MEANS-ENDS ANALYSIS HELPS LLMS PLAN IN CONTINUOUS-STATE DOMAINS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM planning research has traditionally focused on discrete-state domains, where actions form a finite, enumerable set. Existing LLM planning frameworks largely assume actions are enumerable, modeling planning as a search process over candidate actions. However, in continuous-state domains, where actions form a continuum and cannot be enumerated, this assumption breaks down. We find a remedy in Newell and Simon's Means-Ends Analysis (MEA), a general problem-solving technique rooted in heuristics from human psychology. We develop MEDAKA, an MEA-inspired structured prompting framework that encourages recursive goal decomposition, algebraic derivation, and per-step state tracking. To evaluate native LLM planning abilities in continuous-state domains, we develop LIQUIDWORLD, a systematic testbed on liquid mixing. On LIQUIDWORLD, MEDAKA can improve performance across model families, suggesting that classical means-ends analysis elicits LLM-native reasoning abilities that are helpful for continuous-state planning.

1 INTRODUCTION

Large language models (LLMs) have shown remarkable abilities across diverse domains, including commonsense reasoning (29), mathematics (27), and code generation (5). These successes have prompted interest in applying LLMs to more complex cognitive tasks, such as planning (25).

Existing methods for LLM planning are primarily developed for tasks with discrete states and enumerable actions (25). These methods mainly build on an enumerate-and-rank assumption, where they model planning as searching actions over a discrete, enumerable space, and use techniques like state-aware backtracking (22) or external tree structures (28; 13) to aid the search process. However, in many scenarios such as biochemical experimentation, states are continuous, and actions are no longer enumerable. Such a distinction requires LLM capabilities beyond search: while actions cannot be enumerated, they can often be derived algebraically, given a faithful representation of the current state and its deviation from the goal state. In this case, planning natively in such domains requires an LLM to (1) decompose goals into tractable subgoals, (2) faithfully track states across steps, and (3) perform accurate derivations at each step.

An answer to this challenge comes from classical Means-Ends Analysis (MEA) (20), a general problem-solving technique rooted in human psychology. MEA is structurally well-suited to continuous action spaces because it treats undetermined parameters not as candidates to be enumerated and ranked, but as gaps to be closed by reasoning about the constraints relevant to the current subgoal. Concretely, at each step, MEA (1) identifies a gap between the current state and subgoal, (2) proposes an operator (i.e., an action with potentially undetermined parameters) that would close it, (3) recursively resolves any precondition or parameter the operator leaves undetermined, treating each as a new subgoal, and (4) terminates when every parameter is determined. Because the gap is evaluated at each step, MEA also encourages state tracking: the state must be updated and re-examined after every operator application before the next subgoal is identified. In MEA, an undetermined parameter is a gap to be closed by accurately deriving a small system of equations linking the operator's parameters to a local subgoal, rather than requiring the LLM to reason about all constraints simultaneously.

Inspired by MEA, we develop MEDAKA, a structured prompting framework for LLM planning in continuous-state domains that encourages recursive goal decomposition and per-step state tracking. To isolate and evaluate LLM native planning abilities in continuous-state domains, we develop LIQUIDWORLD, a systematic testbed grounded in fluid mixing, a primitive operation to which

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

numerous standard laboratory protocols (e.g., serial dilution, growth medium preparation, and buffer preparation) can be reduced. LIQUIDWORLD problems cover diverse structural characteristics, isolating key reasoning capabilities for planning in continuous-state domains (Table 1): nonlinearity of the mixing constraints, coupling of action parameters, and context length (number of goals and distractor vessels). On LIQUIDWORLD, MEDAKA can yield performance improvement over existing prompting frameworks for LLM planning across both closed-source (Claude (3), GPT (17; 23), Gemini (11)) and open-source model families (Llama (12), Gemma (24)), suggesting that the enumerate-and-rank assumption shared by existing methods is mismatched with continuous-state planning, and that classical means-ends analysis offers a better structural fit. We make our code available at `https://anonymous.4open.science/r/medaka_code-FFD0`.

Our contributions are as follows:

- We identify continuous-state planning as a distinct challenge for LLM planning and characterize the enumerate-and-rank assumption as an architectural mismatch with this setting.
- We introduce LIQUIDWORLD, a systematic testbed for isolating and evaluating LLM native planning in continuous domains, covering diverse problem structures in fluid mixing.
- We propose MEDAKA, a structured prompting framework for LLM planning grounded in Means-Ends Analysis that encourages recursive goal decomposition and per-step state tracking, that can better elicit LLM native planning abilities on LIQUIDWORLD.

2 RELATED WORK

LLM planning frameworks. Current LLM planning frameworks mainly elicit LLM native planning behavior via prompting. Chain-of-Thought (CoT) (27) prompting instructs the model to articulate intermediate reasoning steps before committing to an action and is the default planning framework for modern LLMs. Tree-of-Thoughts (ToT) (28) extends CoT with explicit branching and value-guided search over partial solutions. RAP (13) casts the LLM as a world model and runs Monte Carlo Tree Search over its predicted transitions. Algorithm of Thoughts (AoT) (21) embeds a simulated search trace inside the prompt, while AoT+ (22) augments this with periodic state memoization to mitigate state hallucination. Another line of research in LLM-Modulo (18) frameworks equips LLMs with external solvers or verifiers to offload parameter derivation. For example, LLM3 (26) uses a motion planner to validate LLM-proposed parameterized actions, and PRoC3S (6) uses the LLM to output a constraint-satisfaction program whose parameters are filled in by a constraint-satisfaction solver. The goal of our work is to isolate and elicit native LLM planning abilities in continuous-state domains, not to build a deployable planner. Hence, we do not compare with LLM-Modulo methods, since their external solvers perform the parameter derivations that we aim to measure natively in LLMs.

LLMs in the laboratory. A growing body of research employs LLMs as agents in scientific laboratories. Coscientist (4) equips GPT-4 with tools in internet search, code execution, and robotic instrumentation to autonomously design and run chemistry experiments. ChemCrow (19) augments GPT-4 with 18 expert-designed tools spanning organic synthesis, drug discovery, and materials design, demonstrating the ability to automate the synthesis of small molecules. These systems show that continuous quantities arise naturally when LLMs operate in laboratory settings. They rely on external tools and agentic orchestration, whereas we study the planning ability of the LLM itself.

Classical planning and Means-Ends Analysis. Classical planning provides the formal vocabulary (e.g., operators, preconditions, decomposition) that our framework adapts to LLM-based reasoning. The General Problem Solver (20) introduces Means-Ends Analysis as a recursive difference-reduction procedure. STRIPS (8) formalized symbolic operators with preconditions and effects, and PDDL (1) standardized this representation as a planning language. PDDL 2.1 (9) extended the language with numeric fluents, and Metric-FF (16) provided a heuristic solver for the resulting numeric planning problems. HTN (7) planning introduced hierarchical task networks that recursively decompose compound tasks into primitives. Together, these works supply the symbolic substrate that MEDAKA imports into LLM prompting.

3 METHODS

3.1 PROBLEM FORMULATION

From discrete to continuous-state planning. LLM planning has been studied predominantly in *discrete-state* domains such as BlocksWorld (25), where state transitions are symbolic, not algebraic. LLM planning frameworks, e.g., Chain-of-Thought (27), Tree-of-Thoughts (28), RAP (13) and AoT+ (22) have advanced the state of the art on this classic benchmark, but their core mechanisms (tree search, beam search, in-context simulated search) presuppose a finite action space. However, many planning domains in biochemical laboratories, e.g., liquid handling, involve states governed by continuous numerical quantities. Whether LLM planning methods designed for discrete action selection transfer to such continuous-state domains has not been formally investigated.

