SAGAS: Semantic-Aware Graph-Assisted Stitching for Offline Temporal Logic Planning

Ruijia Liu **Ancheng Hou** **Xiang Yin***
School of Automation and Intelligent Sensing
Shanghai Jiao Tong University, Shanghai 200240, China
{liuruijia,hou.ancheng,yinxiang}@sjtu.edu.cn

Abstract

Linear Temporal Logic (LTL) provides a rigorous framework for specifying long-horizon robotic tasks, yet existing approaches face a trade-off: model-based synthesis relies on accurate labeled transition systems, whereas learning-based methods often require online interaction, task-specific rewards, or specification-conditioned training. We study LTL-specified robotic planning and execution in an offline, model-free setting, where the agent is given only fixed, task-agnostic trajectory fragments, with no dynamics model, task demonstrations, or online data collection. To address this setting, we propose **SAGAS**, a framework that combines the compositionality of symbolic synthesis with the data-driven reachability structure learned from offline trajectories. SAGAS first learns a reusable latent reachability graph and a frozen goal-conditioned executor from fragmented offline data. For each new LTL formula, it performs task-time semantic graph augmentation to ground state-defined propositions on the learned graph and applies Büchi product search to synthesize a cost-aware accepting prefix–suffix waypoint plan executed by the frozen executor. This separation of learning and planning enables zero-shot generalization to unseen LTL formulas and new predicates supported by the offline data, without task-specific rewards, additional data collection, or policy retraining. Experiments on LTL task suites constructed from OGBench locomotion domains show that SAGAS achieves higher success rates and lower execution costs than baselines on diverse unseen tasks, with further evaluation in OGBench Scene demonstrating its applicability to robotic manipulation.

1 Introduction

Linear Temporal Logic (LTL) provides a rigorous framework for specifying long-horizon robotic tasks whose requirements go beyond goal-reaching, such as sequential assembly, persistent surveillance, recurrence, and safety constraints [1, 2, 3]. Automata-theoretic planning handles such specifications by reducing satisfaction to search over the product of a labeled transition system and a Büchi automaton [4, 5, 6]. This offers a compositional and interpretable route to temporal-logic planning, but it assumes access to an accurate transition model whose states carry task-relevant proposition labels and whose transitions are dynamically feasible. In many robotic domains, such a labeled transition model is unavailable. Learning-based alternatives relax this modeling assumption by encoding logical objectives into rewards [7, 8, 9], augmenting policies with automaton states, or decomposing specifications into reusable skills and subgoals [10, 11, 12, 13]. Some of these methods support compositional or zero-shot generalization to new temporal specifications by reusing learned skills, goal-conditioned policies, or symbolic subgoal interfaces. However, this generalization is typically tied to a prescribed skill, goal, or proposition library, or to a distribution of specifications

*Corresponding author: Xiang Yin (yinxiang@sjtu.edu.cn). Project page: cps-sjtu.github.io/SAGAS.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

seen during training. Such an interface can be restrictive when new propositions are defined directly over the robot state at test time, or when the available data consist only of task-agnostic trajectory fragments rather than demonstrations or rollouts for the relevant logical tasks (see Appendix A for extended related work).

We study LTL-specified robotic task planning and execution in an offline, model-free setting. The agent is given only a fixed dataset of task-agnostic trajectory fragments, with no dynamics model, task demonstrations, or online data collection. Rather than assuming a pre-specified library of temporal-logic skills or proposition-conditioned policies, we aim to learn reusable motion structure directly over the state space and compose it at test time according to a newly provided LTL formula. This calls for an abstraction that captures dataset-supported motion connectivity while allowing task-time propositions to be grounded through the state interface.

To address this setting, we propose **SAGAS**, which extends Graph-Assisted Stitching (GAS) [14] from single-goal trajectory stitching to temporal-logic synthesis over offline data. Similar to GAS, SAGAS first learns a temporal-distance representation from fragmented offline trajectories, constructs a latent reachability graph in the learned space, and trains a goal-conditioned low-level policy to execute local transitions between graph waypoints. This offline stage produces a reusable proxy for dataset-supported reachability, while the low-level policy provides a model-free executor for short-range waypoint tracking. Crucially, these reusable components are learned without LTL task information, task rewards, or a pre-specified library of symbolic skills.

Given a new LTL task, SAGAS instantiates the symbolic layer only at test time. It augments the latent reachability graph with task-relevant proposition semantics evaluated on the stored state support, converting the reusable motion graph into a data-supported labeled abstraction suitable for automata-guided synthesis. SAGAS then searches the product of this semantic graph and the Büchi automaton to synthesize an accepting prefix–suffix waypoint plan, where graph costs serve as learned reachability surrogates and automaton transitions encode logical progress. The resulting high-level plan is executed by the pretrained frozen low-level policy. Through this decoupling, formula-specific reasoning is handled by semantic graph augmentation and symbolic product search, while the learned reachability proxy and executor are reused across tasks. Consequently, SAGAS generalizes zero-shot to unseen LTL formulas and new state-defined propositions in both planning and execution, provided that the propositions can be evaluated at task time and are sufficiently supported by the offline data. No task-specific reward design, data collection, or policy retraining is required.

Our contributions are summarized as follows. First, we propose **SAGAS**, a framework that extends GAS-style trajectory stitching to LTL planning and execution by separating reachability learning on fixed, task-agnostic trajectories from task-time symbolic synthesis. Second, we introduce a task-time semantic graph augmentation and cost-aware product-synthesis procedure over the learned latent graph, grounding state-defined propositions through preserved state support and optimizing accepting prefix–suffix waypoint plans using dataset-supported reachability costs. Finally, on LTL task suites constructed from the OGBench AntMaze and HumanoidMaze domains, we show that SAGAS synthesizes and executes prefix–suffix behaviors zero-shot from fragmented offline data, improving finite-lasso execution success and execution cost over three baselines, especially on hard specifications. Further experiments in OGBench Scene demonstrate its applicability to robotic manipulation.

2 Preliminaries

2.1 System model

We consider a discrete-time dynamical system with unknown transition dynamics, defined over a continuous state space $\mathcal{S} \subseteq \mathbb{R}^{d_s}$ and an action space $\mathcal{U}_{\text{act}} \subseteq \mathbb{R}^{d_a}$. The system evolves as $s_{t+1} = f(s_t, a_t)$, where $f : \mathcal{S} \times \mathcal{U}_{\text{act}} \to \mathcal{S}$ is an *unknown* deterministic transition function. To relate the system state to high-level tasks, we define a projection map $\Pi : \mathcal{S} \to \mathcal{X}$, where $\mathcal{X} \subseteq \mathbb{R}^{d_t}$ denotes the *task space* (e.g., robot workspace). While the control inputs act on the high-dimensional state space $\mathcal{S}$, the logical predicates defining the task objectives are specified over the task space $\mathcal{X}$.

2.2 Linear temporal logic task

Atomic propositions. For a test-time LTL specification ϕ, let $\mathcal{AP}_\phi = \{\ell_1, \ldots, \ell_m\}$ denote the finite set of atomic propositions appearing in ϕ. Each proposition $\ell \in \mathcal{AP}_\phi$ is associated with an evaluable predicate over the task space; for concreteness, we represent it as a measurable labeled region $\mathcal{R}_\ell \subseteq \mathcal{X}$. The induced labeling function is $L_\phi(s) = \{\ell \in \mathcal{AP}_\phi \mid \Pi(s) \in \mathcal{R}_\ell\}$. The set can be empty when $\Pi(s)$ lies outside all labeled regions. A state can satisfy several propositions simultaneously, and the same labeling interface describes predicates over joint object configurations.

Linear temporal logic. We specify high-level tasks using Linear Temporal Logic without the "next" operator (LTL$_{-\mathrm{X}}$), a stutter-invariant fragment in which finite repetitions of the same proposition assignment do not affect formula satisfaction. This fragment is widely adopted for robotic tasks in continuous space, where moving between labeled regions takes a variable number of time steps [1, 15]. The syntax of LTL$_{-\mathrm{X}}$ is recursively defined as:

$$\phi ::= \text{true} \mid \ell \in \mathcal{AP}_\phi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \,\mathrm{U}\, \phi_2,$$

where $\neg$, $\wedge$, and $\vee$ denote logical negation, conjunction, and disjunction, respectively, and U is the "until" operator. Common temporal operators such as "eventually" ($\mathrm{F}\phi = \text{true}\,\mathrm{U}\,\phi$) and "always" ($\mathrm{G}\phi = \neg\mathrm{F}\neg\phi$) are derived as standard abbreviations. Given a trajectory $\tau = s_0 s_1 s_2 \cdots$, the sequence of labels $L_\phi(s_0)L_\phi(s_1)L_\phi(s_2)\cdots$ constitutes an infinite word $\sigma \in (2^{\mathcal{AP}_\phi})^\omega$. We write $\tau \models \phi$ if the induced word σ satisfies ϕ according to standard LTL semantics [2]. Any LTL formula ϕ can be translated into a Nondeterministic Büchi Automaton (NBA) $\mathcal{B}_\phi = (Q, Q_0, \Sigma, \delta, F)$, where $\Sigma = 2^{\mathcal{AP}_\phi}$ is the alphabet. An infinite run $q_0 q_1 \ldots$ is accepting if it visits the set of accepting states F infinitely often. If the automaton admits an accepting run, it admits one in lasso form: a finite *prefix* leads from an initial state to an accepting state, and a nonempty *suffix* cycle returns to that state and repeats indefinitely [2, 5, 16].

2.3 Problem setting

We consider an *offline* setting in which the agent has no access to the transition map f and no additional environment interaction is allowed for data collection, policy learning, or planning. Learning and planning must rely solely on a fixed dataset $\mathcal{D} = \{\tau^{(i)}\}_{i=1}^N, \tau^{(i)} = \big(s_0^{(i)}, a_0^{(i)}, s_1^{(i)}, \ldots, a_{H_i-1}^{(i)}, s_{H_i}^{(i)}\big)$, collected by unknown behavior policies. The trajectories are task-agnostic and fragmented: they are not assumed to individually demonstrate the target LTL specification, nor are they labeled with the propositions that may appear in downstream tasks.

At test time, the agent is given an initial state s_0 and an LTL$_{-\mathrm{X}}$ specification ϕ over a finite set of state-evaluable propositions $\mathcal{AP}_\phi$. The associated task-space regions or predicate evaluators define a labeling function $L_\phi : \mathcal{S} \to 2^{\mathcal{AP}_\phi}$ through the projection Π. The formula, propositions, and labeled regions are introduced only at test time. As in standard offline learning, reliable synthesis is limited by the support contained in the fixed dataset $\mathcal{D}$, as further discussed in Appendix D.

We seek a hierarchical solution consisting of a reusable low-level executor learned from $\mathcal{D}$ and a task-specific high-level prefix–suffix plan $\Gamma = (\Gamma_{\mathrm{pre}}, \Gamma_{\mathrm{suf}})$. Following the standard prefix–suffix representation used in automata-based LTL planning [5, 16, 17, 18, 15], Γ_{pre} is a finite sequence of high-level subgoals executed once to reach an accepting automaton state, and Γ_{suf} is a finite cycle of subgoals repeated thereafter. Let τ_Γ denote the physical trajectory induced by executing Γ from s_0 with the learned executor. The desired outcome is $\tau_\Gamma \models \phi$, where satisfaction is evaluated using the test-time labeling function L_ϕ.

Among satisfying prefix–suffix plans, we seek to minimize the execution-cost objective

$$J_{\mathrm{exec}}(\tau_\Gamma; \Gamma) = \lambda\, T_{\mathrm{pre}}(\tau_\Gamma; \Gamma) + (1-\lambda)\, T_{\mathrm{suf}}(\tau_\Gamma; \Gamma), \tag{1}$$

where T_{pre} is the number of environment steps used to complete the prefix, T_{suf} is the number of steps used to complete one suffix cycle, and $\lambda \in [0, 1]$ balances transient and recurrent performance. Since the true execution cost is unavailable during offline planning, the method below optimizes a graph-based surrogate of (1) using dataset-supported reachability costs and evaluates execution cost from realized rollouts. This formulation separates task-time symbolic planning from reusable execution: new formulas and labeling functions determine the high-level prefix–suffix plan Γ, while the low-level executor is learned once from $\mathcal{D}$ and reused without task-specific retraining.

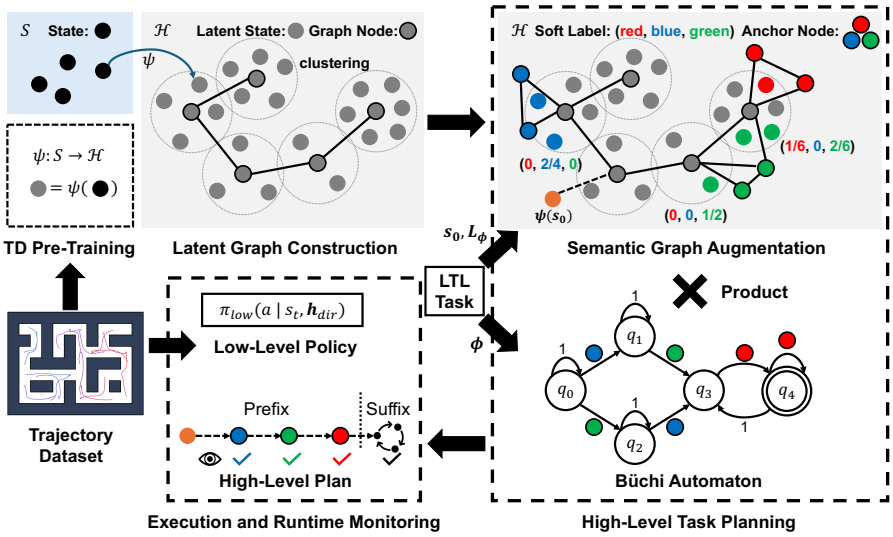


Figure 1: Overview of **SAGAS**. A reusable reachability interface is built from task-agnostic offline data. Given a new LTL formula and labeling function, semantic graph augmentation and Büchi-product search produce an accepting prefix–suffix plan, executed by the frozen low-level policy under runtime monitoring.

3 Our method

3.1 Overview

To address this problem, SAGAS separates offline motion learning from task-time symbolic synthesis (Figure 1). The offline phase constructs a formula-agnostic reachability interface from the fixed trajectory dataset. This interface consists of three reusable objects: a weighted latent reachability graph $\mathcal{H}_{\text{graph}} = (\mathcal{V}, \mathcal{E}, w)$, raw-state support sets $\{\mathcal{D}_v\}_{v \in \mathcal{V}}$ for graph nodes, and a frozen low-level policy π_{low} for executing local graph transitions. The graph and its edge weights summarize dataset-supported local connectivity, while the support sets preserve the raw-state evidence needed to ground future propositions. The low-level policy realizes the selected waypoint transitions in the physical system. Throughout the paper, a *waypoint* denotes a selected latent-graph node used as a high-level subgoal, not a hand-specified task-space point. These components are built once from the task-agnostic dataset and do not depend on downstream LTL formulas, proposition regions, task rewards, or automaton states.

