DON'T MISS A DEADLINE: ONLINE TEMPORAL PLANNING FOR MULTI-GOAL LLM AGENTS UNDER PARTIAL OBSERVABILITY

Anonymous authors
Paper under double-blind review

ABSTRACT

Many real-world applications require Large Language Model (LLM) agents to complete multiple tasks subject to strict temporal constraints. Existing methods often assume a static environment where all task specifications are known a priori. However, in many practical robotic domains, agents only have partial observability due to limited sensory capabilities, and critical task information must be actively gathered through exploration or exploratory actions. To address these challenges, we propose a novel online temporal multi-goal planning framework for LLM agents under partial observability. Specifically, we first construct a partial Simple Temporal Network (STN) derived from the agent's current knowledge base. Then, this partial STN serves as a guide for the LLM to balance the exploration-exploitation trade-off while strictly adhering to temporal constraints during action selection. Additionally, we leverage the STN to monitor action validity; when a constraint violation is detected, semantic feedback is integrated into the prompt to trigger iterative action refinement. To empirically evaluate our approach, we construct a new benchmark, DeadlineBench, which requires LLM agents to execute multi-goal tasks under both temporal constraints and partial observability. Experimental results show that our approach consistently outperforms the state-of-the-art methods in terms of success rate across all benchmarks.

1 INTRODUCTION

Large Language Model (LLM) agents increasingly tackle many long-horizon tasks across various domains such as embodied applications, tool uses and web tasks (Liu et al., 2024; Zhou et al., 2024; Yao et al., 2024). However, completing a workflow requires more than choosing individually valid actions: an action can consume time needed by another goal, and a missed time-window can make that goal permanently unreachable. Successful task completion therefore depends on when actions occur, how concurrent processes are coordinated, and how much time is spent obtaining information. In other words, a multi-goal LLM agent must plan its actions properly subject to the temporal constraints without missing any deadlines of the tasks.

Most of existing settings for LLM agent planning capture diverse goals and interactions, but interaction budgets alone do not capture this temporal coupling. For example, VirtualHome, ALFRED and ALFWorld evaluate household execution, while ScienceWorld studies interactive scientific reasoning (Puig et al., 2018; Shridhar et al., 2020a;b; Wang et al., 2022). Constraint-based planning benchmarks such as TravelPlanner supply requirements with the task (Xie et al., 2024). These settings leave underexplored the joint challenge of coordinating concurrent goals, discovering *temporal constraints* and paying for that discovery with the time needed to act.

To be more realistic, we go one-step further to study this challenge under *partial observability*: the environment enforces its full constraints, but the agent must discover some of them during execution. For example, in the household scenario, preparing tea and sweeping living room before guests arrive requires coordinating search with a background kettle cycle. Starting the kettle later may miss the deadline in a tight schedule even when every action is valid (see Figure 1). The agent must therefore decide both what to do next and whether another information-gathering action is still affordable.

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

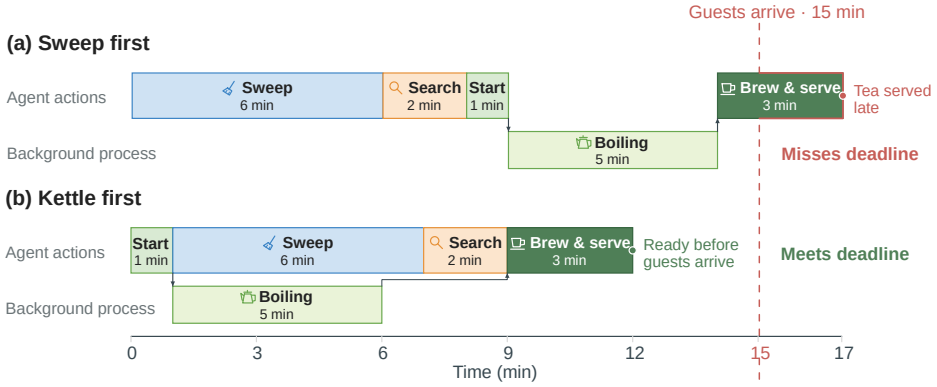

Figure 1: A household example. (a) Legal actions can miss the deadline when ordered poorly. (b) Starting a background process early leaves time for search and chores. See Appendix C for details.

This setting exposes complementary gaps in the existing methods in the literature. For instance, ReAct and Reflexion (Yao et al., 2023b; Shinn et al., 2023) support flexible interaction without maintaining an explicit quantitative temporal model. Symbolic planners and monitors (Fox & Long, 2003; Dechter et al., 1991; Kamath et al., 2025) reason about specified constraints, whereas here the specification itself unfolds through action. Online exploration and replanning (Yoo et al., 2024; Liu et al., 2025) must also account for the cost of acquiring information before a window closes.

In this paper, we address these gaps by maintaining a partial Simple Temporal Network (STN) from public observations. Deterministic rules incorporate confirmed events and discovered bounds; derived deadlines and heuristic margins guide action pruning and prioritization. Guards reject known expired starts and overlong waits, returning semantic feedback for one action repair. This provides guidance under discovered constraints, rather than a feasibility guarantee under hidden ones; network consistency monitoring is not a general action-veto test. Optional same-task experience reuses observed facts and bounds, while model-written advice cannot alter hard constraints.

Our main contributions are threefold: (i) we consider a novel setting coupling multiple goals, hard windows and costly information acquisition, where most of the existing methods fail to solve; (ii) we propose a novel online LLM agent framework that grounds temporal reasoning with STN in observed evidence; and (iii) we construct a new benchmark called DeadlineBench, with 38 household and 33 laboratory tasks verified feasible using the public action interface. Note that our focus here is online discovery and use of constraints, although we are not the first to combine LLMs and temporal planning. In our experiments, we show that our success rate reaches 78.9% and 87.9% in household and laboratory tasks, exceeding the best baseline results by 26.3% and 42.4%, respectively.

2 Related Work

Language-model agents for long-horizon interaction. ReAct (Yao et al., 2023b) interleaves reasoning, actions and observations and remains the default execution loop for interactive agents, and Reflexion (Shinn et al., 2023) summarizes failed episodes into verbal lessons that later trials reuse. ExRAP instead retrieves from a temporal embodied knowledge graph accumulated over earlier episodes and integrates exploration with exploitation for continual embodied instruction following (Yoo et al., 2024). Tree of Thoughts and RAP (Yao et al., 2023a; Hao et al., 2023) recast reasoning as search over candidate solutions, while DEPS and SwiftSage (Wang et al., 2023; Lin et al., 2023) add subgoal decomposition and failure repair for long-horizon embodied tasks. These methods improve which action to take, how to decompose a task, how to search and how to reuse experience; they are evaluated on embodied and tool-use benchmarks whose task structure, observations and interfaces differ substantially from ours (Puig et al., 2018; Shridhar et al., 2020a;b; Wang et al., 2022; Liu et al., 2024; Zhou et al., 2024; Yao et al., 2024; Xie et al., 2024). None of them coordinates several subgoals against one shared environment clock. Recent suites move towards our setting by lengthening horizons and hiding rules (Luo et al., 2026; Han et al., 2026), or by probing

reactions to time pressure (Sehgal et al., 2026), but they do not enforce hard windows whose expiry makes a required subgoal permanently unreachable while background processes keep running.

