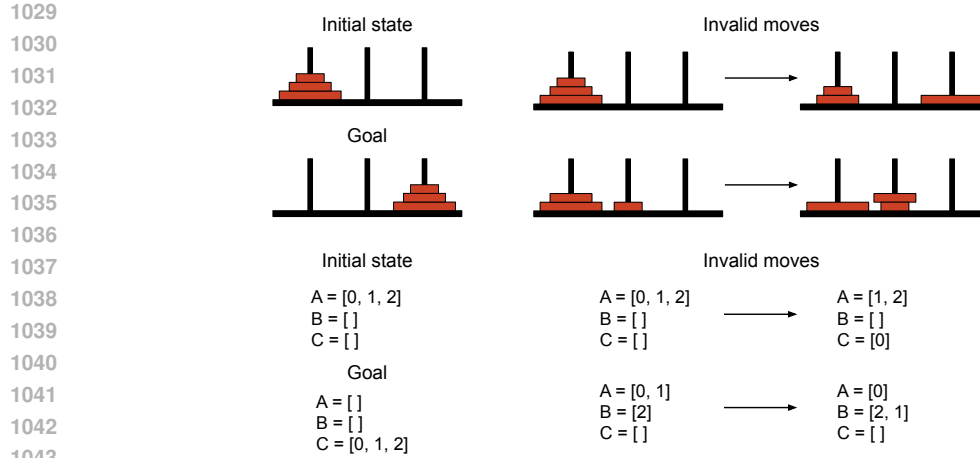


Figure 4: **Graph Traversal.** We investigated two graph traversal tasks utilizing a challenging graph with community structure. **Steppath:** Find shortest path between two nodes, e.g. node 3 and node 7. **Valuepath:** Find shortest path from starting location (e.g., node 10) to location with maximum reward (node 8 in depicted example).

Tower of Hanoi. In the original version of the Tower of Hanoi (ToH) task, there are three pegs and a set of disks of different sizes. The disks are stacked in order of decreasing size on the leftmost peg. The goal is to move all disks to the rightmost peg, such that the disks are stacked in order of decreasing size. There are a couple of rules that determine which moves are considered valid. First, a disk can only be moved if it is at the top of its stack. Second, a disk can only be moved to the top of another stack if it is smaller than the disks in that stack (or if the peg is empty). More complex versions of the task can be created by using a larger number of disks.

We designed an alternative formulation of this task in which the inputs are text-based rather than visual. In this alternative formulation, three lists (A, B, and C) are used instead of the three pegs, and a set of numbers (0, 1, 2, and so on) is used instead of disks of different sizes. The goal is to move all numbers so that they are arranged in ascending order in list C. The rules are isomorphic to ToH. First, a number can only be moved if it is at the end of a list. Second, a number can only be moved to the end of a new list if it is larger than all the numbers in that list. Note that although this novel formulation is isomorphic to ToH (and equally complex), it does not share any surface features with the original ToH puzzle (disks, pegs, etc.), and thus GPT-4 cannot rely on exposure to descriptions of ToH in its training data to solve the problem. We created multiple problem instances

1026 by varying the initial state (the initial positions of the numbers). This resulted in 26 three-disk
 1027 problems and 80 four-disk problems.
 1028



1045 **Figure 5: Tower of Hanoi. Top:** Depiction of the Tower of Hanoi (ToH) puzzle. Disks are stacked
 1046 in order of decreasing size on the leftmost peg. The goal is to move these disks so that they are
 1047 stacked in order of decreasing size on the rightmost peg. Only the disk on the top of the stack may
 1048 be moved, and a disk can only be placed on top of larger disks (or on an empty peg). The version
 1049 shown involves three disks, but more disks can be used (making the task significantly more difficult).
 1050 **Bottom:** Modified text-based version of ToH. Three lists are presented, labelled A, B and C. A set of
 1051 integers is distributed amongst these lists. The goal is to move the numbers so that they are arranged
 1052 in ascending order in list C. Only the number at the end of the list may be moved, and a number can
 1053 only be placed in front of a smaller number. Multiple problem instances were created by varying the
 1054 initial state.

1055
 1056
 1057
 1058
 1059
 1060
 1061
 1062
 1063
 1064
 1065
 1066
 1067
 1068
 1069
 1070
 1071
 1072
 1073
 1074
 1075
 1076
 1077
 1078
 1079

A.5 SUPPLEMENTARY RESULTS

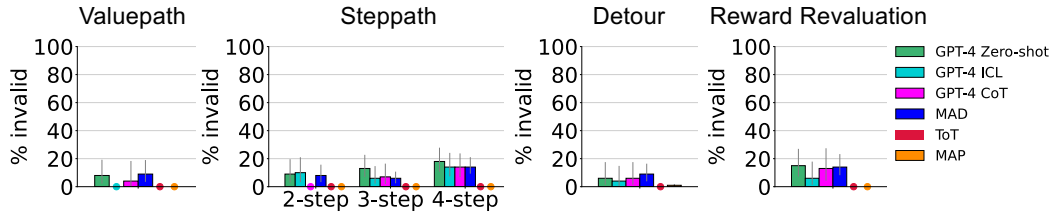


Figure 6: **Graph traversal results.** ‘% invalid’ indicates percentage of moves that are invalid (\downarrow better). GPT-4 Zero-shot, ICL, CoT, and MAD baselines are deterministic, and therefore a single run was performed on all problems. Note that MAP did not employ tree search on the Steppath task, and did not employ task decomposition on any of the graph traversal tasks. Without tree search, MAP’s performance is deterministic, and therefore only a single run was performed on the Steppath task, whereas we performed 5 runs with ToT. Gray error bars reflect 95% binomial confidence intervals (for models evaluated on a single run). Dots reflect values of 0%. For Valuepath, Detour, and Reward Revaluation we performed 10, 10, and 5 runs respectively with MAP and ToT, and present average performance \pm the standard error of the mean (black error bars).