Planning formulation. Let $\mathcal{S}$ be a state space, $\mathcal{A}$ an action space, and $\mathcal{T}\colon \mathcal{S} \times \mathcal{A} \rightharpoonup \mathcal{S}$ a transition function. A planning problem is a tuple $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, s_0, g)$, where $s_0 \in \mathcal{S}$ is the initial state and $g\colon \mathcal{S} \to \{0, 1\}$ is a goal predicate that identifies goal states (10). We denote $\mathcal{A}(s) = \{a \in \mathcal{A} : \mathcal{T}(s, a) \text{ is defined}\}$ as the set of actions applicable in s. A *plan* is a sequence $(a_1, \ldots, a_T)$ such that, setting $s_t = \mathcal{T}(s_{t-1}, a_t)$ for each t, every action is applicable ($a_t \in \mathcal{A}(s_{t-1})$) and the final state satisfies the goal ($g(s_T) = 1$). When the planner is an LLM π, each generated step implicitly performs two sub-tasks:

- **Action selection**: choosing $a_t \in \mathcal{A}(s_{t-1})$ to reduce the gap between s_{t-1} and g;
- **State prediction**: computing $\hat{s}_t \approx \mathcal{T}(s_{t-1}, a_t)$, i.e. simulating the transition in natural language.

This decomposition is central to our analysis: existing methods differ primarily in how they handle action selection, while leaving state prediction to the LLM's unstructured generation. In discrete domains $\mathcal{A}(s)$ is finite for every s, and planning reduces to a search over a finite graph. In continuous-state domains, $\mathcal{A}(s)$ is a continuum, thus enumerating candidate actions is no longer possible; a valid plan may require parameter values that no finite discretisation of $\mathcal{A}$ contains. The planner must *derive* the correct v from the current state and goal, not select it from a finite list. On the state-prediction side, once a_t is fixed, computing $\hat{s}_t = \mathcal{T}(s_{t-1}, a_t)$ becomes a numerical task that existing LLM planning frameworks leave unstructured.

We study continuous-state LLM planning along two axes:

- **LiquidWorld** (§3.2), a systematic testbed for LLM continuous-state planning, grounded in fluid mixing tasks with varying structural characteristics.
- **MEDAKA** (§3.3), a structured prompting framework for LLM continuous-state planning, inspired by Means-Ends Analysis (20). MEDAKA recursively decomposes each goal until all numerical parameters are derived *before* the action is committed.

3.2 LIQUIDWORLD: A SYSTEMATIC TESTBED FOR CONTINUOUS-STATE PLANNING

To study continuous-state LLM planning under controlled conditions, we introduce LIQUIDWORLD, a systematic testbed where the state space, action space, and goal structure are held fixed across instances, while varying structural characteristics such as the number of goal states, nonlinearity of the liquid-mixing constraints, and the presence of distractors add to the difficulty of each instance (Table 1). LIQUIDWORLD is built on fluid mixing, which is the foundation for numerous procedures in biochemical laboratories (e.g., serial dilution, growth medium preparation, and buffer preparation).

Domain description. A LIQUIDWORLD state s consists of n vessels $\{V_1, \ldots, V_n\}$. Each vessel V_i is characterized by a current volume $vol_i \in \mathbb{R}_{\geq 0}$ and a concentration vector $\mathbf{c}_i = (c_i^1, \ldots, c_i^K)$ over K constituents, with $\sum_k c_i^k = 1 (c_i^k \geq 0)$ whenever $vol_i > 0$. A goal g specifies target volumes and concentrations for one or more designated vessels. A vessel has a *pure* composition if $c_i^k = 1$ for some k, and a *mixture* otherwise. A "pouring" action in the domain is denoted as POUR(v, V_i, V_j), which transfers solutions of volume v from V_i to V_j ($0 < v \leq vol_i$). Pouring obeys *mass conservation*: all constituents in the source transfer in proportion to their current concentrations,

State: V1 (capacity 999L): 10 L L1; V2 (capacity 999L): 10 L L2; V3 (capacity 999L): 10 L L3; V4 (capacity 999L): empty

Goal: V4: 5 L, 40% L1, 40% L2, 20% L3

Figure 1: A LIQUIDWORLD instance in the structured text format presented to the LLM. The grammar of this format is given in Figure 2.

```
      <state> ::= <vessel_decl> (";" <vessel_decl>)*
<vessel_decl> ::= NAME "(" "capacity" FLOAT "L" ")" ":" <contents>
   <contents> ::= "empty"
                | FLOAT "L" LID                    // pure vessel
                | FLOAT "L" "," <pct> ("," <pct>)*     // mixture
        <pct> ::= FLOAT "%" LID

       <goal> ::= <goal_clause> (";" <goal_clause>)*
<goal_clause> ::= NAME ":" FLOAT "L" "," <pct> ("," <pct>)*

     <action> ::= "Pour" VOL "L" "from" NAME "to" NAME
          VOL ::= FLOAT | INT "/" INT
```

Figure 2: Grammar of the LIQUIDWORLD problem specification language. NAME is a vessel identifier (e.g. V1), LID a liquid identifier (e.g. L2), and FLOAT/INT are numeric literals. The grammar is implicit in the regex-based parser; we present it here in Backus-Naur Form for clarity.

yielding the post-action state:

$$vol_i' = vol_i - v, \qquad vol_j' = vol_j + v, \qquad c_j^{k\prime} = \frac{vol_j \cdot c_j^k + v \cdot c_i^k}{vol_j + v} \;\; \forall\, k. \tag{1}$$

Problem specification. LIQUIDWORLD instances are stored in structured JSON format (per-vessel volume and composition dictionaries). An equivalent textual rendering is then presented to the LLM to generate plans. Figure 1 shows an example instance in structured text.

To measure reasoning capabilities key to planning in continuous-state domains, we develop problems with five structural characteristics (Table 1): (1) whether to allow direct pouring from source to target ("linear") or (2) require sources to always route through intermediate pools ("nonlinear"); (3) whether the underlying algebraic system has heavy coupling across different sources ("coupling"); (4) whether there is a large number of targets ("multi-goal"); and (5) whether there are excess vessels that are neither targets nor necessary for use ("distractors").

Each LIQUIDWORLD instance exhibits one or more such characteristics, which function as difficulty tags: (1) While pouring actions can be read out iteratively, one liquid at a time, when there is no coupling between the sources, algebraically solving a linear system is required for "coupling" problems. (2) "linear" problems are solvable under a single linear program (LP), which we use only to construct instances with verifiable ground truth, while "nonlinear" ones are more mathematically and structurally complex, since the compositions of the intermediate pools are unknown, resulting in bilinear goal constraints (to and from the pool). "nonlinear" instances are derived from the pooling problem (15), which is an NP-hard process optimization problem (2; 14). (3) Problems tagged with "multi-goal" and "distractors" have longer context length than those that are not.

For "linear"-tagged problems, we apply a parametric generator, whose input includes the grammar of the problem specification language (Figure 2) and the parameters for the start and goal states (number of source vessels, target vessels, and liquids). "nonlinear"-tagged problems also conform to the above grammar but are hand-designed to account for the constraints of intermediate pools.

Solvability verification. After problems are generated, we verify solvability in three stages before any instance enters the LIQUIDWORLD testbed: (1) A *volume check* ensures that the total available volume of each liquid across all sources meets the total demand across all goals. (2) A *concentration check* ensures that no target requires a liquid at a concentration exceeding its maximum concentration across all sources. (3) A *linear-program (LP) feasibility check* (for "linear" problems only) confirms that a non-negative sequence of POUR actions exists and serves as verifiable ground truth: the decision

Table 1: LIQUIDWORLD structure. Each problem is annotated with one or more structurally defined difficulty tags.