At task time, SAGAS receives a task instance (s_0, ϕ, L_ϕ), specifying the initial state, LTL formula, and labeling function. The augmentation stage uses L_ϕ to ground propositions on the preserved state support and incorporates s_0 into a task-specific semantic graph (Section 3.3). This graph is the input to the product planner, which couples it with the task automaton and uses learned reachability costs to guide synthesis of an accepting prefix–suffix plan (Section 3.4). The plan is then passed to the execution layer: its waypoints provide goals for the frozen policy, while its semantic obligations guide runtime monitoring of the realized trajectory (Section 3.5). Algorithm 1 summarizes this task-time workflow, with detailed algorithms provided in Appendix B.

Algorithm 1 SAGAS: task-time planning and execution

Input: Frozen $(\mathcal{H}_{\text{graph}}, \{\mathcal{D}_v\}, \psi, \pi_{\text{low}})$ and offline states $\mathcal{D}_S$; task (s_0, ϕ, L_ϕ); prefix budget K
Output: Product plan $\rho^\star$, trajectory τ, and execution outcome, or search failure

Translate ϕ to the original NBA $\mathcal{B}_\phi$ and normalize its guard terms $\mathcal{G}_\phi$
Compute soft labels $\widehat{P}$ from $\{\mathcal{D}_v\}$ and L_ϕ
Retrieve witness states from $\mathcal{D}_S$ for individual labels and joint terms in $\mathcal{G}_\phi$, forming connected anchor nodes $\mathcal{A}$ with full assignments $L_\phi(s)$
Insert $\mathcal{A}$ and s_0 into the graph to form $(\mathcal{H}_\phi^{\text{sem}}, v_0)$ with $\widehat{P}$
$Q_{\text{aligned}} \leftarrow \bigcup_{q_0 \in Q_0} \delta(q_0, L_\phi(s_0))$
Restrict future guards by support; form $(\widetilde{\mathcal{B}}_\phi, \mathcal{Z}_0)$ from Q_{aligned}
$\mathcal{C}_{\text{pre}} \leftarrow \text{PrefixSearch}(\mathcal{Z}_0, \mathcal{V}_\phi \times \widetilde{F}, K)$; $\mathcal{C} \leftarrow \varnothing$
for all $\rho_{\text{pre}} \in \mathcal{C}_{\text{pre}}$ **do**
  $z_f \leftarrow \text{LastState}(\rho_{\text{pre}})$; $\rho_{\text{suf}} \leftarrow \text{AcceptingSuffix}(z_f)$
  if a valid suffix ρ_{suf} is found **then**
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{(\rho_{\text{pre}}, \rho_{\text{suf}})\}$
  end if
end for
if $\mathcal{C} = \varnothing$ **return** no accepting lasso found within the candidate budget
$\rho^\star \in \arg\min_{\rho \in \mathcal{C}} J_{\text{graph}}(\text{proj}_{\mathcal{V}_\phi}(\rho))$
$(\tau, \text{outcome}) \leftarrow \text{ExecuteAndMonitor}(\rho^\star, \mathcal{H}_\phi^{\text{sem}}, \psi, \pi_{\text{low}}, \mathcal{B}_\phi, L_\phi)$
return $\rho^\star$, τ, and outcome

3.2 Offline reachability interface

We instantiate the support-preserving interface with a GAS-style temporal-distance graph and low-level executor [14]. These objects expose reusable motion structure that can be labeled and searched once a formula is specified. After the offline stage, the graph, support, and executor are frozen and reused across downstream tasks. Temporal-distance learning, filtering, clustering, and executor training are detailed in Appendix B.1.

Latent temporal-distance graph. Let $\psi : \mathcal{S} \to \mathcal{H}$ denote an embedding learned from the offline trajectories, where $\mathcal{H} \subseteq \mathbb{R}^{d_h}$ is the latent representation space. The embedding is trained so that Euclidean distances between embeddings approximate temporal distance, or step-to-go reachability, between the underlying system states. SAGAS uses this latent distance as a proxy for local reachability cost.

Using ψ, SAGAS first applies the GAS temporal-efficiency rule to select trajectory states whose observed motion aligns with the learned reachability geometry. The retained states are then clustered in the latent space at a scale set by H_{TD} to form a weighted graph $\mathcal{H}_{\text{graph}} = (\mathcal{V}, \mathcal{E}, w)$. Each node $v \in \mathcal{V}$ corresponds to a latent cell $\mathcal{C}_v \subseteq \mathcal{H}$, i.e., a cluster of embedded dataset states. With a slight abuse of notation, we also use v to denote the representative point of this cell in latent space. The temporal-distance scale H_{TD} also controls local connectivity: nodes within this distance are connected, and nearest-node links are added between disconnected components. For distinct nodes, the local connection rule and edge costs are

$$\|u - v\|_2 \le H_{\text{TD}} \implies (u, v) \in \mathcal{E}, \qquad w(u, v) = \|u - v\|_2 \quad \forall (u, v) \in \mathcal{E}. \tag{2}$$

The resulting graph summarizes motion connectivity learned from full-state trajectories, with edge weights providing empirical estimates of transition cost.

Raw-state support map. The latent graph alone is insufficient for temporal-logic planning because LTL propositions are evaluated in the original state or task space, not directly in the learned latent space. SAGAS therefore retains the raw dataset support associated with each graph node. Let $\mathcal{D}_S$ denote the set of states appearing in the offline dataset. For each node v, we store a support set $\mathcal{D}_v \subseteq \{s \in \mathcal{D}_S \mid \psi(s) \in \mathcal{C}_v\}$. This support map connects each latent cell to observed physical states, allowing test-time predicates to be evaluated through Π and grounded on the graph.

Frozen low-level executor. SAGAS learns a low-level policy π_{low} from the same dataset and temporal scale H_{TD} to execute local graph transitions. Conditioned on the target waypoint's latent direction, it tracks that waypoint until the execution rule declares completion or the tracking budget is exhausted.

3.3 Semantic graph augmentation

The offline graph $\mathcal{H}_{\text{graph}}$ encodes dataset-supported reachability, but it is not yet a labeled abstraction for LTL synthesis. Given a task instance with initial state s_0, test-time formula ϕ over propositions $\mathcal{AP}_\phi$, and labeling function $L_\phi : \mathcal{S} \to 2^{\mathcal{AP}_\phi}$, SAGAS performs semantic graph augmentation according to the supplied proposition predicates. The augmentation first estimates empirical labels for existing graph nodes and then inserts semantically explicit nodes for the task-relevant labeled regions.

Soft labels for empirical graph semantics. The latent graph provides a compressed representation of the offline dataset, with each node representing a cluster of embedded states. As the embedding ψ has no inverse decoder, a node's latent coordinate cannot be directly converted into a physical state for proposition evaluation. SAGAS therefore evaluates test-time propositions on the node's preserved raw-state support to estimate how strongly the node is associated with each proposition. For an original graph node v, the empirical label estimate, or *soft label*, is written in weighted form as

$$\widehat{P}(\ell \mid v) = \frac{\sum_{s \in \mathcal{D}_v} K_v(s) \mathbf{1}\{\ell \in L_\phi(s)\}}{\sum_{s \in \mathcal{D}_v} K_v(s)}. \tag{3}$$

Here $K_v(s) \geq 0$ is a support-state weight. Intuitively, $\widehat{P}(\ell \mid v)$ measures the weighted proportion of the node's preserved support that satisfies proposition ℓ: values near one indicate that most support states satisfy it, whereas values near zero indicate little empirical association. These soft labels provide graph search with a semantic reference for deciding which nodes to traverse. They are particularly useful for negative literals: when a guard forbids ℓ, the planner uses the estimate to avoid nodes strongly associated with that proposition. Specifically, an original graph node v passes this negative-literal check only when its support has positive total weight and $\widehat{P}(\ell \mid v) \leq \tau_{\text{soft}}$. Appendix B.2 details the screening mechanism.

Anchor nodes for labeled regions. Soft labels characterize an original graph node through its preserved support, which can contain states both inside and outside a proposition region. Consequently, tracking the node's latent representative may reach a state where the desired proposition is false. To provide an explicit semantic target, SAGAS retrieves label-satisfying dataset states and embeds them as *anchor nodes*. Each anchor $a_s = \psi(s)$ retains the full label set $L_\phi(s)$ of its observed witness s and is connected to the graph using the same temporal-distance scale H_{TD}.

The stored label set allows the planner to determine which logical transitions an anchor can witness. For a conjunctive guard term η with positive and negative proposition sets P_η and N_η, a connected anchor is compatible when its witness satisfies $P_\eta \subseteq L_\phi(s)$ and $N_\eta \cap L_\phi(s) = \varnothing$. For example, a transition requiring $a \wedge b$ uses an anchor whose witness state satisfies both propositions simultaneously. SAGAS retains anchors for individual propositions and retrieves additional witnesses as needed for joint requirements. If the required witness cannot be found in the available data or connected to the graph, SAGAS reports the guard term as unsupported by the current abstraction and disables that alternative. Appendix B.3 details witness retrieval and insertion.

Semantic interface for product search. The resulting semantic graph $\mathcal{H}_\phi^{\text{sem}} = (\mathcal{V}_\phi, \mathcal{E}_\phi, w_\phi)$ combines soft labels, anchor witnesses, and inherited reachability costs. It also embeds and connects a temporary start node for s_0; if local insertion yields no connection, the nearest graph node provides an initialization heuristic. This separation keeps the offline reachability graph compact and formula-agnostic. Once a task is specified, soft labels and anchors add semantic resolution only where needed for its propositions, avoiding uniform refinement of the entire graph. Appendix B.5 gives the full construction.

3.4 High-level planning over the semantic graph

With this semantic interface in place, SAGAS turns the test-time LTL task into a search problem over the product of the semantic graph and the task automaton. The graph supplies dataset-supported connectivity and learned reachability costs, while its soft labels and anchors provide semantic evidence for enabling automaton transitions. The resulting product search couples logical progress with dataset-supported reachability, using the learned costs to guide the synthesis of an accepting prefix–suffix plan.

Büchi automaton preprocessing. Given the LTL formula ϕ, SAGAS translates it into a Nondeterministic Büchi Automaton (NBA) $\mathcal{B}_\phi = (Q, Q_0, \Sigma, \delta, F)$. We represent δ symbolically: for each pair (q, q'), the Boolean guard $g_{q,q'}$ is satisfied by exactly those proposition assignments $\sigma \in \Sigma$ for which $q' \in \delta(q, \sigma)$. Before search, SAGAS converts these guards into disjunctive normal form (DNF), so that each conjunctive alternative can be checked independently against the semantic graph. Contradictory terms are discarded, and a term containing positive literals is retained only if a connected anchor witnesses the complete term. The remaining alternatives define the support-restricted automaton $\widetilde{\mathcal{B}}_\phi$, with states $\widetilde{Q} \subseteq Q$ and accepting set $\widetilde{F} = F \cap \widetilde{Q}$. Appendix B.4 details preprocessing.

Product search and prefix–suffix synthesis. SAGAS follows the classical prefix–suffix approach to Büchi acceptance: a prefix reaches an accepting product state, and a suffix returns to that state so that its repetition yields infinitely many accepting visits [5]. Both stages operate on the implicit product system

$$\mathcal{P}_\phi = \mathcal{H}_\phi^{\mathrm{sem}} \otimes \widetilde{\mathcal{B}}_\phi, \qquad \mathcal{Z}_\phi = \mathcal{V}_\phi \times \widetilde{Q}.$$

Here $\mathcal{V}_\phi$ is the node set of the semantic graph, and each product state $(v, q) \in \mathcal{Z}_\phi$ pairs a graph node with a state of the support-restricted automaton.

Prefix search. The prefix search starts from the product states associated with the initial system state s_0 and aims to reach the accepting set $\mathcal{V}_\phi \times \widetilde{F}$. To establish these starting states, SAGAS first reads the observed assignment $L_\phi(s_0)$ in the automaton, obtaining the possible automaton states Q_{init} and the initial product set $\mathcal{Z}_0$:

$$Q_{\mathrm{init}} = \{q' \in \widetilde{Q} \mid \exists q_0 \in Q_0,\ q' \in \delta(q_0, L_\phi(s_0))\}, \qquad \mathcal{Z}_0 = \{(v_0, q) : q \in Q_{\mathrm{init}}\}.$$

Here v_0 is the temporary start node introduced during semantic graph augmentation. The initialization uses the original NBA's initial-state set Q_0 and transition relation δ, consuming $L_\phi(s_0)$ once.

SAGAS then explores the product from $\mathcal{Z}_0$ using A*-style search. Each expansion combines a neighboring graph node v' with an outgoing automaton transition $q \to q'$ whose guard is supported by the semantic evidence at v'.

The semantic check follows the soft-label and anchor rules introduced above. For a guard term η with positive and negative proposition sets P_η and N_η, $\mathrm{PosOK}(v', \eta)$ requires v' to be an anchor whose witness satisfies all propositions in P_η; it is automatically true when P_η is empty. The check $\mathrm{NegOK}(v', \eta)$ requires every forbidden proposition to have soft label at most τ_{soft} at an original graph node with positive total support weight, or to be absent from the observed assignment at an anchor or start node. It is automatically true when N_η is empty; an original graph node without valid support fails a nonempty negative check. Writing $\widetilde{g}_{q,q'}$ for the retained guard and $\mathcal{E}_\phi$ for the semantic graph's edge set, the movement successors satisfy

$$((v, q), (v', q')) \in \mathcal{E}_{\mathcal{P}}^{\mathrm{mov}} \iff (v, v') \in \mathcal{E}_\phi \wedge \exists \eta \in \mathrm{DNF}(\widetilde{g}_{q,q'}) : \mathrm{PosOK}(v', \eta) \wedge \mathrm{NegOK}(v', \eta). \quad (4)$$

Each movement edge inherits the cost $w_\phi(v, v')$ of its semantic-graph edge. SAGAS generates successors from this relation as needed during search, using accumulated edge costs and a graph-distance heuristic to guide expansion toward accepting states. This on-demand construction uses the same transition rules as an explicitly constructed product graph. When an accepting state is reached, the planner reconstructs its prefix and retains it for suffix construction. Because different endpoints can admit different return cycles, SAGAS retains up to K low-cost prefixes, stopping when this budget is reached or the search frontier is exhausted.

Suffix search. At each retained endpoint $z_f = (u, q_f)$, SAGAS first checks whether q_f has a self-loop whose guard is enabled at u. If so, the suffix can stay at the same waypoint with zero graph cost. Otherwise, SAGAS searches for a nonempty path back to z_f using the same movement transitions

and edge costs as in prefix search. Only prefixes with a valid suffix enter the final cost comparison. The complete successor relation and search procedures are given in Appendices B.5 and B.6.

Graph surrogate objective. Each product plan $\rho = (\rho_{\mathrm{pre}}, \rho_{\mathrm{suf}})$ has a waypoint projection $\Gamma = \mathrm{proj}_{\mathcal{V}_\phi}(\rho) = (\Gamma_{\mathrm{pre}}, \Gamma_{\mathrm{suf}})$, obtained by retaining the graph-node component of each product state. SAGAS selects the candidate $\rho^\star$ minimizing the graph surrogate cost

$$J_{\mathrm{graph}}(\Gamma) = \lambda\,\mathrm{Cost}(\Gamma_{\mathrm{pre}}) + (1-\lambda)\,\mathrm{Cost}(\Gamma_{\mathrm{suf}}), \tag{5}$$

where $\mathrm{Cost}(\cdot)$ sums the edge weights of graph movements and $\lambda \in [0,1]$ trades off transient and recurrent efficiency. This objective serves as a graph-level surrogate for the execution objective in Eq. (1), replacing the environment-step counts for the prefix and one suffix traversal with sums of learned reachability costs. Since the actual step counts are unavailable during planning, SAGAS ranks the generated candidates using this surrogate. Detailed search and monitoring procedures are given in Algorithms B.2 and B.3.