1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187

Table 4: Results on Valuepath task. Values within brackets indicate best performance.

Model	% solved problems	% invalid actions	Avg plan steps		
			1-step	2-step	4-step
GPT-4 Zero-shot	54	8	2.5	2.5	5
GPT-4 ICL	91	0	1.75	2.33	4.67
GPT-4 CoT	91	4			
MAD	73	9			
ToT	55(100)	0			
MAP	100	0	1.5	2	4.75

Table 5: Results on Steppath task. Values within brackets indicate best performance.

Model	% solved problems			% invalid actions			Avg plan steps		
	2-step	3 step	4-step	2-step	3-step	4-step	2-step	3-step	4-step
GPT-4 Zero-shot	75	40	20	9	13	18	2.07	4	5.25
GPT-4 ICL	74	74	42	10	6	14	2.14	3.78	4.38
GPT-4 CoT	95	79	47	0	7	14			
MAD	82	79	39	8	6	14			
ToT	67(100)	67(89)	50(63)	0	0	0			
MAP	100	100	95	0	0	0	2.1	3.42	4.5

Table 6: Results on Detour task. Values within brackets indicate best performance.

Model	% solved problems	% invalid actions
GPT-4 Zero-shot	54	6
GPT-4 ICL	38	4
GPT-4 CoT	69	6
MAD	38	9
ToT	33 (100)	0
MAP	85 (100)	1

Table 7: Results on Reward Revaluation task. Values within brackets indicate best performance.

Model	% solved problems	% invalid actions
GPT-4 Zero-shot	31	15
GPT-4 ICL	54	6
GPT-4 CoT	54	13
MAD	46	14
ToT	36 (77)	0
MAP	48 (69)	0

Table 8: Results on ToH. Note that we also include results here for the GPT-4 ICL baseline when prompted with 5 ICL examples (as opposed to 2 examples in the standard version of the baseline). Surprisingly, more ICL examples hurts performance on this task, perhaps due to overfitting to the specific examples (Hasanbeig et al., 2023). Values within brackets indicate best performance.

Model	% solved problems		% invalid actions	
	3-disk	4-disk (OOD)	3-disk	4-disk (OOD)
GPT-4 Zero-shot	11	2	30	50
GPT-4 ICL	46	1	12	41
GPT-4 ICL (5 examples)	38	1	19	41
GPT-4 CoT	42	5	22	39
MAD	25	1	24	43
ToT-MAP	6 (25)	1(4)	4	5
ToT	6 (25)	0 (0)	14	11
MAP	74 (100)	24	0	0

Table 9: Ablation study on ToH with 3 disks. Values within brackets indicate best performance.

Model	% solved problems	% invalid actions
MAP	74 (100)	0
w/o Task Decomposer	50 (67)	0
w/o Tree Search	32 (42)	0
w/o Monitor	27 (33)	31

Table 10: Results on ToH with 3 disks with a smaller LLM (Llama3-70B). MAP with Llama3-70B even outperforms the best GPT-4 baseline (GPT-4 ICL).

Model	% solved problems	% invalid actions
Llama3-70B Zero-shot	19.2	33.8
Llama3-70B ICL	12.5	41.4
Llama3-70B CoT	29.2	33.3
GPT-4 ICL	46	12
Llama3-70B MAP	50	2

Table 11: Results on Mystery Blocksworld domain of PlanBench using a zero-shot version of MAP (no in-context examples were provided for any module).

Model	% solved problems
GPT-4 Zero-shot	0.2
GPT-4 ICL	7.8
MAP Zero-shot	8.2

A.6 ANALYSIS OF FAILURE MODES

To better understand the failure modes displayed by MAP, we analyzed the log files for the Tower of Hanoi (ToH) task (3-disk problems, run accuracy=75%, failed to solve 6/24 problems). We identified the following three general failure modes:

1. Incorrect decomposition: failure to identify subgoals that lie along the optimal path.
2. No progress: taking actions that either move away from or do not make progress toward a subgoal.
3. Falling into loops: visiting a state more than once.

These failure modes were not mutually exclusive. Table 15 shows the number of problems (out of 6 total failures) that involved these failure modes. We then identified which modules were responsible

1242
 1243
 1244
 1245
 1246
 1247
 1248
 1249
 1250
 1251
 1252
 1253
 1254
 1255
 1256
 1257
 1258
 1259
 1260
 1261
 1262
 1263
 1264
 1265
 1266
 1267
 1268
 1269
 1270
 1271
 1272
 1273
 1274
 1275
 1276
 1277
 1278
 1279
 1280
 1281
 1282
 1283
 1284
 1285
 1286
 1287
 1288
 1289
 1290
 1291
 1292
 1293
 1294
 1295

Table 12: Results on ToH 3 disks with baselines also provided with the goal recursion strategy (provided to the Decomposer of MAP).

