TAPE: Tool-Guided Adaptive Planning and Constrained Execution in Language Model Agents

Jongwon Jeong[1] **Jungtaek Kim**[1] **Kangwook Lee**[123*]
[1]University of Wisconsin–Madison [2]KRAFTON [3]Ludo Robotics
{jjeong64, jungtaek.kim}@wisc.edu kangwooklee@krafton.com

Abstract

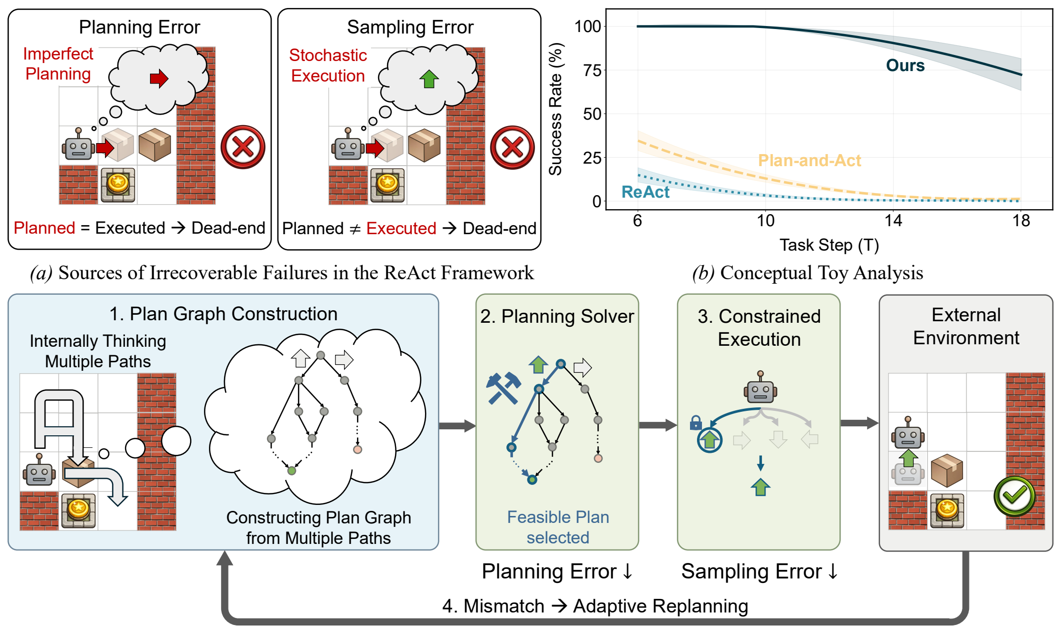
Language Model (LM) agents have demonstrated remarkable capabilities in solving tasks that require multiple interactions with the environment. However, they remain vulnerable in environments where a single error often leads to irrecoverable failure, particularly under strict feasibility constraints. We systematically analyze existing agent frameworks, identifying imperfect planning and stochastic execution as the primary causes. To address these challenges, we propose **T**ool-guided **A**daptive **P**lanning with constrained **E**xecution (**TAPE**). TAPE enhances planning capability by aggregating multiple plans into a graph and employing an external solver to identify a feasible path. During execution, TAPE employs constrained decoding to reduce sampling noise, while adaptively re-planning whenever environmental feedback deviates from the intended state. Experiments across Sokoban, ALFWorld, MuSiQue, and GSM8K-Hard demonstrate that TAPE consistently outperforms existing frameworks, with particularly large gains on hard settings, improving success rates by 21.0 percentage points on hard settings on average, and by 20.0 percentage points for weaker base models on average.

1 Introduction

Language Models (LMs) have evolved into agents capable of understanding and interacting with external environments such as computers (Xi et al., 2025), simulation platforms (Park et al., 2023), and physical robot environments (Hu et al., 2023). In these applications, tools serve as interfaces that enable LM agents to reason, act, and observe within these environments (Li, 2025). For instance, search tools retrieve information from databases (Jin et al., 2025), mouse and keyboard tools interact with computer interfaces (Xie et al., 2024), coding tools execute programs within computers (Gao et al., 2023), and robotic tools perform actions in the physical world (Joublin et al., 2024). By generating executable prompts for tool usage, LM agents can iteratively think and act while receiving observations from the environment, referred to as *ReAct* frameworks (Yao et al., 2023b; Shinn et al., 2023; Hao et al., 2023; Yao et al., 2023a; Zhuang et al., 2024; Qiao et al., 2024; Kim et al., 2025). Through this iterative process, they can solve complex tasks that require planning and adaptation (Parisi et al., 2022; Gao et al., 2023).

Although the ReAct frameworks have shown impressive capabilities in various domains (Liu et al., 2024), these frameworks are highly ineffective when even a few mistakes can cause irrecoverable harms, making it infeasible to achieve the goal thereafter. In practice, feasibility constraints, such as time and cost budgets, limits on tool usage, and strict safety requirements, are one common source of irrecoverable failures. For instance, coding agents often operate under latency or API-cost budgets (Kim et al., 2024; Zheng et al., 2024; Liu et al., 2025), robotic agents must satisfy strict safety constraints (Yang et al., 2024b; Guo et al., 2025), and life-simulation agents face API-usage limits (Park et al., 2023; Choi et al., 2025). Under such constraints, incorrect tool use can exhaust the remaining budget or violate constraints, leaving no feasible sequence of tools that reaches the goal. This motivates our research question:

[*]This work is done while at the University of Wisconsin–Madison.



(a) Sources of Irrecoverable Failures in the ReAct Framework

(b) Conceptual Toy Analysis

(c) Our Framework: Tool-guided Adaptive Planning with constrained Execution (TAPE)

Figure 1: **Overview.** We illustrate our work using Sokoban, where the goal is to push all boxes onto target locations. **(a) Sources of Irrecoverable Failure in the ReAct Framework.** A planning error occurs when the internal reasoning suggests a non-viable action (e.g., pushing a box against a wall); this makes the goal unachievable as the agent cannot pull the box from the wall, while a sampling error arises when LM stochasticity leads to an action deviating from the plan. **(b) Conceptual Toy Analysis.** We model simplified agents by injecting planning and sampling errors into a feasible policy for Sokoban. We measure success rates as the task step T increases, observing that existing frameworks degrade rapidly as T grows. **(c) Our Framework.** TAPE generates and aggregates multiple plans into a graph and uses a solver to select a feasible path, thereby reducing planning errors. Then, it enforces constrained execution to suppress sampling errors.

How can LM agents maximize the success rate on tasks in the presence of irrecoverable failures?

To address this question, we first analyze ReAct frameworks and identify two distinct sources of irrecoverable failures: *planning error* (Valmeekam et al., 2023) and *sampling error* (Hao et al., 2025a). A planning error occurs when an agent's internal planning (i.e., reasoning) is imperfect, leading it to recommend a non-viable action (see Figure 1a, left). A sampling error can occur even when the agent's internal reasoning is correct, because stochastic token generation may generate an action different from the planned one (see Figure 1a, right). By simulating a simplified agent where we inject these errors into a feasible policy, we find that both failures compound as the task horizon increases, reducing the overall success rate (see Figure 1b). Similar issues persist for Plan-and-Act (PA) frameworks (Wang et al., 2023a; Erdogan et al., 2025; Sun et al., 2023). While generating a full plan before execution mitigates sampling errors, PA remains brittle to planning errors, resulting in suboptimal success rates in our analysis (see Figure 1b). Refer to Appendix C for the theoretical validation of this analysis.

To mitigate failures from both error sources, we propose an external **T**ool-guided **A**daptive **P**lanning with constrained **E**xecution framework (**TAPE**), as shown in Figure 1c. Specifically, our framework generates multiple candidate plans and aggregates them into a plan graph, then uses an external solver (e.g., Integer Linear Programming (ILP) (Schrijver, 1998)) to select a feasible path, mitigating failures due to planning errors by optimally selecting among diverse candidates. Next, our method enforces constrained execution by constrained decoding (Willard & Louf, 2023) to the selected next action, which suppresses sampling errors. At each step, if TAPE detects a mismatch between the predicted and realized observations, it updates the plan graph and re-selects a feasible plan. As shown in Figure 1b, our framework achieves a higher success rate than other frameworks.

We evaluate the proposed framework on benchmarks built from Sokoban (Schrader, 2018), ALFWorld (Shridhar et al., 2021), MuSiQue (Trivedi et al., 2022) and GSM8K-Hard (Gao et al., 2023) by adding feasibility constraints, e.g., budgets or tool/action limits, that make mistakes difficult to recover from. Experimental results show that our framework consistently outperforms the ReAct frameworks across all benchmarks. The performance gap is most significant in hard tasks and for weaker base models, proving that our framework effectively mitigates the irrecoverable failures.

2 Problem Formulation

We introduce the definition of an agentic task, formalize the well-known LM agent frameworks, ReAct and Plan-and-Act, and define the errors that arise in LM agents.

2.1 Agentic Task

We consider an *agentic task* in which an LM agent iteratively interacts with an external environment to solve a given problem. We formalize this process as a goal-conditioned Markov Decision Process (G-MDP) (Nasiriany et al., 2019), denoted by $\mathcal{M} = (\mathcal{G}, \mathcal{A}, \mathcal{S}, P, R)$. Here, $\mathcal{G}$ is the goal space, $\mathcal{A}$ is the action space, $\mathcal{S}$ is the state space, P denotes the transition dynamics, and $R : \mathcal{G} \times \mathcal{S} \to \{0, 1\}$ is a goal-dependent reward function, where we write $R_g(s)$. At the beginning of an episode, a goal $g \in \mathcal{G}$ is given. When actions incur costs and the agent operates under budgets, we define a cost function $C : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^k$ and a budget vector $B \in \mathbb{R}^k$. The state $s_t \in \mathcal{S}$ is defined as the interaction history up to timestep t, i.e., $s_t = (a_0, o_0, a_1, o_1, \ldots, a_{t-1}, o_{t-1})$ with $s_0 = \emptyset$. At each timestep $t \in \{0, 1, \ldots\}$, the agent selects an action $a_t \in \mathcal{A}$ conditioned on g and s_t. The environment responds with an observation $o_t = (o_t^{\text{status}}, o_t^{\text{return}})$, where o_t^{status} indicates the execution status (e.g., success or failure), and o_t^{return} denotes the returned value (e.g., tool outputs or error messages). If the task considers budget constraints, the observation includes the remaining-budget component, i.e., $o_t = (o_t^{\text{status}}, o_t^{\text{return}}, o_t^{\text{budget}})$. Given s_t and a_t, the next state s_{t+1} is realized according to $P(\cdot \mid s_t, a_t)$, which corresponds to appending (a_t, o_t) to s_t. An episode $\tau = (s_0, a_0, s_1, \ldots, s_T)$ is successful if it terminates at a state s_T such that $R_g(s_T) = 1$ and, when budget constraints are present, $\sum_{t=0}^{T-1} C(s_t, a_t) \preceq B$, where $\preceq$ denotes element-wise inequality. Such budgets can represent time limits, cost limits, limits on the number of tool calls, or so on (Zheng et al., 2024; Liu et al., 2025; Ma et al., 2026). If the agent enters a dead-end state (e.g., by violating the budget constraint), from which reaching the goal is impossible regardless of subsequent actions and transitions, we terminate the episode at s_T and set $R_g(s_T) = 0$.

2.2 Language Model Agent Framework

An LM agent is defined as a stochastic policy $\pi_\theta(a_t \mid s_t, g)$ that selects an action given the current environment state s_t and goal g. Since the agent cannot observe the concrete return values until it actually interacts with the environment, it must rely on an internal world model to simulate and evaluate potential trajectories beforehand (Hao et al., 2023). To facilitate this internal planning, the agent operates on an abstract state $\hat{s}_t \coloneqq z(s_t) \coloneqq (a_0, o_0^{\text{status}}, \ldots, a_{t-1}, o_{t-1}^{\text{status}})$, which distills the raw history into essential execution statuses. In a constrained G-MDP, this representation is extended to $\hat{s}_t^c \coloneqq z^c(s_t) \coloneqq (a_0, (o_0^{\text{status}}, o_0^{\text{budget}}), \ldots, a_{t-1}, (o_{t-1}^{\text{status}}, o_{t-1}^{\text{budget}}))$ to track resource consumption. We assume the agent has an internal world model $P_\theta(\hat{s}_{t+1} \mid \hat{s}_t, a_t)$, $R_{g,\theta}(\hat{s}_T)$, and $C_\theta(\hat{s}_t^c, a_t)$ that approximate the environment's true transition dynamics P, abstract reward R_g, and cost function C. Using the internal world model, the LM agent can perform internal planning to identify an action path that maximizes rewards.