Temporal constraint reasoning and scheduling. Simple Temporal Networks (Dechter et al., 1991) represent durations, precedence and execution windows as difference constraints between time points and decide feasibility by network consistency, and PDDL2.1 (Fox & Long, 2003) adds durative actions and concurrency to a planning language. We use the former as an evolving internal representation rather than as a planning formalism, and hard constraints enter it only by deterministic compilation of public evidence. The closest language-model system is PLANTOR (Saccon et al., 2025), which generates a Prolog knowledge base, plans symbolically, schedules with mixed-integer programming and executes behaviour trees. Two differences matter. First, PLANTOR lets the model author the symbolic knowledge base, whereas our constraints can be written only by deterministic rules from the instruction and public observations, so the temporal state can be audited independently of the model. Second, PLANTOR plans once the knowledge base exists, whereas our network is built incrementally during execution, and a bound enters the agent's usable specification only after the action that reveals it. Language models have also been studied as deadline-aware schedulers (Zakerian, 2026); there the task set, release times and deadlines are given to the scheduler.

Notice that our setting is the intersection of these research lines: the full timing specification is fixed and enforced by the environment, the agent can discover only part of it through public actions that spend the same time budget, and it must keep a partial temporal state consistent while acting on it. We build that state incrementally as a partial STN and use it to prune candidates, rank goals and gate actions at run time, rather than to schedule a plan once from complete information.

3 Problem Setup

To better present our settings, we separate the environment that evolves from the task the agent must complete: the environment has a state and enforces all constraints, whereas the task specification and the observations determine which parts of that state the agent can use for planning.

3.1 Problem definition

Here we consider a problem domain called *DeadlineBench* where a LLM agent must complete multiple tasks with deadlines in a indoor/room environment, inspired by real-world applications. Formally, a *DeadlineBench* instance of our problem is defined as a tuple $\tau = (\mathcal{W}, \mathcal{A}, \mathcal{G}, \Gamma, D, x)$, whose components specify the environment, the actions, and what must be accomplished.

Environment. The environment $\mathcal{W}$ fixes the room layout, the fixtures at which work is carried out and outputs are delivered, and the containers that hold movable items. It is publicly known; only which container holds which movable item stays hidden, and inspecting a container reveals it.

Action space. The action space $\mathcal{A}$ contains movement, inspection, item handling, waiting, and process actions; executing an action takes its own duration, so every $a \in \mathcal{A}$ has a public positive-integer duration $\delta(a)$.

Task specification. The set $\mathcal{G}$ contains the required goals, Γ is the requirement set: the causal dependencies and the three kinds of hard temporal requirement that Section 3.3 decomposes, D is the global deadline by which the whole episode must finish, and the instruction x is the natural-language text in which the task is presented to the agent.

Time steps. Time is discrete and measured in ticks from the episode origin $t_{\text{origin}} = 0$: an action $a \in \mathcal{A}$ at clock t advances the clock to $t + \delta(a)$. The remaining execution mechanics, including how a wait is charged, are collected in Appendix B.

Foreground and background processes. Task execution contains two kinds of process: A *foreground process*, in $\mathcal{P}_{\text{fg}}$, is carried out by the agent itself as a single non-interruptible action such as movement, inspection, item transfer, sweeping, delivery, or confirmation; a required goal achieved by one is called a direct goal. A background process, in $\mathcal{P}_{\text{bg}}$, instead needs the agent to prepare and start it, after which it progresses with the environment clock while the agent acts elsewhere, so concurrency is part of the task rather than merely an ordering choice.

Each *background process* $p \in \mathcal{P}_{\mathrm{bg}}$ has public starting conditions and is started by the action $\mathrm{start}(p) \in \mathcal{A}$; it has a public cycle duration $\mathrm{dur}(p)$, supports the goal $g_p \in \mathcal{G}$, and runs independently once started. With start and finish times s_p and f_p,

$$f_p = s_p + \delta(\mathrm{start}(p)) + \mathrm{dur}(p). \tag{1}$$

Reaching f_p may still leave foreground work before the goal is satisfied, since only the running stage proceeds without the agent.

3.2 What the agent observes: partial task information

The agent does not receive the instance tuple τ; it observes the publicly known parts of the environment $\mathcal{W}$, the action space $\mathcal{A}$, the instruction x, and the public parts of the task specification: the layout of the rooms, the locations of all fixtures and containers, the required goals with their deadlines d_g, the global deadline D, and the published durations $\delta(a)$ and $\mathrm{dur}(p)$. The placement map Ψ, which gives the container that holds each movable item, stays hidden: an item is described by its task role, and its placement $\Psi(i)$ is revealed only when its container is inspected. Process results and unrevealed bounds such as ℓ_p in Γ are never exposed directly.

During execution, an observation reports the current clock, the agent's room and inventory, discovered item locations, observed process states, currently legal actions, public durations and deadlines, and the locations of declared information sources. A placement $\Psi(i)$, a bound ℓ_p, or a process result therefore enters the agent's information only after the corresponding inspection or check.

At action index $k \geq 0$, the agent receives a public observation o_k. Let $\mathcal{A}_k^{\mathrm{env}} \subseteq \mathcal{A}$ denote the currently legal actions listed in o_k. Before selecting $a_k \in \mathcal{A}_k^{\mathrm{env}}$, the agent has access to the public history $h_k = (o_0, a_0, o_1, \ldots, a_{k-1}, o_k)$, but not to the hidden simulator state. Taking an item requires inspecting its container in the current episode, even if a previous trial recorded its location. Repeated trials may additionally use only prior public experience from the same task (Section 4.4).

3.3 Temporal constraints and success criterion

Events. Every hard requirement in Γ constrains when an event happens. For a required goal $g \in \mathcal{G}$, let c_g be the clock at which g is satisfied; for a completed process whose result must be inspected, let c_p be the clock at which it is checked and r_p the clock at which the revealed response is completed. Together with the process times s_p and f_p of Equation 1, all these times are integer ticks measured from the episode origin $t_{\mathrm{origin}} = 0$.

Constraints. The requirement set Γ contains three kinds of hard temporal constraints.

(a) Published deadlines. The instruction states when each required goal must be completed, so every $g \in \mathcal{G}$ carries a deadline d_g and the constraint

$$c_g - t_{\mathrm{origin}} \leq d_g, \qquad g \in \mathcal{G}, \tag{2}$$

and an episode in which a deadline is missed is unsuccessful even if every individual action is legal.

(b) Inspection-revealed latest starts. A bounded process $p \in \mathcal{P}_{\mathrm{bg}}$ must begin before a latest start ℓ_p that is recorded at a designated information source $\mathrm{src}(p)$; inspecting that source is the only way to learn ℓ_p, and the inspection itself consumes time. The environment enforces

$$s_p - t_{\mathrm{origin}} \leq \ell_p \tag{3}$$

from the beginning, whether or not the bound has been discovered.

(c) Check-and-response windows. A process p whose result is not determined by its inputs has a result $\omega(p)$ that becomes public only when the process is checked. Such a process must be checked within κ_p ticks of finishing, and the response that the check reveals must be completed within a further ρ_p ticks:

$$c_p - f_p \leq \kappa_p, \qquad r_p - c_p \leq \rho_p. \tag{4}$$

The response can start a follow-up process of public duration ϕ_p, which must itself finish before the global deadline D.