Model	% solved problems
GPT-4 Zero-shot	11
GPT-4 Zero-shot w/ goal recursion	23
GPT-4 ICL	46
GPT-4 ICL w/ goal recursion	46
GPT-4 CoT	42
GPT-4 CoT w/ goal recursion	50
MAP w/ goal recursion	74

Table 13: Results on Planbench with ToT on a subset of problems. Values within brackets indicate best performance.

Model	Logistics (30 problems)	Mystery BW (100 problems)
ToT	10.4 (16.7)	0.6 (3)
MAP	53.3	35

Table 14: Results on ToH with 3 disks varying the tree search hyperparameters of MAP. Mean and standard error are reported across 10 runs for computational cost and performance.

Model	Num. calls	Num. input tokens	Num. output tokens	% solved
MAP ($B = 2, L = 1$)	32.04 ± 1.5	$32,630.62 \pm 1,318.4$	$2,197.37 \pm 146.7$	55 ± 6
MAP ($B = 2, L = 2$)	43.46 ± 2.4	$49,537.37 \pm 2,563.7$	$2,727.55 \pm 214.2$	72 ± 2
MAP ($B = 2, L = 3$)	65.65 ± 3.6	$90,406.30 \pm 4,784.7$	$3,845.88 \pm 294.8$	69 ± 2

for these failures. The ‘incorrect decomposition’ failure mode was due to mistakes by the Decomposer (2/6 failures). The ‘no progress’ and ‘falling into loops’ failure modes were due to mistakes made both by the Actor and the Evaluator. Specifically, the Actor sometimes failed to propose at least one action that made progress toward the goal, and the Evaluator sometimes failed to select the action that made progress toward the goal. The Actor and Evaluator made these mistakes at least once for each of the 6 failures. Overall, we found that failures stemmed from errors made by the Decomposer, Actor, and Evaluator modules, whereas the other modules (Monitor, Predictor, and Orchestrator) performed perfectly.

Table 15: Results for number of failures (out of 6 total failures for a given run) of MAP on ToH with 3 disks for each of the three failure modes.

Incorrect decomposition	No progress	Falling into loops
2/6	6/6	5/6

1296 A.7 PROMPTS AND IN-CONTEXT EXAMPLES

1297

1298 A.7.1 TASK DECOMPOSER

1299

1300 For ToH, the TaskDecomposer generated a single subgoal per problem. The in-context examples
1301 included chain-of-thought reasoning (Wei et al., 2022b) based on the goal recursion strategy (Simon,
1302 1975), which is sometimes provided to human participants in psychological studies of problem
1303 solving (Carpenter et al., 1990). The specific prompt and in-context examples are shown below:

1304 Consider the following puzzle problem:

1305 Problem description:

- 1306 - There are three lists labeled A, B, and C.
- 1307 - There is a set of numbers distributed among those three lists.
- 1308 - You can only move numbers from the rightmost end of one list to
1309 the rightmost end of another list.

1310 Rule #1: You can only move a number if it is at the rightmost end
1311 of its current list.

1312 Rule #2: You can only move a number to the rightmost end of a
1313 list if it is larger than the other numbers in that list.

1314 A move is valid if it satisfies both Rule #1 and Rule #2.

1315 A move is invalid if it violates either Rule #1 or Rule #2.

1316

1317 Goal: The goal is to generate a single subgoal from the current
1318 configuration, that helps in reaching the goal configuration using
1319 minimum number of moves.

1320 To generate subgoal use the goal recursion strategy. First if the
1321 smallest number isn't at the correct position in list C, then set
1322 the subgoal of moving the smallest number to its correct position
1323 in list C. But before that, the numbers larger than the smallest
1324 number and present in the same list as the smallest number must
1325 be moved to a list other than list C. This subgoal is recursive
1326 because in order to move the next smallest number to the list
1327 other than list C, the numbers larger than the next smallest
1328 number and present in the same list as the next smallest number
1329 must be moved to a list different from the previous other list and
1330 so on.

1331 Note in the subgoal configuration all numbers should always be in
1332 ascending order in all the three lists.

1333

1334 Here are two examples:

1335 Example 1:

1336 This is the current configuration:

1337 A = [0, 1]

1338 B = [2]

1339 C = []

1340

1341 This is the goal configuration:

1342 A = []

1343 B = []

1344 C = [0, 1, 2]

1345 Answer:

1346 I need to move 0 from list A to list C.

1347 Step 1. Find the numbers to the right of 0 in list A. There is 1
1348 to the right of 0.

1349 Step 2. Find the numbers larger than 0 in list C. There are none.
I will move the numbers found in Step 1 and Step 2 to list B.