ReAct Framework. Following ReAct (Yao et al., 2023b), we decompose the agent's decision-making into two stages, “Think” and “Act”. At each step t, “Think” contains various types of textual reasoning (Yao et al., 2023b; Shinn et al., 2023; Kim et al., 2025) or advanced planning strategy (Hao et al., 2023; Liu et al., 2023b; Yao et al., 2023a; Zhuang et al., 2024; Qiao et al., 2024; Qian et al., 2025; Katz et al., 2024). We interpret this “Think” as *planning* over an internal abstract world model (Hao et al., 2023). Starting from the current interaction state, the agent predicts future abstract states over a lookahead horizon and outputs a planned next action $\hat{a}_t \in \mathcal{A}$ (Liu et al., 2023b). “Act” then samples an executed action conditioned on the state and the planned action. Accordingly, this decision process is formalized as the following:

$$\hat{a}_t \sim \pi_\theta^{\text{Think}}(\cdot \mid s_t, g), \qquad a_t \sim \pi_\theta^{\text{Act}}(\cdot \mid s_t, \hat{a}_t, g). \tag{1}$$

Plan-and-Act Framework. Plan-and-Act (PA) framework leverages an internal abstract world model to perform full planning before step-wise execution (Wang et al., 2023a; Xu et al., 2023; Xiong et al., 2025; Erdogan et al., 2025). Given a goal g, the agent explores the abstract state space and produces an abstract plan $\hat{\tau} = (\hat{s}_0^{\hat{\tau}}, \bar{a}_0, \hat{s}_1^{\hat{\tau}}, \bar{a}_1, \ldots, \hat{s}_T^{\hat{\tau}})$, where the planned action at step t is induced by the internal world model $(P_\theta, R_{g,\theta}, C_\theta)$ through planning, i.e., $\bar{a}_t \sim \pi_\theta^{\text{Think}}(\cdot \mid \hat{s}_t^{\hat{\tau}}, g)$. Once the plan $\hat{\tau}$ is provided in-context, the agent conditions its step-wise decision on the current interaction state s_t and the plan guidance. We model PA's in-context use of the provided plan as follows. When the plan is applicable at step t, meaning that the current abstract state matches the abstract state, $z(s_t) = \hat{s}_t^{\hat{\tau}}$, the PA agent directly follows the planned action with probability $p \in [0, 1]$. Otherwise, it falls back to the ReAct-style decision process. Formally, if $z(s_t) = \hat{s}_t^{\hat{\tau}}$,

$$a_t = \begin{cases} \bar{a}_t, & \text{with probability } p_{\text{follow}}, \\ a_t^{\text{ReAct}}, & \text{with probability } 1 - p_{\text{follow}}, \end{cases} \tag{2}$$

and if $z(s_t) \neq \hat{s}_t^{\hat{\tau}}$, $a_t = a_t^{\text{ReAct}}$, where $\tilde{a}_t \sim \pi_\theta^{\text{Think}}(\cdot \mid s_t, g)$, $a_t^{\text{ReAct}} \sim \pi_\theta^{\text{Act}}(\cdot \mid s_t, \tilde{a}_t, g)$. Here, $\tilde{a}_t$ denotes the planned next action produced by ReAct.

2.3 Errors from Two Different Sources

LM agents are prone to fail agentic tasks due to internal *planning error* and *sampling error* in LMs. In practice, planning error arises from limited planning capability or mismatch between the agent's internal world model and the true world model (Valmeekam et al., 2023). The sampling error can arise from stochastic token generation, prompt sensitivity, and formatting drift (Hao et al., 2025a). To quantify both errors, in G-MDP, we define viable actions as in Definition 1.

Definition 1. *Viable action set is* $\mathcal{A}_{\text{viable}}(s_t) := \Big\{ a \in \mathcal{A} \Big| \exists K \geq 1, \Pr\big(R_g(s_{t+K}) = 1 | s_t, a\big) > 0 \Big\}$.

Using the viable action set, we formalize planning and sampling errors as follows. For simplicity, we assume a constant per-step error rate that is time-invariant.

Assumption 1 (Planning Error)**.** *Let $\hat{a}_t$ denote the agent's planned action at step t. For simplicity, we model the planning error as the constant*

$$\epsilon_{\text{p}} := \Pr\left(\hat{a}_t \notin \mathcal{A}_{\text{viable}}(s_t)\right). \tag{3}$$

Assumption 2 (Sampling Error)**.** *Let $\hat{a}_t$ denote the agent's planned action at step t. For simplicity, we model sampling error by the following constant rates:*

$$\epsilon_{\text{s}} := \Pr(a_t \neq \hat{a}_t). \tag{4}$$

Moreover, we assume that ϵ_{s} does not depend on whether the planned action is viable, i.e., $\Pr(a_t \neq \hat{a}_t \mid \hat{a}_t \in \mathcal{A}_{\text{viable}}(s_t)) = \Pr(a_t \neq \hat{a}_t \mid \hat{a}_t \notin \mathcal{A}_{\text{viable}}(s_t)) = \epsilon_{\text{s}}$.

We further characterize the consequences of sampling error as δ_{b} and δ_{r} in Assumption 3. δ_{b} quantifies how often the sampling error breaks viability when the planned action is viable, while δ_{r} captures how often the sampling error recovers viability when the planned action is non-viable.

Assumption 3. *When the planned action $\hat{a}$ is viable, δ_{b} denotes the probability that deviating from it breaks viability as $\delta_{\text{b}} := \Pr\big(a_t \notin \mathcal{A}_{\text{viable}}(s_t) \bigm| \hat{a}_t \in \mathcal{A}_{\text{viable}}(s_t),\ a_t \neq \hat{a}_t\big)$. On the other hand, when the planned action is non-viable, δ_{r} denotes the probability that deviating from it recovers viability as $\delta_{\text{r}} := \Pr\big(a_t \in \mathcal{A}_{\text{viable}}(s_t) \bigm| \hat{a}_t \notin \mathcal{A}_{\text{viable}}(s_t),\ a_t \neq \hat{a}_t\big)$.*

3 Our Approach: TAPE

In this section, we introduce **TAPE** (**T**ool-guided **A**daptive **P**lanning with constrained **E**xecution), a framework designed to reduce both planning and sampling errors. First, our framework performs (i)

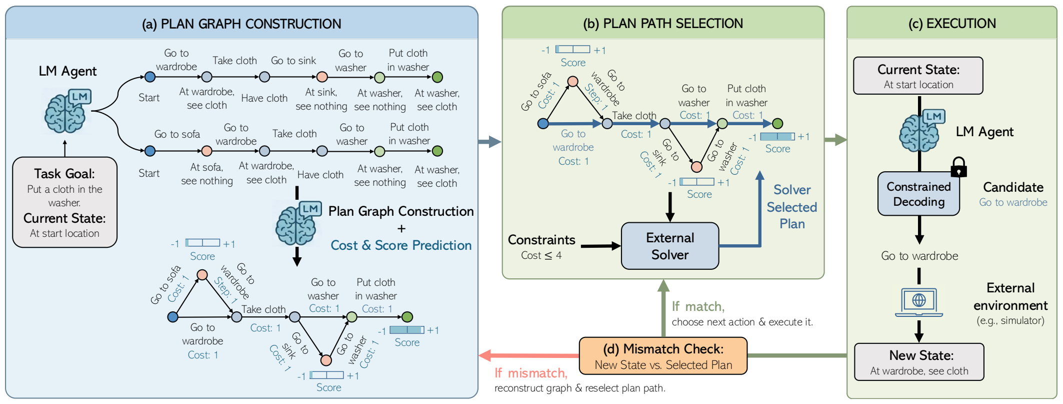

(a) PLAN GRAPH CONSTRUCTION
LM Agent
Task Goal: Put a cloth in the washer.
Current State: At start location
Start
Go to wardrobe
Take cloth
Go to sink
Go to washer
Put cloth in washer
At wardrobe, see cloth
Have cloth
At sink, see nothing
At washer, see nothing
At washer, see cloth
Go to sofa
At sofa, see nothing
Plan Graph Construction + Cost & Score Prediction
Score
Cost: 1
Step: 1
(b) PLAN PATH SELECTION
Constraints Cost ≤ 4
External Solver
Solver Selected Plan
(c) EXECUTION
Current State: At start location
Constrained Decoding
Candidate Go to wardrobe
External environment (e.g., simulator)
New State: At wardrobe, see cloth
(d) Mismatch Check: New State vs. Selected Plan
If match, choose next action & execute it.
If mismatch, reconstruct graph & reselect plan path.

Figure 2: **Overview of TAPE.** The proposed framework consists of four steps. (a) **Plan Graph Construction**: The LM samples multiple trajectories, which are aggregated into a plan graph with predicted costs and scores. (b) **Plan Path Selection**: An external solver (e.g., ILP) identifies the optimal path (blue arrows) subject to constraints. (c) **Constrained Execution**: The agent executes the selected actions using constrained decoding to eliminate sampling errors. (d) **Mismatch Check**: If a mismatch occurs between the predicted and realized state, the agent re-performs plan graph construction and path selection; otherwise, the agent executes the next planned action.

plan graph construction by recombining multiple abstract trajectories generated by an LM, which increases the probability that a feasible plan exists within the planning space. Next, TAPE operates (ii) *plan path selection* on this graph using an external solver, such as Integer Linear Programming (ILP) (Schrijver, 1998), based on predicted costs and rewards. Given the constructed plan graph and its predicted costs and rewards, the solver returns an optimal feasible path within the graph when one exists, effectively reducing planning errors. Then, our framework performs (iii) *constrained execution* by restricting the action space at decoding time (Willard & Louf, 2023) so that only the next prescribed action on the selected path is admissible, thereby suppressing sampling errors. Lastly, to remain robust against mismatches between realized observations and the internal plan, the framework adaptively re-performs (iv) *plan graph construction and path selection* using newly observed information whenever a discrepancy arises. A conceptual overview on ALFWorld (Shridhar et al., 2021) is shown in Figure 2.

Plan Graph Construction. To construct a plan graph $\mathcal{G} = (V, E)$, where $v \in V$ is a state node and $e \in E$ is an edge representing an action $\bar{a}$, TAPE first generates M abstract plans:

$$\hat{\tau}^{(m)} = \left(\hat{s}_0^{(m)}, \bar{a}_0^{(m)}, \hat{s}_1^{(m)}, \bar{a}_1^{(m)}, \ldots, \hat{s}_{L_m}^{(m)}\right), \tag{5}$$

where $m = 1, \ldots, M$, $\hat{s}_\ell^{(m)} \in \hat{\mathcal{S}}$, and $\bar{a}_\ell^{(m)} \in \mathcal{A}$. Then, our framework folds these sampled paths by merging states that share the same core information. Specifically, TAPE merges multiple abstract states $\hat{s}$ into a single node $v \in V$ if they represent the same observation and task progress. For example, in ALFWorld, many distinct states (e.g., different intermediate tool outputs or error messages) correspond to the same node representing the agent's current location, the objects in the inventory, and the task progress (such as whether a required object has been found, cleaned, or placed). We denote this merging function as $f_\theta : \hat{\mathcal{S}} \to \mathcal{V}$. For the merged nodes, TAPE reassigns edges as $e_{ij} = \left(v_i \xrightarrow{\bar{a}_\ell^{(m)}} v_j\right) \in E$, where $v_i = f_\theta(\hat{s}_\ell^{(m)})$ and $v_j = f_\theta(\hat{s}_{\ell+1}^{(m)})$ for each consecutive pair in the sampled plans.

Score and Cost Prediction. Using the internal abstract world model $(P_\theta, R_{g,\theta}, C_\theta)$ in LMs, TAPE assigns a scalar reward to each node $u \in V$ and a predicted cost to each edge $e \in E$. Specifically, for terminal nodes v_{ter}, our framework estimates the expected reward as the following:

$$\hat{r}_\theta(u) := \mathbb{E}_\theta[R_{g,\theta}(\hat{s}) \mid f_\theta(\hat{s}) = v_{\text{ter}}], \tag{6}$$

representing the expected reward of being in v_{ter}. For each edge $e = (v \xrightarrow{a} v') \in E$, TAPE predicts the cost vector as $\hat{\mathbf{c}}_\theta(e) := \mathbb{E}_\theta[\mathbf{C}_\theta(\hat{s}, a) \mid f_\theta(\hat{s}) = v]$, which estimates the expected budget consumption of executing the edge-labeled action at node v. In constrained settings, these predicted costs $\{\hat{\mathbf{c}}_\theta(e)\}_{e \in E}$ are used to enforce feasibility under the remaining budget $\mathbf{b}_t$.