Causal dependencies. Completing a process must precede satisfying the goal it supports:

$$c_{g_p} - f_p \geq 0, \qquad p \in \mathcal{P}_{\mathrm{bg}}. \tag{5}$$

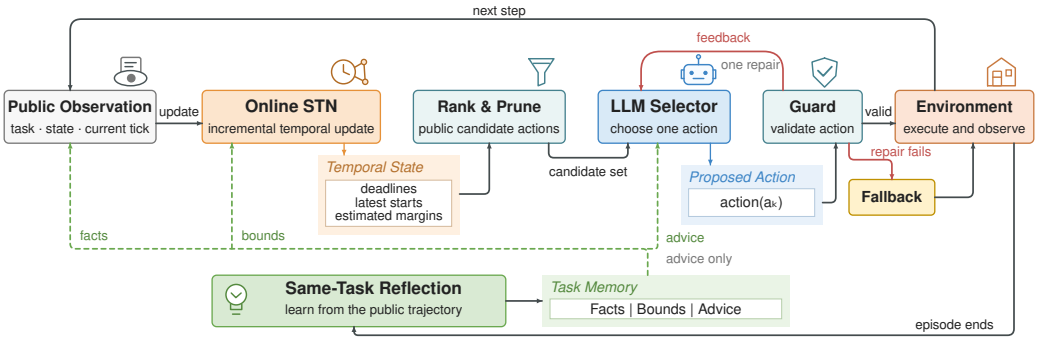

Figure 2: Overview of our online-STN agent framework. Public evidence incrementally updates the STN; its temporal summary guides deterministic candidate construction and a frozen LLM selector. The guard returns a rejected proposal for at most one repair before invoking a deterministic fallback, and environment feedback starts the next decision step. After an episode, same-task reflection routes observed facts and publicly revealed bounds back to the public state and STN, while textual advice is supplied only to the selector.

Since delivering or confirming a result is itself a foreground action, a background process contributes to success only after the agent has also performed every required following action.

Success criterion. An episode succeeds when the agent completes every required goal and mandatory inspection, and the resulting event times satisfy Equations 2–5 for the full requirement set Γ, including the bounds ℓ_p, κ_p and ρ_p and the outcomes $\omega(p)$ that the agent never observed. The environment evaluates these conditions against the full instance τ, so partial observability limits the information available for planning, not the rules that determine success.

4 METHOD

Here we starts from an overview of our framework and then describe our three modules in details.

4.1 OVERVIEW OF ONLINE-STN AGENT FRAMEWORK

Figure 2 gives the execution loop of Online-STN. At each decision step, the agent receives a public observation containing the current world state and clock. The temporal network incrementally incorporates confirmed events and newly revealed bounds into a partial STN, then derives a temporal summary of deadlines, latest starts, and estimated margins. Deterministic rules use this summary to rank unfinished goals and prune the currently legal actions. The resulting candidate set, together with the observation and temporal summary, is passed to the frozen LLM selector, which proposes one grounded action.

The proposed action is checked against the candidate set and a narrow temporal guard. A rejected proposal is returned to the selector with a semantic reason for at most one repair; if repair also fails, a deterministic priority rule supplies a fallback action. The accepted action is executed in the environment, and the new observation closes the online loop by updating both the public state and the STN. Thus, the method repeatedly revises its schedule as information becomes available rather than committing to a complete plan from the initial partial specification.

When an episode ends, the optional reflection module supports another trial of the same task. It stores observed world facts and publicly revealed temporal bounds separately from a short model-written scheduling suggestion. On the next trial, facts augment the public state, publicly revealed bounds re-enter the STN through the usual consistency check, and textual advice is shown only to the LLM. This separation is central: only deterministic routines can modify the temporal network, whereas model output can influence action selection but cannot create a hard constraint, event time, or action precondition. Sections 4.2–4.4 describe the components of Figure 2 in turn: the temporal network, the LLM action selector, and same-task reflection.

4.2 TEMPORAL NETWORK FOR MAINTAINING INFORMATION ON TASK DEADLINES

The temporal network keeps the agent's temporal state: a public state for grounding and routing, and a partial STN over the evidence observed so far.

Partial STN model. The temporal network $\mathcal{N}_k$ is a partial STN whose nodes are named event time points—origin, process starts and finishes, goals, checks and responses—and whose edges are difference constraints $t_v - t_u \leq c$. Following standard STN inference (Dechter et al., 1991), all-pairs shortest-path closure decides consistency and yields each event's feasible window,

$$t_v \in \big[-\mathrm{dist}(v, \mathrm{origin}), \mathrm{dist}(\mathrm{origin}, v)\big]. \tag{6}$$

It covers starts, finishes, durations, precedence, deadlines and revealed bounds, but not resource capacity, stochastic durations or disjunctive constraints. Initialised at reset with the origin and the published deadlines—a goal without a tighter deadline is bounded by the episode horizon—it is revised only by public evidence, through deterministic rules that each pass a consistency check, and it records the feedback-confirmed event times (Appendix A).

Temporal queries from LLM agents. The network supplies the windows of Equation 6, the confirmed event times and the latest start or response bound established by discovered or remembered evidence; an unknown bound stays absent rather than being guessed. Together with a heuristic estimate of the remaining work these form the summary q_k of Figure 2, so the LLM and the deterministic rules share the same public temporal evidence.

4.3 LLM ACTION SELECTOR FOR BALANCING EXPLORATION AND EXPLOITATION

The selector turns the temporal state into one executable action: deterministic logic builds and checks the candidate set, and the frozen LLM chooses within it, which preserves the LLM's ability to interpret observations while keeping both temporal checks reproducible.

Candidate action construction. Let $\mathcal{A}_k^{\mathrm{env}}$ be the actions currently legal under the public interface. The selector removes starts whose known upper bound has expired and ranks unfinished goals by estimated margin. If an action at the current location directly advances the most urgent goal, the candidate set may be reduced to it, but it is still passed to the LLM; otherwise fixed rules prioritise routing to a known prerequisite, starting an idle process, retrieving or delivering a finished product, searching for a missing prerequisite and waiting for the next predictable completion, and the filtered legal set is kept when none applies. Movement follows the shortest route on the public room graph, and unreachable processes leave the ranking. These rules rank the next step; they do not commit to a schedule.

Estimated time margin. For each unfinished goal, the selector subtracts an estimated remaining process and post-process cost—published durations plus small fixed allowances for delivery or confirmation—from the ticks left before its deadline or the episode horizon, capped by a known latest-start bound. This is a heuristic margin rather than optimal slack: it cannot account for an undiscovered requirement.

Model input and grounding. At each decision step, the LLM receives the instruction, the current observation, recent public action feedback, the temporal summary with the goal ranking, and the candidate actions with their interactable argument values; it never receives hidden placements, unread bounds, unrevealed outcomes or the full action cross-product. A response is accepted only when its action name and every argument exactly match a candidate, which also enforces the candidate's public preconditions and movement adjacency.

Temporal guard, repair, and fallback. The guard rejects exactly two kinds of proposal: a start whose latest-start bound has passed, and a wait longer than the estimated margin of the most urgent goal. It is not a general action veto and does not certify feasibility under hidden constraints. A parse, grounding or guard failure returns a semantic reason for one repair, so a decision step invokes the selector at most twice; if repair also fails, a deterministic fallback picks from the same candidate set by fixed priority, skipping unreachable or temporally invalid processes and waiting one tick if nothing survives. The fallback is a recovery rule, not a success guarantee.

4.4 Reflection module for reasoning on task facts and experiences

The reflection module is optional and is invoked once after an episode. Its memory is indexed by task, so it supports repeated trials without introducing cross-task information. It contains two independently switchable channels, enabling the ablations in Section 5.3 to separate factual recall from temporal experience.

Channel A: Public world facts. This channel stores item–container pairs that appeared in the public observation stream, excluding transient locations such as the inventory. At the start of the next trial, remembered locations fill missing entries in the public state; observations from the current episode always take precedence. The memory can therefore avoid searches of unrelated containers, but it does not bypass any inspection required by the environment in the new episode.

Channel B: Temporal experiences. This channel stores two distinct forms of experience. Firstly, after each episode, the LLM receives the public trajectory, terminal goal state, elapsed time, and public STN summary and produces one short, goal-conditioned scheduling suggestion. At most three suggestions whose required goals match the current task are added to the selector prompt. They remain advice: they cannot alter a constraint, event time, deadline, or action precondition.