Tag	Criterion	#Probs
linear	The volumes to pour from each source into each target vessel to reach the goal can be obtained by solving a linear system.	136
nonlinear	Sources reach targets only through intermediate pools whose compositions are chosen by the plan; the pour volumes and intermediate compositions cannot be obtained from a linear system.	53
coupling	Liquids recur across sources ($\#$*source–liquid pairs* $-$ $\#$*liquids* ≥ 3), so the pour volumes must be solved as one coupled system.	106
multi-goal	≥ 5 target vessels.	76
distractor	≥ 2 vessels that are neither targets nor used by the reference plan.	28
Total		189

variables are the volumes poured from each source (to each target in the multi-target case), with equality constraints requiring each goal composition to be met exactly and inequality constraints capping the total volume drawn from each source. For multi-target problems, a joint LP is solved so that targets sharing sources are simultaneously satisfiable. Instances that fail any check are discarded and resampled. The Full LP formulations are given in Appendix A.5."nonlinear" problems are hand-solved to check feasibility, after passing the previous two checks.

Plan validator. Requiring no external solvers or LLM-based judges, the validator parses a candidate plan into a sequence of POUR actions, applies Eq. (1) step by step, and checks whether each goal vessel matches its target. Specifically, for every goal vessel, (i) the absolute volume of each liquid component must match its target within $\epsilon = 0.05\,\mathrm{L}$, and (ii) the total vessel volume must be within the same tolerance. Target volumes range from 1 to $5\,\mathrm{L}$, so this corresponds to 1–5% relative error. For "nonlinear" instances, it additionally checks that the liquid passed through the designated pool.

3.3 MEDAKA: MEANS-ENDS DECOMPOSITION FOR CONTINUOUS-STATE PLANNING

Section 3.1 argued that continuous-state planning requires deriving action parameters rather than selecting actions from an enumerable set. To facilitate planning in the continuous-state domain, we adopt a primitive from Means-Ends Analysis (MEA) (20), a general problem-solving approach rooted in human psychology that is built around parameter derivation instead of search.

MEA identifies the gap between the current state and the goal and proposes an operator that closes that gap. Here, an "operator" is an action but potentially with underspecified parameters. For example, an operator in LIQUIDWORLD can be "Pour to V3" but with the exact amount unknown. If the operator's preconditions are not satisfied, or parameters are not yet derived, a recursive subgoal is created to meet these preconditions or derive these parameters. While the original MEA formulation was symbolic, its logic naturally extends to numerical settings: a missing real-valued action parameter is a gap to close, and is algebraically derivable from the current state and goal.

We instantiate MEA as a structured prompting protocol called MEDAKA (Figure 3). MEDAKA realizes MEA through two mechanisms: *recursive goal decomposition* drives macro control flow, and *per-step state tracking* audits the traces to facilitate faithful derivations. The complete algorithmic logic of MEDAKA is presented in Algorithm 1 and Appendix B.1.

Recursive goal decomposition. At each planning step, MEDAKA instructs the LLM to (1) compute $D = \textsc{Diff}(s_{t-1}, g)$, the set of unsatisfied goal conditions, (2) propose an operator o that would reduce one of these gaps, and (3) test whether o is *applicable*, i.e., all of o's preconditions satisfied and all of its numerical parameters determined. If o is not applicable, then the LLM recursively sets up subgoals whose satisfaction makes o applicable. Recursion terminates when a fully parameterized, precondition-satisfying operator is reached, and only then does the LLM commit to an action a.

This scheme encourages every undetermined parameter to be made an explicit subgoal and fully specified before an action is committed, rather than guessed and incrementally amended. It places reasoning effort before commitment, reducing the propagation of errors into the generated plan.

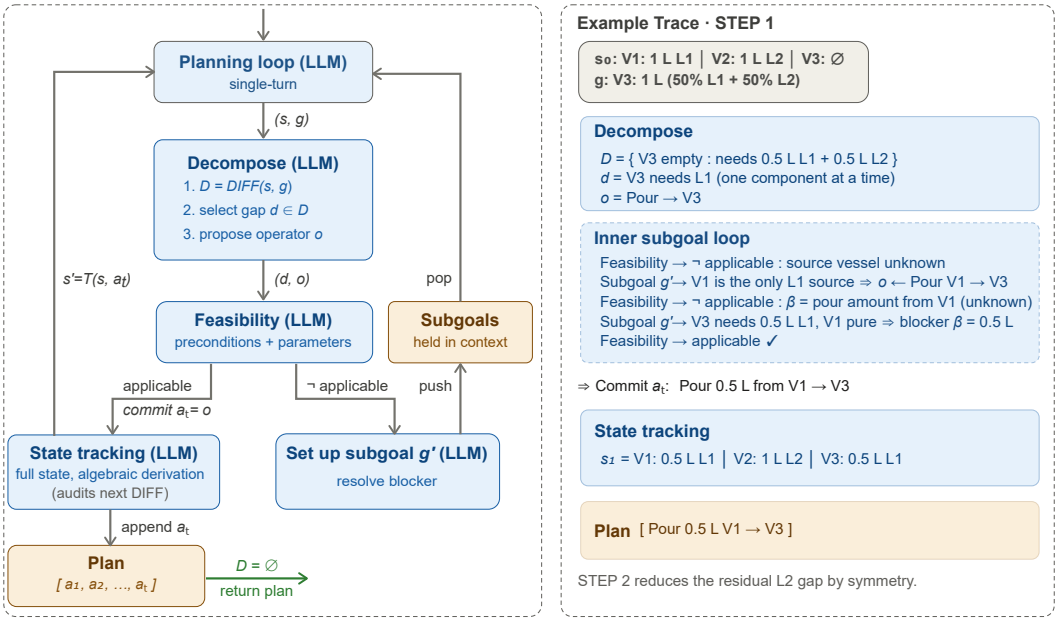

Figure 3: Overview of MEDAKA. **Left:** the LLM recursively decomposes the goal by identifying a gap d between the state s and goal g and proposes an operator o. A plan is returned when no gap remains. Feasibility tests whether o is applicable in s: when blocked, the LLM derives a subgoal g' that resolves the blocker algebraically and updates o, with deferred subgoals pushed onto a subgoal stack. Once applicable, an action $a_t = o$ is committed and the state tracker emits the full successor $s' = T(s, a_t)$. **Right:** An example trace through STEP 1 of a LIQUIDWORLD problem.

This marks a deviation from the workflow established by planning frameworks like AoT (21) and AoT+ (22), which place emphasis on backtracking when the plan goes awry.

Per-step state tracking. Per-step state tracking governs all numerical computation in MEDAKA, encouraging faithful algebraic derivations at every planning step. It instructs the LLM to evaluate the transition equations (Eq. 1) against the tracked state in both directions (Appendix B.1): backward, by setting up and solving equations for action parameters; forward, by deriving the complete successor state $s_t = \mathcal{T}(s_{t-1}, a_t)$ after every committed action a_t, listing the volume and composition of every vessel, not only those affected by the action. This contrasts with AoT+, which caches states only at heuristically chosen intervals during its in-context simulated search. In MEDAKA, the state output is instead a *computational* mechanism that constrains subsequent reasoning: because the next step's DIFF is computed from the recorded s_t, a hallucinated state either produces an inconsistent gap or surfaces as a contradiction in later steps.

4 EXPERIMENTS

4.1 SETUP

We compare MEDAKA against two prompting-based LLM planning methods: Chain-of-Thought (CoT) (27) and AoT+ (22). CoT is the standard sequential reasoning baseline that represents the default LLM reasoning behavior, while AoT+ is a recent state-of-the-art prompting framework for LLM planning that augments reasoning with in-context search and periodic state memoization. Both methods were originally developed and evaluated primarily in discrete symbolic planning domains. We do not compare with LLM-Modulo methods since external solvers confound native parameter derivation abilities key to planning. We evaluate across seven LLMs spanning both proprietary and open-weight families across different model sizes: `Claude-Sonnet-4.6`, `Gemini-3-Flash`, `GPT-5.4`, `GPT-4o`, `LLaMA-3.3-70B`, `Gemma4-4B`, and `Gemma4-2B`. Since our question is which reasoning structure helps continuous-state planning, we disable thinking mode for all models

so that their pretrained reasoning patterns do not override the tested prompting methods' intended behavior. For each (model, method, problem) triplet, we perform three repeated rollouts and measure `pass@1` (accuracy) and `pass@3`, where a pass is determined by the plan validator described in Section 3.2.