Plan Path Selection. Given the graph $\mathcal{G} = (V, E)$, the current node v_0, and the set of terminal nodes $V_{\text{ter}} \subseteq V$, our framework selects a single directed walk on $\mathcal{G}$ by solving an optimization problem via an external solver (e.g., Integer Linear Programming (ILP) (Schrijver, 1998)). To obtain a tractable finite-size ILP, the solver optimizes over a finite horizon $L_{\max} \geq 1$ using a time-expanded formulation. For each edge $e = (v \xrightarrow{a} v') \in E$ and step $\ell \in \{0, 1, \ldots, L_{\max} - 1\}$, let $x_{e,\ell} \in \{0, 1\}$ indicate whether the walk takes edge e at step ℓ. $\text{src}(e) = v$ and $\text{tgt}(e) = v'$ are the source and target of e. Formally, in the G-MDP setting, the ILP formulation is in Appendix B.

Constrained Execution. Given the optimal path selected by the solver, denoted as $\pi^\star = (v_0^{\pi^\star} \xrightarrow{\bar{a}_0^\star} v_1^{\pi^\star} \xrightarrow{\bar{a}_1^\star} \cdots)$, TAPE executes the plan by restricting the admissible action set to the prescribed action whenever the path is applicable. Let v_t be the current node in the environment at step t. If the current node matches the planned node (i.e., $v_t = v_t^{\pi^\star}$), we enforce the LM π_θ^{Act} to generate the planned action $\bar{a}_t^\star$ as a_t. This is implemented via constrained decoding (Willard & Louf, 2023), which constrains the LM to follow $\bar{a}_t^\star$ by fixing the tool choice and enforcing the exact tool-call format.

Mismatch Check and Replanning. In practice, a mismatch between the predicted state in the plan and the realized state can invalidate the current path. For example, the environment might transition to an unexpected state, or the remaining budget might evolve differently than estimated. To address this, our framework verifies the consistency between the real state node $\hat{s}_{t+1}$ and the predicted state node $v_{t+1}^{\pi^\star}$. Specifically, if TAPE confirms that v_{t+1} matches $v_{t+1}^{\pi^\star}$, TAPE proceeds to execute the next planned action along $\pi^\star$. Conversely, if our framework detects a mismatch, TAPE regenerates multiple abstract plans, constructs a new plan graph with updated costs and rewards, and solves for a new optimal path via the external solver.

4 Empirical Analysis

We compare TAPE against two representative agent frameworks: (i) **ReAct** (Yao et al., 2023b) which interleaves reasoning and acting, and (ii) **Plan-and-Act (PA)** (Erdogan et al., 2025) which executes a pre-generated plan. We note that various advanced prompting or reflection techniques can be orthogonally integrated into these frameworks; in this paper, we adapt Xiong et al. (2025) for the implementation. Therefore, we focus on evaluating the fundamental structural differences between the frameworks.

For the evaluation, we consider four benchmarks characterized by frequent irrecoverable states: Sokoban, ALFWorld, GSM8K-Hard, and MuSiQue. **Sokoban** (Schrader, 2018) is a classic planning puzzle requiring the agent to push boxes to target locations without creating deadlocks. **ALFWorld** (Shridhar et al., 2021) involves embodied decision-making in a simulated household environment. We also include **MuSiQue** (Trivedi et al., 2022) for multi-hop factual reasoning using retrieval tools and **GSM8K-Hard** (Gao et al., 2023) for mathematical reasoning using arithmetic tools. We report the success rate for each task. For more details about the experimental setting, see Section E.1.

4.1 Overall Results

The overall results on agentic benchmarks using `gpt-4.1-mini` (OpenAI, 2025) are summarized in Figure 3. We compare TAPE with ReAct and PA across four benchmarks characterized by frequent irrecoverable states, and find that TAPE consistently outperforms the baselines. In particular, TAPE improves over ReAct, suggesting that our method mitigates both planning and sampling errors, whereas ReAct suffers from their accumulation. Additionally, PA frequently outperforms ReAct, which may indicate that providing an explicit plan as in-context guidance helps reduce sampling errors. Furthermore, TAPE consistently surpasses both ReAct and PA. As shown in Section G.2, we additionally compare against Best-of-N (Wang et al., 2023b; Kang et al., 2025) of ReAct and PA,

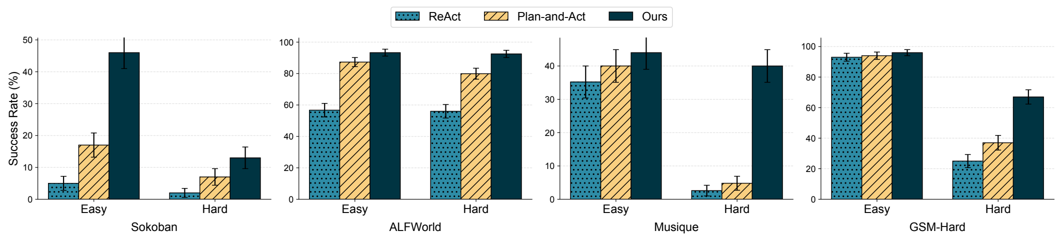


Figure 3: **Success rates across four agentic tasks.** We evaluate our framework against ReAct and Plan-and-Act on Sokoban, ALFWorld, Musique, and GSM-Hard. We use `gpt-4.1-mini` for LM backbone. We find that TAPE consistently demonstrates superior performance over existing frameworks in both easy and hard settings.

Table 1: **Planning and sampling error analysis on Sokoban.** We estimate planning and sampling error rates across all states and find that both errors occur in existing frameworks. TAPE substantially reduces both errors, showing an improvement in success rate. Best is in **bold** and second-best is underlined.

Framework	Planning Error (%) ↓	Sampling Error (%) ↓	Success Rate (%) ↑
ReAct	50.7 ± 1.8	8.3 ± 1.0	5.0 ± 2.2
Plan-and-Act	$\underline{47.7 \pm 1.8}$	$\underline{4.7 \pm 0.8}$	$\underline{17.0 \pm 3.8}$
TAPE	$\mathbf{36.7 \pm 1.9}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{46.0 \pm 5.0}$

which sample the same number of plans at each step as TAPE, and observe that TAPE still consistently outperforms them. Overall, these results suggest that **TAPE successfully minimizes the planning and sampling errors inherent in existing frameworks**. Notably, we observe substantial performance gains even in scenarios where existing frameworks achieve near-zero success rates, indicating that TAPE can elevate the agent's effective planning capability by enabling the discovery of viable paths in otherwise unsolved instances.

4.2 Detailed Analysis

Planning and Sampling Errors. We compare planning and sampling errors across agent frameworks, as summarized in Table 1. We use Sokoban for this analysis because it admits an oracle shortest-path planner, which enables us to directly estimate the errors. Overall, lower planning and sampling error rates are associated with higher success rates. This result highlights that **two sources of errors in Section 2.3 exist in practice and directly affect task success**. When comparing ReAct and PA, PA reduces the sampling error by roughly 43.4% and slightly reduces the planning error by 3.0%, which is accompanied by a gain in the success rate. This aligns with our claim in Proposition 2 that providing an in-context plan helps mitigate sampling errors while it does not effectively mitigate planning errors. Most importantly, TAPE significantly reduces planning errors while nearly eliminating sampling errors, leading to the largest improvement in the success rate. These results empirically support our analysis in Proposition 3 that **plan selection with a solver in a plan graph reduces planning errors and constrained execution suppresses sampling errors**. The implementation details of our error estimators in Sokoban are provided in Section G.1.

Task Difficulty. We analyze how the performance of each framework varies with task difficulty as shown in Figure 3. As the difficulty increases, all frameworks exhibit a performance decline. Specifically, the success rates of ReAct and PA drop by an average of 55.4% and 52.4%, respectively. In contrast, TAPE consistently maintains higher success rates across all tasks, with an average decrease of 27.7%. This result indicates that **TAPE effectively mitigates planning errors as tasks become hard and a mistaken action becomes more critical**.

Impact of LM's Planning Ability. We evaluate TAPE across several LMs with varying degrees of reasoning (or planning) capabilities (See Section E.1 for LM backbones). In Figure 4a, the models are arranged in ascending order of their capability, as evidenced by the monotonic increase in the success rate of ReAct from 22.0% to 66.0%. As illustrated in Figure 4a, TAPE consistently

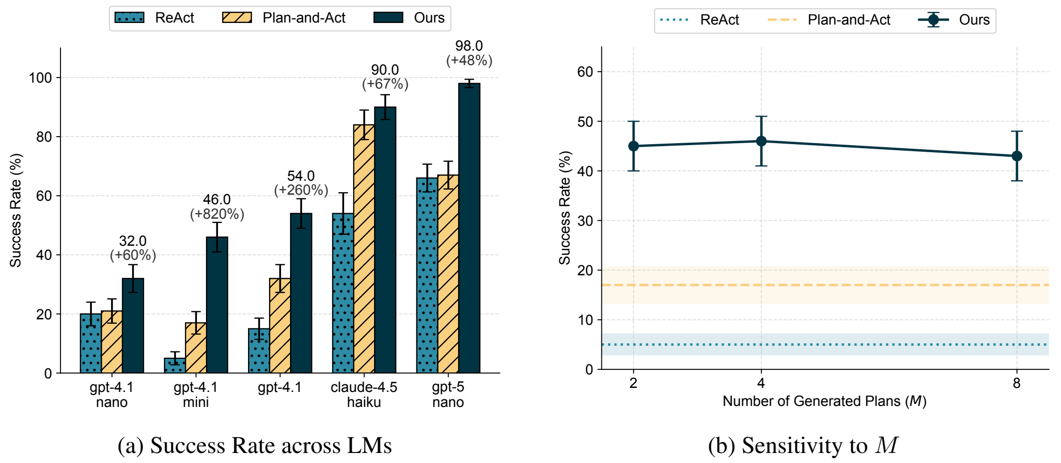


Figure 4: **Cross-model success rates and sensitivity to the number of generated plans in Sokoban. (a)** Success rates of TAPE and baselines across LMs with different planning capabilities. TAPE consistently improves over other frameworks, with larger gains on weaker models, indicating effective mitigation of planning errors. **(b)** Sensitivity of TAPE to the number of generated plans M used to construct the plan graph. The best performance is achieved at $M = 4$, suggesting that moderate plan aggregation via node merging effectively expands the candidate action space.

outperforms the baselines across all models. Notably, our method demonstrates exceptional efficacy on less capable models with limited planning ability; for instance, on `gpt-4.1-mini`, TAPE achieves a relative improvement over ReAct. Even for highly capable models like `gpt-5-nano`, which already exhibit strong baseline performance, our method yields a significant gain, relatively improving +48% over ReAct. This suggests that **TAPE effectively mitigates planning errors due to models with lower reasoning ability**, while eliminating remaining errors in stronger models.

Sensitivity to M. We investigate the impact of the number of generated plans (M) used for graph construction on the success rate. As shown in Figure 4b, we find that $M = 4$ yields the optimal performance. The performance gain observed when increasing M from 2 to 4 is attributed to the aggregation of action candidates; as nodes merge, the number of outgoing edges increases. This reduces the planning errors for each node, supporting Proposition 3. However, we observe a performance decline at $M = 8$. This degradation may stem from the reliance on LMs for graph construction. As the number of paths increases, the LLM struggles to maintain global consistency, leading to reduced graph completeness and accumulated construction errors.

5 CONCLUSION

In this paper, we proposed TAPE, a framework designed to mitigate planning and sampling errors in LM agents. By aggregating multiple plans into a graph and employing Integer Linear Programming, TAPE identifies feasible paths, thereby reduces the planning error. Furthermore, TAPE utilizes constrained execution to substantially reduce sampling error and performs adaptive replanning to handle environment and LM's knowledge mismatches. Our experiments demonstrate that TAPE significantly outperforms the ReAct framework and the Plan-and-Act framework, particularly in complex tasks and for models with limited planning capabilities.

ACKNOWLEDGMENTS

This work was supported by National Science Foundation (NSF) award DMS-2023239, NSF CAREER award CCF-2339978, the Amazon Research Award, the Google Cloud Research Credits Program, and a grant from FuriosaAI. In addition, it used resources through CIS251382 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by NSF awards OAC-2138259, OAC-2138286, OAC-2138307, OAC-2137603, and OAC-2138296.