Secondly, the channel stores structured latest-start and latest-response records that the episode publicly discovered. These records are copied from environment observations rather than generated by the LLM. At reset they are replayed through the same consistency-preserving insertion procedure as newly discovered bounds, allowing candidate ranking to use previously observed temporal information before the agent spends time rediscovering it. Keeping structured bounds separate from textual advice preserves the deterministic provenance of every hard constraint. Appendix F specifies storage and retrieval in detail.

5 Experiments

5.1 Setup

Benchmark. We evaluate two suites of 38 household and 33 laboratory tasks; Appendix B documents the interface and the feasibility certification, and Appendix C the two environments. Methods that reuse experience across trials run three trials per task, and the remaining methods run one. Every method was evaluated in three independent rounds that each visit every task; rates average the rounds with the trial index kept separate, so a first-trial rate is measured over 114 household and 99 laboratory episodes. The ablation tables use the same suites; each row is a single evaluation unless noted below. All methods use the same public interface, task-specific tick deadline and 40-step cap, without oracle access. LLM calls use deepseek-v4-flash at most 128 generated tokens.

Baselines. ReAct (Yao et al., 2023b) has one trial and no memory. Reflexion (Shinn et al., 2023) has three trials, adding one verbal lesson after each episode; its empty-memory first prompt equals ReAct's. The two rows marked ‡ are our adaptations of published systems, PLANTOR and Agent-C, both ported to this benchmark's interface so that they can act under hidden item locations and latent bounds; Appendix E states what each adaptation changes and why it makes the baseline stronger. Both adapted rows have one trial. Adapted Agent-C (Kamath et al., 2025) has one trial and no cross-episode memory. Online-STN with experience also has three trials. Each episode resets the environment; memory never crosses task IDs. First trials provide the cold-start comparison, while later trials evaluate experience. Port fidelity and unequal trial budgets qualify comparisons.

A success run requires every goal completed on time and every mandatory source inspected otherwise it is marked as a failure. Failure classes are (i-a) deadline termination with a refused start, (i-b) step exhaustion with a refused start, (i-c) deadline termination without refusal, and (ii) step exhaustion without refusal. These are mechanical outcomes, not proof of a particular reasoning error. Temporal feasibility (no refused expired start or unused check/response window) is distinct from success, since inaction can avoid a violation. Protocol details are in Appendix D.

5.2 Main results

Table 1 reports success rate, episode steps and irrelevant inspections averaged over three independent evaluation rounds that each visit every task; the first and third trials are reported separately. On the

Table 1: Success rate (%), steps and irrelevant inspections, averaged over three independent rounds with trials kept separate; trial 1 has empty memory. Irrelevant: no required item in the container (in the laboratory, no required clue). ‡ denotes our adaptation of the published system.

		Household			Laboratory		
Method	**Trial**	**Success rate**	**Steps**	**Irrel.**	**Success rate**	**Steps**	**Irrel.**
ReAct	1st	36.8	17.3	0.79	43.4	28.3	0.99
Reflexion	1st	28.1	17.4	0.65	43.4	27.9	1.06
	3rd	48.2	18.1	0.70	45.5	27.3	0.91
PLANTOR‡	1st	52.6	18.0	1.95	27.3	28.9	0.39
Agent-C‡	1st	19.3	26.5	1.56	32.3	34.7	1.09
Online-STN+Exp. (ours)	1st	67.5	20.1	1.10	72.7	26.1	0.67
	3rd	78.9	17.5	0.16	87.9	24.2	0.30

Table 2: Network-maintenance ablation: one trial per task, no prior experience. Metrics and definitions follow Table 1; model and budgets follow Section 5.1, and only network maintenance varies.

	Household			Laboratory		
Variant	**Success rate**	**Steps**	**Irrel. insp.**	**Success rate**	**Steps**	**Irrel. insp.**
Static-STN	44.7	21.9	1.21	30.3	34.3	0.36
Rolling-STN	68.4	22.8	1.16	30.3	34.3	0.45
Online-STN	73.7	19.6	1.16	72.7	26.2	0.73

first trial, Online-STN reaches 67.5% on household tasks and 72.7% on laboratory tasks, against 52.6% and 43.4% for the best baselines. Household success grows with experience, reaching 78.9% on the third trial, whereas laboratory success rises from 72.7% on the first trial to 87.9%. On the third trial the margins over the best baselines are 26.3 and 42.4 percentage points. Reflexion improves household success from 28.1% on the first trial to 48.2% on the third, whereas adapted Agent-C reaches 19.3% and 32.3%. ReAct and first-trial Reflexion differ by 8.7 percentage points on household despite identical prompts, so differences of this size should not be read as significant.

The cost columns follow the same ordering. Adapted Agent-C spends the longest episodes (26.5 household and 34.7 laboratory steps) and, together with the adapted PLANTOR baseline, the most irrelevant inspections (1.56 and 1.95 on household), which matches its laboratory profile of budget exhaustion. Our episodes shorten with experience, from 20.1 to 17.5 household steps and from 26.1 to 24.2 laboratory steps, and irrelevant inspections fall from 1.10 to 0.16 and from 0.67 to 0.30; among the methods that complete most tasks these are the lowest values in both domains.

5.3 Ablations on STN and memory

Temporal network maintenance. Table 2 compares three network-maintenance variants, one trial per task with no prior experience and a shared executor, model, observations and budgets. Static-STN keeps the initial graph, Rolling-STN rebuilds it from public history at each decision, and Online-STN retains confirmed events and discovered constraints in a persistent graph. Household success rises from 44.7% (Static-STN) to 68.4% (Rolling-STN) and 73.7% (Online-STN), whereas in the laboratory both reconstruction variants stay at 30.3% against 72.7%. Retaining an evolving temporal state therefore helps most where bounds are inspection-revealed and responses outcome-dependent, and it shortens episodes without changing search cost. Appendix G gives the failure breakdown and scope.

Episodic experience channels. Table 3 compares both channels on the third trial: A retains observed locations and B combines replayed bounds with goal-matched advice. Variants start with empty memory, so the comparison concerns a repeated task rather than transfer; the both-channels row averages the three rounds of Section 5.2, the others are single runs. In the household domain, location memory cuts irrelevant inspections and episodes but does not beat the no-memory variant, and rule reflection alone changes little; only both channels together exceed it, reaching 78.9%. The laboratory separates them differently: neither channel alone changes the outcome, while both re-

Table 3: Experience ablation on the third trial; the first trial has empty memory. A and B are the two channels of Section 4.4, and each variant's name states which are on. Metrics follow Table 1.

	Household			Laboratory		
Variant	**Success rate**	**Steps**	**Irrel. insp.**	**Success rate**	**Steps**	**Irrel. insp.**
Online-STN (neither)	73.7	19.8	1.13	72.7	26.0	0.61
Location memory (A)	68.4	17.1	0.24	72.7	26.1	0.27
Rule reflection (B)	71.1	21.6	1.29	75.8	25.7	0.52
Full experience (A+B)	78.9	17.5	0.16	87.9	24.2	0.30

Table 4: Failure shares (%) per domain, pooling all registered episodes. Classes follow Section 5.1; window miss sums i-a–i-c. ‡ denotes our adaptation of the published system.

		i-a	**i-b**	**i-c**	**ii**	**window miss**
Household	ReAct	11.4	1.8	50.0	0.0	63.2
	Reflexion	11.4	0.6	48.0	0.0	59.9
	PLANTOR‡	34.2	0.0	13.2	0.0	47.4
	Agent-C‡	0.0	0.0	78.8	0.0	78.8
	Online-STN+Exp. (ours)	0.9	3.5	20.2	0.0	24.6
Laboratory	ReAct	29.3	19.2	7.1	1.0	55.6
	Reflexion	31.3	10.8	13.8	0.7	55.9
	PLANTOR‡	66.7	3.0	3.0	0.0	72.7
	Agent-C‡	0.0	2.0	2.0	63.6	4.0
	Online-STN+Exp. (ours)	0.0	0.0	2.4	14.8	2.4

duce irrelevant inspections, and only both together raise success to 87.9%. Remembering where an item was found reduces search while temporal experience changes how it is scheduled, but neither suffices alone. Appendix G reports the per-channel numbers and caveats.