3.5 Plan execution and runtime monitoring

SAGAS follows the waypoint projection of $\rho^\star$ with the frozen low-level policy, using runtime monitoring to decide when to advance. Waypoints inherited from the offline graph guide intermediate motion and may be skipped when locally reachable within H_{TD}. The task-time anchors selected by the plan instead serve as mandatory semantic checkpoints, where both latent proximity and the required positive labels under L_ϕ must be verified. As the policy tracks each selected target, the monitor starts from the confirmed source automaton state, $S = \{q_i\}$, and updates $S \leftarrow \bigcup_{q\in S} \delta(q, L_\phi(s_t))$ on each new observation using the original NBA. Reaching the target advances the plan only if its planned automaton state q_j belongs to S and the guard checks pass. On confirmation, the monitor resets to $S = \{q_j\}$ and processes subsequent observations for the next segment.

Throughout this motion, the monitor also checks negative obligations until their release. To help the executor avoid violations between waypoints, navigation tracking adds a bounded repulsive bias away from nearby anchors associated with active forbidden labels. If an anchor is missed, tracking times out, or an active negative obligation is violated, the planned product transition is reported as unrealized. After completing the prefix, the executor repeats the suffix under the same monitoring rules. A cycle is counted as complete only when the suffix endpoint is reached and the monitor confirms a return to the planned accepting state from the labels observed during that cycle. Appendix B.7 gives the full execution procedure.

Proposition 1 (Soundness of synthesis and monitored execution) *For every product plan ρ returned by SAGAS, the initial label and the guards along its prefix and repeated suffix define a nonempty set of infinite label words accepted by the original Büchi automaton $\mathcal{B}_\phi$. During execution, the labels observed between consecutive targets confirmed by the monitor admit a run of this automaton connecting their planned automaton states. If the prefix is completed and the monitor confirms completion of infinitely many suffix cycles, then $\tau_\Gamma \models \phi$.*

Proof sketch. Each selected guard term is satisfiable and implies an original NBA guard. Any label word consistent with the initial label and selected guard sequence therefore admits an accepting run along the plan's automaton states. During execution, induction on the monitor updates shows that each confirmed segment admits an automaton run between its planned source and target states, allowing successive runs to be concatenated at their shared states. This run follows the observed physical labels and visits the accepting state on every confirmed suffix cycle, so infinitely many such cycles imply $\tau_\Gamma \models \phi$. Appendix C gives the full proof.

4 Experiments

Our experiments examine the reliability, efficiency, and generality of zero-shot LTL planning from fixed offline data. **Q1:** Does semantic product planning improve LTL satisfaction under a shared offline backbone? **Q2:** Does joint graph–automaton cost optimization reduce capped execution cost relative to decoupled logic-first planning? **Q3:** How does performance vary across dataset regimes, maze scales, and robot dynamics? **Q4:** Can the same semantic interface handle overlapping regions and joint object configurations?

Table 1: Aggregate results over 8 AntMaze and 6 HumanoidMaze environments. SR is finite-lasso execution success rate; NCC is normalized capped cost. Entries are means $\pm$ cross-setting standard deviations. Dark/light gray shading marks the best/second-best method within each domain–difficulty group; higher SR and lower NCC are better.

Domain	Method	Easy		Medium		Hard	
		SR (%)	NCC	SR (%)	NCC	SR (%)	NCC
AntMaze	SAGAS	75.4 ± 10.6	0.26 ± 0.11	75.1 ± 14.0	0.26 ± 0.13	65.0 ± 13.0	0.38 ± 0.13
	LF-GAS	73.5 ± 11.5	0.28 ± 0.12	71.5 ± 12.6	0.30 ± 0.13	55.9 ± 14.4	0.46 ± 0.14
	GA-GAS	67.1 ± 9.0	0.36 ± 0.09	66.1 ± 12.3	0.36 ± 0.13	53.6 ± 12.7	0.49 ± 0.13
	AP-HIQL	56.0 ± 24.8	0.45 ± 0.25	50.7 ± 24.8	0.50 ± 0.25	39.8 ± 20.4	0.63 ± 0.20
HumanoidMaze	SAGAS	64.8 ± 14.1	0.37 ± 0.11	57.8 ± 13.7	0.45 ± 0.14	49.5 ± 16.9	0.54 ± 0.16
	LF-GAS	63.0 ± 14.9	0.40 ± 0.14	56.7 ± 14.2	0.46 ± 0.13	43.2 ± 14.0	0.60 ± 0.13
	GA-GAS	59.5 ± 14.3	0.44 ± 0.13	50.2 ± 10.2	0.53 ± 0.09	41.2 ± 14.5	0.63 ± 0.12
	AP-HIQL	55.7 ± 27.1	0.49 ± 0.25	50.8 ± 27.7	0.54 ± 0.25	37.7 ± 21.7	0.67 ± 0.19

4.1 Experimental setup

Environments and datasets. We evaluate on OGBench [19] AntMaze, HumanoidMaze, and Scene using the official task-agnostic offline datasets, collected independently of any LTL formula. Navigation covers eight AntMaze settings across the `navigate`, `stitch`, and `explore` regimes and six HumanoidMaze settings across `navigate` and `stitch`, with multiple maze sizes in each domain. Scene contains a robot arm, a cube, drawer, window, and two buttons that toggle the drawer and window locks; its dataset consists of scripted interactions with these objects. Within each setting, all graphs, representations, and policies are frozen and reused across tasks, without task-specific data collection or retraining.

Compared methods. To compare high-level planning strategies, LF-GAS and GA-GAS share SAGAS's graph and frozen executor. LF-GAS first selects an automaton route and then connects its transitions through the graph; GA-GAS greedily advances automaton transitions through nearest semantic anchors without product search. We also compare with AP-HIQL, which adapts LTL-instructed goal reaching [10] using a separate HIQL policy [20].

Task suites. For navigation, we generate unseen $\mathrm{LTL}_{-\mathrm{X}}$ tasks by randomly instantiating and composing reachability, sequencing, safety, and recurrence templates. Easy tasks use one template, Medium tasks conjoin up to three, and Hard tasks up to four; Easy and Medium use at most five regions, while Hard allows eight. Regions are sampled to be disjoint to simplify feasibility control. All methods receive the same tasks and initial states: 100 tasks per setting and difficulty, totaling 4,200 cases. We also evaluate 320 overlapping-region tasks across AntMaze Giant-Stitch and Large-Explore, and 120 Scene tasks with joint object-state predicates. The Scene predicates describe joint object configurations in a 7D mixed discrete–continuous space.

Evaluation. We report finite-lasso execution success rate (SR), determined by independent post-rollout verification of the labels along each saved state trajectory. For recurrent tasks, success requires prefix completion followed by $M = 2$ suffix traversals in both navigation suites and $M = 1$ in Scene. Navigation efficiency is measured by normalized capped cost (NCC): $\min\{T_{\mathrm{pre}} + T_{\mathrm{suf}}, 2C_{\max}\}/(2C_{\max})$ when the prefix and first suffix traversal are completed, and 1 otherwise, where $C_{\max}$ is the configured cost budget. Lower NCC therefore reflects both shorter executions and fewer capped failures. Appendix E gives task designs, baselines, and suite-specific parameters.

4.2 Results and analysis

Table 1 summarizes the navigation comparison as means and cross-setting standard deviations; Appendix F reports the per-setting results.

Q1: LTL satisfaction. SAGAS achieves the highest aggregate SR in every domain–difficulty group, improving Hard-task SR over LF-GAS by 9.1 percentage points in AntMaze and 6.3 in HumanoidMaze. These gains support joint reasoning about logical progress and physical reachability when several events, avoidance constraints, and suffix cycles must be coordinated. Paired comparisons on all 4,200 cases confirm significant aggregate SR gains over all three baselines in both domains after Holm correction (largest corrected $p = 0.032$; Appendix F.2).

Q2: Execution cost. SAGAS also obtains the lowest aggregate NCC in every domain–difficulty group. Relative to LF-GAS, Hard-task NCC decreases from 0.46 to 0.38 in AntMaze and from 0.60 to 0.54 in HumanoidMaze. Because NCC also penalizes failures, we separately compare execution length on tasks solved by both methods. Across all difficulties, SAGAS uses on average 103.3 fewer environment steps in AntMaze and 210.0 fewer in HumanoidMaze for the prefix plus the first suffix traversal, supporting the use of joint graph–automaton costs to select shorter executable plans (Appendix F.3). Appendix I examines planning time and candidate-budget sensitivity; Appendix H.2 relates graph cost to execution length.

Q3: Dataset regime, maze size, and robot dynamics. In AntMaze, AP-HIQL is most competitive on the longer, coherent Navigate trajectories. SAGAS's SR advantage grows from 4.11 percentage points on Navigate to 22.44 on Stitch and 52.17 on Explore, consistent with the value of stitching behavior from fragmented or exploratory data (Appendix F.4). Within AntMaze Navigate, SAGAS achieves 63.7% SR on the Large maze and 55.3% on Giant. Across all settings, its aggregate SR is 71.8% in AntMaze and 57.4% in HumanoidMaze, where it leads in each difficulty group. Most SAGAS failures occur during execution. Appendices G and H examine these failures and the effects of soft-label screening and guard-aware steering.

Q4: Predicate generalization. SAGAS handles simultaneous proposition requirements in the overlap suite, finding 292/320 accepting lassos and achieving 81.3% SR (260/320). The same interface also supports joint object configurations in Scene: it finds 120/120 lassos and achieves 67.5% SR (81/120) while coordinating manipulation under the objects' lock dependencies. Table F.8 in the appendix summarizes the results; Appendices E.6 and E.7 give the protocols, and Appendix F.5 provides confidence intervals and per-family analyses.

4.3 Qualitative case study

We use a patrol task in `antmaze-giant-stitch` to illustrate how SAGAS plans and executes a long-horizon route from short offline trajectory fragments. The robot must repeatedly patrol three regions in order and visit three others in sequence at least once. The middle region of this second sequence is off limits until the robot visits a designated checkpoint. Writing e_i for the proposition that the robot is in region i, the task is

$$\underbrace{\mathrm{G}(\mathrm{F}(e_1 \wedge \mathrm{F}(e_2 \wedge \mathrm{F}e_3)))}_{\text{Infinite Patrol}} \wedge \underbrace{\mathrm{F}(e_4 \wedge \mathrm{F}(e_5 \wedge \mathrm{F}e_6))}_{\text{Sequential Visit}} \wedge \underbrace{(\neg e_5 \mathrm{U} e_7)}_{\text{Conditional Safety}} . \tag{6}$$

Figure 2 shows the resulting trajectory. Along the blue prefix, the robot reaches e_4, detours to the checkpoint e_7, and only then enters e_5 and continues to e_6. With the one-time sequence complete, it follows the red patrol loop through e_1, e_2, and e_3. No individual trajectory in the offline dataset demonstrates this complete temporal pattern; SAGAS constructs it by stitching fragments together. Additional qualitative case studies are provided in Appendix J.

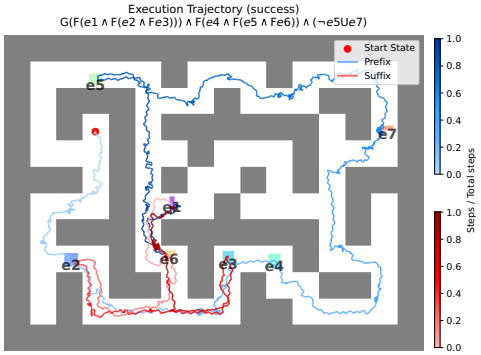


Figure 2: Execution trajectory for Eq. (6), with the prefix shown in blue and the recurrent suffix shown in red.

5 Conclusion

We presented **SAGAS**, an offline, model-free framework for LTL planning and execution from fixed, task-agnostic trajectory fragments. Task-time semantic graph augmentation grounds new predicates on preserved state support, while cost-aware Büchi product search synthesizes plans for the frozen executor. This separation enables zero-shot generalization across unseen formulas and data-supported predicates without task-specific data collection or policy retraining. Experiments in fixed-topology OGBench environments show higher finite-lasso execution success rates and lower execution costs than baselines in locomotion, with further evaluations demonstrating support for overlapping predicates and robotic manipulation. Future work will investigate whether this generalization can extend beyond tasks and predicates to unseen layouts and changing topology. Appendix D discusses the assumptions and limitations in detail.

Acknowledgments and Disclosure of Funding

This work was supported by the National Science and Technology Major Project (2025ZD1600700) and the National Natural Science Foundation of China (62573291, 62533017). The authors declare no competing interests.

References

[1] Marius Kloetzer and Calin Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transactions on Automatic Control*, 53(1):287–297, 2008.

[2] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[3] Guy Scher and Hadas Kress-Gazit. Warehouse automation in a day: from model to implementation with provable guarantees. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pages 280–287. IEEE, 2020.

[4] Georgios E Fainekos, Hadas Kress-Gazit, and George J Pappas. Temporal logic motion planning for mobile robots. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2020–2025. IEEE, 2005.

[5] Stephen L Smith, Jana Tŭmová, Calin Belta, and Daniela Rus. Optimal path planning for surveillance with temporal-logic constraints. *The International Journal of Robotics Research*, 30(14):1695–1708, 2011.

[6] Jiming Ren, Haris Miller, Karen M Feigh, Samuel Coogan, and Ye Zhao. LTL-D*: Incrementally optimal replanning for feasible and infeasible tasks in linear temporal logic specifications. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4495–4502. IEEE, 2024.

[7] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-constrained reinforcement learning. *arXiv preprint arXiv:1801.08099*, 2018.

[8] Cameron Voloshin, Abhinav Verma, and Yisong Yue. Eventual discounting temporal logic counterfactual experience replay. In *International Conference on Machine Learning*, pages 35137–35150. PMLR, 2023.

[9] Ameesh Shah, Cameron Voloshin, Chenxi Yang, Abhinav Verma, Swarat Chaudhuri, and Sanjit A Seshia. LTL-constrained policy optimization with cycle experience replay. *Transactions on Machine Learning Research*, 2025.

[10] Wenjie Qiu, Wensen Mao, and He Zhu. Instructing goal-conditioned reinforcement learning agents with temporal logic objectives. *Advances in Neural Information Processing Systems*, 36: 39147–39175, 2023.

[11] Mathias Jackermeier and Alessandro Abate. DeepLTL: Learning to efficiently satisfy complex LTL specifications for multi-task RL. In *The Thirteenth International Conference on Learning Representations*, 2025.

[12] Jason Xinyu Liu, Ankit Shah, Eric Rosen, Mingxi Jia, George Konidaris, and Stefanie Tellex. Skill transfer for temporal task specification. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2535–2541. IEEE, 2024.

[13] Zijian Guo, İlker Işık, H M Sabbir Ahmad, and Wenchao Li. One subgoal at a time: Zero-shot generalization to arbitrary linear temporal logic requirements in multi-task reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 38, pages 77500–77529, 2025.