4.2 MAIN RESULTS

Table 2: `pass@1`(%) on LIQUIDWORLD grouped by difficulty tag, computed as the mean pass rate over all rollouts on problems with that tag, 3 rollouts per problem. A problem may carry one or more tags (Table 1). Shaded rows mark MEDAKA ("MED") and the performance of the best method per model is marked in bold. Appendix A.2 reports `pass@3`(%).

Tag	**Method**	Claude Sonnet 4.6	Gemini 3 Flash	GPT 5.4	GPT 4o	LLaMA 3.3 70B	Gemma4 4B	Gemma4 2B
linear	CoT	95.1	84.8	**91.9**	32.1	31.1	71.8	32.6
	AoT+	96.6	91.2	90.0	23.5	20.3	67.4	28.7
	MED	**97.3**	**94.1**	90.9	**45.1**	**50.5**	**79.9**	**37.7**
nonlinear	CoT	33.3	16.4	28.9	9.4	**15.1**	14.5	10.1
	AoT+	55.3	21.4	**49.1**	10.1	10.7	**15.1**	11.3
	MED	**87.4**	**50.9**	46.5	**15.7**	**15.1**	12.6	**11.9**
coupling	CoT	60.4	39.6	54.7	2.5	2.5	34.9	4.7
	AoT+	73.6	49.4	**62.9**	2.5	1.6	28.9	4.4
	MED	**90.3**	**68.9**	**62.9**	**10.7**	**14.2**	**38.1**	**10.4**
multi-goal	CoT	50.4	31.1	43.4	15.8	14.5	20.2	**14.9**
	AoT+	65.8	37.3	**56.6**	14.5	10.5	21.9	11.4
	MED	**89.5**	**60.5**	54.8	**16.7**	**18.4**	**24.6**	9.6
distractor	CoT	73.8	66.7	72.6	10.7	25.0	57.1	**34.5**
	AoT+	88.1	67.9	**83.3**	7.1	15.5	53.6	20.2
	MED	**94.0**	**77.4**	76.2	**42.9**	**42.9**	**63.1**	33.3
Total	CoT	77.8	65.6	74.3	25.7	26.6	55.7	26.3
	AoT+	85.0	71.6	**78.5**	19.8	17.6	52.7	23.8
	MED	**94.5**	**82.0**	**78.5**	**36.9**	**40.6**	**61.0**	**30.5**

As shown in Table 2, MEDAKA yields the best overall performance (under ties) on LIQUIDWORLD, highlighting its potential to enhance general LLM planning abilities in continuous-state domains. Our results also confirm that the structural characteristics induce problem difficulty: across all tested methods and models, problems tagged "nonlinear", "coupling", and "multi-goal" have pass rates substantially below the overall average.

Problem structural difficulty tags show cross-model performance heterogeneity that reflects discrepancies in reasoning capabilities. For instance, while frontier models including `Claude-Sonnet-4.6`, `Gemini-3-Flash`, and `GPT-5.4` are saturated on "linear" problems, they exhibit significantly lower vanilla CoT performance on "nonlinear" problems, but applying MEDAKA substantially improves them. Notably, `Claude-Sonnet-4.6` reaches 87.4% `pass@1`, suggesting it does possess the underlying reasoning capabilities needed to solve planning problems with nonlinear constraints, and MEDAKA successfully elicits these capabilities where CoT fails to.

In contrast, older and open-weights models including `GPT-4o`, `LLaMA-3.3-70B`, `Gemma4-4B`, and `Gemma4-2B` are less capable on "linear" problems, but MEDAKA successfully improves their performance while AoT+ cannot, though to a level that is still below the frontier models. However, for "nonlinear" problems, MEDAKA improves the closed-source `GPT-4o` (9.4% to 15.7%) but fails to yield meaningful improvement for the tested open-weights models, suggesting that they inherently lack the reasoning capabilities to solve these problems.

For "coupling" problems, we observe across-the-board gains compared to vanilla CoT when applying MEDAKA, showing a larger improvement than AoT+ (except on `GPT-5.4`). Since "coupling"

problems require algebraic solving of equation systems, such gains corroborate MEDAKA's design, where faithful algebraic derivation is encouraged.

Our results also show MEDAKA's potential in facilitating management and reasoning over long context. We observe substantial gains for "multi-goal" problems (`Claude-Sonnet-4.6`: 50.4% to 89.5%) except `GPT-4o` and `Gemma` models. MEDAKA also raises performance on problems with distractor vessels to a near-perfect 94.0% for `Claude-Sonnet-4.6` and induces a near 4-fold improvement for `GPT-4o` (10.7% to 42.9%). For both difficulty tags, neither MEDAKA nor AoT+ improve `Gemma4-2B`, suggesting its long-context reasoning capabilities are limited by its size.

Furthermore, Figure 4 shows that MEDAKA uses more tokens than baselines while consistently improving performance (except `GPT-5.4`). MEDAKA uses roughly 1.8 to 3.8× as many total tokens as CoT (median; Table 7), so we compare it against CoT with best-of-3 oracle selection (`pass@3`). Even so, MEDAKA's `pass@1` exceeds CoT's `pass@3` on 4/7 models (all except `GPT-5.4` and the two `Gemma4` models); e.g., for `Claude-Sonnet-4.6` and `Gemini-3-Flash`, CoT reaches 86.8 and 67.2 `pass@3` (Table 4 in Appendix A.2), versus MEDAKA's 94.5 and 82.0 `pass@1`. This suggests that, for most models, MEDAKA uses test-time compute more effectively than CoT.

In summary, these results suggest that MEDAKA can elicit LLM reasoning capabilities, e.g., resolving nonlinear constraints, faithful algebraic derivation, and long-context management/reasoning, that are crucial for planning in continuous-state domains.

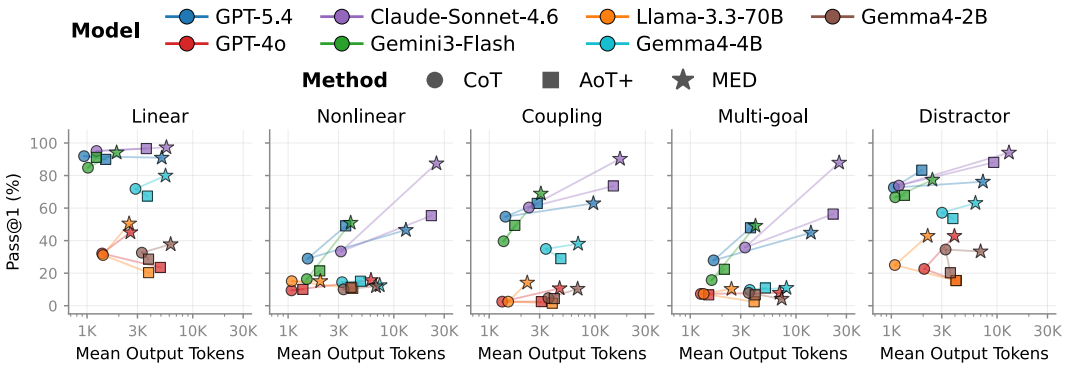

Figure 4: `pass@1` vs. mean output tokens on LIQUIDWORLD. Mean output tokens per trial (log-scale x) vs. Pass@1 (y). Markers denote methods, colors represent models, and thin lines connect methods of the same model. MEDAKA is abbreviated "MED".