5.4 Analysis on failures

Table 4 separates failure outcomes by pooling all registered trials per method across the evaluation rounds, whereas the main table reports success separately for each trial.

Our window-miss share is the lowest in both domains (24.6% household and 2.4% laboratory), and the two domains fail differently: household failures are dominated by deadline stalls without refusal (20.2%), whereas laboratory failures are dominated by step exhaustion (14.8%) rather than missed windows. The adapted baselines fail in their own characteristic ways: adapted PLANTOR shifts failures toward refused starts (34.2% and 66.7%), and adapted Agent-C exhausts the step budget in 63.6% of its laboratory episodes. The classes therefore describe outcomes rather than proving why a policy failed. Systematic candidate pruning, singleton choices and fallback all contribute to behaviour, so success characterises the complete agent rather than the LLM alone; tasks within a domain are related, and one model and decoding configuration are used, so these numbers compare methods on the stated task sets rather than performance over an independent task population.

6 Conclusion

We presented Online-STN, an online planning method for multi-goal LLM agents under partial observability and a shared environment clock. Its core is a partial Simple Temporal Network maintained from public evidence alone, whose deadlines, latest starts and margin estimates shape candidate construction, action selection and guarding; optional same-task experience reuses what earlier trials publicly observed. On the household and laboratory suites of DeadlineBench, the method outperforms every compared system. Under partial observability, an agent should therefore accumulate the temporal obligations it discovers and use them to shape exploration, not only to veto actions.

AI USE STATEMENT

Generative AI tools assisted this work in four limited ways.

Writing. Generative AI tools assisted with language editing and improving the clarity of the manuscript. All resulting text was reviewed and approved by the authors.

Literature discovery. They suggested candidate related work. The authors verified every cited reference against its primary source; no citation relies solely on a model-generated claim.

Ideation and implementation. They assisted in proposing method variants and ablation designs, and in implementing and running the benchmark, agent, and analysis code, including refactoring, test scaffolding, and diagnostic scripts.

Benchmark construction. They assisted the authors in designing DeadlineBench task templates, including scenario families, goals, layouts, hidden item placements, latent temporal bounds, and outcome branches. The task instances were then instantiated from author-designed templates and reviewed by the authors against the acceptance criteria in Appendix C. Their feasibility was certified using the full-information oracle.

The authors specified the research goals, designed and reviewed the benchmark tasks, reviewed all AI-assisted material, and take full responsibility for the manuscript, its claims, and its artifacts. All reported results were produced by the authors' evaluation code on frozen task files and cross-checked against saved run artifacts.

See the ICLR 2027 AI Policy for Authors for further details.

ETHICS STATEMENT

This work does not involve human subjects, does not collect or release personal data, and does not train or fine-tune any model. The benchmark environments are synthetic, text-based abstractions of the two task domains it defines — the household and laboratory domains, both of which are evaluated here — and the tasks are generated from templates rather than derived from real operational records. No sensitive, proprietary, or personally identifiable data is used at any stage. The domains are stylised and are not intended to represent the safety requirements of real instrumentation; we make no claim about the suitability of any method evaluated here for deployment in a physical laboratory or production setting.

REPRODUCIBILITY STATEMENT

The task schema, public interaction records and evaluation protocol support reproduction.

Environment and tasks. All task instances are frozen JSON files distributed with the code. Each instance carries its own witness from the full-information oracle of Section B.2: the witness action sequence, its action count, its completion tick and the resulting tick slack are stored in the instance, and the manifest of each frozen set repeats them per instance together with the oracle verdict. An instance enters a reported set only when the oracle finds a trajectory that fits both its step budget and its tick deadline.

Recorded traces. Each episode records public observations, executed actions, model prompts and responses, terminal status, action count and elapsed environment time. Success and failure classes are aggregated according to Section 5.1. No credentials are recorded.

Protocol. Section 5.1 states, per table, the domain, the model, the number of episodes, whether within-task memory was used, and the number of samples. All methods share one model client, one public observation interface, one candidate representation and one per-task step budget, and no method can reach hidden instance state at any point.

Scope of the reported numbers. The evaluation uses fixed, released task sets rather than a held-out sample, so the numbers characterise the mechanism; the adapted Agent-C row is our port of the published system rather than the published code; and every reported number comes from a single external model at temperature 0, so we make no significance claim.

REFERENCES

Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 1991.

Maria Fox and Derek Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 2003.

Yi Han, Yan Wang, Lingfei Qian, Haohang Li, Yupeng Cao, Yueru He, Xueqing Peng, Nanhan Shen, Yitao Xu, Yankai Chen, et al. Can LLM Agents Be CFOs? Benchmarking Long-Horizon Resource Allocation in an Uncertain Enterprise Environment. *arXiv preprint arXiv:2603.23638*, 2026.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with Language Model is Planning with World Model. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, 2023.

Adharsh Kamath, Sishen Zhang, Calvin Xu, Shubham Ugare, Gagandeep Singh, and Sasa Misailovic. Enforcing Temporal Constraints for LLM Agents. *arXiv preprint arXiv:2512.23738*, 2025.

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. SwiftSage: A Generative Agent with Fast and Slow Thinking for Complex Interactive Tasks. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2023.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. AgentBench: Evaluating LLMs as Agents. In *International Conference on Learning Representations*, 2024.

Xiaotian Liu, Ali Pesaranghader, Hanze Li, Punyaphat Sukcharoenchaikul, Jaehong Kim, Tanmana Sadhu, Hyejeong Jeon, and Scott Sanner. Open-World Planning via Lifted Regression with LLM-Inferred Affordances for Embodied Agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2025.

Haotian Luo, Huaisong Zhang, Xuelin Zhang, Haoyu Wang, Zeyu Qin, WenJie Lu, Guozheng Ma, Haiying He, Yingsha Xie, Qiyang Zhou, Zixuan Hu, Hongze Mi, Yibo Wang, Naiqiang Tan, Hong Chen, Yi R. Fung, Chun Yuan, and Li Shen. UltraHorizon: Benchmarking Agent Capabilities in Ultra Long-Horizon Scenarios. In *Forty-third International Conference on Machine Learning*, 2026.

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. VirtualHome: Simulating Household Activities via Programs. In *2018 IEEE/cvf conference on computer vision and pattern recognition*. IEEE, 2018.

Enrico Saccon, Matteo Saveriano, Edoardo Lamon, Luigi Palopoli, and Marco Roveri. Combining Large Language Models and Symbolic Reasoning for Multi-Robot Temporal Planning through Explainable Knowledge Bases. *arXiv preprint arXiv:2502.19135*, 2025.

Neil KR Sehgal, Sharath Chandra Guntuku, and Lyle Ungar. Real-Time Deadlines Reveal Fragile Temporal Adaptation in LLM Strategic Dialogues. *arXiv preprint arXiv:2601.13206*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2023.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *2020 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*. IEEE, 2020a.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. *arXiv preprint arXiv:2010.03768*, 2020b.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. ScienceWorld: Is your Agent Smarter than a 5th Grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, Explain, Plan and Select: Interactive Planning with LLMs Enables Open-World Multi-Task Agents. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2023.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. TravelPlanner: A Benchmark for Real-World Planning with Language Agents. In *Proceedings of the 41st International Conference on Machine Learning*. JMLR.org, 2024.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*, 2023b.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. *arXiv preprint arXiv:2406.12045*, 2024.