References

Anthropic. System card: Claude Sonnet 4.5. https://www-cdn.anthropic.com/963373e433e489a87a10c823c52a0a013e9172dd.pdf, 2025.

Xinyun Chen, Chen Liang, Adams Wei Yu, Denny Zhou, Dawn Song, and Quoc V Le. Neural symbolic reader: Scalable integration of distributed and symbolic representations for reading comprehension. In *International Conference on Learning Representations*, 2019.

Soyeon Choi, Kangwook Lee, Oliver Sng, and Joshua M Ackerman. Infected Smallville: How disease threat shapes sociality in LLM agents. *arXiv preprint arXiv:2506.13783*, 2025.

Gautier Dagan, Frank Keller, and Alex Lascarides. Dynamic planning with a LLM. *arXiv preprint arXiv:2308.06391*, 2023.

Lutfi Eren Erdogan, Hiroki Furuta, Sehoon Kim, Nicholas Lee, Suhong Moon, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-Act: Improving planning of agents for long-horizon tasks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2025.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided Language Models. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 10764–10799. PMLR, 2023.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pre-trained Large Language Models to construct and utilize world models for model-based task planning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Weihang Guo, Zachary Kingston, and Lydia E Kavraki. CaStL: Constraints as specifications through LLM translation for long-horizon task and motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2025.

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 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Yilun Hao, Yongchao Chen, Yang Zhang, and Chuchu Fan. Large Language Models can solve real-world planning rigorously with formal verification tools. In *Proceedings of the Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, 2025a.

Yilun Hao, Yang Zhang, and Chuchu Fan. Planning anything with rigor: General-purpose zero-shot planning with LLM-based formalized programming. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025b.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. WebVoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Yafei Hu, Quanting Xie, Vidhi Jain, Jonathan Francis, Jay Patrikar, Nikhil Keetha, Seungchan Kim, Yaqi Xie, Tianyi Zhang, Hao-Shu Fang, et al. Toward general-purpose robots via foundation models: A survey and meta-analysis. *arXiv preprint arXiv:2312.08782*, 2023.

Sukai Huang, Nir Lipovetzky, and Trevor Cohn. Planning in the dark: Llm-symbolic planning pipeline without experts. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can Language Models resolve real-world github issues? In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Frank Joublin, Antonello Ceravola, Pavel Smirnov, Felix Ocker, Joerg Deigmoeller, Anna Belardinelli, Chao Wang, Stephan Hasler, Daniel Tanneberg, and Michael Gienger. CoPAL: corrective planning of robot actions with Large Language Models. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

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 *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.

Minki Kang, Jongwon Jeong, Seanie Lee, Jaewoong Cho, and Sung Ju Hwang. Distilling LLM agent into small models with retrieval and code tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Michael Katz, Harsha Kokel, Kavitha Srinivas, and Shirin Sohrabi Araghi. Thought of Search: Planning with Language Models through the lens of efficiency. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Jeonghye Kim, Sojeong Rhee, Minbeom Kim, Dohyung Kim, Sangmook Lee, Youngchul Sung, and Kyomin Jung. ReflAct: World-grounded decision making in LLM agents via goal-state reflection. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. An LLM compiler for parallel function calling. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.

Xinzhe Li. A review of prominent paradigms for LLM-based agents: Tool use, planning (including RAG), and feedback learning. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, 2025.

Jiaju Lin, Haoran Zhao, Aochi Zhang, Yiting Wu, Huqiuyue Ping, and Qin Chen. AgentSims: An open-source sandbox for large language model evaluation. *arXiv preprint arXiv:2308.04026*, 2023.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. LLM+P: Empowering Large Language Models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023a.

Tengxiao Liu, Zifeng Wang, Jin Miao, I Hsu, Jun Yan, Jiefeng Chen, Rujun Han, Fangyuan Xu, Yanfei Chen, Ke Jiang, et al. Budget-aware tool-use enables effective agent scaling. *arXiv preprint arXiv:2511.17006*, 2025.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

Zhihan Liu, Hao Hu, Shenao Zhang, Hongyi Guo, Shuqi Ke, Boyi Liu, and Zhaoran Wang. Reason for future, act for now: A principled framework for autonomous llm agents with provable sample efficiency. *arXiv preprint arXiv:2309.17382*, 2023b.

Yichuan Ma, Linyang Li, Peiji Li, Xiaozhe Li, Qipeng Guo, Dahua Lin, Kai Chen, et al. Timely Machine: Awareness of time makes test-time scaling agentic. *arXiv preprint arXiv:2601.16486*, 2026.

Soroush Nasiriany, Vitchyr Pong, Steven Lin, and Sergey Levine. Planning with goal-conditioned policies. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Kazuma Obata, Tatsuya Aoki, Takato Horii, Tadahiro Taniguchi, and Takayuki Nagai. Lip-LLM: Integrating linear programming and dependency graph with Large Language Models for multi-robot task planning. *IEEE Robotics and Automation Letters*, 2024.

OpenAI. GPT-4.1. https://openai.com/index/gpt-4-1/, 2025.

Aaron Parisi, Yao Zhao, and Noah Fiedel. TALM: Tool augmented Language Models. *arXiv preprint arXiv:2205.12255*, 2022.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023.

Jinghua Piao, Yuwei Yan, Jun Zhang, Nian Li, Junbo Yan, Xiaochong Lan, Zhihong Lu, Zhiheng Zheng, Jing Yi Wang, Di Zhou, et al. AgentSociety: Large-scale simulation of LLM-driven generative agents advances understanding of human behaviors and society. *arXiv preprint arXiv:2502.08691*, 2025.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. ADaPT: As-needed decomposition and planning with Language Models. In *Proceedings of the 2024 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, 2024.

Haofu Qian, Chenjia Bai, Jiatao Zhang, Fei Wu, Wei Song, and Xuelong Li. Discriminator-guided embodied planning for LLM agent. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

Shuofei Qiao, Runnan Fang, Ningyu Zhang, Yuqi Zhu, Xiang Chen, Shumin Deng, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Agent planning with world knowledge model. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language Models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Max-Philipp B. Schrader. gym-sokoban. https://github.com/mpSchrader/gym-sokoban, 2018.

Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

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.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. AdaPlanner: Adaptive planning from feedback with Language Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics (TACL)*, 2022.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of Large Language Models-a critical investigation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-Solve prompting: Improving zero-shot chain-of-thought reasoning by Large Language Models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023a.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023b.

Zhilin Wang, Yu Ying Chiu, and Yu Cheung Chiu. Humanoid Agents: Platform for simulating human-like generative agents. *arXiv preprint arXiv:2310.05418*, 2023c.

Brandon T Willard and Rémi Louf. Efficient guided generation for Large Language Models. *arXiv preprint arXiv:2307.09702*, 2023.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 2025.

Yu Xia, Yiran Jenny Shen, Junda Wu, Tong Yu, Sungchul Kim, Ryan A Rossi, Lina Yao, and Julian McAuley. SAND: Boosting LLM agents with self-taught action deliberation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 3062–3077, 2025.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Weimin Xiong, Yifan Song, Qingxiu Dong, Bingchan Zhao, Feifan Song, Xun Wang, and Sujian Li. MPO: Boosting LLM agents with meta plan optimization. *arXiv preprint arXiv:2503.02682*, 2025.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. ReWOO: Decoupling reasoning from observations for efficient augmented Language Models. *arXiv preprint arXiv:2305.18323*, 2023.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-Agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024a.

Ziyi Yang, Shreyas S Raman, Ankit Shah, and Stefanie Tellex. Plug in the safety chip: Enforcing constraints for LLM-driven robot agents. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024b.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree Of Thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in Language Models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023b.

Yao Zhang, Zijian Ma, Yunpu Ma, Zhen Han, Yu Wu, and Volker Tresp. WebPilot: A versatile and autonomous multi-agent system for web task execution with strategic exploration. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2025.

Yuanhang Zheng, Peng Li, Ming Yan, Ji Zhang, Fei Huang, and Yang Liu. Budget-constrained tool learning with planning. In *Findings of the Association for Computational Linguistics (ACL)*, 2024.

Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A Rossi, Somdeb Sarkhel, and Chao Zhang. ToolChain*: Efficient action space navigation in Large Language Models with A* search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

A Related Work

Agentic Tasks. Agentic tasks refer to problems in which an agent interacts with the external environment to accomplish the goal Yao et al. (2023b). Recent advancements have demonstrated the possibility of LMs' agentic ability across diverse domains, including robotic manipulation (Hu et al., 2023), graphical user interface (GUI) navigation (He et al., 2024), and software engineering (Jimenez et al., 2024; Yang et al., 2024a). Specifically, in life simulations, agents typically role-play as Non-Player Characters (NPCs), exhibiting human-like behaviors (Park et al., 2023; Choi et al., 2025).

However, the impressive capabilities of these agents often rely on unrestricted resource consumption, which poses a significant barrier to practical deployment. For instance, Park et al. (2023) shows that a two-day simulation of 25 agents incurred thousands of dollars in API costs. Similarly, subsequent studies noted that the latency induced by such complex reasoning loops renders real-time interaction infeasible (Wang et al., 2023c; Lin et al., 2023; Piao et al., 2025). These examples highlight that in real-world scenarios, agents cannot be deployed with unbounded resources.

Instead, agents must operate under constraints, such as monetary budgets, latency limits, or safety protocols (Zheng et al., 2024). Operating under these constraints fundamentally changes the problem: the agent can no longer rely on exhaustive trial-and-error, and small planning or sampling errors can compound into constraint violations and irrecoverable states (Valmeekam et al., 2023; Hao et al., 2025a). Our work focuses on improving the success rate of LM agents in such constrained settings by mitigating such error compounding.

Language Model Agents. LM agents solve complex tasks by interacting with external environments, such as computers and databases, via tools (Yao et al., 2023b; Schick et al., 2023). The ReAct framework (Yao et al., 2023b) established a baseline by interleaving reasoning and acting, allowing agents to respond to immediate observations. However, relying solely on step-by-step generation often makes agents susceptible to planning errors (Valmeekam et al., 2023) and sampling errors (Hao et al., 2025a).

To reduce both errors, some works have been proposed. To reduce planning errors, some works incorporate trained world knowledge models for planning (Qiao et al., 2024; Qian et al., 2025), search over multiple candidate thoughts (Yao et al., 2023a), perform deliberation over candidate action sequences using imagined rollouts or heuristic search (Hao et al., 2023; Liu et al., 2023b; Zhuang et al., 2024; Katz et al., 2024), or use reflection mechanisms to identify failures and revise subsequent behavior (Shinn et al., 2023; Xia et al., 2025; Kim et al., 2025). To mitigate sampling errors, Plan-and-Solve strategies generate a high-level plan before execution to improve global coherence and reduce logical inconsistencies during execution (Wang et al., 2023a; Erdogan et al., 2025; Xu et al., 2023; Xiong et al., 2025). Some works address both errors by combining planning before execution with online plan refinement during execution (Sun et al., 2023; Prasad et al., 2024; Zhang et al., 2025). Despite these improvements, these methods typically do not enforce hard feasibility constraints during plan selection, and execution can still suffer from sampling errors even when the plan is feasible, since action generation remains stochastic and is not constrained to follow the plan.

In contrast, our framework enforces hard feasibility at plan selection by using a formal solver to select a constraint-feasible path over a plan graph constructed from multiple candidate plans, thereby reducing the planning errors. It further reduces sampling errors by constraining decoding to the selected action, with replanning when observations mismatch the plan.

Neural-Symbolic Approaches. Neural-symbolic approaches integrate external solvers into learning-based models to incorporate symbolic structure and formal constraints (Chen et al., 2019; Kambhampati et al., 2024). For LLMs, LMs usually act as a translator that converts natural-language task descriptions into formal planning inputs for external solvers, such as the Planning Domain Definition Language (PDDL) (Liu et al., 2023a; Dagan et al., 2023; Guan et al., 2023; Guo et al., 2025; Huang et al., 2025; Katz et al., 2024), Satisfiability Modulo Theories (SMT) (Hao et al., 2025a;b), or Linear Programming (LP) (Obata et al., 2024). Most prior neural-symbolic LM planning frameworks use LMs primarily as translators that map natural language into a single formal specification or solver-consistent plan, and treat execution as a downstream procedure.