[14] Seungho Baek, Taegeon Park, Jongchan Park, Seungjun Oh, and Yusung Kim. Graph-assisted stitching for offline hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 2391–2408. PMLR, 2025.

[15] Xusheng Luo, Yiannis Kantaros, and Michael M Zavlanos. An abstraction-free method for multirobot temporal logic optimal control synthesis. *IEEE Transactions on Robotics*, 37(5): 1487–1507, 2021.

[16] Cristian Ioan Vasile and Calin Belta. Sampling-based temporal logic path planning. In *2013 IEEE/RSJ international conference on intelligent robots and systems*, pages 4817–4822. IEEE, 2013.

[17] Yiannis Kantaros and Michael M Zavlanos. Sampling-based optimal control synthesis for multirobot systems under global temporal tasks. *IEEE Transactions on Automatic Control*, 64 (5):1916–1931, 2019. doi: 10.1109/TAC.2018.2853558.

[18] Yiannis Kantaros and Michael M Zavlanos. STyLuS*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems. *The International Journal of Robotics Research*, 39(7):812–836, 2020.

[19] Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. OGBench: Benchmarking offline goal-conditioned RL. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

[20] Seohong Park, Dibya Ghosh, Benjamin Eysenbach, and Sergey Levine. HIQL: Offline goal-conditioned RL with latent states as actions. *Advances in Neural Information Processing Systems*, 36:34866–34891, 2023.

[21] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning with deterministic μ-calculus specifications. In *2012 American Control Conference (ACC)*, pages 735–742. IEEE, 2012.

[22] Mihail Pivtoraiko, Ross A. Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009. doi: 10.1002/rob.20285.

[23] Wolfgang Hönig, Joaquim Ortiz-Haro, and Marc Toussaint. db-A*: Discontinuity-bounded search for kinodynamic mobile robot motion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 13540–13547, 2022. doi: 10.1109/IROS47612.2022.9981577.

[24] Rahul Shome and Lydia E. Kavraki. Asymptotically optimal kinodynamic planning using bundles of edges. In *IEEE International Conference on Robotics and Automation*, pages 9988–9994, 2021. doi: 10.1109/ICRA48506.2021.9560836.

[25] Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, pages 2107–2116. PMLR, 2018.

[26] Marco Bagatella, Andreas Krause, and Georg Martius. Directed exploration in reinforcement learning from linear temporal logic. *Transactions on Machine Learning Research*, 2025.

[27] Ruixuan Miao, Xu Lu, Cong Tian, Bin Yu, and Zhenhua Duan. T4NMTD: Transition-centric reinforcement learning for non-Markovian task decomposition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 24388–24395, 2026.

[28] Mathias Jackermeier, Mattia Giuri, Jacques Cloete, and Alessandro Abate. Zero-shot instruction following in RL via structured LTL representations. In *The Fortieth Annual Conference on Neural Information Processing Systems*, 2026. To appear.

[29] Alessandro Abate, Giuseppe De Giacomo, Mathias Jackermeier, Jan Křetínský, Maximilian Prokop, and Christoph Weinhuber. Semantically labelled automata for multi-task reinforcement learning with LTL instructions. In *Proceedings of the Thirty-Fifth International Joint Conference on Artificial Intelligence*, pages 3–11, 2026. doi: 10.24963/ijcai.2026/1.

[30] Jacques Cloete, Mathias Jackermeier, Ioannis Havoutis, and Alessandro Abate. PlatoLTL: Scaling LTL-guided multi-task RL. *arXiv preprint arXiv:2601.22891v3*, 2026.

[31] Taylor Bergeron, Zachary Serlin, and Kevin Leahy. Comp-LTL: Temporal logic planning via zero-shot policy composition. *arXiv preprint arXiv:2408.04215*, 2024.

[32] Matteo Pannacci, Andrea Fanti, Elena Umili, and Roberto Capobianco. Grounding LTL tasks in sub-symbolic RL environments for zero-shot generalization. *Reinforcement Learning Journal*, 7, 2026.

[33] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in Neural Information Processing Systems*, 30, 2017.

[34] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit Q-learning. In *The Tenth International Conference on Learning Representations*, 2022.

[35] Benjamin Eysenbach, Tianjun Zhang, Sergey Levine, and Ruslan Salakhutdinov. Contrastive learning as goal-conditioned reinforcement learning. *Advances in Neural Information Processing Systems*, 35:35603–35620, 2022.

[36] Seohong Park, Tobias Kreiman, and Sergey Levine. Foundation policies with Hilbert representations. In *International Conference on Machine Learning*, pages 39737–39761. PMLR, 2024.

[37] Wonchul Shin and Yusung Kim. Guide to control: Offline hierarchical reinforcement learning using subgoal generation for long-horizon and sparse-reward tasks. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 4217–4225. International Joint Conferences on Artificial Intelligence Organization, 2023.

[38] Taegeon Park, Seungho Baek, Jongchan Park, Seungjun Oh, and Yusung Kim. Temporal distance-aware subgoal generation for offline hierarchical reinforcement learning. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pages 2305–2314, 2025.

[39] Sungyoon Kim, Yunseon Choi, Daiki E Matsunaga, and Kee-Eung Kim. Stitching sub-trajectories with conditional diffusion model for goal-conditioned offline RL. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 13160–13167, 2024.

[40] Evgenii Opryshko, Junwei Quan, Claas A. Voelcker, Yilun Du, and Igor Gilitschenski. Test-time graph search for goal-conditioned reinforcement learning. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 94754–94774. PMLR, 2026.

[41] Zeyu Feng, Hao Luan, Kevin Yuchen Ma, and Harold Soh. Diffusion meets options: Hierarchical generative skill composition for temporally-extended tasks. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10854–10860. IEEE, 2025.

[42] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in Neural Information Processing Systems*, 34:20132–20145, 2021.

[43] Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In *International Conference on Computer Aided Verification*, pages 53–65. Springer, 2001.

Appendix

A Related work . . . 14
B Algorithmic details . . . 16
B.1 Offline graph construction and executor training . . . 16
B.2 Soft-label screening for negative literals . . . 17
B.3 Anchor witness construction . . . 17
B.4 Support-aware automaton preprocessing . . . 18
B.5 Semantic product transition rules . . . 18
B.6 Prefix–suffix search procedure . . . 20
B.7 Plan execution and runtime monitoring . . . 21
B.8 Complexity analysis . . . 23
C Soundness of synthesis and monitored execution . . . 24
D Scope and limitations . . . 25
E Experimental protocols and implementation details . . . 26
E.1 Navigation environments and datasets . . . 26
E.2 Navigation task generation . . . 27
E.3 Compared methods . . . 28
E.4 Evaluation protocol and metrics . . . 28
E.5 Parameters and computing resources . . . 28
E.6 Overlapping-region tasks and evaluation . . . 29
E.7 Scene manipulation tasks and evaluation . . . 29
F Detailed task results . . . 30
F.1 Navigation benchmark results . . . 30
F.2 Paired comparisons and confidence intervals . . . 32
F.3 Execution length on common-success tasks . . . 33
F.4 SAGAS and AP-HIQL across AntMaze datasets . . . 35
F.5 Overlapping-region and Scene manipulation results . . . 36
G Failure-mode analysis . . . 36
H Component and sensitivity analysis . . . 37
H.1 Soft-label screening and guard-aware steering . . . 37
H.2 Graph cost and realized execution length . . . 38
H.3 Graph resolution and data subsampling . . . 38
H.4 Predicate constraints and support subsampling . . . 39
I Planning time and search complexity . . . 40
I.1 Task-time planning . . . 40
I.2 Offline graph and automaton sizes . . . 41
I.3 Product-graph sizes and reachability . . . 42
I.4 Search strategies and candidate budget . . . 42
J Case studies . . . 43
J.1 Product-space planning versus decoupled planning . . . 43
J.2 Generalization across predicate geometries . . . 44

A Related work

Automata-based planning and formal synthesis. Classical temporal-logic planning constructs a finite transition system, forms its product with an automaton for the LTL specification, and searches for an accepting prefix–suffix run [1, 4, 5, 16]. Some planners sample continuous configurations rather than discretizing the workspace in advance [21, 15]; others sample states of a discrete product rather than constructing it in full [17, 18]. SAGAS uses the same product-search principle with a semantic graph built from fixed offline data.

Motion planning with reusable motions and controls. Motion planners can reuse feasible motions or controls to reduce the work required for each new query. State-lattice planners precompute motions between neighboring lattice states [22], while db-A* allows bounded gaps between successive motion primitives that are subsequently closed by trajectory optimization [23]. Bundles of Edges [24] reuses sampled controls rather than fixed connections: for each goal, it grows a new tree by forward propagation and provides an asymptotic optimality guarantee. SAGAS instead reuses a reachability graph learned from fixed trajectories to plan for new LTL specifications without query-time propagation.

LTL-guided reinforcement learning and reward machines. Learning-based approaches often encode logical progress through reward shaping, automaton states, or reward machines [7, 25, 26, 8, 9]. Transition-centric decomposition methods also use automaton transitions as reusable subtasks [27]. These approaches support exploration and policy learning for temporal objectives, typically through online interaction or training for a target specification or task distribution. SAGAS instead learns a reusable low-level executor and handles each new specification through graph search at test time.

Zero-shot LTL generalization and policy composition. Zero-shot LTL methods reuse learned behavior under new specifications. Goal-conditioned agents can follow LTL instructions without additional training over the LTL task space [10], while DeepLTL conditions Büchi-structure-aware policies on truth-assignment sequences to handle finite- and infinite-horizon formulas [11]. Related universal policies use Boolean-formula sequences or semantically labelled automata [28, 29]. PlatoLTL generalizes across predicate instances by encoding their task parameters in the goal embedding [30].

Other methods compose subgoal solvers or task primitives. GenZ-LTL decomposes unseen specifications into reach-avoid subgoals solved through safe RL [13]. Comp-LTL combines minimum-violation task primitives by constructing a pruned transition system and searching its product with the specification automaton [31]. Skill-transfer and modular policy methods likewise reuse learned primitives across temporal tasks [12].

SAGAS reuses a latent reachability graph and low-level executor learned from fixed task-agnostic data, handling each specification through task-time semantic graph augmentation and product search. This also separates predicate learning from planning: learned grounding methods map sub-symbolic observations to propositions [32], whereas SAGAS takes a supplied labeling function and evaluates it on preserved states to construct anchors and soft labels. New predicates can therefore be introduced at task time wherever the offline data provide sufficient support.

Offline goal-conditioned RL and graph-based stitching. Hindsight experience replay makes a trajectory useful for goals other than the one originally pursued by relabeling it with a goal the agent achieved [33]. Offline RL methods such as implicit Q-learning instead address how to learn a policy when the data are fixed [34]. For goal reaching, contrastive RL estimates goal-conditioned values from state–action and goal embeddings [35]; HIQL and foundation policies based on Hilbert representations learn goal-reaching behavior directly from offline trajectories [20, 36]. Fragmented demonstrations make long-horizon execution difficult, motivating hierarchical methods that generate subgoals or stitch short trajectories [37, 38, 39]. GAS builds a latent reachability graph from a temporal-distance representation and searches it for single-goal reaching [14]. TTGS likewise combines learned policies with test-time graph search to bridge long horizons [40]. SAGAS adds a support-preserving semantic interface to the GAS backbone: proposition-grounded anchors and soft labels enable cost-aware Büchi product synthesis with positive and negative literals and accepting suffix cycles.

Diffusion-based planning from offline data. For goal-reaching tasks, SSD [39] stitches together sub-trajectories generated by a diffusion model conditioned on the goal and a target value. DOPPLER [41] uses diffusion-generated trajectories as candidate options for LTL tasks. It learns to score each option given the current state and the part of the formula that remains to be satisfied. In SAGAS, the formula enters after training: the planner searches the product of the labeled reachability graph and the formula's Büchi automaton for an accepting prefix–suffix plan.

B Algorithmic details

B.1 Offline graph construction and executor training

The offline interface consists of a weighted reachability graph, preserved state support, and a local policy for tracking selected graph nodes. We instantiate the graph and policy with GAS [14], using a shared temporal-distance representation to define graph connections and policy subgoals. The preserved support links each node to observed physical states, allowing SAGAS to evaluate new predicates after the representation, graph, and policy have been frozen.

Temporal-distance representation. The temporal-distance representation measures proximity in terms of reachability: two states that are close in task space may still require many steps to connect. In the learned embedding $\psi : \mathcal{S} \to \mathcal{H}$, with $\mathcal{H} \subseteq \mathbb{R}^{d_h}$, the distance

$$d_\psi(s, g) = \|\psi(s) - \psi(g)\|_2 \tag{B.1}$$

estimates the minimum number of environment steps needed to reach g from s. We parameterize the goal-conditioned value as

$$V(s, g) = -d_\psi(s, g), \tag{B.2}$$

and fit ψ to offline transitions with an IQL-style expectile temporal-difference objective [34, 36].

TD-aware graph construction. To reduce the influence of detours in the recorded trajectories, graph construction first selects states whose observed motion is directionally consistent at the local scale H_{TD}. Let $\mathcal{F}(s_t, d)$ denote the first subsequent state on the same trajectory at latent distance at least d from s_t. For each s_t, we compare two latent displacements along that trajectory: one to $\mathcal{F}(s_t, H_{\mathrm{TD}})$ and the other to the state recorded H_{TD} steps later. Their cosine similarity defines the temporal-efficiency score

$$\theta_{\mathrm{TE}}(s_t) = \cos(\psi(\mathcal{F}(s_t, H_{\mathrm{TD}})) - \psi(s_t), \psi(s_{t+H_{\mathrm{TD}}}) - \psi(s_t)) \,. \tag{B.3}$$

We retain s_t for clustering when the two displacement vectors are closely aligned.

A retained state joins the nearest cluster if the latent distance to its representative is at most $H_{\mathrm{TD}}/2$; otherwise it starts a new cluster. Each cluster mean becomes a graph node. Clustering is performed jointly across trajectories, so a node can represent nearby latent states from different recorded fragments. We connect distinct nodes within latent distance H_{TD} of one another. If the graph is disconnected, we repeatedly add the shortest link from the growing main component to any remaining component.

Each edge $(u, v) \in \mathcal{E}$ receives weight $w(u, v) = \|u - v\|_2$, the distance between the two cluster means. This weight serves as a surrogate for the number of environment steps needed to move between the represented states. The product planner uses the same cost convention for local edges and links between components.

Support-preserving graph interface. A cluster representative cannot be directly evaluated by state-defined predicates. SAGAS preserves raw states from the cluster to supply the corresponding semantic evidence:

$$\mathcal{D}_v \subseteq \{\, s \in \mathcal{D}_S \mid \psi(s) \in \mathcal{C}_v \,\}. \tag{B.4}$$

Here $\mathcal{D}_S$ is the set of states in the offline dataset, and $\mathcal{D}_v$ may contain either all states assigned to the cluster or a bounded subset.