4.3 QUALITATIVE FAILURE MODES

To better understand how search-style prompts fail in continuous-state planning, we inspected failed trajectories on "linear"-tagged problems from `GPT-4o` and `LLaMA-3.3-70B`, which are the two models where MEDAKA has the most improvement over AoT+ and CoT on these problems. We analyze cases where CoT or AoT+ failed but MEDAKA succeeded or produced a substantially more consistent trace. Table 6 in Appendix A.4 summarizes the dominant failure modes and provides example outputs. The errors are not merely arithmetic slips: they reflect a mismatch between searching candidate actions and deriving continuous parameters, where valid pour volumes can only be determined by deriving mass-balance equations rather than enumeration.

Trial-and-error parameter search. A common AoT+ failure is to treat continuous pour volumes as quantities to be guessed, tested, and locally amended. This behavior is encouraged by the backtracking structure of AoT+: when an intermediate state does not match the goal, the model often attempts another nearby action rather than re-deriving the entire set of pour parameters. In the first example of Table 6, the model reaches the target volume of 2 L but with composition $(20, 25, 55)\%$ instead of $(13.43, 30.52, 56.05)\%$, then adds small corrective pours and attempts to undo the overshoot.

State hallucination and drift. This appears in both CoT and AoT+ traces. Although AoT+ explicitly summarizes intermediate states, this summary an unstructured generation of the LLM and

Table 3: Ablation of MEDAKA on `Claude-Sonnet-4.6` and `Gemini-3-Flash`, 3 trials per problem. "− Recursive Module" drops the recursive goal decomposition module; "− State Tracking" drops the per-step state tracking module, which includes all numeric computation instructions. Each cell reports `pass@1` (%) on the full LIQUIDWORLD with the absolute drop (in percentage points) relative to full MEDAKA in parentheses. See Table 5 in Appendix A.3 for per-tag ablation results.

	Claude-Sonnet-4.6	Gemini-3-Flash
MEDAKA	94.5	82.0
− State Tracking	90.5 (−4.0)	76.0 (−6.0)
− Recursive Module	91.0 (−3.5)	75.1 (−6.9)

can become unreliable when state updates require numerical computation. In the second example of Table 6, the model empties V_8 but later reuses it as if it still contained liquid.

Locally correct but globally inconsistent arithmetic. A third failure mode is that the model performs locally correct computations while producing a globally inconsistent plan. This occurs when the model reasons one action at a time rather than solving the coupled conservation constraints over all liquids simultaneously. In the third example of Table 6, the final volume reaches 4 L, but the composition is $(36.7, 40.0, 23.3)\%$ instead of $(36.71, 26.67, 36.62)\%$. The error arises because liquids can appear in multiple sources: increasing one component may also introduce excess of another component. Thus, satisfying the target requires solving the global coupled system, not greedily matching one component at a time.

4.4 ABLATION STUDIES

We perform an ablation study on MEDAKA's two core components (recursive goal decomposition and per-step state tracking) using `Claude-Sonnet-4.6` and `Gemini-3-Flash`, the two models with the best MEDAKA performance and largest improvement over CoT. We develop two ablated versions: "− State Tracking" drops the per-step state-tracking module, which includes all numerical computation instructions, and "− Recursive Module" drops all instructions related to recursive goal decomposition, retaining per-step tracking as an 'algebra-aware' CoT. As shown in Table 3, both ablation configurations outperform CoT and AoT+, suggesting that the recursive decomposition (even without any numerical computation instructions) and state tracking modules each help LLMs on continuous-state planning. Note that since each ablation retains one module, their contributions need not be additive.

5 DISCUSSION

We studied LLM planning in continuous-state domains, where actions form a continuum and do not satisfy the enumerate-and-rank assumption of existing LLM planning frameworks. We developed MEDAKA, a structured prompting framework for LLM planning inspired by classical Means-Ends Analysis, and LIQUIDWORLD, a systematic testbed for LLM continuous-state planning grounded in fluid mixing, covering diverse scenarios and measuring key LLM reasoning capabilities through structural difficulty tags. MEDAKA can outperform existing prompting-based baselines across difficulty tags, surpass CoT's oracle-selected `pass@3` on four of seven models, and retain gains over baselines when either MEDAKA component is ablated. All these results corroborate that recursive goal decomposition and per-step state tracking, as encouraged by Means-Ends Analysis, both help LLMs plan in continuous-state domains.

While set in fluid mixing, LIQUIDWORLD is designed as a controlled testbed rather than a benchmark: it isolates parameter derivation with deterministic, fully specified transitions, so that failures can be attributed to LLM reasoning. Though LIQUIDWORLDalready exposes systematic failures under CoT using a simple transition function, it could be extended with stochastic transitions, partial observability, or additional constraint types. Finally, pairing MEDAKA with external solvers or verifiers is a separate direction from ours: we restrict comparisons to single-turn prompting methods because our goal is to characterize the native planning abilities of LLMs.

AI USE STATEMENT

In this work, we used generative AI tools to generate synthetic data sets. Specifically, the "linear"-tagged problems' generation and verification pipeline is executed by an AI coding agent, using hand-implemented code. We have not used generative AI tools to propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, assist with translation, clean or reformat dataset, support qualitative and thematic data analysis, interpret results, and the rest of the required disclosure tasks (help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs,) are not applicable to this work.

Additionally, we used generative AI tools to edit a research paper to improve readability. We have reviewed all AI-assisted work. LLM-generated synthetic data have all been verified for solvability by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

The authors of this paper adhere to the ICLR Code of Ethics in every way.

REPRODUCIBILITY STATEMENT

As efforts made to ensure reproducibility, we presented the prompts for the methods as well as LLM configurations in the Appendices, and open-sourced the code with a link present at the introduction section.

REFERENCES

[1] Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins Sri, Anthony Barrett, Dave Christianson, et al. Pddl—the planning domain definition language. Technical report, 1998.

[2] Mohammed Alfaki and Dag Haugland. Strong formulations for the pooling problem. *Journal of Global Optimization*, 56(3):897–916, 2013.

[3] Anthropic. Introducing Claude Sonnet 4.6. `https://www.anthropic.com/news/claude-sonnet-4-6`, February 2026. Large language model. Model ID: claude-sonnet-4-6. Accessed 2026-09-20.

[4] Daniil A Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023.

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[6] Aidan Curtis, Nishanth Kumar, Jing Cao, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Trust the PRoC3S: Solving long-horizon robotics problems with LLMs and constraint satisfaction. In *Conference on Robot Learning*, pages 1362–1383. PMLR, 2025.

[7] Kutluhan Erol, James Hendler, and Dana S. Nau. HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI)*, pages 1123–1128, 1994.

[8] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[9] Maria Fox and Derek Long. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124, 2003.

[10] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

[11] Google. Gemini 3 Flash: frontier intelligence built for speed. `https://blog.google/products/gemini/gemini-3-flash`, December 2025. Large language model. Accessed 2026-09-20.

[12] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[13] Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173, 2023.

[14] Dag Haugland. The computational complexity of the pooling problem. *Journal of Global Optimization*, 64(2):199–215, 2016.

[15] Co A Haverly. Studies of the behavior of recursion for the pooling problem. *Acm sigmap bulletin*, (25):19–28, 1978.

[16] Jörg Hoffmann. The metric-ff planning system: Translating"ignoring delete lists"to numeric state variables. *Journal of artificial intelligence research*, 20:291–341, 2003.

[17] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[18] Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Paul Saldyt, and Anil B Murthy. Position: Llms can't plan, but can help planning in llm-modulo frameworks. In *Forty-first International Conference on Machine Learning*, 2024.

[19] Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. Augmenting large language models with chemistry tools. *Nature machine intelligence*, 6(5):525–535, 2024.

[20] Allen Newell and Herbert A Simon. Gps, a program that simulates human thought. pages 415–428, 1995.

[21] Bilgehan Sel, Ahmad Al-Tawaha, Vanshaj Khattar, Ruoxi Jia, and Ming Jin. Algorithm of thoughts: enhancing exploration of ideas in large language models. In *Proceedings of the 41st International Conference on Machine Learning*, pages 44136–44189, 2024.