In contrast, our framework keeps a diverse set of candidate plans, folds them into a plan graph, and solves a constraint-feasible path selection problem over this graph. We further integrate planning and execution by constraining decoding to the selected action and replanning upon plan-observation mismatches, which is critical under strict feasibility constraints where deviations are irrecoverable.

B ILP FORMULATION

In G-MDP setting, the ILP formulation is as follows:

$$\max_{x} \sum_{\ell=0}^{L_{\max}-1} \sum_{e\in E} \hat{r}_\theta\big(\mathrm{tgt}(e)\big)\, x_{e,\ell} \tag{7}$$

$$\text{s.t.} \quad \sum_{e\in E} x_{e,\ell} = 1, \quad \forall \ell = 0,\dots,L_{\max}-1, \tag{8}$$

$$\sum_{e\in E:\ \mathrm{src}(e)=v_0} x_{e,0} = 1, \tag{9}$$

$$\sum_{e\in E:\ \mathrm{tgt}(e)\in V_{\mathrm{ter}}} x_{e,L_{\max}-1} = 1, \tag{10}$$

$$\sum_{e\in E:\mathrm{tgt}(e)=v} x_{e,\ell-1} = \sum_{e\in E:\mathrm{src}(e)=v} x_{e,\ell}, \tag{11}$$

$$\forall v \in V, \forall \ell \in \{1,\dots,L_{\max}-1\}\, x_{e,\ell} \in \{0,1\}, \forall e \in E,\ \forall \ell. \tag{12}$$

Equation (7) maximizes the total accumulated reward of the visited nodes. The constraints ensure the validity of the selected path. Specifically, Equation (8) ensures that the agent takes exactly one action at each step. Equation (9) and Equation (10) specify that the path starts at v_0 and ends at one of the terminal nodes in V_{ter} at step $L_{\max}$. Finally, Equation (11) ensures that if the agent arrives at node v at step $\ell - 1$, it must depart from v at step ℓ.

In the constrained G-MDP setting, the agent aims to maximize the expected reward while strictly adhering to a specified budget (e.g., token limit or search depth). We extend the path selection formulation by incorporating the predicted edge costs. Recall that for each edge $e \in E$, our model estimates a cost $\hat{c}_\theta(e)$, representing the expected resource consumption of traversing that edge. Let $\mathbf{b}_t$ denote the remaining budget vector at the current step. We formulate the constrained path selection problem by adding a linear budget constraint to the original ILP:

$$\max_{x} \sum_{\ell=0}^{L_{\max}-1} \sum_{e\in E} \hat{r}_\theta\big(\mathrm{tgt}(e)\big), x_{e,\ell} \tag{13}$$

$$\text{s.t.} \sum_{\ell=0}^{L_{\max}-1} \sum_{e\in E} \hat{\mathbf{c}}_\theta(e), x_{e,\ell} \preceq \mathbf{b}_t, \tag{14}$$

Constraints Equation (8) to Equation (12).

Here, Equation (14) enforces the *budget constraint*, ensuring that the cumulative predicted cost of the selected walk does not exceed $\mathbf{b}_t$. The remaining structural constraints (Equation (8) to Equation (12)) are identical to those in the unconstrained formulation, guaranteeing that the solution forms a valid directed walk from the start node v_0 to a goal node in V_g.If the solver finds no solution satisfying Equation (14) (i.e., all valid paths to the goal exceed the budget), it returns an infeasibility status, which can be handled by a fallback policy or by re-planning with relaxed constraints.

C THEORETICAL ANALYSIS

In this section, we analyze the success probability of agents within the framework in Section 2.2. Based on the planning and execution errors in Section 2.3, we explain how these errors propagate and reduce the overall success probability. We define the success probability as follows.

Definition 2 (Success Probability). *Let $\tau = (s_0, a_0, s_1, \dots)$ denote a random trajectory generated by the agent, and let $T_g \coloneqq \min\{t \geq 0 : R_g(s_t) = 1\}$ be the first goal-reaching step, with the convention $\min \emptyset \coloneqq \infty$. We define the success event as $\mathcal{S} \coloneqq \{T_g < \infty\} \cap \bigcap_{t=0}^{T_g - 1} \{a_t \in \mathcal{A}_{\text{viable}}(s_t)\}$, and we refer to $\Pr(\mathcal{S})$ as the success probability.*

C.1 EXISTING FRAMEWORKS

From Section 2, we derive the upper bound of the success probability of the ReAct framework as follows.

Proposition 1. *Assume that any successful trajectory must take at least T action selections. Then, the ReAct success probability is bounded by U_{ReAct}:*

$$\Pr(\mathcal{S}) \leq U_{\text{ReAct}} \coloneqq \Big((1 - \epsilon_{\text{p}})(1 - \epsilon_{\text{s}}\delta_{\text{b}}) + \epsilon_{\text{p}}\epsilon_{\text{s}}\delta_{\text{r}} \Big)^T, \tag{15}$$

where equality holds when the task exactly ends at T. If $(1 - \epsilon_{\text{p}})\delta_{\text{b}} \geq \epsilon_{\text{p}}\delta_{\text{r}}$, U_{ReAct} increases as ϵ_{p} and ϵ_{s} decrease.

Proposition 1 implies that as T grows, per-step errors compound and sharply reduce the overall success rate. Also, if $(1 - \epsilon_{\text{p}})\delta_{\text{b}} \geq \epsilon_{\text{p}}\delta_{\text{r}}$, the success probability can be increasing by reducing planning and sampling error. $(1 - \epsilon_{\text{p}})\delta_{\text{b}} \geq \epsilon_{\text{p}}\delta_{\text{r}}$ is reasonable since deviations from the planned one are more likely to break viability than to recover it in agents.

Plan-and-Act (PA) framework utilizes a pre-generated plan as in-context guidance. This guidance effectively reduces execution stochasticity, lowering the sampling error. This comparison is formalized in Proposition 2.

Proposition 2. *Let $\alpha \coloneqq \Pr(z(s_t) = \hat{s}_t^{\hat{\tau}})$ denote the probability that the plan is aligned at each step. Under the same assumptions in Proposition 1, we have the upper bound of success probability for PA as $U_{\text{PA}} \coloneqq \big((1 - \epsilon_{\text{p}})(1 - \epsilon_{\text{s,PA}}\delta_{\text{b}}) + \epsilon_{\text{p}}\epsilon_{\text{s,PA}}\delta_{\text{r}} \big)^T$, where $\epsilon_{\text{s}_{\text{PA}}} = (1 - \alpha p_{\text{follow}})\epsilon_{\text{s}}$. Consequently, we have the following:*

$$U_{\text{PA}} \geq U_{\text{ReAct}}, \tag{16}$$

where equality holds when $\alpha = 0$ or $p_{\text{follow}} = 0$.

Proposition 2 suggests that the upper bound of success probability of PA is higher than that of ReAct due to reducing the sampling errors from ϵ_{s} to $\epsilon_{\text{s}_{\text{PA}}}$. However, note that the planning error ϵ_{p} remains.

C.2 OUR FRAMEWORK

TAPE aggregates N sampled plans into a plan graph. This structure increases the diversity of action candidates at each step. Let v_t denote the planning node in the plan graph corresponding to the current state s_t, i.e., $v_t = f_\theta(s_t)$. We define the set of candidate actions at v_t as $\hat{\mathcal{A}}(v_t) \coloneqq \{\bar{a} \in \mathcal{A} \mid \exists\, (v_t \xrightarrow{\bar{a}} u_{t+1}) \in E\}$, and let $d(v_t) \coloneqq |\hat{\mathcal{A}}(v_t)|$ be the number of distinct actions proposed by the aggregated plans. Note that $d(v_t) \geq 1$ for any non-terminal node v_t.

Proposition 3. *Assume that a task requires T steps, the external solver selects a viable action whenever one exists, and constrained decoding eliminates sampling error (i.e., $\epsilon_{\text{s}} \approx 0$). Then, the upper bound of the success probability for TAPE is given by $U_{\text{ours}} \coloneqq \prod_{t=0}^{T-1} \big(1 - (\epsilon_{\text{p}})^{d(v_t)}\big)$. Consequently, we have*

$$U_{\text{ours}} \geq U_{\text{PA}} \geq U_{\text{ReAct}}, \tag{17}$$

where the equality between U_{ours} and U_{PA} holds if and only if $d(v_t) = 1$ and $\epsilon_{\text{s,PA}}\delta_{\text{b}} = \epsilon_{\text{s,PA}}\delta_{\text{r}} = 0$.

Proposition 3 confirms that TAPE theoretically guarantees the highest success probability among the compared frameworks by exponentially reducing planning errors from ϵ_{p} to $(\epsilon_{\mathrm{p}})^{d(v_t)}$ via selecting a viable action from multiple candidates, while also eliminating sampling errors $\epsilon_{\mathrm{s}} \approx 0$ via constrained decoding.

D PROOFS OF THEORETICAL ANALYSIS

In this section, we provide detailed proofs for the propositions presented in Appendix C. We define the event of selecting a viable action at step t as $\mathcal{V}_t := \{a_t \in \mathcal{A}_{\text{viable}}(s_t)\}$. According to Definition 2, the success probability for a task requiring T_g steps is given by $\Pr(\mathcal{S}) = \Pr(\cap_{t=0}^{T_g-1} \mathcal{V}_t)$. Assuming the Markov property and independence of errors at each step, we focus on deriving the single-step success probability $P(\mathcal{V}_t)$.

D.1 PROOF OF PROPOSITION 1

Proof. Let $\hat{a}_t$ be the action planned by the agent (intent), and a_t be the action actually executed. Also, let E_{plan} be the event that the planned action is viable, i.e., $\hat{a}_t \in \mathcal{A}_{\text{viable}}(s_t)$, and let E_{exec} be the event that the executed action matches the plan, i.e., $a_t = \hat{a}_t$. We define two independent events based on the error sources as follows: $\Pr(E_{\text{plan}}) = 1 - \epsilon_{\mathrm{p}}$ and $\Pr(E^c_{\text{plan}}) = \epsilon_{\mathrm{p}}$, and $\Pr(E^c_{\text{exec}}) = \epsilon_{\mathrm{s}}$.

The viable action event $\mathcal{V}_t$ can occur in two disjoint cases. First, when the plan is viable, the executed action a_t remains viable if execution succeeds (E_{exec}) or if execution fails but does not break viability. The latter occurs with probability $1 - \delta_{\mathrm{b}}$.

$$\Pr(\mathcal{V}_t \mid E_{\text{plan}}) = (1 - \epsilon_{\mathrm{s}}) \cdot 1 + \epsilon_{\mathrm{s}} \cdot (1 - \delta_{\mathrm{b}}) = 1 - \epsilon_{\mathrm{s}}\delta_{\mathrm{b}}. \tag{18}$$

Second, when the plan is non-viable, the executed action a_t becomes viable only if execution fails (E^c_{exec}) and accidentally recovers viability (with probability δ_{r}).

$$\Pr(\mathcal{V}_t \mid E^c_{\text{plan}}) = (1 - \epsilon_{\mathrm{s}}) \cdot 0 + \epsilon_{\mathrm{s}} \cdot \delta_{\mathrm{r}} = \epsilon_{\mathrm{s}}\delta_{\mathrm{r}}. \tag{19}$$

By the Law of Total Probability, the probability of selecting a viable action at step t is:

$$\begin{aligned}
\Pr(\mathcal{V}_t) &= \Pr(\mathcal{V}_t \mid E_{\text{plan}}) \Pr(E_{\text{plan}}) + \Pr(\mathcal{V}_t \mid E^c_{\text{plan}}) \Pr(E^c_{\text{plan}}) && (20)\\
&= (1 - \epsilon_{\mathrm{s}}\delta_{\mathrm{b}})(1 - \epsilon_{\mathrm{p}}) + (\epsilon_{\mathrm{s}}\delta_{\mathrm{r}})\epsilon_{\mathrm{p}} && (21)\\
&= (1 - \epsilon_{\mathrm{p}})(1 - \epsilon_{\mathrm{s}}\delta_{\mathrm{b}}) + \epsilon_{\mathrm{p}}\epsilon_{\mathrm{s}}\delta_{\mathrm{r}}. && (22)
\end{aligned}$$

Since the task requires at least $T \leq T_g$ steps, the overall success probability is the product of single-step probabilities up to T:

$$\begin{aligned}
\Pr(\mathcal{S}) &\leq \prod_{t=0}^{T-1} \Pr(\mathcal{V}_t) && (23)\\
&= \Big((1 - \epsilon_{\mathrm{p}})(1 - \epsilon_{\mathrm{s}}\delta_{\mathrm{b}}) + \epsilon_{\mathrm{p}}\epsilon_{\mathrm{s}}\delta_{\mathrm{r}}\Big)^T && (24)\\
&=: U_{\text{ReAct}}. && (25)
\end{aligned}$$

Equality holds if the task ends exactly at step T, i.e. $T = T_g$.