Minjong Yoo, Jinwoo Jang, Wei-Jin Park, and Honguk Woo. Exploratory Retrieval-Augmented Planning For Continual Embodied Instruction Following. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2024.

Reza Zakerian. When Do LLM Agents Help? Deadline-Aware Mixed-Criticality Task Scheduling at the Autonomous-Vehicle Edge. *arXiv preprint arXiv:2608.19557*, 2026.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *International Conference on Learning Representations*, 2024.

A THE TEMPORAL NETWORK

The network stores difference constraints of the form $t_v - t_u \leq c$. Equality constraints such as the process law of Equation equation 1 are added as a pair of opposing upper bounds, and absolute times are anchored by the origin time point $t_{\text{origin}} = 0$, so the published requirement of Equation equation 2 is stored as $c_g - t_{\text{origin}} \leq d_g$. Consistency is decided by all-pairs shortest-path closure: the network is consistent exactly when the closure contains no negative cycle. The closure is recomputed after every addition, so propagation is global rather than lazy; the networks here are small enough that this is not a bottleneck, and we make no claim about incremental propagation efficiency. A candidate constraint is tried on a clone of the network and rejected if it would make the network inconsistent, so the network can never carry a contradiction.

Insertion rules. Public process specifications identify the device, input, published duration and goal of each process. A started process p contributes its process law

$$t_{\text{finish}(p)} - t_{\text{start}(p)} = \delta(\text{start}(p)) + \text{dur}(p), \tag{7}$$

and the precedence requirement $t_{\text{goal}(p)} - t_{\text{finish}(p)} \geq 0$, with the observed start tick fixing the corresponding event time. Inspection can reveal a latest-start constraint $t_{\text{start}(p)} - t_{\text{origin}} \leq \ell$, and a revealed latest-response bound is represented analogously on the response event. Each bound is keyed by its public clue identifier, which makes repeated observations idempotent, and a verified bound replayed from an earlier trial follows the same insertion path as a newly observed one. Checks, response actions and goal-completion observations similarly fix their event times, which are never relaxed after acceptance.

One derived quantity is computed from the network for the scheduler: for each unfinished goal, the latest feasible start of the process that serves it, obtained by relaxing the deadline d_g through the process duration and the intervening precedence edges. It is derived from the network and is distinct from a bound ℓ_p that inspection of a source may reveal. The estimated remaining margin that the selector uses is defined in Section 4.3; it is a scheduling heuristic built on hand-set allowances rather than an optimal slack.

Building and revising the network. The network is built from public information only, and it is revised by exactly the operations that Algorithm 1 names. At reset, $\mathcal{N}_0 = \text{InitFromPublicInfo}(x, o_0)$ contains the origin and the published deadlines; at each decision step the temporal summary $q_k = \text{TemporalSummary}(\mathcal{N}_k, \beta_k)$ is derived from the current network and public state; and after the environment returns the next observation and feedback, $\text{Revise}(\mathcal{N}_k, \hat{a}_k, fb_k, o_{k+1})$ inserts the confirmed event times and the publicly revealed bounds. Every insertion passes the consistency check above, so the network carries only constraints that public evidence supports. In the pseudo-code, o_k is the public observation at decision step k, β_k is the public state maintained from the observation history, $\mathcal{N}_k$ is the current STN, q_k is its temporal summary, and h_k is the public interaction history.

B ACTION SPACE AND SETTINGS

DeadlineBench contains two embodied suites, one household and one laboratory. They share a single interface, a single clock, a single step budget and a single evaluation procedure, and they instantiate the same formal object: required goals over a partially observed world, a travel graph, processes with published durations, and hard temporal requirements of the three kinds in Section 3.3. One tick is 15 seconds and every action costs its published duration, so time is spent rather than merely elapsed. The two suites differ in what they withhold and in the shape of their temporal requirements, and Appendix C records the concrete parameters behind each of them.

B.1 THE ACTION VOCABULARY

Each suite exposes a small, fixed action vocabulary, listed in Table 5. Every method, including each adapted baseline, receives the same interface and the same parsing rules: an action is executed only if the composed block matches a currently legal public action exactly.

The household domain names its devices directly, so a start action identifies the appliance it starts. The laboratory domain uses the generic start_process and confirm_process with a target argument,

Algorithm 1 Execution loop of Online-STN.

Require: instruction x, environment $\mathcal{E}$, step budget B

$o_0 \leftarrow \mathrm{Observe}(\mathcal{E})$, $\beta_0 \leftarrow \mathrm{PublicState}(o_0)$
$\mathcal{N}_0 \leftarrow \mathrm{InitFromPublicInfo}(x, o_0)$
for $k \leftarrow 0, \ldots, B-1$ **do**
  $q_k \leftarrow \mathrm{TemporalSummary}(\mathcal{N}_k, \beta_k)$
  $\mathcal{A}_k^{\mathrm{cand}} \leftarrow \mathrm{PruneCandidates}(o_k, \beta_k, q_k)$
  $\hat{a}_k \leftarrow \mathrm{Select}(x, o_k, h_k, q_k, \mathcal{A}_k^{\mathrm{cand}})$
  if $\neg\mathrm{Accept}(\hat{a}_k, \mathcal{A}_k^{\mathrm{cand}}, q_k)$ **then**
    $\hat{a}_k \leftarrow \mathrm{RepairOnce}(\hat{a}_k, \mathrm{RejectionReason})$
  end if
  if $\neg\mathrm{Accept}(\hat{a}_k, \mathcal{A}_k^{\mathrm{cand}}, q_k)$ **then**
    $\hat{a}_k \leftarrow \mathrm{Fallback}(\mathcal{A}_k^{\mathrm{cand}}, \beta_k, q_k)$
  end if
  $o_{k+1}, fb_k \leftarrow \mathrm{Step}(\mathcal{E}, \hat{a}_k)$
  $\beta_{k+1} \leftarrow \mathrm{Update}(\beta_k, o_{k+1})$
  $\mathcal{N}_{k+1} \leftarrow \mathrm{Revise}(\mathcal{N}_k, \hat{a}_k, fb_k, o_{k+1})$ {deterministic public evidence only}
  if $\mathrm{Terminal}(o_{k+1})$ **then**
    break
  end if
end for
$\mathrm{Reflect}$(public episode summary) {optional; once per episode}

Table 5: Action vocabularies by domain; arguments in parentheses, duration arguments in ticks.

Domain	Count	Actions
Household	11	move, wait(ticks), inspect, take, place, fill_kettle, start_kettle, start_oven, start_washer, confirm_washer, sweep_carpet
Laboratory	9	move, wait(ticks), inspect, take, place, start_process, check_result, apply_reagent, confirm_process

and check_result and apply_reagent exist only there, where they implement the outcome-conditional mechanism of Section 3.3. A wait may consume between 1 and 32 ticks; the public menu offers 1 and 4 ticks and, while a process is running, the wait that ends exactly at its next completion, and a wait advances the clock by exactly its stated duration within one decision step. An action whose block matches no currently legal public action advances the clock by one tick and is counted as an interface diagnostic rather than as a temporal failure (Section 5.1).

B.2 Feasibility certification

A full-information oracle computed by A* search knows the hidden placements and process results, but acts through the same public action interface and must still perform every mandatory inspection. A witness is an oracle action sequence that succeeds within the episode's step budget and the global deadline D. All 71 evaluated instances have such a witness; for each one we record its action count, completion time, and global slack $D - c_{\max}$, where $c_{\max} = \max_{g \in \mathcal{G}} c_g$ is the last goal-satisfaction time. An instance is causally feasible when its goals are reachable within the step budget, and temporally feasible when a causally feasible sequence also satisfies Equations equation 2–equation 5. The oracle certifies solvability, not difficulty, and is not a competing agent.