Low-level executor training and deployment. The low-level policy learns to move toward subgoals specified by their latent direction. For each training state s_t, we select $s_{\mathrm{sub}} = \mathcal{F}(s_t, H_{\mathrm{TD}})$ from the same recorded trajectory, using the graph's temporal-distance scale rather than a fixed number of steps. The policy receives the normalized direction toward this subgoal:

$$\vec{h}_{\mathrm{dir}} = \frac{\psi(s_{\mathrm{sub}}) - \psi(s_t)}{\|\psi(s_{\mathrm{sub}}) - \psi(s_t)\|}. \tag{B.5}$$

We train the direction-conditioned critic with the intrinsic reward

$$r^{\mathrm{dir}}(s_t, s_{t+1}, \vec{h}_{\mathrm{dir}}) = \left\langle \psi(s_{t+1}) - \psi(s_t), \vec{h}_{\mathrm{dir}} \right\rangle, \tag{B.6}$$

which measures progress along the requested direction in the embedding. The DDPG+BC actor objective combines high estimated value with a behavior-cloning term that keeps actions close to those in the dataset, following the same regularization principle as TD3+BC [42]. Unless noted otherwise, we use the training settings of GAS [14].

At execution time, the selected graph waypoint v replaces the training subgoal. The frozen policy receives the same form of directional input, recomputed from the current state at each step:

$$\vec{h}_{\mathrm{dir}}(s_t, v) = \frac{v - \psi(s_t)}{\|v - \psi(s_t)\|_2}. \tag{B.7}$$

B.2 Soft-label screening for negative literals

For an original graph node v, the soft label measures the weighted fraction of its preserved states that satisfy proposition ℓ:

$$\widehat{P}(\ell \mid v) = \frac{\sum_{s \in \mathcal{D}_v} K_v(s)\, \mathbf{1}\{\ell \in L_\phi(s)\}}{\sum_{s \in \mathcal{D}_v} K_v(s)}, \tag{B.8}$$

where $K_v(s) \geq 0$ is the weight of support state s. For region predicates, the indicator tests whether $\Pi(s) \in \mathcal{R}_\ell$. When a guard term contains $\neg\ell$, the planner admits v only if the estimate has a positive denominator and does not exceed τ_{soft}. At anchors, the planner instead checks the complete witness label set and rejects any anchor whose witness satisfies a forbidden proposition. Appendix B.5 combines these rules in the product successor relation.

By default, we use $K_v(s) \equiv 1$, giving each preserved state equal weight within the cluster. The weighted definition also allows greater emphasis on states near the representative, for example through $K_v(s) = K_\sigma(\|\psi(s) - v\|_2)$ with $K_\sigma(r) = \exp(-r^2/(2\sigma^2))$ and $\sigma > 0$.

B.3 Anchor witness construction

Anchors provide concrete witnesses for positive literals. A cluster may contain satisfying states even when tracking its representative reaches a state where the proposition is false. An anchor instead targets the embedding of a specific observed state whose labels satisfy the requirement.

For each proposition ℓ, SAGAS scans the offline dataset and collects candidate states

$$\mathcal{S}_\ell^{\mathrm{cand}} = \{\, s \in \mathcal{D}_S \mid \ell \in L_\phi(s) \,\}.$$

If the pool is empty, the planner marks ℓ unavailable as a positive requirement under the current dataset.

For a nonempty pool, retrieval continues until N_s connected anchors have been accepted, the per-proposition candidate-attempt limit is reached, or the pool is exhausted. Each candidate is embedded as

$$a_{s_j^{(\ell)}} = \psi(s_j^{(\ell)}), \qquad s_j^{(\ell)} \in \mathcal{S}_\ell^{\mathrm{cand}},$$

and tested for connections to the offline graph at the same temporal-distance scale:

$$\mathcal{N}_{\mathrm{TD}}(a_{s_j^{(\ell)}}) = \{\, v \in \mathcal{V} \mid \|a_{s_j^{(\ell)}} - v\|_2 \leq H_{\mathrm{TD}} \,\}.$$

A candidate with a nonempty neighbor set is inserted as an anchor and connected to these neighbors, with edge weights given by their latent distances. Its support contains the witness state alone, and its label set is that state's complete assignment:

$$a_{s_j^{(\ell)}} \in \mathcal{A}_\ell, \qquad \mathcal{D}_{a_{s_j^{(\ell)}}} = \{s_j^{(\ell)}\}, \qquad \lambda_{\mathcal{A}}(a_{s_j^{(\ell)}}) = L_\phi(s_j^{(\ell)}).$$

The full assignment allows an anchor to belong to several per-proposition sets, supporting overlapping predicates. Retrieval also includes propositions that occur only negatively, whose anchors can guide avoidance during execution. Candidates without temporal-distance neighbors are discarded; if none connects, the positive requirement remains unavailable.

Joint-guard retrieval. When a guard term requires several propositions simultaneously, separate witnesses for each proposition are insufficient. For a normalized guard term η, let P_η and N_η denote its sets of required and forbidden propositions. The joint candidate pool is

$$\mathcal{S}_\eta^{\mathrm{cand}} = \{s \in \mathcal{D}_S : P_\eta \subseteq L_\phi(s),\ N_\eta \cap L_\phi(s) = \varnothing\}.$$

For terms containing positive literals, SAGAS retrieves from this pool using the connection rule above. Accepted anchors can be reused by compatible guard terms and included in each applicable per-proposition set, so the final $\mathcal{A}_\ell$ may exceed the initial budget N_s. If no connected witness is found, the planner disables that term while retaining any supported alternatives. Terms without positive literals require no positive anchor.

B.4 Support-aware automaton preprocessing

SAGAS translates ϕ with `ltl2ba` [43] and then restricts future transitions to those supported by the semantic graph. Before this restriction, the original nondeterministic Büchi automaton is

$$\mathcal{B}_\phi = (Q, Q_0, \Sigma, \delta, F),$$

where Q is the finite state set, $Q_0 \subseteq Q$ the initial-state set, $\Sigma = 2^{\mathcal{AP}_\phi}$ the alphabet of proposition assignments, $\delta : Q \times \Sigma \to 2^Q$ the transition function, and $F \subseteq Q$ the accepting set. Under standard Büchi semantics [2], a run $q_0 q_1 \ldots$ over an infinite word $\sigma_0 \sigma_1 \ldots \in \Sigma^\omega$ satisfies $q_0 \in Q_0$ and $q_{t+1} \in \delta(q_t, \sigma_t)$ for all $t \geq 0$. It is accepting when it visits F infinitely often, i.e., $\mathrm{Inf}(q_0 q_1 \ldots) \cap F \neq \emptyset$.

Boolean guards represent the transitions without enumerating every proposition assignment. For each state pair (q, q'), the guard $g_{q,q'}$ is satisfied by exactly the assignments that enable the transition:

$$\mathsf{Sat}(g_{q,q'}) = \{\sigma \in \Sigma \mid q' \in \delta(q, \sigma)\}.$$

Thus, $q' \in \delta(q, \sigma)$ iff $\sigma \models g_{q,q'}$. To check alternative ways of satisfying a guard independently, SAGAS converts it into disjunctive normal form:

$$g_{q,q'} \equiv \bigvee_{\eta \in \mathrm{DNF}(g_{q,q'})} \eta, \qquad \eta = \bigwedge_{\ell \in P_\eta} \ell \wedge \bigwedge_{\ell \in N_\eta} \neg \ell,$$

where P_η and N_η contain the propositions required to be true and false, respectively. The successor oracle and runtime monitor use these same sets to check the selected term.

Preprocessing treats each satisfiable DNF term as a separate transition and discards contradictory terms. A term containing positive literals is retained only when a connected anchor witnesses the complete term; any term removed for lack of support is reported to the user. For each state pair, the retained guard $\widetilde{g}_{q,q'}$ is the disjunction of the surviving terms, giving $\widetilde{g}_{q,q'} \Rightarrow g_{q,q'}$. This restriction applies to movement and dwell guards after the initial observation.

The initial assignment is already known from s_0, so the original NBA processes it using Q_0 and δ without requiring an offline anchor. State pruning then considers reachability from the resulting automaton states and the existence of return cycles through accepting states. The resulting support-restricted automaton $\widetilde{\mathcal{B}}_\phi$ has state set $\widetilde{Q} \subseteq Q$ and accepting set $\widetilde{F} = F \cap \widetilde{Q}$. Appendices I.2 and I.3 report the automaton sizes before preprocessing and during product search.

Algorithm B.1 summarizes semantic graph augmentation using the normalized guard terms.

B.5 Semantic product transition rules

The successor oracle combines graph connectivity with the semantic checks that enable automaton transitions. It generates movement successors on demand during search and also checks whether an accepting endpoint admits a dwell suffix at the same waypoint, as described in Section 3.4.

Let $\mathcal{H}_\phi^{\mathrm{sem}} = (\mathcal{V}_\phi, \mathcal{E}_\phi, w_\phi)$ be the semantic graph and $\widetilde{\mathcal{B}}_\phi$ the support-restricted automaton from Appendix B.4, with state set $\widetilde{Q}$, accepting set $\widetilde{F}$, and retained guards $\widetilde{g}_{q,q'}$. Semantic evidence depends on the node type: original nodes $\mathcal{V}$ use soft labels $\widehat{P}$, inserted anchors $\mathcal{A} = \bigcup_{\ell \in \mathcal{AP}_\phi} \mathcal{A}_\ell$ use witness assignments $\lambda_{\mathcal{A}}$, and the temporary start node v_0 uses $L_\phi(s_0)$. The implicit product system is

$$\mathcal{P}_\phi = (\mathcal{Z}_\phi, \mathcal{E}_\mathcal{P}, c_\mathcal{P}), \qquad \mathcal{Z}_\phi = \mathcal{V}_\phi \times \widetilde{Q}.$$

Algorithm B.1 Task-time semantic graph augmentation

Input: Latent graph $\mathcal{H}_{\text{graph}} = (\mathcal{V}, \mathcal{E}, w)$, preserved supports $\{\mathcal{D}_v\}$, offline states $\mathcal{D}_S$, embedding ψ, initial state s_0, labeling function L_ϕ, normalized guard terms $\mathcal{G}_\phi$, temporal-distance scale H_{TD}, initial per-proposition anchor budget N_s and candidate-attempt limit

Output: Semantic graph $\mathcal{H}_\phi^{\text{sem}}$, start node v_0, anchor sets and witness label sets, soft-label estimates, unavailable propositions $\mathcal{U}_{\text{unav}}$

Initialize anchor sets $\mathcal{A}_\ell \leftarrow \emptyset$ for all $\ell \in \mathcal{AP}_\phi$
Estimate soft labels $\widehat{P}(\ell \mid v)$ for all original graph nodes v and propositions ℓ using their raw supports and L_ϕ
for all $\ell \in \mathcal{AP}_\phi$ **do**
  Retrieve candidate states from $\mathcal{D}_S$ satisfying ℓ under L_ϕ
  while $|\mathcal{A}_\ell| < N_s$, untried candidates remain, and the candidate-attempt limit is not reached **do**
    Embed the next candidate as $a_{s_j^{(\ell)}} = \psi(s_j^{(\ell)})$
    if $a_{s_j^{(\ell)}}$ has temporal-distance neighbors within H_{TD} **then**
      Insert it as an anchor with singleton support, witness label set $L_\phi(s_j^{(\ell)})$, and local graph edges within H_{TD}
      $\mathcal{A}_\ell \leftarrow \mathcal{A}_\ell \cup \{a_{s_j^{(\ell)}}\}$
    end if
  end while
end for
for all DNF terms η containing positive literals **do**
  Retrieve connected witnesses satisfying all positive and no negative literals
  Insert reusable joint anchors and update the applicable anchor sets $\mathcal{A}_\ell$; record η unsupported if no witness connects
end for
Set $\mathcal{U}_{\text{unav}} \leftarrow \{\ell \in \mathcal{AP}_\phi \mid \mathcal{A}_\ell = \emptyset\}$ from the final anchor sets
Embed s_0 as $v_0 = \psi(s_0)$ and connect it using the same temporal-distance scale
if v_0 has no incident graph edge **then**
  Connect v_0 to the nearest existing graph node as an initialization heuristic
end if
Form $\mathcal{H}_\phi^{\text{sem}}$ from the augmented graph, anchors, soft labels, and v_0
return $\mathcal{H}_\phi^{\text{sem}}$ and its semantic auxiliary data

Its initial states pair v_0 with the automaton states reached after processing $L_\phi(s_0)$:

$$Q_{\text{init}} = \{q' \in \widetilde{Q} \mid \exists q_0 \in Q_0,\ q' \in \delta(q_0, L_\phi(s_0))\}, \qquad \mathcal{Z}_0 = \{(v_0, q) : q \in Q_{\text{init}}\}.$$

A movement from $z = (v, q)$ to $z' = (v', q')$ requires a semantic-graph edge and at least one retained guard term enabled at the destination:

$$(z, z') \in \mathcal{E}_{\mathcal{P}}^{\text{mov}} \iff (v, v') \in \mathcal{E}_\phi \ \wedge\ \exists \eta \in \text{DNF}(\widetilde{g}_{q,q'}) \text{ such that } \text{Enable}_\phi(v, v', \eta) = \text{true}. \quad \text{(B.9)}$$

The cost of an enabled movement transition is inherited from the semantic graph:

$$c_{\mathcal{P}}(z, z') = w_\phi(v, v'). \quad \text{(B.10)}$$

For a conjunctive term η with required and forbidden proposition sets P_η and N_η, the enablement check separates the positive and negative requirements:

$$\text{Enable}_\phi(v, v', \eta) = \text{PosOK}(v', \eta) \wedge \text{NegOK}(v', \eta). \quad \text{(B.11)}$$

The negative-literal check depends on the type of successor node:

$$\text{NegOK}(v', \eta) = \begin{cases} \text{true}, & N_\eta = \emptyset, \\ \bigwedge_{\ell \in N_\eta} \left(\widehat{P}(\ell \mid v') \le \tau_{\text{soft}}\right), & v' \in \mathcal{V}^+, \\ \lambda_{\mathcal{A}}(v') \cap N_\eta = \emptyset, & v' \in \mathcal{A}, \\ L_\phi(s_0) \cap N_\eta = \emptyset, & v' = v_0, \\ \text{false}, & \text{otherwise}. \end{cases} \quad \text{(B.12)}$$

Here $\mathcal{V}^+$ contains original graph nodes with $\sum_{s \in \mathcal{D}_v} K_v(s) > 0$, for which the soft-label estimate is defined. A guard term without negative literals passes this check automatically. Positive literals, in

contrast, require a connected anchor whose witness satisfies them all:

$$\mathrm{PosOK}(v', \eta) = [P_\eta = \varnothing] \ \vee \ [v' \in \mathcal{A} \ \wedge \ P_\eta \subseteq \lambda_{\mathcal{A}}(v')].$$

Together, the two checks evaluate the complete term at an anchor and apply the soft-label screen at an original node.

At an accepting endpoint, the additional suffix-only dwell relation is

$$\mathcal{E}_{\mathcal{P}}^{\mathrm{dwell}} = \left\{ (z, z) : \begin{array}{l} z = (u, q),\ q \in \widetilde{F},\ \exists \eta \in \mathrm{DNF}(\widetilde{g}_{q,q}) : \\ \mathrm{PosOK}(u, \eta) \wedge \mathrm{NegOK}(u, \eta) \end{array} \right\}.$$

A dwell keeps the same waypoint and has zero graph cost, without requiring a self-edge in the semantic graph. The complete relation is $\mathcal{E}_{\mathcal{P}} = \mathcal{E}_{\mathcal{P}}^{\mathrm{mov}} \cup \mathcal{E}_{\mathcal{P}}^{\mathrm{dwell}}$: prefix and nonempty return searches expand movement edges, while dwell is tested separately at accepting endpoints.