To show monotonicity, let $f(\epsilon_{\mathrm{s}}) = U_{\text{ReAct}}$. We take the derivative with respect to ϵ_{s}

$$\frac{\partial}{\partial \epsilon_{\mathrm{s}}} f(\epsilon_{\mathrm{s}}) = \left(-(1 - \epsilon_{\mathrm{p}})\delta_{\mathrm{b}} + \epsilon_{\mathrm{p}}\delta_{\mathrm{r}}\right)^T. \tag{26}$$

For the success probability to increase as ϵ_{s} decreases (i.e., derivative is negative), we require

$$-(1 - \epsilon_{\mathrm{p}})\delta_{\mathrm{b}} + \epsilon_{\mathrm{p}}\delta_{\mathrm{r}} \leq 0 \iff (1 - \epsilon_{\mathrm{p}})\delta_{\mathrm{b}} \geq \epsilon_{\mathrm{p}}\delta_{\mathrm{r}}. \tag{27}$$

Similarly, we can show that the upper bound of success probability increases as ϵ_{p} decreases. This confirms the condition stated in Proposition 1. □

D.2 Proof of Proposition 2

Proof. According to the Plan-and-Act mechanism defined in Equation (2), the agent behavior is divided into two cases based on plan alignment and following probability.

First, following the plan occurs when the plan is aligned ($z(s_t) = \hat{s}_t^{\hat{\tau}}$) and the agent chooses to follow it. We denote this event as F. The probability of this event is $\Pr(F) = \alpha p_{\text{follow}}$. In this case, $a_t = \bar{a}_t$. Since $\bar{a}_t$ is deterministically selected from the pre-generated plan, the execution sampling error is eliminated ($\epsilon_{\text{s}} \to 0$). However, the pre-generated plan itself is subject to the same planning error ϵ_{p} as the ReAct framework. Thus, the success probability for this case is derived by setting $\epsilon_{\text{s}} = 0$ in the ReAct single-step probability as

$$\Pr(\mathcal{V}_t \mid F) = (1 - \epsilon_{\text{p}})(1 - 0) + \epsilon_{\text{p}}(0) = 1 - \epsilon_{\text{p}}. \tag{28}$$

Second, fallback to ReAct occurs when the plan is not aligned or the agent chooses not to follow. We denote this event as R. The probability is $\Pr(R) = 1 - \alpha p_{\text{follow}}$. We denote this event as R. In this case, $a_t = a_t^{\text{ReAct}}$, and the success probability is identical to that of the ReAct framework derived in Proposition 1 as

$$\Pr(\mathcal{V}_t \mid R) = (1 - \epsilon_{\text{p}})(1 - \epsilon_{\text{s}}\delta_{\text{b}}) + \epsilon_{\text{p}}\epsilon_{\text{s}}\delta_{\text{r}}. \tag{29}$$

By the Law of Total Probability, the single-step success probability for PA is

$$\begin{aligned}\Pr(\mathcal{V}_t) &= \Pr(F)\Pr(\mathcal{V}_t \mid F) + \Pr(R)\Pr(\mathcal{V}_t \mid R) &(30)\\ &= (\alpha p_{\text{follow}})(1 - \epsilon_{\text{p}}) + (1 - \alpha p_{\text{follow}})\Big[(1 - \epsilon_{\text{p}})(1 - \epsilon_{\text{s}}\delta_{\text{b}}) + \epsilon_{\text{p}}\epsilon_{\text{s}}\delta_{\text{r}}\Big]. &(31)\end{aligned}$$

We can rearrange the terms to isolate ϵ_{s}:

$$\begin{aligned}\Pr(\mathcal{V}_t) &= (\alpha p_{\text{follow}})(1 - \epsilon_{\text{p}}) + (1 - \alpha p_{\text{follow}})(1 - \epsilon_{\text{p}}) + (1 - \alpha p_{\text{follow}})\left[-(1 - \epsilon_{\text{p}})\epsilon_{\text{s}}\delta_{\text{b}} + \epsilon_{\text{p}}\epsilon_{\text{s}}\delta_{\text{r}}\right] &(32)\\ &= (1 - \epsilon_{\text{p}}) - (1 - \epsilon_{\text{p}})(1 - \alpha p_{\text{follow}})\epsilon_{\text{s}}\delta_{\text{b}} + \epsilon_{\text{p}}(1 - \alpha p_{\text{follow}})\epsilon_{\text{s}}\delta_{\text{r}}. &(33)\end{aligned}$$

Now, we define the effective sampling error for PA as $\epsilon_{\text{s,PA}} := (1 - \alpha p_{\text{follow}})\epsilon_{\text{s}}$. Substituting this into Equation (33), we have

$$\Pr(\mathcal{V}_t) = (1 - \epsilon_{\text{p}})(1 - \epsilon_{\text{s,PA}}\delta_{\text{b}}) + \epsilon_{\text{p}}\epsilon_{\text{s,PA}}\delta_{\text{r}}. \tag{34}$$

Finally, since the task requires T steps, we take the product over $t = 0$ to $T - 1$, we have

$$U_{\text{PA}} = \Big((1 - \epsilon_{\text{p}})(1 - \epsilon_{\text{s,PA}}\delta_{\text{b}}) + \epsilon_{\text{p}}\epsilon_{\text{s,PA}}\delta_{\text{r}}\Big)^T. \tag{35}$$

The comparison $U_{\text{PA}} \geq U_{\text{ReAct}}$ follows directly from $\epsilon_{\text{s,PA}} \leq \epsilon_{\text{s}}$ and equality holds when $\alpha p_{\text{follow}} = 0$. □

D.3 Proof of Proposition 3

Proof. Let $\hat{\mathcal{A}}(v_t)$ be the set of $d(v_t)$ candidate actions generated at step t. A planning failure occurs for the entire set only if *all* candidates in $\hat{\mathcal{A}}(v_t)$ are non-viable. Assuming the generation of each candidate is independent given the state, the probability that all candidates fail is $(\epsilon_{\text{p}})^{d(v_t)}$. Consequently, the probability that there exists at least one viable action in the candidate set is:

$$\Pr(\exists a \in \hat{\mathcal{A}}(v_t) : a \in \mathcal{A}_{\text{viable}}) = 1 - (\epsilon_{\text{p}})^{d(v_t)}. \tag{36}$$

Based on the assumptions in Proposition 3, the external solver successfully identifies a viable action if one exists in the set and constrained execution eliminates sampling error (i.e., $\epsilon_{\text{s}} \approx 0$), ensuring the selected action is executed exactly. Thus, the single-step success probability for TAPE is exactly $1 - (\epsilon_{\text{p}})^{d(v_t)}$. Taking the product over T steps yields

$$U_{\text{Ours}} = \prod_{t=0}^{T-1} \Big(1 - (\epsilon_{\text{p}})^{d(v_t)}\Big). \tag{37}$$

From Proposition 2, we already established $U_{\mathrm{PA}} \geq U_{\mathrm{ReAct}}$. We now focus on proving $U_{\mathrm{Ours}} \geq U_{\mathrm{PA}}$.

Since we assume $(1-\epsilon_{\mathrm{p}})\delta_{\mathrm{b}} \geq \epsilon_{\mathrm{p}}\delta_{\mathrm{r}}$, the success probability of PA at step t is maximized when the effective sampling error is zero ($\epsilon_{\mathrm{s,PA}} = 0$). Substituting $\epsilon_{\mathrm{s,PA}} = 0$ into the term for PA gives the upper bound:

$$(1-\epsilon_{\mathrm{p}})(1-\epsilon_{\mathrm{s,PA}}\delta_{\mathrm{b}}) + \epsilon_{\mathrm{p}}\epsilon_{\mathrm{s,PA}}\delta_{\mathrm{r}} \leq 1-\epsilon_{\mathrm{p}}. \tag{38}$$

Next, for TAPE, since $d(v_t) \geq 1$ and $0 \leq \epsilon_{\mathrm{p}} < 1$, it follows that $(\epsilon_{\mathrm{p}})^{d(v_t)} \leq \epsilon_{\mathrm{p}}$. Therefore, we have

$$1-(\epsilon_{\mathrm{p}})^{d(v_t)} \geq 1-\epsilon_{\mathrm{p}}. \tag{39}$$

Combining Equation (38) and Equation (39), we have

$$1-(\epsilon_{\mathrm{p}})^{d(v_t)} \geq 1-\epsilon_{\mathrm{p}} \geq (1-\epsilon_{\mathrm{p}})(1-\epsilon_{\mathrm{s,PA}}\delta_{\mathrm{b}}) + \epsilon_{\mathrm{p}}\epsilon_{\mathrm{s,PA}}\delta_{\mathrm{r}}. \tag{40}$$

Since Equation (40) satisfies for all steps $t = 0 \ldots T-1$, we conclude $U_{\mathrm{Ours}} \geq U_{\mathrm{PA}}$. Also, we can see that $1-(\epsilon_{\mathrm{p}})^{d(v_t)} = (1-\epsilon_{\mathrm{p}})(1-\epsilon_{\mathrm{s,PA}}\delta_{\mathrm{b}}) + \epsilon_{\mathrm{p}}\epsilon_{\mathrm{s,PA}}\delta_{\mathrm{r}}$ holds when $d(v_t) = 1$ and $\epsilon_{\mathrm{s,PA}}\delta_{\mathrm{b}} = \epsilon_{\mathrm{s,PA}}\delta_{\mathrm{r}} = 0$. □

E Experiment Details

E.1 Experimental Setup

Tasks and Datasets. Tasks and Datasest details are explained below. The detailed statistics are summarized in Table.

- **Sokoban (Schrader, 2018).** is a task pushing all boxes onto goal tiles using four primitive actions (`U`, `D`, `L`, `R`). To make the task irrecoverable, we define success as solving the instance within two steps of the optimal solution length. We construct two difficulty by the controlling optimal step count $T^\star$: *easy* instances have $T^\star = 6$, while *hard* instances have $T^\star = 10$. For evaluation, we construct 10 maps for each task difficulty and run 10 times for each map.
- **ALFWorld (Shridhar et al., 2021).** This task is a synthetic text-based household embodied tasks where the agent executes environment-specific actions to complete a given goal. To align with our assumption that the agent knows the abstract world model in agentic tasks, we modify the environment to provide the object availability information for each location and the basic information of the dependency of the action. We define difficulty by the action budget relative to the optimal length $T^\star$: *easy* instances use a looser budget, while *hard* instances use a tighter budget. For evaluation,
- **GSM-Hard (Gao et al., 2023)** is a mathemathical reasoning problem dataset that converts some numerical values larger so that the problem cannot be solve easily without arithmetic tools. We cast as an agentic task by equipping the agent with arithmetic tools ($+$, $-$, $\times$, $/$). For each operator, we provide two tool variants: a *fast* tool with lower success probability and a *slow* tool with higher success probability. We define *hard* tasks as those with a tight time budget that incentivizes using fast tools, and *easy* tasks as those with a loose time budget.
- **MuSiQue (Trivedi et al., 2022)** is a factual multi-hop reasoning dataset, where the agent can query a retriever to obtain supporting information. To create an agentic setting with explicit cost–quality tradeoffs, we construct five synthetic retrievers ranging from fast, cheap, but inaccurate to slow, expensive, but accurate. Analogous to GSM-8K, we define *hard* tasks as those requiring fast and cheap solving, and *easy* tasks as those allowing a larger time budget and higher retrieval cost.

Language Model Backbones. We use five LM backbones: `gpt-4.1-nano`, `gpt-4.1-mini`, `gpt-4.1` (OpenAI, 2025), `gpt-5-nano` (Singh et al., 2025), and `claude-4.5-haiku` (Anthropic, 2025).

Inference. We set the sampling temperature to 0.3 for inference in all experiments except `gpt-5-nano` models. For `gpt-5-nano`, as we do not change the temperature, we set the default value (it is unknown). Other parameters, such as top-p and repetition penaltiy, are set as the default value (both are 1).

E.2 Baselines

ReAct (Yao et al., 2023b). ReAct is the framework that make LM agents solve the tasks by interleaving thought and actions to interact with the external environments. There are various types of implementation for the thoughts and acts (Yao et al., 2023b; Shinn et al., 2023; Kim et al., 2025), we adopt the prompting technique from Xiong et al. (2025). Detailed prompts are summarized in Prompt E.1.