1350 Hence I will move 1 from list A to list B. Also numbers should
1351 be in ascending order in list B.
1352 Subgoal:
1353 A = [0]
1354 B = [1, 2]
1355 C = []
1356
1357 Example 2:
1358
1359 This is the current configuration:
1360 A = [1]
1361 B = [0]
1362 C = [2]
1363
1364 This is the goal configuration:
1365 A = []
1366 B = []
1367 C = [0, 1, 2]
1368
1369 Answer:
1370 I need to move 0 from list B to list C.
1371 Step 1. Find the numbers to the right of 0 in list B. There are
1372 none.
1373 Step 2. Find the numbers larger than 0 in list C. There is 2
1374 which is larger than 0.
1375 I will move the numbers found in Step 1 and Step 2 to list A.
1376 Hence, I will move 2 from list C to list A. Also numbers should
1377 be in ascending order in list A.
1378 Subgoal:
1379 A = [1, 2]
1380 B = [0]
1381 C = []
1382
1383 Here is the task:
1384
1385 This is the current configuration:
1386 A = [0, 1, 2]
1387 B = []
1388 C = []
1389
1390 This is the goal configuration:
1391 A = []
1392 B = []
1393 C = [0, 1, 2]
1394
1395 Answer:
1396
1397 A.7.2 ACTOR
1398
1399 The Actor was prompted to propose $B = 2$ distinct actions. In some instances, the Actor failed
1400 to propose two distinct actions. In those cases, we iteratively scaled the temperature by a factor of
1401 0.1. This was done for a maximum of 10 attempts or until two distinct actions were produced. If the
1402 Actor was not able to propose two distinct actions even after 10 attempts, we then used only a single
1403 action. Please note that although the in-context examples involve both actions and resulting states,
the Actor is only responsible for proposing actions. The specific prompt and in-context examples
for the ToH task are shown below:
Consider the following puzzle problem:
Problem description:
- There are three lists labeled A, B, and C.

1404 - There is a set of numbers distributed among those three lists.
1405 - You can only move numbers from the rightmost end of one list to
1406 the rightmost end of another list.
1407 Rule #1: You can only move a number if it is at the rightmost end
1408 of its current list.
1409 Rule #2: You can only move a number to the rightmost end of a
1410 list if it is larger than the other numbers in that list.
1411 A move is valid if it satisfies both Rule #1 and Rule #2.
1412 A move is invalid if it violates either Rule #1 or Rule #2.
1413 Goal: The goal is to end up in the goal configuration using
1414 minimum number of moves.
1415 Here are two examples:
1416
1417 Example 1:
1418 This is the starting configuration:
1419 A = [0, 1]
1420 B = [2]
1421 C = []
1422 This is the goal configuration:
1423 A = []
1424 B = []
1425 C = [0, 1, 2]
1426 Here is the sequence of minimum number of moves to reach the goal
1427 configuration from the starting configuration:
1428 Move 2 from B to C.
1429 A = [0, 1]
1430 B = []
1431 C = [2]
1432 Move 1 from A to B.
1433 A = [0]
1434 B = [1]
1435 C = [2]
1436 Move 2 from C to B.
1437 A = [0]
1438 B = [1, 2]
1439 C = []
1440 Move 0 from A to C.
1441 A = []
1442 B = [1, 2]
1443 C = [0]
1444 Move 2 from B to A.
1445 A = [2]
1446 B = [1]
1447 C = [0]
1448 Move 1 from B to C.
1449 A = [2]
1450 B = []
1451 C = [0, 1]
1452 Move 2 from A to C.
1453 A = []
1454 B = []
1455 C = [0, 1, 2]
1456
1457

1458 Example 2:
1459 This is the starting configuration:
1460 A = [1]
1461 B = [0]
1462 C = [2]
1463 This is the goal configuration:
1464 A = []
1465 B = []
1466 C = [0, 1, 2]
1467 Here is the sequence of minimum number of moves to reach the goal
1468 configuration from the starting configuration:
1469
1470 Move 2 from C to A.
1471 A = [1, 2]
1472 B = [0]
1473 C = []
1474 Move 0 from B to C.
1475 A = [1, 2]
1476 B = []
1477 C = [0]
1478 Move 2 from A to B.
1479 A = [1]
1480 B = [2]
1481 C = [0]
1482 Move 1 from A to C.
1483 A = []
1484 B = [2]
1485 C = [0, 1]
1486 Move 2 from B to C.
1487 A = []
1488 B = []
1489 C = [0, 1, 2]
1490
1491 Here is the task:
1492 This is the starting configuration:
1493 A = [0, 1, 2]
1494 B = []
1495 C = []
1496 This is the goal configuration:
1497 A = [0]
1498 B = [1, 2]
1499 C = []
1500
1501 Give me only two different valid next moves possible from the
1502 starting configuration that would help in reaching the goal
1503 configuration using as few moves as possible.
1504 Your answer should be in the format as below:
1505 1. Move <N> from <src> to <trg>.
1506
1507 **A.7.3 MONITOR**
1508 The Monitor was prompted with chain-of-thought reasoning in which each of the rules of the task
1509 were checked before determining action validity. We stored the actions deemed valid by the Monitor
1510 in a separate buffer, and we terminated the action proposal loop (Algorithm 1) when there were two
1511 distinct actions in this buffer, or exceeded a maximum of 10 interactions with the Monitor. After

1512 termination of the action proposal loop, if the buffer didn't contain two distinct actions, we used the
1513 only action in the buffer. If the buffer was empty, we used the action(s) proposed by the Actor at
1514 the last attempt. The following text was used as a prompt and in-context examples in the ToH task:

1515 Consider the following puzzle problem:
1516

1517 Problem description:

- 1518 - There are three lists labeled A, B, and C.
- 1519 - There is a set of numbers distributed among those three lists.
- 1520 - You can only move numbers from the rightmost end of one list to
1521 the rightmost end of another list.

1522 Rule #1: You can only move a number if it is at the rightmost end
1523 of its current list.

1524 Rule #2: You can only move a number to the rightmost end of a
1525 list if it is larger than the other numbers in that list.

1526 A move is valid if it satisfies both Rule #1 and Rule #2.

1527 A move is invalid if it violates either Rule #1 or Rule #2.

1528 Goal: The goal is to check if the proposed move satisfies or
1529 violates Rule #1 and Rule #2 and based on that if it is a valid
1530 or invalid move.