C Environments

This section records what each suite withholds and how its parameters are bounded. The two five-task additions were designed after the earlier families had been analysed, and each was accepted

only against a stated criterion: in the household addition the fixed-order baseline must fail while our method starts the bounded process inside its window, and in the laboratory addition our cold start must fail while the experienced one succeeds. They are designed suites rather than held-out samples, and every task in them is kept in the reported sets. The frozen evaluation sets contain 38 household and 33 laboratory tasks, each with a uniform 40-step budget. The household domain combines hidden item locations, direct chores, and background appliances under shared deadlines; 13 of its 38 tasks additionally carry a latest-start bound in an optional notice, which the environment enforces whether or not the agent opens it. The laboratory domain combines hidden sample or reagent locations with instrument processes. Its 33 tasks carry 31 latest-start bounds of 4 to 16 ticks across 27 tasks, and in 22 of the 33 a protocol log that the task requires the agent to read is what reveals a bound; every task includes a hidden process outcome that selects a response branch. Household tasks therefore emphasize search and concurrent scheduling, whereas laboratory tasks additionally require revising a schedule after an outcome is observed. The domains have different action spaces and failure modes, so we report them separately rather than treating them as two difficulty levels or pooling them into one ranking.

Requirement parameters. The three requirement kinds of Section 3.3 are instantiated with bounded values. Household goal deadlines d_g are stated in the instruction and are specific to each task; in the laboratory the horizon is $D = 78$ ticks on 28 of the 33 tasks and 88–96 ticks on the remaining five, and the goal deadlines are stated against it. Latest starts are bounded: the 17 household bounds ℓ_p lie between 4 and 17 ticks, and the 31 laboratory bounds between 4 and 16 ticks. Check-and-response windows occur in the laboratory set, where the check window is $\kappa_p \in \{2, 5\}$ ticks, the response window is $\rho_p \in \{1, 5\}$ ticks, and the follow-up duration is $\phi_p \in \{0, 4, 6, 8\}$ ticks, with $\phi_p = 0$ when the response starts no follow-up process.

Table 6 summarises the two domains.

Table 6: What each domain withholds and how it becomes known. The domains are reported separately rather than pooled into a single score.

Domain	Unknown to the agent	How it becomes known
Household	item locations; latest-start bounds	inspecting containers and optional notices
Laboratory	which container holds each movable item, the latest start that binds a background process, together with which of two response chains the finished assay implies	inspecting containers, reading the experiment log, running the assay and then checking its result

The evaluated sets. Two domains are evaluated. The household set contains 38 tasks and the laboratory set contains 33. Three properties of these sets bear on how the numbers may be read.

All tasks are frozen. Each evaluated instance is a byte-frozen task-file copy distributed with the benchmark. At freeze time, we re-verified the feasibility of every instance with the full-information oracle from Section B.2: all 38 household and 33 laboratory instances support a valid trajectory respecting the step budget and tick deadline, provided the oracle operates exclusively through the public interface.

The step cap is uniform and is a safety valve. Both sets carry a single step budget of 40 for every task. It is at or above the oracle witness action count of every task in both sets (13 to 22 actions in household, 15 to 25 in laboratory), although an evaluated policy can still exhaust it. The binding budget is instead the tick deadline, which is per task and is not uniform: 13 of the 38 household tasks carry at most two ticks of slack over the oracle's own completion tick, while the laboratory tasks carry between 41 and 57 ticks, so what binds there is the discovered latest-start bound rather than the global deadline. We therefore report window misses and step-budget exhaustion separately throughout and never fold them into a single failure count.

They exercise different constraint classes. The household set carries published deadlines and an unknown location, and its temporal difficulty comes from the order in which background chores

and discoveries consume one clock; 13 of its 38 tasks additionally hide a latest-start bound in a notice that the agent may or may not open, and the environment enforces that bound either way, but the notice is not itself a success condition. In the laboratory set the same class is both present and declared: 22 of the 33 tasks key a bound to an experiment log that the task requires the agent to read, and all 33 carry a process whose result is revealed only by checking it, and whose revealed result selects one of two response chains; until the agent has run the assay and checked it, neither the required response nor the time it needs is known, so a constraint can become known late and change what the remaining schedule must contain. Between them the two domains instantiate every class of constraint that this paper formalises; only the laboratory set asks an agent to act on a constraint class whose existence it must first establish.

Household: an unknown location. The household domain combines several finite-duration chores, some of which run unattended in an appliance once started, with movable items hidden in containers. What the agent lacks is where those items are. The domain therefore probes whether a method can keep a parallel chore schedule running while still paying to find out, since every container opened is time not spent on a deadline.

Laboratory: an unknown bound and an unknown branch. The laboratory domain contains tasks in which a background assay must be started, checked at a reader, and then resolved through one of two response chains. Two kinds of incompleteness appear here: a latest-start bound that binds a background process and is recorded only in an experiment log, and the branch itself, which is not knowable until the assay has finished and its result has been read. The domain therefore probes whether a method goes looking for a constraint it does not yet hold, writes it into the schedule it is already running, and then revises its plan in response to a fact that arrives late.

How the domains relate. The domains are not successive difficulty levels. Household tasks isolate a state-estimation cost—finding an item—whose temporal consequence is the time spent searching. Laboratory tasks additionally hide a temporal bound and an outcome-dependent branch. The latter correspond to the inspection-revealed latest starts and check-and-response windows in Section 3.3; an unknown item location is instead a missing action precondition. Because the domains also differ in action space, episode length, deadlines, and failure modes, we interpret comparisons within each domain and do not combine them into one ranking. The same structure explains why information acquisition must be judged against the time available for the affected goal or process rather than against the global deadline alone: household tasks can leave little global slack, whereas laboratory tasks leave more but bind on the local bounds that inspection reveals, so the agent has to track discovered local obligations alongside the global clock.

Worked example. Figure 1 instantiates the instance tuple with a household task: prepare tea and sweep the living room before guests arrive. The room graph connects the kitchen, the living room, and the storage area; the kettle is a fixture that hosts the boiling process, and the tea leaves are a movable item hidden in one of the searchable containers. The placement map Ψ gives the container that holds the leaves, and that entry stays hidden until the agent searches the container. Boiling has a deterministic result here, so the hidden results of Section 3.3 do not affect the plan.

The instruction x asks for tea and a clean living room, and these two required goals form $\mathcal{G}$. The action space $\mathcal{A}$ includes movement, search, sweeping, filling and starting the kettle, brewing, and serving. The background processes $\mathcal{P}_{\mathrm{bg}}$ contain the kettle's boiling process p, which runs in the kettle after the agent reaches and prepares it; its cycle duration $\mathrm{dur}(p)$ is five minutes. The requirement set Γ states that tea can be served only after the leaves have been found and boiling has finished, and that both goals must be completed by the global deadline $D = 60$ ticks; one tick is 15 seconds.

The public observation initially exposes the rooms, fixtures, containers, the instruction and the deadline, but not the placement of the tea leaves. Searching consumes two minutes and reveals the tea leaves' location. If the agent first spends six minutes sweeping, then searches, fills and starts the kettle for one minute, waits through five minutes of boiling, and finally spends three minutes brewing and serving, every action is locally legal but the sequence misses the deadline (Figure 1a). If it starts the kettle first, the background process runs while the agent sweeps and searches, allowing both goals to finish in time (Figure 1b). The example shows why the agent must reason jointly about hidden information, foreground work, and the overlap created by background processes.