Plan-and-Act (Wang et al., 2023a; Erdogan et al., 2025). Plan-and-Act (PA) first generates the a plan and use it as in-context prompt. Then, LM agent refers the plan to interact with the environment. For inteaction part, the prompt is the same as ReAct. Detailed prompts for the plan generation phase are summarized in Prompt E.2.

Prompt E.1: ReAct (Sokoban)

Interact with Sokoban environment to solve a task (placing every box onto goals.)

Sokoban rules (task + mechanics)
- Task objective: place every box onto goals. The puzzle is solved when all boxes are on goals.
- Action mechanics: each action moves the player exactly one cell in the chosen direction (U/D/L/R).
- U: move up, e.g., (x, y) -¿ (x, y + 1)
- D: move down, e.g., (x, y) -¿ (x, y - 1)
- L: move left, e.g., (x, y) -¿ (x - 1, y)
- R: move right, e.g., (x, y) -¿ (x + 1, y)
- The action name must match the coordinate change (U increases y, D decreases y, R increases x, L decreases x).
- Walls: the player cannot move into a wall cell. If a wall occupies the destination cell, the action is invalid and the player does not move.
- Boxes: if the destination cell has a box, the player attempts to push it one cell further in the same direction.
- The push succeeds only if the cell behind the box is empty floor or a goal.
- If the cell behind the box is a wall or another box, the push is invalid and nothing moves.
- A push is only possible when the player is on the cell immediately adjacent to the box from the opposite side of the push direction.
- The player cannot pull boxes and cannot push two boxes at once.
- Some pushes can create deadlocks (for example, pushing a box into a corner where it cannot reach any goal).

Examples (coordinate outcomes, using U/D/L/R only):
- Empty move, R: player at (x, y), no wall/box at (x + 1, y). Action R -¿ player at (x + 1, y); box on goal location unchanged (goals are unaffected by moves).
- Empty move, L: player at (x, y), no wall/box at (x - 1, y). Action L -¿ player at (x - 1, y); box on goal location unchanged (goals are unaffected by moves).
- Empty move, U: player at (x, y), no wall/box at (x, y + 1). Action U -¿ player at (x, y + 1); box on goal location unchanged (goals are unaffected by moves).
- Empty move, D: player at (x, y), no wall/box at (x, y - 1). Action D -¿ player at (x, y - 1); box on goal location unchanged (goals are unaffected by moves).
- Wall block, U: player at (x, y), wall at (x, y + 1). Action U -¿ invalid, player stays at (x, y); box on goal location unchanged.
- Wall block, D: player at (x, y), wall at (x, y - 1). Action D -¿ invalid, player stays at (x, y); box on goal location unchanged.
- Wall block, L: player at (x, y), wall at (x - 1, y). Action L -¿ invalid, player stays at (x, y); box on goal location unchanged.
- Wall block, R: player at (x, y), wall at (x + 1, y). Action R -¿ invalid, player stays at (x, y); box on goal location unchanged.
- Push succeeds, R: player at (x, y), box at (x + 1, y), no wall/box at (x + 2, y). Action R -¿ player at (x + 1, y), box moves to (x + 2, y); if (x + 2, y) is a goal, box on goal location becomes (x + 2, y), otherwise unchanged.
- Push succeeds, L: player at (x, y), box at (x - 1, y), no wall/box at (x - 2, y). Action L -¿ player at (x - 1, y), box moves to (x - 2, y); if (x - 2, y) is a goal, box on goal location becomes (x - 2, y), otherwise unchanged.
- Push succeeds, U: player at (x, y), box at (x, y + 1), no wall/box at (x, y + 2). Action U -¿ player at (x, y + 1), box moves to (x, y + 2); if (x, y + 2) is a goal, box on goal location becomes (x, y + 2), otherwise unchanged.
- Push succeeds, D: player at (x, y), box at (x, y - 1), no wall/box at (x, y - 2). Action D -¿ player at (x, y - 1), box moves to (x, y - 2); if (x, y - 2) is a goal, box on goal location becomes (x, y - 2), otherwise unchanged.
- Push blocked by wall, R: player at (x, y), box at (x + 1, y), wall at (x + 2, y). Action R -¿ invalid, player stays at (x, y), box stays at (x + 1, y); box on goal location unchanged.

- Push blocked by wall, L: player at (x, y), box at (x - 1, y), wall at (x - 2, y). Action L -¿ invalid, player stays at (x, y), box stays at (x - 1, y); box on goal location unchanged.
- Push blocked by wall, U: player at (x, y), box at (x, y + 1), wall at (x, y + 2). Action U -¿ invalid, player stays at (x, y), box stays at (x, y + 1); box on goal location unchanged.
- Push blocked by wall, D: player at (x, y), box at (x, y - 1), wall at (x, y - 2). Action D -¿ invalid, player stays at (x, y), box stays at (x, y - 1); box on goal location unchanged.
- Push blocked by box, R: player at (x, y), box at (x + 1, y), another box at (x + 2, y). Action R -¿ invalid, player stays at (x, y), boxes stay at (x + 1, y) and (x + 2, y); box on goal location unchanged.
- Push blocked by box, L: player at (x, y), box at (x - 1, y), another box at (x - 2, y). Action L -¿ invalid, player stays at (x, y), boxes stay at (x - 1, y) and (x - 2, y); box on goal location unchanged.
- Push blocked by box, U: player at (x, y), box at (x, y + 1), another box at (x, y + 2). Action U -¿ invalid, player stays at (x, y), boxes stay at (x, y + 1) and (x, y + 2); box on goal location unchanged.
- Push blocked by box, D: player at (x, y), box at (x, y - 1), another box at (x, y - 2). Action D -¿ invalid, player stays at (x, y), boxes stay at (x, y - 1) and (x, y - 2); box on goal location unchanged.

Instruction
You are the player in a Sokoban environment, and your goal is to place every box on a goal within a limited number of actions (within step remaining). That means you need to make the same number of boxes on goals by placing all boxes onto goals. If a box is on a goal, it is considered satisfied.
At each turn, you will receive the current observation.
Observation format (all coordinates are (x, y)):
- wall location: (x1, y1), (x2, y2), ...
- player location: (x, y)
- box location: (x3, y3), ...
- goal location: (x4, y4), ...
- box on goal location: (x5, y5), ...
- Step remaining: ¡steps_remaining¿
The "box location" list includes only boxes not on goals. The "goal location" list includes all goals. You may choose between two outputs: "Thought" or "Action".

When you choose "Thought", you must:
Plan the full solution (overall path) so that all boxes reach goals within the remaining steps, using the Sokoban rules (task + mechanics).
From that full plan, explicitly predict only the next 1 step: how the immediate action will change the player location, box location, and box on goal location.
Based on that planning and the current observation, decide the immediate next action to take.
IMPORTANT:
- Your Thought MUST match the given observation exactly. No hallucination about positions or adjacency is allowed.
- If a push is feasible immediately, you MUST choose the move that pushes.
- If you planned a push in the previous Thought and the current observation still allows that push, you MUST continue and execute it.
- Do NOT rewrite the plan from scratch unless the environment changed and the old plan became infeasible.
Your output must follow exactly:
Thought: ¡your reasoning¿
Action: ¡U—D—L—R¿"

Prompt E.2: Plan-and-Act (Sokoban)

Interact with Sokoban environment to solve a task (placing every box onto goals.)
{{sokoban_rule}}

Instruction You are the player in a Sokoban environment, and your goal is to place every box on a goal within a limited number of actions (within step remaining). That means you need to make the same number of boxes on goals by placing all boxes onto goals. If a box is on a goal, it is considered satisfied.
At each turn, you will receive the current observation. Observation format (all coordinates are (x, y)): - wall location: (x1, y1), (x2, y2), ... - player location: (x, y) - box location: (x3, y3), ... - goal location: (x4, y4), ... - box on goal location: (x5, y5), ... - Step remaining: ¡steps_remaining¿ The ”box location” list includes only boxes not on goals. The ”goal location” list includes all goals.
You are in planning mode. Using the Sokoban rules, plan the full solution (overall path) so that all boxes reach goals within the remaining steps, using the Sokoban rules (task + mechanics). Based on this plan, you generate the full sequence of actions.

F Implementation Details of TAPE

F.1 Prompt Examples

Prompts used in our framework, especially Sokoban task, are introduced in Prompt F.1, Prompt F.2, Prompt F.3, and Prompt F.4. Other prompts for are introduced in

Prompt F.1: State Projector

Extract the current exact player location, box locations, goal locations, and box on the goals' location, and prompt from the given history.

Prompt F.2: Sample Plans

Interact with Sokoban environment to solve a task (placing every box onto goals.)
{{sokoban_rule}}

Instruction

You are the player in a Sokoban environment, and your goal is to place every box on a goal within a limited number of actions (within step remaining). That means you need to make the same number of boxes on goals by placing all boxes onto goals. If a box is on a goal, it is considered satisfied.

At each turn, you will receive the current observation. Observation format (all coordinates are (x, y)): - wall location: (x1, y1), (x2, y2), ... - player location: (x, y) - box location: (x3, y3), ... - goal location: (x4, y4), ... - box on goal location: (x5, y5), ... - Step remaining: ¡steps_remaining¿ The "box location" list includes only boxes not on goals. The "goal location" list includes all goals.

You are in planning mode. Using the Sokoban rules, plan the full solution (overall path) so that all boxes reach goals within the remaining steps. Generate {{num_plans}} diverse and valid plans. For each plan, you must **precise** reason based on the location of player, box, goal, and the wall, and the Sokoban rules (task + mechanics). Then, generate the full solution (overall path) so that all boxes reach goals within the remaining steps, using the Sokoban rules (task + mechanics). Based on this plan, you generate the full sequence of actions. Generate as **diverse** as possible while maintaining success within given remaining steps. Each plan MUST have a full action sequence (at most remaining steps of actions). Provide exactly {{num_plans}} plans labeled "Plan 1:" ,..., "Plan {{num_plans}}:".

IMPORTANT: Use the action-coordinate rules exactly. Actions update the player location: U (moving up) moves the player (x, y) -¿ (x, y+1), D (moving down) moves the player (x, y) -¿ (x, y-1), R (moving right) moves the player (x, y) -¿ (x+1, y), L (moving left) moves the player (x, y) -¿ (x-1, y). Always align your verbal directions (up/down/left/right) with these coordinate changes. When describing relative positions, use the coordinate conventions: "above" means larger y, "below" means smaller y, "right" means larger x, "left" means smaller x. Validate each step in the plan by explicitly computing the next (x, y) from the action; if the destination cell is a box, check the cell behind it and treat the move as a valid push if that cell is empty or a goal, otherwise the move is invalid and must not appear in the action sequence. If the player pushes a box, the box will move one cell in the same direction as the player unless blocked by a wall or another box. After any move (including a push), the player and any box must occupy different cells; they can never share the same location.

Prompt F.3: Build Plan Graph

Simulate Sokoban plans, produce per-plan step sequences and generate the graph.

{{sokoban_rule}}

IMPORTANT: Use the action-coordinate rules exactly. Actions update the player location: U (moving up) moves the player (x, y) -¿ (x, y+1), D (moving down) moves the player (x, y) -¿ (x, y-1), R (moving right) moves the player (x, y) -¿ (x+1, y), L (moving left) moves the player (x, y) -¿ (x-1, y). Always align your verbal directions (up/down/left/right) with these coordinate changes. When describing relative positions, use the coordinate conventions: "above" means larger y, "below" means smaller y, "right" means larger x, "left" means smaller x. Validate each step in the plan by explicitly computing the next (x, y) from the action; if the destination cell is a box, check the cell behind it and treat the move as a valid push if that cell is empty or a goal, otherwise the move is invalid and must not appear in the action sequence. If the player pushes a box, the box will move one cell in the same direction as the player unless blocked by a wall or another box. After any move (including a push), the player and any box must occupy different cells; they can never share the same location.

Instructions

- Simulate each plan step-by-step using the Sokoban rules and the observation. - Build each plan's step sequence as alternating entries: node -¿ action -¿ node -¿ action -¿ ... - A node entry must include a node id and the predicted observation text. - Node ids must be unique across all plans (no duplicate node_id between plans). - IMPORTANT: Observations must include ONLY player location and box location (no walls, no goals, no box-on-goal). - The observation text must start with a short "Thought: ..." line that explains how the previous action moves player/box while considering walls. - An action entry must include only the action (U/D/L/R). No separate thought key in action entries. - Preserve ALL actions from every plan (do not drop steps).