1531 Here are two examples:

1532 Example 1:

1533 This is the initial configuration:

1534 A = []
1535 B = [1]
1536 C = [0, 2]

1537 Proposed move:

1538 Move 0 from C to B.

1539 Answer:

1540 First check whether the move satisfies or violates Rule #1. Index
1541 of 0 in list C is 0. Length of list C is 2. The difference in
1542 length of list C and index of 0 in list C is 2, which is not equal
1543 to 1. Hence 0 is not at the rightmost end of list C, and the move
1544 violates Rule #1.

1545 Next check whether the move satisfies or violates Rule #2. For
1546 that compute the maximum of list B, to which 0 is moved. Maximum
1547 of list B is 1. 0 is not larger than 1. Hence the move violates
1548 Rule #2.

1549 Since the Move 0 from list C to list B violates both Rule #1 and
1550 Rule #2, it is invalid.

1551 Example 2:

1552 This is the initial configuration:

1553 A = []
1554 B = [1]
1555 C = [0, 2]

1556 Proposed move:

1557 Move 2 from C to B.

1558 Answer:

1559 First check whether the move satisfies or violates Rule #1. Index
1560 of 2 in list C is 1. Length of list C is 2. The difference in
1561 length of list C and index of 2 in list C is 1. Hence 2 is at the
1562 rightmost end of list C, and the move satisfies Rule #1.

1566 Next check whether the move satisfies or violates Rule #2. For
1567 that compute the maximum of list B, to which 2 is moved. Maximum
1568 of list B is 1. 2 is larger than 1. Hence the move satisfies
1569 Rule #2.
1570 Since the Move 2 from list C to list B satisfies both Rule #1 and
1571 Rule #2, it is valid.

1572

1573 Here is the task:

1574

1575 This is the initial configuration:

1576

A = []

1577

B = [0, 1]

1578

C = [2]

1579

1580 Proposed move:

1581

Move 1 from B to A.

1582

Answer:

1583

1584 A.7.4 PREDICTOR

1585

1586 The Predictor was prompted to predict the next state, given the current state and the proposed
1587 action. The following text was used as a prompt and in-context examples in the ToH task:

1588

Consider the following puzzle problem:

1589

Problem description:

1590

- There are three lists labeled A, B, and C.

1591

- There is a set of numbers distributed among those three lists.

1592

- You can only move numbers from the rightmost end of one list to
1593 the rightmost end of another list.

1594

Rule #1: You can only move a number if it is at the rightmost end
1595 of its current list.

1596

Rule #2: You can only move a number to the rightmost end of a
1597 list if it is larger than the other numbers in that list.

1598

Goal: The goal is to predict the configuration of the
1599 three lists, if the proposed move is applied to the current
1600 configuration.

1601

1602 Here are two examples:

1603

1604 Example 1:

1605

This is the current configuration:

1606

A = []

1607

B = [1]

1608

C = [0, 2]

1609

1609 Proposed move:

1610

Move 2 from list C to list B.

1611

1612 Answer:

1613

A = []

1614

B = [1, 2]

1615

C = [0]

1616

1616 Example 2:

1617

1618 This is the current configuration:

1619

A = []

1620 B = [1]
1621 C = [0, 2]
1622 Proposed move:
1623 Move 1 from list B to list A.
1624
1625 Answer:
1626 A = [1]
1627 B = []
1628 C = [0, 2]
1629
1630 Here is the task:
1631 This is the current configuration:
1632 A = []
1633 B = [0, 1]
1634 C = [2]
1635 Proposed move:
1636 Move 1 from list B to list A.
1637
1638 Answer:
1639
1640 A.7.5 EVALUATOR
1641
1642 For the Evaluator, in the ToH task, GPT-4 was prompted to generate a heuristic function that could
1643 be used to estimate the distance between the current state and the goal state. The description of this
1644 heuristic function was then included in the Evaluator's prompt. In the graph traversal tasks, we
1645 used only in-context examples, with no heuristic function prompt.
1646
1647 The following text shows prompt and in-context examples used in the ToH task. The first User
1648 message prompts GPT-4 to generate a heuristic function. After GPT-4's response, another User
1649 message prompts the Evaluator to use this function:
1650
1651 USER: Consider the following puzzle problem:
1652
1653 Problem description:
1654 - There are three lists labeled A, B, and C.
1655 - There is a set of numbers distributed among those three lists.
1656 - You can only move numbers from the rightmost end of one list to
1657 the rightmost end of another list.
1658 Rule #1: You can only move a number if it is at the rightmost end
1659 of its current list.
1660 Rule #2: You can only move a number to the rightmost end of a
1661 list if it is larger than the other numbers in that list.
1662 A move is valid if it satisfies both Rule #1 and Rule #2.
1663 A move is invalid if it violates either Rule #1 or Rule #2.
1664
1665 Goal: The goal is to predict the minimum number of valid
1666 moves required to reach the goal configuration from the current
1667 configuration.
1668
1669 Here are two examples:
1670
1671 Example 1:
1672 This is the current configuration:
1673 A = [0, 1, 2]
1674 B = []
1675 C = []

1674 This is the goal configuration:

1675 A = []

1676 B = []

1677 C = [0, 1, 2]

1678 Answer:

1679 The minimum number of valid moves required to reach the goal
1680 configuration from the current configuration is 7.