During expansion of (v, q), the oracle examines its semantic-graph neighbors and outgoing retained guards, returning the successors that satisfy Eq. (B.9). Prefix and return searches share this procedure, and the dwell test reuses its semantic checks at the current waypoint.

B.6 Prefix–suffix search procedure

SAGAS selects a complete lasso by comparing prefixes together with their return suffixes. It first collects accepting prefixes, then searches for a suffix at each retained endpoint, using the same successor oracle for both stages.

Prefix candidate collection. The prefix search runs A* from $\mathcal{Z}_0$ toward the accepting set $\mathcal{Z}_F = \mathcal{V}_\phi \times \widetilde{F}$, using product-edge costs $c_{\mathcal{P}}$. When an accepting state $z_f = (u, q_f)$ is removed from the priority queue for expansion, its predecessor chain yields a candidate prefix. Retaining up to K distinct accepting endpoints allows the final choice to account for differences in suffix cost. Since the search stores one predecessor chain per product state, each endpoint contributes at most one prefix.

Suffix cycle construction. At each retained endpoint $z_f = (u, q_f)$, an enabled dwell provides a zero-cost graph suffix. If no dwell is available, Dijkstra search starts from the enabled movement successors of z_f and seeks the lowest-cost nonempty path back to z_f. A prefix becomes a complete candidate only when one of these suffixes is found.

Candidate selection. Each complete candidate $\rho = (\rho_{\mathrm{pre}}, \rho_{\mathrm{suf}})$ retains its selected guard terms and has waypoint projection $\Gamma = \mathrm{proj}_{\mathcal{V}_\phi}(\rho)$. Following the prefix–suffix objective [15], SAGAS compares candidates by

$$J_{\mathrm{graph}}(\Gamma) = \lambda\, \mathrm{Cost}(\Gamma_{\mathrm{pre}}) + (1 - \lambda)\, \mathrm{Cost}(\Gamma_{\mathrm{suf}}),$$

where Cost sums semantic-graph edge weights for movements and assigns zero to a dwell. The planner returns the lowest-cost candidate under this objective.

The budget K balances search effort against exploring additional accepting endpoints that may yield a lower-cost complete lasso. Appendix I.4 evaluates this trade-off by comparing feasibility, lasso cost, and runtime across candidate budgets and against exhaustive finite-product search.

Prefix-search heuristic. At a product state (v, q), the A* heuristic estimates the remaining cost of reaching an accepting product state by considering the first positive requirement along a possible path. To identify these requirements, let $\mathcal{C}_0(q)$ contain q and the states reachable from it in the preprocessed automaton using only guard terms with no positive literals. From these states, we collect into $\mathcal{B}_+(q)$ the propositions appearing positively in outgoing guard terms whose destination can still reach an accepting state. These propositions are candidates for the first positive requirement on a path to acceptance.

If $q \in \widetilde{F}$ or $\mathcal{C}_0(q) \cap \widetilde{F} \neq \emptyset$, acceptance is possible in the automaton without another positive requirement, and we use $h(v, q) = 0$. We also use zero when $\mathcal{B}_+(q) = \emptyset$. Otherwise, the heuristic is the distance to the nearest anchor for any of the candidate propositions:

$$h(v, q) = \min_{\ell \in \mathcal{B}_+(q)} \mathrm{dist}_\phi(v, \mathcal{A}_\ell), \tag{B.13}$$

where $\mathrm{dist}_\phi(v, \mathcal{A}_\ell)$ is the shortest-path distance from v to any anchor in $\mathcal{A}_\ell$, computed on the semantic graph without automaton constraints. These distances are precomputed for each relevant anchor set and reused during search.

When the heuristic is nonzero, every accepting product path must eventually take a first transition requiring a positive proposition $\ell \in \mathcal{B}_+(q)$. That transition ends at an anchor in $\mathcal{A}_\ell$, so the graph cost accumulated by then is at least $\mathrm{dist}_\phi(v, \mathcal{A}_\ell)$ and hence at least $h(v, q)$. The remaining edges have nonnegative costs, making $h(v, q)$ a lower bound on the total cost of reaching acceptance. The zero cases satisfy this bound directly, so the heuristic is admissible.

Algorithm B.2 gives the complete prefix–suffix search procedure.

Algorithm B.2 Product-space prefix–suffix search

Input: Semantic graph $\mathcal{H}_\phi^{\mathrm{sem}} = (\mathcal{V}_\phi, \mathcal{E}_\phi, w_\phi)$ with $\{\mathcal{A}_\ell\}$, $\lambda_{\mathcal{A}}$, $\widehat{P}$, and start node v_0; original NBA $\mathcal{B}_\phi$, initial label $L_\phi(s_0)$, unavailable requirements, soft-label threshold τ_{soft}, prefix budget K, trade-off λ

Output: Product plan $\rho^\star$, including product states and guard terms, with waypoint projection $\mathrm{proj}_{\mathcal{V}_\phi}(\rho^\star)$, or failure

$Q_{\mathrm{aligned}} \leftarrow \bigcup_{q_0 \in Q_0} \delta(q_0, L_\phi(s_0))$
Restrict future guards by support, retain aligned initial states, and precompute $h(v, q)$
Set $Q_{\mathrm{init}} = Q_{\mathrm{aligned}} \cap \widetilde{Q}$ and $\mathcal{Z}_0 = \{(v_0, q) : q \in Q_{\mathrm{init}}\}$
Run A* on the implicit semantic product graph from $\{(v_0, q) : q \in Q_{\mathrm{init}}\}$, retaining up to K accepting prefix candidates
Initialize candidate plan set $\mathcal{C} \leftarrow \emptyset$
for all product prefix ρ_{pre} ending at $z_f = (u, q_f)$ **do**
    Initialize $\rho_{\mathrm{suf}} \leftarrow$ none
    if a feasible suffix-only symbolic dwell option exists at z_f **then**
        Set ρ_{suf} to the dwell at z_f, retaining the enabled self-loop guard
    else
        Search the same implicit product graph for a nonempty return cycle through z_f and set ρ_{suf} with its guard trace if one is found
    end if
    if ρ_{suf} is found **then**
        Add $(\rho_{\mathrm{pre}}, \rho_{\mathrm{suf}})$ to $\mathcal{C}$
    end if
end for
if $\mathcal{C} = \emptyset$ **then**
    return no accepting lasso found within the candidate budget
end if
Select $\rho^\star \in \arg\min_{\rho \in \mathcal{C}} J_{\mathrm{graph}}(\mathrm{proj}_{\mathcal{V}_\phi}(\rho))$
return $\rho^\star$ with its waypoint projection $\mathrm{proj}_{\mathcal{V}_\phi}(\rho^\star)$ and selected guard terms

B.7 Plan execution and runtime monitoring

The frozen low-level policy follows the waypoints of $\rho^\star$, while the monitor uses the observed trajectory to decide when to advance along the product plan. The plan retains its automaton states and selected guard terms alongside the waypoint projection $\Gamma = \mathrm{proj}_{\mathcal{V}_\phi}(\rho^\star)$. Repeating the suffix unfolds this projection into

$$\Gamma_{\mathrm{pre}} \oplus \Gamma_{\mathrm{suf}}^{\omega} = (v_0, v_1, \ldots),$$

where v_0 is the start.

Waypoint selection and tracking. Let $k \geq 1$ index the next unconfirmed waypoint and $k' \geq k$ the selected tracking target. Waypoints inherited from the offline graph guide intermediate motion and may be skipped when a later target lies within H_{TD}. Selected anchors and accepting endpoints are mandatory semantic checkpoints that the tracker cannot bypass. For the current state s_t and target $v_{k'}$, the nominal tracking displacement is

$$\Delta_t(s_t, v_{k'}) = v_{k'} - h_t, \tag{B.14}$$

where $h_t = \psi(s_t)$ denotes the current latent state. This displacement supplies the nominal direction to the frozen low-level policy. The monitor confirms the target using the checks below before advancing k to $k' + 1$; the indices continue across repeated suffix cycles.

Monitoring and progress confirmation. While the policy tracks a selected waypoint, the monitor follows the automaton progress induced by the observed state labels. Consider a segment from the last confirmed product state (v_i, q_i) to the selected target (v_j, q_j), where j may exceed $i+1$ if intermediate waypoints are skipped. Starting from $S_i(t_i) = \{q_i\}$ after processing the source label, the monitor processes each subsequent observation using the original NBA:

$$S_i(t+1) = \bigcup_{q \in S_i(t)} \delta(q, L_\phi(s_{t+1})).$$

Here, t_i is the source confirmation time, and $S_i(t)$ contains the automaton states reachable from q_i under the labels observed since that confirmation.

During tracking, let $\mathcal{N}_{\mathrm{act}}(t)$ contain the propositions required to be false by the currently active guards of the plan. The monitor checks

$$L_\phi(s_t) \cap \mathcal{N}_{\mathrm{act}}(t) = \emptyset. \tag{B.15}$$

This set is updated as the automaton progresses and also supplies the forbidden labels for guard-aware steering. A violation of these active negative requirements terminates execution.

Target confirmation combines physical arrival with the automaton progress recorded in S_i. For an anchor target a_s with selected incoming guard η, the reach and positive-label checks are

$$\|\psi(s_t) - a_s\|_2 < \varepsilon_{\mathrm{anc}}, \qquad P_\eta \subseteq L_\phi(s_t),$$

where P_η contains the positive literals of η. A non-anchor target uses the waypoint-tracking reach condition. In either case, confirmation at time t_j requires the reach and guard checks to pass and $q_j \in S_i(t_j)$, so the observed segment connects the planned automaton endpoints. The next segment starts from $\{q_j\}$ and processes only subsequent observations, without consuming the endpoint label again. If the target remains unconfirmed when the local tracking budget expires, execution terminates. Lemma 2 establishes that successive confirmations preserve a valid run of the original NBA over the observed labels.

After completing the prefix, the tracker repeats the suffix under the same monitoring rules. Each cycle is confirmed only when the suffix endpoint is reached and the labels observed during that cycle admit a return to the planned accepting automaton state. A dwell cycle is likewise confirmed using a new physical observation. Appendix C formalizes the resulting execution guarantee.

Guard-aware steering. To help the executor avoid forbidden regions between waypoints in navigation, guard-aware steering adds bounded repulsion to the nominal displacement $\Delta_t(s_t, v_{k'})$. Anchors of active forbidden propositions serve as avoidance landmarks. For each $\ell \in \mathcal{N}_{\mathrm{act}}(t)$ with $\mathcal{A}_\ell \neq \emptyset$, the nearest such anchor is

$$v_\ell^{\mathrm{near}}(h_t) = \arg\min_{v \in \mathcal{A}_\ell} \|h_t - v\|_2. \tag{B.16}$$

The contributions from these anchors form the repulsion vector

$$r_t = \sum_{\substack{\ell \in \mathcal{N}_{\mathrm{act}}(t) \\ \mathcal{A}_\ell \neq \emptyset}} \beta \left(1 - \frac{\|h_t - v_\ell^{\mathrm{near}}(h_t)\|_2}{\rho_{\mathrm{rep}}}\right)_+^p \frac{h_t - v_\ell^{\mathrm{near}}(h_t)}{\|h_t - v_\ell^{\mathrm{near}}(h_t)\|_2 + \epsilon}, \tag{B.17}$$

where ρ_{rep} is the repulsion radius, β sets its strength, p controls how sharply it increases near an anchor, and $(x)_+ = \max(x, 0)$. Clipping this vector to a maximum norm $r_{\max}$ gives the policy direction

$$\widetilde{d}_t = \mathrm{normalize}\big(\Delta_t(s_t, v_{k'}) + \mathrm{clip}_{r_{\max}}(r_t)\big), \tag{B.18}$$

where clipping rescales r_t only when its norm exceeds $r_{\max}$, and normalization uses a small numerical tolerance. The resulting direction combines motion toward the target with bounded deflection away from forbidden anchors. When steering is disabled or no active forbidden proposition has anchors, we set $r_t = 0$ and the policy receives the normalized waypoint displacement.

Algorithm B.3 gives the full execution procedure.

Algorithm B.3 Plan execution and runtime monitoring

Input: Product plan $\rho^\star$ with waypoint projection $\Gamma = \mathrm{proj}_{\mathcal{V}_\phi}(\rho^\star)$, planned automaton states $\{q_i\}$ and guard terms $\{\eta_i\}$, original NBA $\mathcal{B}_\phi$, anchors $\mathcal{A}$, embedding ψ, frozen low-level policy π_{low}, thresholds $H_{\mathrm{TD}}, \varepsilon_{\mathrm{anc}}$, labeling function L_ϕ, local budget H_{low}, steering setting

Output: Executed trajectory τ and execution outcome, including completion and violation statistics

Unfold the prefix followed by repeated suffix cycles into waypoint and guard streams, starting at (v_0, q_0)
For a dwell suffix, repeat the endpoint and its guard at successive waypoint indices
Initialize next unconfirmed index $k \leftarrow 1$, selected target index $k' \leftarrow 1$, and local tracking timer $c \leftarrow 0$
Initialize $S \leftarrow \{q_0\}$, where (v_0, q_0) starts $\rho^\star$ after processing $L_\phi(s_0)$
for $t = 0, 1, \ldots$ until timeout, violation, or the prescribed finite suffix-evaluation criterion is met **do**
    Embed current state $h_t \leftarrow \psi(s_t)$
    On each new observation after the last confirmation, update $S \leftarrow \bigcup_{q \in S} \delta(q, L_\phi(s_t))$
    Determine active obligations from automaton progress; terminate if no feasible branch remains
    Let $\mathcal{N}_{\mathrm{act}}(t)$ be the union of forbidden labels in the active guard obligations
    if $L_\phi(s_t) \cap \mathcal{N}_{\mathrm{act}}(t) \neq \emptyset$ **then**
        Record a semantic violation and terminate the rollout as failed
    end if
    If the target's reach and guard checks pass but $q_{k'} \notin S$, report an unrealized transition and terminate
    On a fresh observation, count a suffix cycle when its accepting endpoint passes the reach, guard, and automaton checks
    if target anchor $v_{k'}$ is close and all its required positive labels are verified under $L_\phi(s_t)$ **then**
        Mark the checkpoint verified; set $k \leftarrow k' + 1$ and reset $S \leftarrow \{q_{k'}\}$, $c \leftarrow 0$
    else if non-anchor target $v_{k'}$ is reached and all preceding semantic checkpoints are confirmed **then**
        Mark waypoint progress verified; set $k \leftarrow k' + 1$ and reset $S \leftarrow \{q_{k'}\}$, $c \leftarrow 0$
    end if
    Select the furthest waypoint $k' \geq k$ within H_{TD}, up to and including the next pending semantic checkpoint; default to $k' = k$
    Compute $\widetilde{d}_t$ by normalizing waypoint displacement plus bounded repulsion when steering is enabled; otherwise normalize waypoint displacement alone
    Apply action $a_t \sim \pi_{\mathrm{low}}(\cdot \mid s_t, \widetilde{d}_t)$ and observe s_{t+1}
    Increment c; if $c > H_{\mathrm{low}}$ before target confirmation, record the current product plan as unrealized and terminate as failed
    Record task-space labels, anchor completions, suffix-cycle completions, and any violation
end for
return trajectory τ and execution outcome

B.8 Complexity analysis

Task-time computation has two main components: grounding predicates on offline states and searching the semantic product graph.