Return JSON only: { "plan_sequences": [{ "plan_id": "plan1", "steps": [{ "kind": "node", "node_id": "nodeX", "thought": "Initial states" "observation": "player location: (x, y)
n box location: (x1, y1), ..." }, { "kind": "action", "action": "U" }, { "kind": "node", "node_id": "nodeY", "thought": "After moving up (U), player is at (x, y+1) unless blocked; box moves to (x1, y1+1) if pushed." "observation": "player location: (x, y+1)
n box location: (x1, y1+1), ..." }] }] },

- After all plan sequences are built, merge identical nodes when the observation text matches exactly. - Preserve ALL actions from every plan (do not drop steps). - Construct the full graph (nodes + edges) from the merged nodes. - Generate a thought for each edge based on the transition between observations. - Mark goal nodes explicitly with "is_goal": true in full_graph nodes. - JSON keys must appear in this order: reasoning, merge_log, full_graph. - In merge_log entries, put "reason" first, then "kept_node", then "merged_nodes".

{ "reasoning": "overall reasoning for node merging and graph construction", "merge_log": [{ "reason": "same observation text", "kept_node": "nodeX", "merged_nodes": ["nodeXX", ...] }], "full_graph": { "nodes": [{ "id": "nodeX", "observation": "player location: (x, y)
nbox location: (x1, y1), ...", "is_goal": true or false }], "edges": [{ "from": "nodeX", "to": "nodeY", "thought": "why this step is taken", "action": "¡U/D/L/R¿" }] } }

Prompt F.4: Annotate Graph

You score all the states in the graph. Higher score means closer to solving. Goal: move all boxes onto goals (use goal locations from the observation). Assign score 1.0 to goal states. Assign score -1.0 if, starting from this state, there is no valid sequence of actions that can ever reach any goal state (deadlocked/unreachable). All other states should have score 0.0. Consider wall locations, blocked pushes, and required pushing routes when judging reachability. Only use scores -1.0, 0.0, or 1.0 (no other values). For each state, provide a short reasoning sentence before assigning its score. Return JSON only with both reasons and scores, for example: { "reasons": { "s0": "reasoning for s0: whether any sequence can reach a goal, considering walls, blocked pushes, and deadlock signals (e.g., box stuck in a corner with no goal)", ... },g "scores": { "s0": 0.0, ... } }.

Table 2: **Example of sampling error.** Although the plan specifies the immediate next action as R, the executed action is U.

Thoughts	Planned	Executed
The player is at (6, 4), right below the box at (6, 5). The player can push the box right from (6, 5) to (7, 5), which is the goal. This push will complete the task. The immediate next action is to push the box right by moving right from (6, 4) to (7, 4), pushing the box from (6, 5) to (7, 5).	R	U

Table 3: **Comparing TAPE with best-of-N integration on the ReAct and PA frameworks (Wang et al., 2023b).** Results are on **Sokoban (easy)**, and we report **mean ± standard error**. For a compute-matched comparison, the Best-of-N variants sample the same number of plans per step (M) as TAPE. Even under this matched sampling budget, TAPE consistently achieves higher success rates, suggesting that the gains are not solely due to increased sampling.

Method	Success Rate (%)
TAPE ($M = 4$)	$\mathbf{46.0 \pm 5.0}$
Plan-and-Act	20.0 ± 4.0
Plan-and-Act ($M = 4$)	$\underline{22.0 \pm 4.1}$
ReAct	4.0 ± 2.0
ReAct ($M = 4$)	8.0 ± 2.7

G Additional Experiment

G.1 Planning & Sampling Error Estimation

This section describes how we extract the intended action from `Thought` and how we compute planning and sampling error rates reported in Table 1. Also, we do the qualitative analysis of each errors.

Setting. For each step, we extract the intended next action $\hat{a}_t \in \{U, D, L, R\}$ from the agent's `Thought` using `gpt-4.1-mini` as a parser.

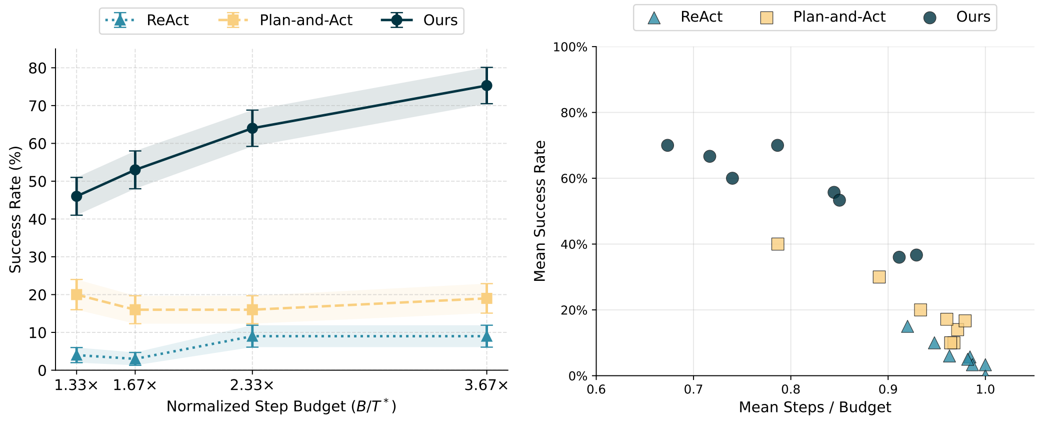
Then, given the current state s_t and the intended action $\hat{a}_t$, we simulate a one-step transition to obtain $s'_{t+1} = f(s_t, \hat{a}_t)$. We run a Breadth-First Search (BFS) oracle from s'_{t+1} to compute the minimum remaining steps to reach the goal, denoted $b(s'_{t+1})$. Let $B(t+1)$ denote the remaining step budget at time $t+1$. We mark $\hat{a}_t$ as non-viable if $d(s'_{t+1}) > B(t+1)$ or if $b(s'_{t+1}) = \infty$, and we count it as a planning error.

Lastly, let a_t denote the executed action from `Action`. We define the sampling error indicator at step t as $\mathbf{1}_{[a_t \neq \hat{a}_t]}$ and compute the sampling error rate by averaging this indicator over steps.

Sampling Error Example. In our analysis, we observe that sampling errors often manifest as a mismatch between the action implied by the LM's plan and the action actually executed. Table 2 shows a representative example: the *Thoughts* column contains the LM-generated planning trace, which clearly implies the next action R (pushing the box right to the goal), while the executed action deviates to U. Comparing the planned action inferred from the LM's reasoning with the executed action reveals such stochastic deviations at execution time, which we refer to as *sampling error*.

G.2 Additional Empirical Results

Comparison with Best-of-N Methods. We compare TAPE against compute-matched best-of-N variants of ReAct and PA (Wang et al., 2023b; Kang et al., 2025), as shown in Table 3. At each step, these baselines sample the same number of plans (M) as TAPE and select the best outcome, yielding a roughly comparable trajectory-sampling budget (and hence similar inference-token usage). On Sokoban (easy) with M=4, TAPE achieves 46.0% success, outperforming PA (20.0%) and PA-best-of-4 (22.0%), as well as ReAct (4.0%) and ReAct-best-of-4 (8.0%). These results indicate that



(a) Success under Larger Step Budgets (b) Success–Cost Trade-off

Figure 5: **Impact of larger step budgets and success-cost trade-off on Sokoban.** **(a)** Success rate as a function of the normalized step budget $B/S_{\min}$, where $S_{\min}$ denotes the oracle minimum number of steps (here $S_{\min} = 6$ and $B \in \{8, 10, 14, 22\}$). Error bars indicate $\pm$ standard error across runs. Our framework is more higher success rate as step budget increases. **(b)** Success rate versus step consumption (steps used divided by the budget). Methods closer to the top-left achieve higher success while consuming fewer steps per budget, indicating a better success–cost trade-off. TAPE can be both efficient and powerful compared to other frameworks.

TAPE's gains are not solely attributable to increased sampling, but rather to improved plan selection and execution under feasibility constraints.

Success under Larger Step Budgets. In Figure 5a, as we increase the step budget (i.e., provide more slack beyond the oracle minimum steps), we observe that ReAct and Plan-and-Act largely plateau, exhibiting only marginal changes in success despite the additional budget. In contrast, TAPE improves monotonically from 46% to 75% success as the normalized step budget $B/S_{\min}$ increases, indicating that it can effectively exploit extra steps for recovery rather than getting stuck in irrecoverable failures. This trend is consistent with our error decomposition: TAPE reduces planning errors that transition the agent into dead-end (non-viable) states (as measured by our Sokoban dead-end oracle) and mitigates sampling-induced execution deviations via constrained decoding and mismatch-triggered replanning. Consequently, TAPE lowers the probability of entering irrecoverable dead-ends early, allowing additional budget to translate into sustained gains in task success.

Success–Cost Trade-off. In Figure 5b, the x-axis measures step consumption (steps used divided by the budget), so better methods lie toward the top-left (higher success with lower cost consumption). we find that TAPE is closer to this region, achieving higher success while using fewer steps per budget on average compared to ReAct and PA. This result indicates that TAPE does not only improve the success rate but also enhances the efficiency.

G.3 Qualitative Analysis

Graph Example. Figure 6 illustrates an example of the graph constructed by TAPE for Sokoban. Nodes represent Sokoban states, where the goal is denoted as G, boxes as \$, the player as @, and walls as $\#$. Edges represent the actions. Given that 5 steps remain, the blue path shown in Figure 6 is the one selected by the external solver. This example demonstrates that our framework is capable of finding a feasible path by using formal solver when the graph is well-constructed.

Qualitative Comparison. Figure 7 provides a representative trajectory-level comparison between ReAct and TAPE under the same action budget. ReAct makes an early mistake (at $t = 5$), after which the subsequent actions continue to deviate and the agent fails to reach the goal within the budget, terminating in failure. In contrast, TAPE reaches the goal within the same budget by selecting a feasible plan and executing it more reliably, avoiding irrecoverable states. Overall, this

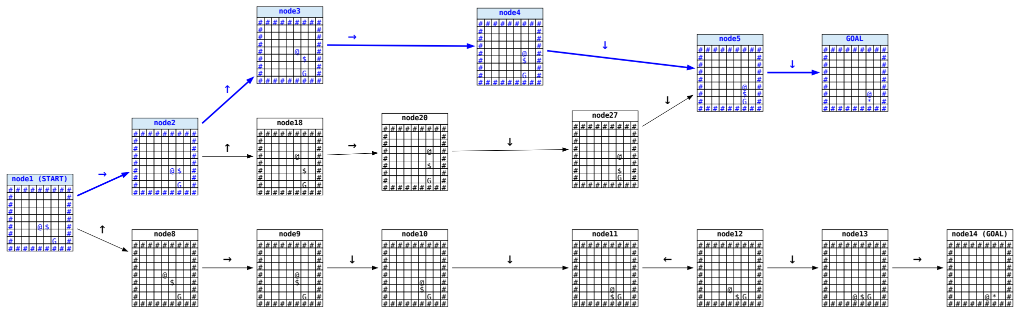
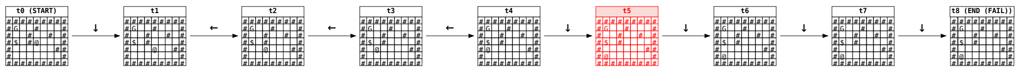
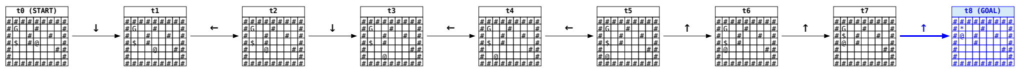


Figure 6: **Graph construction and plan selection example in Sokoban**. Graph is constructed by multiple paths from LLM lookahead plans. Then, a blue path is selected by a formal solver (ILP). When performing constrained execution, TAPE can accurately performs its plan, achieving the goal.



(a) ReAct



(b) Our Framework

Figure 7: **Qualitative comparison between ReAct and TAPE.** We visualize representative execution trajectories under the same action budget ($B = 8$). **(a) ReAct** makes a mistake at t=5 (red), after which it repeatedly deviates and fails to reach the goal within the budget, terminating at t=8 (END/FAIL). **(b) TAPE** selects a feasible path and executes it reliably, reaching the goal within the same budget at t=8 (blue).

qualitative evidence supports our quantitative findings in Section 4 that TAPE mitigates both planning and sampling errors.