1681

1682

1683 Example 2:

1684 This is the current configuration:

1685 A = [1, 2]

1686 B = [0]

1687 C = []

1688 This is the goal configuration:

1689 A = []

1690 B = []

1691 C = [0, 1, 2]

1692 Answer:

1693 The minimum number of valid moves required to reach the goal
1694 configuration from the current configuration is 4.

1695

1696

1697 What heuristic function can be used to estimate the minimum number
1698 of valid moves required to reach the goal configuration from a
1699 given current configuration?

1700

1701 **GPT-4:** A suitable heuristic function for this problem is the "sum
1702 of the distances" heuristic. This heuristic estimates the minimum
1703 number of valid moves required to reach the goal configuration by
1704 calculating the sum of the distances each number needs to travel
1705 to reach its final position in the goal configuration.

1706 Here's how the heuristic function works:

1707 1. For each number in the current configuration, determine its
1708 current position (list and index) and its goal position (list and
1709 index) in the goal configuration.

1710 2. Calculate the distance between the current position and the
1711 goal position for each number. The distance can be calculated
1712 as the absolute difference between the indices of the current and
1713 goal positions, plus a penalty if the number needs to move to a
1714 different list.

1715 3. Sum the distances calculated in step 2 for all numbers.

1716 The heuristic function will return the sum of the distances, which
1717 is an estimate of the minimum number of valid moves required to
1718 reach the goal configuration from the current configuration.

1719 This heuristic is admissible because it never overestimates
1720 the cost of reaching the goal configuration. It considers the
1721 minimum number of moves required for each number to reach its goal
1722 position, without taking into account the constraints imposed by
1723 the rules of the puzzle. Therefore, the actual number of moves
1724 required to reach the goal configuration will always be greater
1725 than or equal to the heuristic value.

1726

1727

1728 USER: This is the current configuration:
1729 A = [0]
1730 B = []
1731 C = [1, 2]
1732 This is the goal configuration:
1733 A = [0]
1734 B = [1, 2]
1735 C = []
1736 Use the heuristic function to predict the minimum number of valid
1737 moves required to reach the goal configuration from the current
1738 configuration.
1739 Please provide your answer according to the heuristic function in
1740 the format as below:
1741 The minimum number of valid moves required to reach the goal
1742 configuration from the current configuration is <N>.
1743
1744 A.7.6 ORCHESTRATOR
1745
1746 The Orchestrator was prompted to determine whether the current subgoal was achieved by the
1747 current state. The following text was used as a prompt and in-context examples in the ToH task:
1748 Consider the following puzzle problem:
1749
1750 Problem description:
1751 - There are three lists labeled A, B, and C.
1752 - There is a set of numbers distributed among those three lists.
1753 - You can only move numbers from the rightmost end of one list to
1754 the rightmost end of another list.
1755 Rule #1: You can only move a number if it is at the rightmost end
1756 of its current list.
1757 Rule #2: You can only move a number to the rightmost end of a
1758 list if it is larger than the other numbers in that list.
1759
1760 Goal: The goal is to predict whether the current configuration
1761 matches the goal configuration or not.
1762 Here are two examples:
1763 Example 1:
1764 This is the current configuration:
1765 A = []
1766 B = []
1767 C = [0, 1, 2]
1768 This is the goal configuration:
1769 A = []
1770 B = []
1771 C = [0, 1, 2]
1772 Answer: The current configuration matches the goal configuration.
1773 Hence yes.
1774 Example 2:
1775 This is the current configuration:
1776 A = [0, 1]
1777 B = [2]
1778 C = []
1779
1780 This is the goal configuration:
1781 A = []

1782 B = []
1783 C = [0, 1, 2]
1784
1785 Answer: The current configuration doesn't match the goal
1786 configuration. Hence no.
1787
1788 Here is the task:
1789 This is the current configuration:
1790 A = []
1791 B = [0, 1, 2]
1792 C = []
1793 This is the goal configuration:
1794 A = []
1795 B = []
1796 C = [0, 1, 2]
1797
1798 Answer:
1799
1800 **A.7.7 ZERO-SHOT PROMPT**
1801 An example prompt for the GPT-4 zero-shot baseline is shown below:
1802
1803 Consider the following puzzle problem:
1804 Problem description:
1805 - There are three lists labeled A, B, and C.
1806 - There is a set of numbers distributed among those three lists.
1807 - You can only move numbers from the rightmost end of one list to
1808 the rightmost end of another list.
1809 Rule #1: You can only move a number if it is at the rightmost end
1810 of its current list.
1811 Rule #2: You can only move a number to the rightmost end of a
1812 list if it is larger than the other numbers in that list.
1813 A move is valid if it satisfies both Rule #1 and Rule #2.
1814 A move is invalid if it violates either Rule #1 or Rule #2.
1815 Goal: The goal is to end up in the configuration where all
1816 numbers are in list C, in ascending order using minimum number
1817 of moves.
1818 This is the starting configuration:
1819 A = [0, 1, 2]
1820 B = []
1821 C = []
1822 This is the goal configuration:
1823 A = []
1824 B = []
1825 C = [0,1,2]
1826 Give me the sequence of moves to solve the puzzle from the
1827 starting configuration, updating the lists after each move.
1828 Please try to use as few moves as possible, and make sure to
1829 follow the rules listed above. Please limit your answer to a
1830 maximum of 10 steps.
1831 Please format your answer as below:
1832 Step 1. Move <N> from <src> to <tgt>.
1833 A = []
1834 B = []
1835 C = []

1836 A.7.8 ICL PROMPT

1837
1838 An example prompt for the GPT-4 ICL baseline is shown below:

1839 Consider the following puzzle problem:

1840
1841 Problem description:

- 1842 - There are three lists labeled A, B, and C.
- 1843 - There is a set of numbers distributed among those three lists.
- 1844 - You can only move numbers from the rightmost end of one list to the rightmost end of another list.