Semantic preprocessing evaluates predicates on the preserved node supports and the full anchor-retrieval pool. Let $N_{\mathrm{sup}} = |\bigcup_{v \in \mathcal{V}} \mathcal{D}_v|$ and $N_{\mathrm{data}} = |\mathcal{D}_S|$ denote their respective sizes. Soft-label computation requires $O(|\mathcal{AP}_\phi| N_{\mathrm{sup}})$ predicate calls, while labeling the full retrieval pool requires $O(|\mathcal{AP}_\phi| N_{\mathrm{data}})$ calls, reusing support assignments where available.

Let G count the normalized guard terms examined during joint-witness retrieval, with at most b_η literals per term. Given the state assignments, a direct scan requires at most $O(G b_\eta N_{\mathrm{data}})$ label checks. Checking each candidate's connectivity adds a range search and processing of its local neighbors. The anchor budget, candidate-attempt limit, and pool size bound the number of candidates examined. After DNF conversion, guard preprocessing is linear in the normalized representation.

For product search, let $n = |\mathcal{V}_\phi|$ and $m = |\mathcal{E}_\phi|$ denote the semantic-graph size, $r = |\widetilde{Q}|$ the number of preprocessed automaton states, and $d_B = \max_{q \in \widetilde{Q}} |\mathrm{Out}(q)|$ the maximum number of outgoing guard terms from an automaton state. The product contains at most

$$N_{\mathcal{P}} \leq nr \tag{B.19}$$

states. Each movement successor combines a graph edge with an outgoing guard term, so the number $M_{\mathcal{P}}$ of generated movement edges is bounded by

$$M_{\mathcal{P}} \leq mrd_B. \tag{B.20}$$

With a binary heap, one A* or Dijkstra search has worst-case time $O(M_{\mathcal{P}} \log N_{\mathcal{P}})$, plus guard-evaluation costs. Guard normalization, support pruning, and soft-label computation precede search, so each guard check uses stored values and costs time proportional to its number of literals.

The prefix heuristic requires distances to b relevant anchor sets. These can be precomputed by b multi-source shortest-path searches on the semantic graph, taking $O(b\, m \log n)$ time with a binary heap.

SAGAS runs one prefix search and up to K suffix return searches, one for each retained accepting endpoint. The resulting worst-case search time is

$$O((K+1)M_{\mathcal{P}} \log N_{\mathcal{P}})\,. \tag{B.21}$$

In practice, the graph-distance heuristic can reduce product-state expansion, and an enabled dwell removes the need for a return search at that endpoint. Dwell testing adds no product states and checks at most d_B self-loop terms per endpoint, separately from the movement-edge bound.

For a fixed task and network, the computational cost of execution and monitoring grows linearly with the rollout horizon: each step computes the state embedding, updates the monitor and tracking target, and queries the policy, with guard-aware steering applied when enabled.

C Soundness of synthesis and monitored execution

The proof of Proposition 1 connects the synthesized plan to the labels observed during execution. We first show that the plan's guard sequence admits accepting runs of the original NBA. We then show that the monitor preserves a valid run over the actual observations whenever it confirms a target. Infinitely many confirmed suffix cycles make this run accepting, establishing satisfaction of the task formula.

Throughout, L_ϕ is the task's labeling function and $\mathcal{B}_\phi = (Q, Q_0, \Sigma, \delta, F)$ is the original NBA, with $\mathcal{L}(\mathcal{B}_\phi) = \mathcal{L}(\phi)$ and $\Sigma = 2^{\mathcal{AP}_\phi}$. The support-restricted automaton retains states $\widetilde{Q} \subseteq Q$ and accepting states $\widetilde{F} = F \cap \widetilde{Q}$.

Soundness of the planned guard sequence. Let $\rho = (\rho_{\mathrm{pre}}, \rho_{\mathrm{suf}})$ be a returned product plan, unfolded as

$$\rho_{\mathrm{pre}} \oplus \rho_{\mathrm{suf}}^{\omega} = (v_0, q_0)(v_1, q_1) \cdots .$$

The prefix ends at (v_p, q_p) with $q_p \in \widetilde{F}$, and the suffix returns to this product state on each repetition. Let η_i be the guard term selected for the transition from (v_i, q_i) to (v_{i+1}, q_{i+1}), including the self-loop guard for a dwell suffix. Product initialization consumes $L_\phi(s_0)$ before search, so $q_0 \in \delta(q^{\mathrm{init}}, L_\phi(s_0))$ for some $q^{\mathrm{init}} \in Q_0$.

Lemma 1 (Soundness of the synthesized guard sequence) *For the returned plan ρ, the set of infinite words $\widehat{\sigma} \in \Sigma^\omega$ satisfying*

$$\widehat{\sigma}_0 = L_\phi(s_0), \qquad \widehat{\sigma}_{i+1} \models \eta_i \quad \textit{for all } i \geq 0$$

is nonempty, and every such word is accepted by $\mathcal{B}_\phi$.

Proof. Automaton preprocessing expresses each original guard as a disjunction of satisfiable conjunctive terms and retains the supported alternatives. The movement and dwell rules select among these terms, so each planned transition satisfies $\eta_i \Rightarrow \widetilde{g}_{q_i,q_{i+1}} \Rightarrow g_{q_i,q_{i+1}}$. Consequently, a word satisfying the selected guards gives $q_{i+1} \in \delta(q_i, \widehat{\sigma}_{i+1})$ at every transition, as in the standard product construction [2, 15]. Together with the initialization transition on $\widehat{\sigma}_0$, these transitions form a run from $q^{\mathrm{init}} \in Q_0$. The run visits $q_p \in \widetilde{F} \subseteq F$ on every suffix repetition and is therefore accepting.

To construct a word satisfying the guards, choose one satisfying assignment for each term in the finite prefix and suffix, prepend $L_\phi(s_0)$, and repeat the suffix assignments indefinitely. Each retained term is satisfiable, so this construction establishes nonemptiness. □

Soundness of confirmed execution segments. During physical execution, the monitor reads $\sigma_t = L_\phi(s_t)$ at each observation time t. These labels are obtained from the actual rollout τ_Γ, where

$\Gamma = \mathrm{proj}_{\mathcal{V}_\phi}(\rho)$. A segment between selected waypoints can span several physical steps. The monitor follows the original NBA over all labels in that segment and checks that they connect its planned automaton endpoints.

Definition 1 (Confirmation of targets and suffix cycles) *Let $\kappa_0 = 0 < \kappa_1 < \cdots$ be the plan indices of successive confirmed targets, and let t_r be the confirmation time of target κ_r, with $t_0 = 0$ for the initial product state. This sequence may be finite or infinite; gaps between indices correspond to skipped intermediate waypoints. Starting from a confirmed target, the monitor initializes $S_r(t_r) = \{q_{\kappa_r}\}$ and updates it with each subsequent observation:*

$$S_r(t+1) = \bigcup_{q \in S_r(t)} \delta(q, \sigma_{t+1}).$$

Target κ_{r+1} is confirmed at time t_{r+1} when the reach and guard checks in Appendix B.7 pass and $q_{\kappa_{r+1}} \in S_r(t_{r+1})$. The next segment begins with the singleton $\{q_{\kappa_{r+1}}\}$ and continues with observations after t_{r+1}.

After the prefix, a suffix cycle is confirmed when these checks establish a return to its accepting endpoint (v_p, q_p). Each cycle includes at least one new physical observation, including when the waypoint remains unchanged in a dwell suffix.

Write $\delta^*(U, w)$ for the set of automaton states reachable from $U \subseteq Q$ after reading a finite label word w. The following lemma identifies the monitor's state set with this reachability relation and connects successive confirmations to a run over the observed trajectory.

Lemma 2 (Soundness of monitored progress) *During the segment starting at target κ_r, the monitor in Algorithm B.3 maintains*

$$S_r(t) = \delta^*(\{q_{\kappa_r}\}, \sigma_{t_r+1} \cdots \sigma_t)\,.$$

Whenever target κ_{r+1} is confirmed, the observed segment admits a run from q_{κ_r} to $q_{\kappa_{r+1}}$. These segment runs concatenate with the initialization transition to form a run of the original NBA over all labels observed up to that confirmation.

Proof. At $t = t_r$, the segment word is empty and both sides equal $\{q_{\kappa_r}\}$. Each update applies δ to every currently reachable state using the next observed label. Induction on the observation time therefore establishes the equality throughout the segment. At confirmation, membership of $q_{\kappa_{r+1}}$ in this set gives a run between the two planned automaton states over precisely $\sigma_{t_r+1} \cdots \sigma_{t_{r+1}}$.

The initialization transition covers σ_0 and ends at q_0. Each confirmed segment extends the preceding run from its final automaton state, and the next segment begins with the following observation. Concatenating these runs therefore yields a run from an original initial state over the full label sequence observed so far. □

LTL satisfaction through repeated suffix execution. The segment-wise result extends to infinite executions because every confirmed suffix cycle returns to the same accepting automaton state.

Proposition 2 (LTL satisfaction from verified recurrence) *Let τ_Γ be an execution of a lasso returned by SAGAS. If execution completes the prefix and the monitor confirms infinitely many suffix cycles according to Definition 1, then $\tau_\Gamma \models \phi$.*

Proof. Lemma 2 provides compatible finite runs between consecutive confirmed targets. Concatenating them with the initialization transition constructs a run over the physical label sequence. Since every suffix cycle includes a new observation, infinitely many confirmed cycles extend this run over the entire infinite trajectory. The infinitely many returns to $q_p \in \widetilde{F} \subseteq F$ make the run accepting, and $\mathcal{L}(\mathcal{B}_\phi) = \mathcal{L}(\phi)$ gives $\tau_\Gamma \models \phi$. □

Together, these results prove Proposition 1.

D Scope and limitations

The logical guarantees in Appendix C concern the planned guards and the labels observed during execution. Completing a task also requires the offline data to support the necessary motions and the learned executor to perform them reliably.

Predicate evaluation and offline support. SAGAS assumes that a predicate evaluator is available at task time and uses it to label states preserved from the offline dataset. To ground a guard term containing positive literals, the planner needs a dataset state that satisfies the complete term and can be connected to the graph as an anchor. Sparse coverage can leave a term without such a witness; disabling that alternative may then prevent a plan from being found even when the task is physically feasible.

Learned reachability and execution. Graph edges describe estimated reachability between latent waypoint regions. Executing an edge, however, requires the frozen policy to reach its target from the particular physical state produced by preceding actions. Errors in the learned reachability estimate or in waypoint tracking can therefore cause execution to fail even when the planner has found an accepting lasso. The failures analyzed in Appendix G motivate more reliable local tracking and suffix plans whose transitions can be repeated consistently.

Planning cost and search budget. SAGAS returns the lowest graph-cost lasso among the candidates retained under budget K. A finite budget can omit an accepting endpoint with a cheaper complete lasso, so increasing K trades additional computation for the opportunity to find a better plan (Appendix I.4). The objective uses learned temporal-distance costs as estimates of execution length. A guarantee of optimal physical execution would additionally require assumptions relating these costs to realized step counts.

Negative constraints. When a guard requires $\neg\ell$, the soft-label screen admits an original graph node when the weighted fraction of its preserved states satisfying ℓ is at most τ_{soft}. With a positive threshold, some of those states may therefore violate the guard. Even when all preserved states satisfy $\neg\ell$, the motion between waypoints may encounter states where ℓ holds. The threshold controls how nodes are screened from the data; it does not bound the probability of a violation during execution. The monitor checks active negative obligations on the states actually reached, while navigation additionally uses guard-aware steering to bias motion away from forbidden regions. Assessing recorded transition segments during planning could help account for violations between waypoints.

Execution failures and recovery. SAGAS terminates a rollout when a guard is violated or waypoint tracking times out. A future extension could replan after a timeout, using the current physical state and the automaton states consistent with the observed trajectory to seek another feasible continuation of the task.

Recurrent specifications. We evaluate recurrent tasks by requiring the executor to complete the prefix and traverse the suffix M times within the rollout budget (Appendix E.4). This finite-lasso criterion tests whether the planned behavior can be sustained over the evaluation horizon. Satisfaction of the infinite-horizon specification follows when the monitor continues to confirm suffix cycles indefinitely, as established in Proposition 2.

Environment scope. The evaluation uses state-based predicates and a fixed environment topology for each dataset. Generalization to unseen layouts or changing topology remains untested and would require adapting the reachability interface to the new connectivity. Visual or language-based predicate grounding and planning under partial observability are further directions for extending the framework.

Deployment scope. Moving from the simulated benchmarks to physical robots would require validating the learned executor under real operating conditions and integrating independent safety monitors and fail-safe controllers.

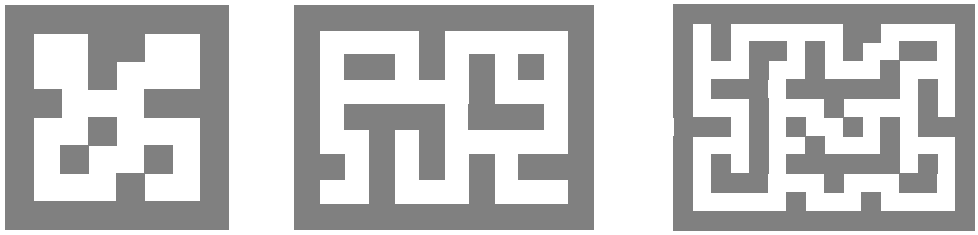
E Experimental protocols and implementation details

The navigation benchmark compares end-to-end LTL planning across robot dynamics and offline dataset regimes. The overlapping-region and Scene suites examine predicate generalization through simultaneous region membership and joint object configurations. Appendix F reports the results for all three suites.

E.1 Navigation environments and datasets

We use the `antmaze` and `humanoidmaze` locomotion domains from OGBench [19]. AntMaze has an 8-DoF agent with 29-dimensional state observations. Its eight benchmark settings combine *medium*, *large*, and *giant* maze layouts with the available `navigate`, `stitch`, and `explore` datasets.

HumanoidMaze has a 21-DoF agent with 69-dimensional state observations and uses the same family of layouts with the available `navigate` and `stitch` datasets. Figure E.1 shows the layouts. The methods learn from offline trajectories and receive no obstacle maps.



(a) *Medium* (b) *Large* (c) *Giant*

Figure E.1: Maze layouts used in the OGBench locomotion domains. Gray regions indicate walls and white regions indicate navigable space.

All datasets contain task-agnostic trajectories collected independently of the test formulas. The AntMaze data regimes differ in the behavior they provide: `navigate` contains noisy-expert maze traversals, `stitch` contains short fragments that require composition at task time, and `explore` contains random trajectories with broad coverage but low-quality behavior.

E.2 Navigation task generation

We generate $\mathrm{LTL}_{-\mathrm{X}}$ tasks by randomly instantiating and composing the templates in Table E.1. The difficulty groups in Table E.2 control the number of propositions and the number of template instances combined by conjunction. Each proposition corresponds to a geometric region centered in a sampled free maze cell, with regions constrained to be pairwise disjoint.

Each environment–difficulty pair contains 100 tasks shared by all methods. The generator checks that each formula is syntactically well formed and that its automaton has a nonempty language. Every task that passes these checks is retained and evaluated by all methods.

Table E.1: Parameterized LTL task templates used by the generator.