[22] Bilgehan Sel, Ruoxi Jia, and Ming Jin. LLMs can plan only if we tell them. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

[23] Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

[24] Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, et al. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

[25] Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. volume 36, pages 38975–38987, 2023.

[26] Shu Wang, Muzhi Han, Ziyuan Jiao, Zeyu Zhang, Ying Nian Wu, Song-Chun Zhu, and Hangxin Liu. Llm^3: Large language model-based task and motion planning with motion failure reasoning. In *2024 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 12086–12092. IEEE, 2024.

[27] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[28] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

[29] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 4791–4800, 2019.

A Appendix

A.1 Algorithmic Description of Medaka

The Medaka procedure is summarized in Algorithm 1.

Algorithm 1 Medaka: Means-Ends Decomposition for continuous-state LLM planning

Require: initial state s_0, goal g, LLM π, step budget $T_{\max}$, subgoal depth budget $c_{\max}$
Ensure: action sequence $a_{1:T}$ such that $g(s_T) = 1$, or Fail

$\textsc{State}_0 \leftarrow s_0$
for $t = 1, 2, \ldots, T_{\max}$ **do**
  [Reason]:
  $D \leftarrow \pi.\textsc{Diff}(\textsc{State}_{t-1},\ g)$ ▷ unsatisfied goal conditions
  if $D = \emptyset$ **then**
    return $a_{1:t-1}$
  end if
  The LLM selects a gap $d \in D$
  $o \leftarrow \pi.\textsc{ProposeOperator}(d,\ \textsc{State}_{t-1})$
  $c \leftarrow 0$
  while $\neg\textsc{applicable}(o)$ **and** $c < c_{\max}$ **do** ▷ $\textsc{applicable}(o) \triangleq$ all preconditions met $\wedge$ all parameters determined
    Identify blocker β in a ▷ unmet precondition or undetermined parameter
    $g' \leftarrow$ subgoal: "make β satisfied"
    Derive g' via algebraic solving on $(\textsc{State}_{t-1},\ g)$ ▷ parameter values computed under per-step state tracking
    Update o with derived value
    $c \leftarrow c + 1$
  end while
  if $\neg\textsc{applicable}(o)$ **then**
    return Fail
  end if
  [Action] $a_t \leftarrow o$
  $\textsc{State}_t \leftarrow \pi.\textsc{ComputeFullState}(\textsc{State}_{t-1},\ a_t)$ ▷ complete state: every vessel, not only those affected by a_t
end for
return Fail ▷ step budget exceeded

A.2 Pass@3 results

Table 4 reports Pass@3 for every model and method, computed over the same n=3 rollouts as Table 2: a problem is counted as solved if at least one rollout produces a plan accepted by the validator.

A.3 Per-tag ablation results

Table 5 breaks the ablation of Table 3 down by difficulty tag (Table 1).

A.4 Failure Mode Example

Table 6 shows examples for each representative failure mode.

A.5 Problem Generator: Linear Programming Solvability Formulations

Before any instance enters the testbed, the generator verifies solvability via three checks (Section 3.2). The first two are simple necessary conditions; the third is a linear program that is both necessary and sufficient.

Table 4: Pass@3 (%) on LIQUIDWORLD by difficulty tag. Shaded rows mark MEDAKA ("Med").

Tag	Method	Claude Sonnet 4.6	Gemini 3 Flash	GPT 5.4	GPT 4o	LLaMA 3.3 70B	Gemma4 4B	Gemma4 2B
linear	CoT	95.6	86.0	94.9	41.9	39.0	79.4	42.6
	AoT+	97.8	91.2	94.9	29.4	27.9	77.2	39.0
	MED	98.5	94.1	94.9	62.5	68.4	87.5	54.4
nonlinear	CoT	64.2	18.9	49.1	11.3	15.1	17.0	11.3
	AoT+	88.7	22.6	73.6	11.3	15.1	15.1	13.2
	MED	96.2	58.5	75.5	17.0	15.1	13.2	13.2
coupling	CoT	76.4	42.5	67.9	6.6	5.7	38.7	9.4
	AoT+	91.5	50.0	81.1	4.7	3.8	34.0	9.4
	MED	96.2	72.6	81.1	18.9	26.4	44.3	16.0
multi-goal	CoT	72.4	32.9	60.5	17.1	17.1	21.1	15.8
	AoT+	89.5	38.2	77.6	17.1	13.2	23.7	13.2
	MED	96.1	65.8	78.9	25.0	22.4	27.6	17.1
distractor	CoT	85.7	67.9	78.6	32.1	35.7	64.3	50.0
	AoT+	100.0	67.9	96.4	21.4	21.4	60.7	28.6
	MED	100.0	78.6	89.3	57.1	57.1	67.9	50.0
Total	CoT	86.8	67.2	82.0	33.3	32.3	61.9	33.9
	AoT+	95.2	72.0	88.9	24.3	24.3	59.8	31.7
	MED	97.9	84.1	89.4	49.7	53.4	66.7	42.9

Table 5: per-tag ablation results of MEDAKA (`pass@1`, %, 3 trials per problem). Columns as in Table 3.

	Claude-Sonnet-4.6			Gemini-3-Flash		
Tag	MEDAKA	− State	− Recursive	MEDAKA	− State	− Recursive
linear	97.3	93.6	94.1	94.1	91.4	93.1
nonlinear	87.4	82.4	83.0	50.9	36.5	28.9
coupling	90.3	88.4	89.3	68.9	58.2	56.0
multi-goal	89.5	86.0	87.7	60.5	46.9	43.4
distractor	94.0	90.5	92.9	77.4	69.0	67.9
Total	94.5	90.5	91.0	82.0	76.0	75.1

Volume sufficiency. For each liquid type $k \in \{1, \ldots, K\}$, let $a_k = \sum_i vol_i \cdot c_i^k$ be the total available volume of liquid k across all sources, and $d_k = \sum_j g_j^k$ the total demand across all goal vessels. The check requires $a_k \geq d_k$ for every k.

Concentration feasibility. For each liquid k, let $\bar{c}^k = \max_i c_i^k$ be the maximum fraction of k across all source vessels. For each goal vessel j with target fraction $\hat{c}_j^k$, the check requires $\hat{c}_j^k \leq \bar{c}^k$. This is a necessary condition because pouring preserves mixture ratios: no sequence of pours can concentrate a liquid above its highest source fraction.

Table 6: Representative failed traces illustrating three qualitative failure modes.