1845 Rule #1: You can only move a number if it is at the rightmost end
1846 of its current list.

1847 Rule #2: You can only move a number to the rightmost end of a
1848 list if it is larger than the other numbers in that list.

1849 A move is valid if it satisfies both Rule #1 and Rule #2.

1850 A move is invalid if it violates either Rule #1 or Rule #2.

1851 Goal: The goal is to end up in the configuration where all
1852 numbers are in list C, in ascending order using minimum number
1853 of moves.

1854 Here are two examples:

1855
1856 Example 1:

1857 This is the starting configuration:

1858 A = [0, 1]

1859 B = [2]

1860 C = []

1861 This is the goal configuration:

1862 A = []

1863 B = []

1864 C = [0, 1, 2]

1865 Here is the sequence of minimum number of moves to reach the goal
1866 configuration from the starting configuration:

1867 Move 2 from B to C.

1868 A = [0, 1]

1869 B = []

1870 C = [2]

1871

1872 Move 1 from A to B.

1873 A = [0]

1874 B = [1]

1875 C = [2]

1876

1877 Move 2 from C to B.

1878 A = [0]

1879 B = [1, 2]

1880 C = []

1881

1882 Move 0 from A to C.

1883 A = [1]

1884 B = [1, 2]

1885 C = [0]

1886

1887 Move 2 from B to A.

1888 A = [2]

1889 B = [1]

1890 C = [0]

1891

1892 Move 1 from B to C.

1893 A = [2]

1890 B = []
1891 C = [0, 1]
1892
1893 Move 2 from A to C.
1894 A = []
1895 B = []
1896 C = [0, 1, 2]
1897
1898 Example 2:
1899 This is the starting configuration:
1900 A = [1]
1901 B = [0]
1902 C = [2]
1903 This is the goal configuration:
1904 A = []
1905 B = []
1906 C = [0, 1, 2]
1907 Here is the sequence of minimum number of moves to reach the goal
1908 configuration from the starting configuration:
1909 Move 2 from C to A.
1910 A = [1, 2]
1911 B = [0]
1912 C = []
1913 Move 0 from B to C.
1914 A = [1, 2]
1915 B = []
1916 C = [0]
1917 Move 2 from A to B.
1918 A = [1]
1919 B = [2]
1920 C = [0]
1921 Move 1 from A to C.
1922 A = []
1923 B = [2]
1924 C = [0, 1]
1925 Move 2 from B to C.
1926 A = []
1927 B = []
1928 C = [0, 1, 2]
1929
1930
1931 Here is the task:
1932 This is the starting configuration:
1933 A = [0, 1, 2]
1934 B = []
1935 C = []
1936 This is the goal configuration:
1937 A = []
1938 B = []
1939 C = [0,1,2]
1940
1941 Give me the sequence of moves to solve the puzzle from the
1942 starting configuration, updating the lists after each move.
1943 Please try to use as few moves as possible, and make sure to

1944 follow the rules listed above. Please limit your answer to a
1945 maximum of 10 steps.
1946
1947 Please format your answer as below:
1948 Step 1. Move <N> from <src> to <tgt>.
1949 A = []
1950 B = []
1951 C = []
1952
1953 **A.7.9 CoT ICL PROMPT**
1954 An example prompt for the GPT-4 CoT ICL baseline is shown below:
1955 Consider the following puzzle problem:
1956
1957 Problem description:
1958 - There are three lists labeled A, B, and C.
1959 - There is a set of numbers distributed among those three lists.
1960 - You can only move numbers from the rightmost end of one list to
1961 the rightmost end of another list.
1962 Rule #1: You can only move a number if it is at the rightmost end
1963 of its current list.
1964 Rule #2: You can only move a number to the rightmost end of a
1965 list if it is larger than the other numbers in that list.
1966 A move is valid if it satisfies both Rule #1 and Rule #2.
1967 A move is invalid if it violates either Rule #1 or Rule #2.
1968
1969 Goal: The goal is to end up in the configuration where all
1970 numbers are in list C, in ascending order using minimum number
1971 of moves.
1972 Here are two examples:
1973
1974 Example 1:
1975 This is the starting configuration:
1976 A = [0, 1]
1977 B = [2]
1978 C = []
1979 This is the goal configuration:
1980 A = []
1981 B = []
1982 C = [0, 1, 2]
1983
1984 Here is the sequence of minimum number of moves along with
1985 reasoning for each move to reach the goal configuration from the
1986 starting configuration:
1987
1988 I need to move 0 from A to C. But before that I need to move the
1989 number present to the right of 0, which is 1 to B. There is a
1990 number larger than 1 already present in list B. Hence I first need
1991 to move 2 from B to C.
1992 Move 2 from B to C.
1993 A = [0, 1]
1994 B = []
1995 C = [2]
1996
1997 I need to move 0 from A to C. But before that I need to move the
1998 number present to the right of 0, which is 1 to B.
1999 Move 1 from A to B.
2000 A = [0]
2001 B = [1]
2002 C = [2]