D EVALUATION PROTOCOL DETAILS

Fairness protocol. Some compared rows are our own ports of published systems rather than the published implementations, and a comparison is only meaningful if a port changes nothing except the agent; Appendix E states what each port changes. The protocol is applied identically to every method: only public observations reach an agent, and the oracle's plan, the hidden placements, the hidden bounds and the hidden outcome are never exposed; the task files are byte-frozen and released with the code; the step budget and the tick horizon come from the task file with no method-specific override; the number of trials per task is the value registered per method and is recorded in the manifest; and every method sees the same action vocabulary and the same evaluation interface.

Model and generation settings. All methods use the same frozen external model, deepseek-v4-flash, temperature 0 and a 128-token generation cap. The selector receives action words and current interactable values; only exact matches to the candidate set and public interface are executed. Model latency does not advance the environment clock, which is defined by action durations.

The selector prompt. Every decision is composed from public information only: the task instruction; the current public observation, with the candidate menu reported separately from it; a guidance block carrying the network summary and the recommended next focus, both recomputed from the network after the latest observation; the four most recent public feedback records; whichever experience channel is switched on for the run; and the catalogue of currently legal actions together with their public interactable values. The model composes one action, which is executed only if it matches a candidate exactly. A composed action can be rejected twice over: it may fail to parse against the catalogue, or the network may guard it because it would violate a known hard constraint. In either case the rejection reason is returned to the model and it is asked again, up to the registered repair budget; if that budget is exhausted, a deterministic fallback drawn from the surviving candidates is executed instead and recorded as a fallback rather than as a model decision. No step of this assembly reads the hidden placements, the hidden bounds, the hidden outcome or the oracle's plan.

Protocol and metrics. Trial counts are given in the main table; every episode resets the environment, with memory restricted to the same task. Success requires all goals on time and all mandatory sources inspected. Main success rates are reported by trial, whereas failure classes pool all registered episodes of each method. The window-miss group includes deadline termination with a refused start, step exhaustion with a refused start, and deadline termination without refusal. Pure step exhaustion is reported separately to distinguish failure to complete from an explicit temporal-window violation. Avoiding violations alone does not imply completion, so the taxonomy explains success rates rather than replacing them.

E BASELINE ADAPTATION

Some of the compared rows are ports of published systems rather than artefacts released for this benchmark: the two rows marked ‡, the adapted PLANTOR and the adapted Agent-C, are our adaptations of the corresponding published methods. In each case the port removes an obstacle that is unrelated to the method's own mechanism — an interface mismatch, or an assumption that the evaluated setting violates — and nothing else. The adapted method is therefore a stronger baseline here than its published form would be, and the comparisons it supports are conservative with respect to our own method.

PLANTOR, adapted. PLANTOR plans once from a validated symbolic knowledge base and then schedules that static plan with a constraint solver. It assumes that the knowledge base grounds every action precondition, which is exactly what a partially observed task withholds: an item's location is unknown until it is found, so a plan that names the item cannot be executed at all. Our port keeps the published planning core unchanged — a symbolic partial-order plan, scheduled once by OR-Tools CP-SAT and executed by a behaviour-tree-style executor that checks action preconditions — and adds only what is needed to run it: goal resolution read from the public task specification rather than a fixed job table, per-goal deadlines read from the task rather than a single household field, laboratory chains priced by the route the executor actually walks rather than by a constant, and a fixed container-search macro in front of the plan. It still never replans: no observation arriving after reset changes its plan or its schedule, which is the published behaviour, so the row shows what static planning achieves once the only added ingredient is search.

Agent-C, adapted. Agent-C compiles temporal safety properties into a symbolic specification and filters the model's choices against it, rejecting any candidate action that would make completion impossible. As published, that specification is supplied. Here it has to be acquired, because an agent sees only public observations and the bounds that matter are revealed by inspecting sources. Our adaptation keeps the published mechanism — a symbolic encoding, a per-step admissibility check solved with z3, constrained selection over the surviving candidates, and a monitor over live obligations — and changes only its input: the specification compiler accepts a constraint only when it is derivable from the public observation stream, so the same filter runs over an incrementally discovered specification instead of an oracle one, and the selector is additionally given the recent public action–feedback pairs that every other method receives. Both changes remove a disadvantage that has nothing to do with the method's filtering mechanism, so the row is a strong version of Agent-C rather than a weakened one.

F EXPERIENCE DETAILS

The network and selector above operate within a single episode. For settings in which the same task is attempted more than once, we additionally study an optional experience store. Its scope is one task identifier and it is reinitialised for every new task: no fact, rule, or model history crosses task boundaries. The store has two independently switchable channels, kept separate because they have different roles.

Channel A — public world facts. Item-to-container pairs that the agent actually observed during a previous episode of the same task. Only pairs that appeared in the agent's own public observation stream are stored; a transient inventory is explicitly excluded. Before each decision, stored facts for the current task are merged into the agent's routing state under the rule that the current episode wins: an observation made in the current episode always overrides a remembered one.

Channel B — temporal experience. Free-text scheduling advice produced once per episode by the language model from the public trajectory, the terminal goal status, and the public network summary. A rule is stored with the goal set it applies to and the episode that supports it. At decision time, the three most recent rules whose required goals match the current task are retrieved. Rules are injected into the prompt as advice only: they cannot alter a hard constraint, a deadline, a confirmed event time, or an action precondition. The same channel also carries, in structured form, the bounds that a previous trial publicly discovered: each one is a public latest-start or latest-response record revealed by the environment. It is replayed into the network at reset and passes the same consistency guard as a freshly discovered bound. That replay is deterministic and contains nothing the agent did not observe in an earlier trial of the same task; the model-authored text never enters the network.

Observability is preserved. Experience never bypasses the environment's partial observability. Remembering that an item is in a particular container is a routing hint: the agent must still travel to the room and perform an inspect action before a take becomes legal, and the environment admits take only from containers opened during the current episode. Experience therefore removes search cost, not the observation requirement, and this distinction is enforced by the environment preconditions.

G ABLATION DETAILS

Failure profiles. The two laboratory network ablations record no window miss and end 69.7% of their episodes in pure step exhaustion, against 21.2% for Online-STN, which misses a window in 6.1% of its episodes. Avoiding an explicit violation is therefore not sufficient: a policy can remain unsuccessful without ever attempting a temporally invalid action. These rates support the completion advantage of the persistent representation, but they do not show that rebuilding necessarily loses historical evidence, since Rolling-STN also receives the public history. The lower laboratory inspection counts of the two reconstruction variants similarly reflect budgets consumed by repeated process starts and repositioning rather than by search.

Scope of the network comparison. This comparison concerns the agents defined by their network-maintenance rules, not an expressiveness advantage of persistence over reconstruction: graphs containing the same time points, confirmed times and difference constraints have the same STN closure. The empirical distinction is the temporal state that each variant maintains and supplies to the shared

selector, and aggregate success alone does not identify which missing or different constraint accounts for a failed reconstruction-based decision.

Interpreting the experience ablation. The household success gaps among these variants are at most 5.3 percentage points, which is within the variation observed across evaluation rounds of an identical first-trial prompt, so we read the success ordering descriptively and make no significance claim; channel B also bundles bound replay with textual advice, so the comparison does not separate their individual effects. In the laboratory domain neither channel alone changes the outcome (72.7% for location memory and 75.8% for rule reflection, against 72.7% without memory) while both reduce irrelevant inspections, from 0.61 to 0.27 and 0.52. A single channel is therefore not sufficient there either: remembering where material was found, or which bound was observed, does not by itself schedule the response that an assay outcome requires within its local window.