Failure mode 1: Trial-and-error repair after volume saturation
Example Initial state: V_1: 10 L L_2; V_2: 10 L $(40\%L_1, 60\%L_3)$; V_3: 10 L L_3; V_4: empty. Goal: $V_4 = 2$ L $(13.43\%L_1, 30.52\%L_2, 56.05\%L_3)$. `pour 1.0L V2 -> V4` $\Rightarrow$ V_4 : 1 L, $(40, 0, 60)\%$ `pour 0.5L V1 -> V4` $\Rightarrow$ V_4 : 1.5 L, $(26.7, 33.3, 40)\%$ `pour 0.5L V3 -> V4` $\Rightarrow$ V_4 : 2 L, $(20, 25, 55)\%$ *volume reached, composition wrong* `pour 0.1L V2 -> V4` $\Rightarrow$ V_4 : 2.1 L, $(21.0, 23.8, 55.2)\%$ *repair after saturation* `pour 0.1L V1 -> V4` $\Rightarrow$ V_4 : 2.2 L, $(20, 27.3, 52.7)\%$ *continued local repair* `pour 0.2L V4 -> V1` $\Rightarrow$ V_4 : 2 L, $(20, 27.3, 52.7)\%$ *attempted undo by pouring mixture back*
Failure mode 2: State hallucination from source depletion
Example Initial state: 100 source vessels, each 1 L pure liquid; V_8 contain L_4. Goal: V_{34}= 2L (50%L4, 50%L12), V_{51} = 2L (50%L15, 50%L13) . . . ; V_{82} = 2L (50%L20, 50%L4). `pour 1L V12 -> V34` $\Rightarrow$ V_{12} empty `pour 1L V8 -> V34` $\Rightarrow$ V_8 empty L_4 *source depleted* `pour 1L V50 -> V51` $\Rightarrow$ V_{50} empty `pour 1L V15 -> V51` $\Rightarrow$ V_{15} empty `pour 1L V13 -> V82` $\Rightarrow$ V_{13} empty `pour 1L V8 -> V82` $\Rightarrow$ *invalid action: reuses already-empty* V_8
Failure mode 3: Local correctness without global mass balance
Example Initial state: V_1: 10 L $(40\%L_1, 60\%L_2)$; V_2: 10 L $(70\%L_1, 30\%L_3)$; V_3: 10 L L_3; V_4: empty. Goal: $V_4 = 4$ L with $(36.71\%L_1, 26.67\%L_2, 36.62\%L_3)$. `pour 2.667L V1 -> V4` $\Rightarrow$ V_4 : 2.667 L, $(40, 60, 0)\%$ `pour 0.573L V2 -> V4` $\Rightarrow$ V_4 : 3.24 L, $(45.3, 49.4, 5.3)\%$ `pour 0.76L V3 -> V4` $\Rightarrow$ V_4 : 4 L, $(36.7, 40.0, 23.3)\%$ *volume reached, concentration not*

Single-target LP. For a single goal vessel with target component volumes $(g^1, \ldots, g^K)$, the LP is:

$$\min_{x_1,\ldots,x_n} \quad \sum_{i=1}^{n} x_i \tag{2}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} x_i \cdot c_i^k = g^k, \quad \forall\, k \in \{1, \ldots, K\} \tag{3}$$

$$0 \le x_i \le tot_i, \quad \forall\, i \in \{1, \ldots, n\} \tag{4}$$

where x_i is the volume poured from source i, tot_i is the total volume in source i, and c_i^k is the fraction of liquid k in source i. The objective is a dummy minimisation; only feasibility matters.

Joint LP (multiple targets). When the goal specifies $m > 1$ target vessels that may share sources, the decision variables become x_{ij} (volume poured from source i to target j), giving $n \times m$ variables:

$$\min_{x_{ij}} \quad \sum_{i,j} x_{ij} \tag{5}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} x_{ij} \cdot c_i^k = g_j^k, \quad \forall\, j \in \{1, \ldots, m\},\ \forall\, k \in \{1, \ldots, K\} \tag{6}$$

$$\sum_{j=1}^{m} x_{ij} \le tot_i, \quad \forall\, i \in \{1, \ldots, n\} \tag{7}$$

$$x_{ij} \ge 0, \quad \forall\, i, j \tag{8}$$

Constraint (6) requires each target to receive exactly the right volume of each liquid component. Constraint (7) ensures that the total volume drawn from each source across all targets does not exceed its available volume. An instance is admitted to the testbed only if the LP returns a feasible solution.

A.6 TOKEN USAGE DETAILS

We report per-trial token usage for all methods and models for LIQUIDWORLD in Table 7.

Table 7: **Per-trial token usage on LIQUIDWORLD** . Each cell reports the median (and Q1–Q3 interquartile range) across all problems and trials.

Model	Method	Input	Output	Total
Claude-4.6	CoT	962 (876–1425)	1231 (1019–1392)	2185 (1898–2931)
	AoT+	1463 (1357–1873)	1726 (1342–15891)	3134 (2734–17580)
	MED	2381 (2269–2831)	4498 (2432–17725)	6857 (4679–20482)
Gemini3-Flash	CoT	931 (839–1448)	1028 (892–1393)	1989 (1731–2855)
	AoT+	1308 (1181–1874)	1225 (1032–1776)	2543 (2250–3746)
	MED	2206 (2079–2764)	1888 (1555–2954)	4053 (3665–5685)
GPT-5.4	CoT	872 (794–1313)	966 (626–1516)	1859 (1446–2777)
	AoT+	1206 (1104–1751)	1415 (998–3438)	2640 (2052–5152)
	MED	2083 (1981–2674)	4981 (3209–8815)	7013 (5186–10987)
GPT-4o	CoT	873 (795–1314)	915 (771–1108)	1849 (1617–2474)
	AoT+	1247 (1118–2053)	1121 (813–1640)	2551 (1945–3536)
	MED	2084 (1982–2605)	2116 (1582–3277)	4402 (3546–6069)
Llama-3.3-70B	CoT	926 (827–1630)	1151 (879–1404)	2189 (1847–2854)
	AoT+	1251 (1139–1888)	1828 (1396–2949)	3507 (2754–4734)
	MED	2110 (2008–2714)	2029 (1527–2707)	4290 (3843–5116)
Gemma4-4B	CoT	957 (865–1464)	2735 (2153–3700)	4121 (3126–5098)
	AoT+	1324 (1196–1900)	3672 (2793–5127)	5466 (4145–6872)
	MED	2221 (2105–2790)	5473 (3944–7690)	7839 (6284–10539)
Gemma4-2B	CoT	956 (863–1462)	3355 (2486–3895)	4512 (3617–5313)
	AoT+	1322 (1195–1888)	3805 (3089–4534)	5488 (4507–6229)
	MED	2220 (2093–2778)	5739 (4375–7590)	8219 (6775–10188)

B Implementation Details

We report the complete system and user prompts for all methods tested. These prompts are identical across all models.

B.1 Complete LLM Prompt

We provide the system and user prompts for AoT+, MEDAKA, and CoT. All proprietary models (GPT-4o, GPT-5.4, Claude Sonnet 4.6, Gemini 3 Flash) were accessed via their providers' official APIs under standard terms of service. The open-weight models LLaMA-3.3-70B (Llama 3.3 Community License) and Gemma4-2B/4B (Gemma Terms of Use) were accessed via Featherless AI under standard terms of service. We use temperature=0 for the rollouts.

System prompt

```
You are a planning agent.

## Domain Description:

# LiquidWorld Planning Domain

## Description:
You have vessels with fixed capacities. Vessels hold homogeneous
    mixtures of liquids (L1, L2, L3, ...).
All liquids have the same density and blend when mixed. You pour a
    specified volume from one vessel to another.

## Rules:
1. Pour only from non-empty vessels.
2. Specify volume in L to pour (e.g., 0.5 L).
3. Poured liquid preserves the mixture ratio of the source.
When pouring from a mixture, ALL liquid types transfer in
    proportion to their concentration.
4. Destination must have enough remaining capacity.
5. Any non-empty vessel can serve as a source, including a goal
    vessel that has been partially filled.
A goal vessel only needs to match its target state at the END of
    the plan; intermediate contents are unconstrained.

## State and Goal Format:
- Pure liquid: Vi (capacity x L): y L Ln
- Mixture: Vi (capacity x L): y L, z% L1, w% L2
- Empty: Vi (capacity x L): empty
- Goal: Vi: x L, y% L1, z% L2

## Action Format:
- Pour x L from Vi to Vj
```