1998 I need to move 0 from A to C. There is a number larger than 0
1999 already present in list C. Hence I first need to move 2 from C
2000 to B.
2001 Move 2 from C to B.
2002 A = [0]
2003 B = [1, 2]
2004 C = []
2005 There is no number to the right of 0 in A, and there is no number
2006 larger than 0 in C. Hence, I can move 0 from A to C.
2007 Move 0 from A to C.
2008 A = []
2009 B = [1, 2]
2010 C = [0]
2011 I need to move 1 from B to C. But before that I need to move the
2012 number present to the right of 1, which is 2 to A.
2013 Move 2 from B to A.
2014 A = [2]
2015 B = [1]
2016 C = [0]
2017 There is no number to the right of 1 in B, and there is no number
2018 larger than 1 in C. Hence, I can move 1 from B to C.
2019 Move 1 from B to C.
2020 A = [2]
2021 B = []
2022 C = [0, 1]
2023 There is no number to the right of 2 in A, and there is no number
2024 larger than 2 in C. Hence, I can move 2 from A to C.
2025 Move 2 from A to C.
2026 A = []
2027 B = []
2028 C = [0, 1, 2]
2029 Example 2:
2030 This is the starting configuration:
2031 A = [1]
2032 B = [0]
2033 C = [2]
2034 This is the goal configuration:
2035 A = []
2036 B = []
2037 C = [0, 1, 2]
2038 Here is the sequence of minimum number of moves along with
2039 reasoning for each move to reach the goal configuration from the
2040 starting configuration:
2041 I need to move 0 from B to C. There is a number larger than 0
2042 already present in list C. Hence I first need to move 2 from C
2043 to A.
2044 Move 2 from C to A.
2045 A = [1, 2]
2046 B = [0]
2047 C = []
2048 There is no number to the right of 0 in B, and there is no number
2049 larger than 0 in C. Hence, I can move 0 from B to C.
2050 Move 0 from B to C.
2051 A = [1, 2]

2052 B = []
2053 C = [0]
2054
2055 I need to move 1 from A to C. But before that I need to move the
2056 number present to the right of 1, which is 2 to B.
2057 Move 2 from A to B.
2058 A = [1]
2059 B = [2]
2060 C = [0]
2061
2062 There is no number to the right of 1 in A, and there is no number
2063 larger than 1 in C. Hence, I can move 1 from A to C.
2064 Move 1 from A to C.
2065 A = []
2066 B = [2]
2067 C = [0, 1]
2068
2069 There is no number to the right of 2 in B, and there is no number
2070 larger than 2 in C. Hence, I can move 2 from B to C.
2071 Move 2 from B to C.
2072 A = []
2073 B = []
2074 C = [0, 1, 2]
2075
2076 Here is the task:
2077
2078 This is the starting configuration:
2079 A = [0, 1, 2]
2080 B = []
2081 C = []
2082
2083 This is the goal configuration:
2084 A = []
2085 B = []
2086 C = [0, 1, 2]
2087
2088 Give me the sequence of moves to solve the puzzle from the
2089 starting configuration, updating the lists after each move.
2090 Please try to use as few moves as possible, and make sure to
2091 follow the rules listed above. Please limit your answer to a
2092 maximum of 10 steps.
2093
2094 Please format your answer as below:
2095 Step 1. Move <N> from <src> to <tgt>.
2096 A = []
2097 B = []
2098 C = []

2099 A.8 COMPUTATIONAL COST: THINKING FAST AND (VERY) SLOW

2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500

Tables 16 and 17 show various cost metrics for both MAP vs. baseline models, and compares these cost metrics with performance. To address the significant computational cost of MAP, we also developed a more efficient version that cached and re-used results for redundant prompts. This was done for all modules except the Actor and the TaskDecomposer. This version of the model was significantly more efficient, while retaining the same level of performance.

2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159

Table 16: Average per-problem computational cost (\pm the standard error of the mean) on ToH with 3 disks. 5 runs were done for MAP.

Model	Num. calls	Num. input tokens	Num. output tokens	% solved
GPT-4 ICL	1 ± 0.0	810.88 ± 0.1	190.38 ± 15.4	46
GPT-4 CoT	1 ± 0.0	1309.88 ± 0.1	422.42 ± 22.8	42
ToT	45.75 ± 0.6	26649.88 ± 454.0	7523.96 ± 322.4	6
MAP	148.6 ± 8.2	$109,090.025 \pm 6,567.2$	$14,543.57 \pm 844.6$	74 ± 3
MAP (efficient)	41.99 ± 3.6	47242.43 ± 3848.4	2766.34 ± 324.5	77 ± 3

Table 17: Average per-problem computational cost (\pm the standard error of the mean) incurred by each module of MAP on ToH with 3 disks.

Module	Num. calls	Num. input tokens	Num. output tokens
Actor	24.68 ± 1.5	$38,274.62 \pm 2,796.8$	758.69 ± 54.1
Monitor	46.26 ± 2.8	$32,135.44 \pm 1,933.2$	$7,131.85 \pm 429.6$
Predictor	29.48 ± 1.6	$11,284.65 \pm 601.3$	477.48 ± 25.7
Evaluator	22.52 ± 1.2	$17,054.15 \pm 917.6$	$5,730.11 \pm 334.3$
Orchestrator	24.67 ± 1.2	$9,553.3 \pm 466.7$	316.0 ± 16.0
Task Decomposer	1 ± 0.0	787.88 ± 0.0	129.44 ± 2.0