AoT+ user prompt

```
# Algorithm of Thoughts Plus (AoT+) Planning System

## Planning Task:
Given an initial state and a goal state, create a plan to transform
     the initial state into the goal state.
Use the Algorithm of Thoughts Plus approach to solve this planning
    problem.

### Key Instructions:
```

```
1. Think step by step to explore possible actions
2. Periodically track and restate the current state (using STATE
    notation)
3. Explore different paths when needed
4. Backtrack if a path doesn't seem promising
5. Build towards the goal state systematically
6. For each action you take, start the line with "ACTION:" followed
     by the action description
7. After each action, show the new state using "STATE x.y:"
    notation
8. Conclude with a "Plan summary:" section that lists all the
    actions in numbered format

## Examples:
Initial state: V1 (capacity 1L): 1 L L1; V2 (capacity 1L): 1 L L2;
    V3 (capacity 2L): empty
Goal state: V1: 0.5 L L1; V2: 0.5 L L2; V3: 1 L, 50% L1, 50% L2

STATE 1: Pour 4 L from V1 to V3
Let's try a different approach.

STATE 2: Pour 0.5 L from V1 to V3
STATE 2.1: Pour 1 L from V2 to V3

This approach isn't working well. Let's backtrack from STATE 2
STATE 2.2: Pour 0.5 L from V2 to V3

Final check: V1: 0.5 L L1V2: 0.5 L L2V3: 1 L, 50% L1, 50% L2 has
    been achieved.
...
Plan summary:
1. Pour 0.5 L from V1 to V3
2. Pour 0.5 L from V2 to V3

---

## Problem to Solve:

Initial state: V1 (capacity 999L): 10 L L1; V2 (capacity 999L): 10
    L L2; V3 (capacity 999L): 10 L L3; V4 (capacity 999L): empty
Goal state: V4: 5 L, 40% L1, 40% L2, 20% L3

STATE 0: V1 (capacity 999L): 10 L L1; V2 (capacity 999L): 10 L L2;
    V3 (capacity 999L): 10 L L3; V4 (capacity 999L): empty

Solve this step by step, tracking the state periodically. Remember
    to:
1. Prefix each action with "ACTION:"
2. Update the state after significant changes using "STATE x.y:"
    notation
3. End with a "Plan summary:" section

Start your solution:
```

MEDAKA user prompt

```
# MEDAKA Planning
## Planning Task:
Given an initial state and a goal state, create a plan to transform
     the initial state into the goal state.
```

```
Use the MEA planning format below.

### MEA Planning Format

Think step by step to explore possible actions.
You MUST follow this format strictly for every planning step.

### Key Instructions:
1. Compare the current state with the goal state and list all
    differences (Diff)
2. Pick one difference and search for an operator that reduces it
3. Judge whether the operator is directly solvable
(solvable = all preconditions met AND all parameters determined)
4. If not solvable, set "make the operator solvable" as a subgoal
and repeat from step 2 for that subgoal until a solvable operator
    is found
5. Execute the solvable operator, update the state, and return to
    step 1 for remaining differences
6. If the result deviates from expectation, diagnose the cause and
    adjust.

Each step:
[REASON]
- Diff: compare current state (from STATE k-1) with goal and list
    all differences (Diff)
- Operator: pick one action to reduce a gap; judge whether it is
    directly solvable
- Subgoal: what blocks the operator? State the blocker.
First, list each operand's per-unit contribution to the result.
Then set up equations, solve for parameters, and confirm all parts
are fully derived and the action moves closer to the goal.
Repeat (Subgoal -> Operator) until a solvable operator is found.
[ACTION] State the solvable operator with its computed parameter
STATE k: Write out the FULL new state (k is the step number).
You may show brief arithmetic before the STATE line when computing
new compositions for mixtures.

End your solution with a "Plan summary:" section listing only the
    actions from your successful path.

## Example:
Initial state: V1 (capacity 1L): 1 L L1; V2 (capacity 1L): 1 L L2;
    V3 (capacity 2L): empty
Goal state: V3: 1 L, 50% L1, 50% L2

STATE 0: V1 (capacity 1L): 1 L L1; V2 (capacity 1L): 1 L L2; V3 (
    capacity 2L): empty

STEP 1:
[REASON]
- Diff: V3 empty, goal 0.5L L1 + 0.5L L2 | V1 excess 0.5L | V2
    excess 0.5L
- Operator: Pour to V3 -- not solvable (V3 needs L1 + L2; source
    unknown)
- Subgoal: determine sources. Sources with L1: V1 only. Sources
    with L2: V2 only.
No shared components across sources -> solve one at a time. Start
    with L1.
-> Operator: Pour from V1 to V3 -- not solvable (amount unknown)
-> Subgoal: V3 needs (1L * 50% = 0.5L) L1; V1 is pure L1 (
    concentration = 100%) -> pour amount = 0.5 / 1.0 = 0.5L
-> Operator: Pour 0.5L from V1 to V3 -- solvable (V1 has 1L >= 0.5L,
     V3 cap 2L >= 0.5L)
```

```
[ACTION] Pour 0.5 L from V1 to V3
STATE 1:
V1: 1 - 0.5 = 0.5L L1
V3 receives 0.5L pure L1: L1 = 0.5*100% = 0.5L; total = 0.5L
V1: 0.5L L1 | V2: 1L L2 | V3: 0.5L L1

STEP 2:
[REASON]
- Diff: V3 has 0.5L L1 + 0L L2, goal 0.5L L1 + 0.5L L2 -> remaining:
    0.5L L2
- Operator: Pour to V3 -- not solvable (amount unknown)
- Subgoal: Sources with L2: V2 only (pure L2). Amount unknown.
V3 needs (1L * 50% = 0.5L) L2; V2 is pure L2 (concentration = 100%)
    -> pour amount = 0.5 / 1.0 = 0.5L
-> Operator: Pour 0.5L from V2 to V3 -- solvable (V2 has 1L >= 0.5L,
    V3 cap 2L >= 0.5L)
[ACTION] Pour 0.5 L from V2 to V3
STATE 2:
V2: 1 - 0.5 = 0.5L L2
V3 receives 0.5L pure L2: L2 = 0.5*100% = 0.5L; total = 0.5 + 0.5 =
    1L, L1 = 0.5/1 = 50%, L2 = 0.5/1 = 50%
V1: 0.5L L1 | V2: 0.5L L2 | V3: 1L 50%L1 50%L2

Plan summary:
1. Pour 0.5 L from V1 to V3
2. Pour 0.5 L from V2 to V3

---

## Problem to Solve:

Initial state: V1 (capacity 999L): 10 L L1; V2 (capacity 999L): 10
    L L2; V3 (capacity 999L): 10 L L3; V4 (capacity 999L): empty
Goal state: V4: 5 L, 40% L1, 40% L2, 20% L3

STATE 0: V1 (capacity 999L): 10 L L1; V2 (capacity 999L): 10 L L2;
    V3 (capacity 999L): 10 L L3; V4 (capacity 999L): empty

Solve this step by step using the MEA planning format above.
    Remember:
1. For each step, use the MEA loop inside [REASON], then [ACTION] ->
    STATE k:
2. End with a "Plan summary:" section

Start your solution:
```

CoT user prompt

```
# Vanilla Chain-of-Thought Planning

## Planning Task:
Given an initial state and a goal state, create a plan to reach the
    goal.

Think step by step. End with a "Plan summary:" section listing only
    your final actions.

## Example:
Initial state: V1 (capacity 1L): 1 L L1; V2 (capacity 1L): 1 L L2;
    V3 (capacity 2L): empty
```

```
Goal state: V1: 0.5 L L1; V2: 0.5 L L2; V3: 1 L, 50% L1, 50% L2

V3 needs 1L at 50/50, so it needs 0.5L of L1 and 0.5L of L2. V1 is
    pure L1 with 1L, and the goal wants V1 to keep 0.5L, so I pour
    0.5L from V1 to V3. Now V3 has 0.5L L1. Similarly V2 is pure L2
    and should keep 0.5L, so I pour 0.5L from V2 to V3. That gives
    V3 = 1L at 50/50, and both V1 and V2 keep 0.5L. Done.

Plan summary:
1. Pour 0.5 L from V1 to V3
2. Pour 0.5 L from V2 to V3

---

## Problem to Solve:
Initial state: V1 (capacity 999L): 10 L L1; V2 (capacity 999L): 10
    L L2; V3 (capacity 999L): 10 L L3; V4 (capacity 999L): empty
Goal state: V4: 5 L, 40% L1, 40% L2, 20% L3

Think step by step. Finish with:
Plan summary:
1. ...
```