LTL formula structure	**Semantic description**
$\mathrm{F}\ell_i$	**Reach:** Eventually visit region i.
$\mathrm{G}(\neg\ell_i)$	**Safety:** Always avoid region i.
$\mathrm{F}(\ell_1 \wedge \mathrm{F}(\ell_2 \wedge \cdots \wedge \mathrm{F}\ell_m))$	**Sequence:** Visit $1 \to 2 \to \cdots \to m$.
$\mathrm{F}\ell_1 \wedge \mathrm{F}\ell_2 \wedge \cdots \wedge \mathrm{F}\ell_m$	**Coverage:** Visit m regions in any order.
$\neg\ell_{\text{avoid}}\,\mathrm{U}\,\ell_{\text{goal}}$	**Conditional:** Avoid until goal is reached.
$\mathrm{GF}(\ell_1 \wedge \mathrm{F}(\ell_2 \wedge \cdots \wedge \mathrm{F}\ell_m))$	**Patrol:** Infinitely revisit $1 \to \cdots \to m$.
$\mathrm{F}(\ell_1 \vee \ell_2 \vee \cdots \vee \ell_m)$	**Choice:** Visit at least one region in set.
$\mathrm{FG}\ell_i$	**Persistence:** Eventually stay in region i.
$\mathrm{F}(\ell_1 \wedge \mathrm{F}(\ell_2 \wedge \mathrm{FG}\ell_3))$	**Sequence-to-persist:** Visit a sequence and eventually remain in the final region.
$\bigwedge_i \mathrm{F}\ell_i \wedge \bigwedge_{i>1}(\neg\ell_i\,\mathrm{U}\,\ell_{i-1})$	**Strict order:** Visit multiple regions while enforcing a prescribed order.
$\bigwedge_i \mathrm{F}\ell_i \wedge \bigwedge_{j>1}(\neg\ell_1\,\mathrm{U}\,\ell_j)$	**Last visit:** Visit all regions while delaying region 1 until the others are reached.

Table E.2: Difficulty groups for generated LTL test cases.

Difficulty	**Logical composition**	**Labeled regions**
Easy	One template instance, no conjunction of multiple templates	At most 5
Medium	Conjunction of up to 3 template instances	At most 5
Hard	Conjunction of up to 4 template instances	At most 8

E.3 Compared methods

SAGAS, LF-GAS, and GA-GAS use the same latent graph and frozen policy within each dataset. They differ in how they select the high-level plan, allowing a comparison of planning strategies under a common reachability interface.

LF-GAS (Logic-first + GAS) first selects an accepting prefix–suffix sequence in the Büchi automaton, favoring sequences with fewer transitions. It maps the positive requirements to anchors and connects them by shortest paths in the latent graph, using the same node-level forbidden-label pruning as SAGAS. Graph costs thus guide motion between anchors, while the automaton sequence is chosen independently of those costs.

GA-GAS (Greedy anchor + GAS) selects the locally feasible outgoing Büchi transition with the lowest current GAS shortest-path cost to its required proposition, using the same forbidden-label pruning. Transitions without positive literals are taken in place when their negative requirements pass the check. The greedy rule constructs the prefix and a suffix returning to the same accepting graph node and Büchi state.

AP-HIQL (Automaton path + HIQL goal reaching) adapts LTL-instructed goal reaching [10] using a separate frozen HIQL [20] policy. It first selects an accepting prefix–suffix label sequence in the automaton and chooses dataset states from the required regions as goals. The policy then reaches these goals in order while monitoring forbidden labels. This provides an end-to-end comparison with a different goal-conditioned architecture.

E.4 Evaluation protocol and metrics

All reusable components are learned from the offline dataset and frozen before evaluation, including the HIQL policy used by AP-HIQL. For each test case, every method receives the same formula, task-space labeling function, and initial state.

The navigation benchmark and product-size experiments initialize product search at an initial state of the NBA. The overlap and Scene suites first advance the NBA with $L_\phi(s_0)$, following Section 3.4. In all three suites, final verification reads the physical label sequence from $L_\phi(s_0)$ onward.

Labels are evaluated on the executed states. Success rate (SR) is the fraction of tasks meeting the finite-lasso criterion: prefix completion and M verified suffix traversals, with $M = 2$ unless otherwise specified. A rollout that reaches its environment-step limit before meeting this criterion counts as a failure.

Normalized capped cost (NCC) evaluates the prefix and first valid suffix traversal under a common budget. For a rollout that completes both, let T_{pre} and T_{suf} denote the respective environment-step counts. Its NCC is

$$\mathrm{NCC} = \frac{\min\{\lambda T_{\mathrm{pre}} + (1-\lambda)T_{\mathrm{suf}},\, C_{\max}\}}{C_{\max}},$$

where $C_{\max}$ is the capped evaluation budget shared by all methods within a setting. Navigation uses $\lambda = 0.5$ and the setting-specific budget $C_{\max}$, giving the equivalent expression $\min\{T_{\mathrm{pre}} + T_{\mathrm{suf}}, 2C_{\max}\}/(2C_{\max})$. We assign NCC 1 when the prefix or first valid suffix traversal is not completed. Lower NCC therefore reflects shorter executions and fewer capped failures.

In navigation, SR requires $M = 2$ suffix traversals, whereas NCC uses the one-cycle criterion. A rollout that completes the first suffix cycle but fails during the second counts as an SR failure, although its NCC can remain below 1.

E.5 Parameters and computing resources

Table E.3 lists the navigation parameters shared by SAGAS, LF-GAS, and GA-GAS. Backbone training follows the GAS implementation for the corresponding OGBench domains [14]; the methods reuse the same trained graph and policy within each dataset.

Table E.3: Task-time semantic graph augmentation, planning, and execution parameters for the navigation benchmark.

Parameter	**AntMaze**	**HumanoidMaze**
Task cases per environment–difficulty pair	100	100
Anchor source	Dataset retrieval	Dataset retrieval
Initial anchors per proposition N_s	3	3
Maximum candidate attempts per proposition	50	50
Soft-label threshold τ_{soft}	0.05	0.20
Temporal-distance scale H_{TD}	8	32
Prefix candidates K	5	5
Prefix–suffix trade-off λ	0.5	0.5
Suffix traversals for finite evaluation M	2	2
Maximum rollout steps	8000	12000 / 16000 / 20000
Guard-aware steering radius ρ_{rep}	64	64
Guard-aware steering gain β	0.1	0.01
Guard-aware steering power p	1.0	1.0
Guard-aware steering max norm $r_{\max}$	0.2	0.02

The three HumanoidMaze rollout budgets correspond to `medium`, `large`, and `giant` layouts, respectively. Navigation experiments ran on an Ubuntu 24.04 server with dual Intel Xeon Gold 6426Y CPUs, 256 GB RAM, and one NVIDIA RTX 3090 GPU with 24 GB memory. The following protocols specify the settings used for the overlap and Scene suites.

E.6 Overlapping-region tasks and evaluation

All four task families use the intersection of regions a and b as a target. The agent must reach the overlap ($\mathrm{F}(a \wedge b)$), avoid c until it reaches the overlap ($\neg c \,\mathrm{U}\, (a \wedge b)$), revisit it repeatedly ($\mathrm{GF}(a \wedge b)$), or reach it and then visit d ($\mathrm{F}((a \wedge b) \wedge \mathrm{F}d)$). In AntMaze Giant-Stitch and Large-Explore, we sample 40 configurations per family in each environment, giving 320 tasks. Region shapes include circles, convex polygons, and non-convex polygons. Before planning, we check that the target intersection is nonempty and the initial position is clear, then fix the regions and initial state for each task.

We use the AntMaze settings in Table E.3 for the number of prefix candidates K, guard-aware steering, and maximum rollout steps. For overlapping regions, we use a soft-label threshold of $\tau_{\mathrm{soft}} = 0.01$. We run each task once and report finite-lasso SR as defined in Appendix E.4.

E.7 Scene manipulation tasks and evaluation

In OGBench `scene-play-v0`, a robot arm manipulates a cube, a drawer, a window, and two buttons (Figure E.2). Each button toggles the lock on either the drawer or the window, so moving one of these objects can require first pressing its button. The offline dataset consists of scripted interactions with these objects, collected independently of the LTL tasks. We use the released GAS Scene representation and low-level policy and build a reachability graph that retains the dataset states assigned to each node.

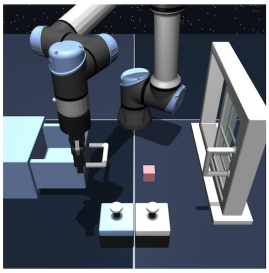


Figure E.2: Scene workspace, cropped from the OGBench overview [19].

Predicates over joint object configurations. We define the task predicates on a 7D mixed discrete–continuous space: the cube's 3D position, the drawer and window positions, and the two binary button states.

The predicates b_0 and b_1 indicate whether the corresponding button is on. We label the drawer open (d_o) at positions at or below -0.10 and closed (d_c) at or above -0.08. For the window, the open predicate (w_o) holds at or above 0.12 and the closed predicate (w_c) at or below 0.08. Between its two thresholds, the drawer or window is labeled neither open nor closed.

We choose three fixed cube positions A, B, and C from the offline data. The predicates c_A, c_B, and c_C each require the cube to be within 0.06 of the corresponding position; c_{A+} uses a wider radius of 0.10 around A. For the cube-in-drawer predicate c_D, the cube must lie inside a box that moves with the drawer handle: $x \in [0.21, 0.45]$, $y - y_{\text{handle}} \in [-0.27, -0.07]$, and $z \in [0, 0.15]$.

Table E.4 defines the ten joint configurations. Each row specifies the object predicates that must hold simultaneously.

Table E.4: Joint object configurations used in the Scene tasks.

Symbol	Main feature	Full joint condition
D_r	Drawer ready	$b_0 \wedge d_c \wedge w_c \wedge c_{A+}$
D_o	Drawer open	$b_0 \wedge d_o \wedge w_c \wedge c_{A+}$
W_r	Window ready	$b_1 \wedge d_c \wedge w_c \wedge c_{A+}$
W_o	Window open	$b_1 \wedge d_c \wedge w_o \wedge c_{A+}$
C_A	Cube at A, objects closed	$c_A \wedge d_c \wedge w_c$
C_B	Cube at B, objects open	$c_B \wedge d_o \wedge w_o$
C_{CD}	Cube at C, drawer open	$c_C \wedge b_1 \wedge d_o \wedge w_c$
C_{CW}	Cube at C, window open	$c_C \wedge b_1 \wedge d_c \wedge w_o$
C_D	Cube stored, window closed	$c_D \wedge b_0 \wedge d_c \wedge w_c$
C_{DW}	Cube stored, window open	$c_D \wedge b_0 \wedge d_c \wedge w_o$

Fixed task design. Each of the six task families contains four hand-written LTL formulas that combine different predicates. Table E.5 gives one example per family. In the two-stage example, $\mathrm{F}(D_r \wedge \mathrm{F}D_o)$, the drawer must first be closed and later open. At both stages, button 0 must be on, the window closed, and the cube near A.

We pair each of the 24 formulas with all five official OGBench Scene initial configurations, giving 20 tasks per family and 120 in total. The initial configurations supply the starting states; the LTL formulas specify the goals.

Table E.5: Representative Scene formulas, one from each of the six task families.

Task family	Representative formula
Single-stage configuration	$\mathrm{F}D_o$
Ordered two-stage manipulation	$\mathrm{F}(D_r \wedge \mathrm{F}D_o)$
Ordered three-stage manipulation	$\mathrm{F}(C_A \wedge \mathrm{F}(C_{CD} \wedge \mathrm{F}C_{CW}))$
Choice then configuration	$\mathrm{F}((D_r \vee W_r) \wedge \mathrm{F}D_o)$
Negative-until	$\neg c_D \,\mathrm{U}\, C_{CD}$
Recurrent suffix	$\mathrm{GF}(D_o \wedge \mathrm{F}W_o)$

Execution and verification. When selecting anchors, we give priority to states immediately following an observed change in the relevant object and retain examples with different configurations of the other objects. We use a soft-label threshold of $\tau_{\text{soft}} = 0.01$ for negative literals. Task-time planning is timed from predicate-evaluator initialization to the end of product search, including semantic graph augmentation (Table I.2).

We run each task once, with at most 8,000 rollout steps and no guard-aware steering. An independent verifier evaluates the same predicates on the recorded state trajectory to check finite-lasso success. For recurrent tasks, this requires completing the prefix and one suffix traversal ($M = 1$). We report SR over all 120 tasks.

F Detailed task results

F.1 Navigation benchmark results

Tables F.1 and F.2 report the AntMaze and HumanoidMaze results, with 100 generated LTL tasks per environment–difficulty pair. SR and NCC follow the definitions in Appendix E.4.

SAGAS leads the aggregate AntMaze results with 71.8% SR and 0.30 NCC, compared with 67.0% and 0.35 for LF-GAS. Its advantage is most pronounced in `giant-stitch` and `large-explore`, where logical choices interact with reachability and coverage. HumanoidMaze has the same aggregate ordering: SAGAS reaches 57.4% SR and 0.45 NCC, followed by LF-GAS at 54.3% and 0.49. Some individual HumanoidMaze settings favor a baseline, while SAGAS leads overall and on the Hard-task summary in Table 1.

Table F.1: AntMaze LTL benchmark results. Each environment–difficulty setting contains 100 test cases. SR is finite-lasso execution success rate; NCC is normalized capped cost for prefix completion plus one suffix traversal. Dark/light gray shading marks the best/second-best method within each setting; higher SR and lower NCC are better.

Environment	Difficulty	SAGAS		LF-GAS		GA-GAS		AP-HIQL	
		SR (%)	NCC	SR (%)	NCC	SR (%)	NCC	SR (%)	NCC
Medium-Navigate	Easy	78.0	0.21 ± 0.37	79.0	0.21 ± 0.37	68.0	0.35 ± 0.45	70.0	0.27 ± 0.41
	Medium	83.0	0.17 ± 0.34	74.0	0.27 ± 0.41	60.0	0.42 ± 0.48	70.0	0.26 ± 0.40
	Hard	65.0	0.38 ± 0.45	60.0	0.42 ± 0.46	54.0	0.48 ± 0.47	61.0	0.42 ± 0.45
	All	75.3	0.25 ± 0.40	71.0	0.30 ± 0.42	60.7	0.42 ± 0.47	67.0	0.32 ± 0.43
Large-Navigate	Easy	73.0	0.27 ± 0.39	67.0	0.31 ± 0.42	59.0	0.41 ± 0.46	77.0	0.26 ± 0.39
	Medium	63.0	0.37 ± 0.44	67.0	0.38 ± 0.44	68.0	0.37 ± 0.44	71.0	0.32 ± 0.42
	Hard	55.0	0.50 ± 0.45	49.0	0.54 ± 0.45	45.0	0.58 ± 0.45	51.0	0.51 ± 0.45
	All	63.7	0.38 ± 0.44	61.0	0.41 ± 0.44	57.3	0.45 ± 0.46	66.3	0.36 ± 0.44
Giant-Navigate	Easy	61.0	0.41 ± 0.45	62.0	0.39 ± 0.43	55.0	0.47 ± 0.46	62.0	0.41 ± 0.43
	Medium	57.0	0.43 ± 0.43	56.0	0.42 ± 0.43	53.0	0.50 ± 0.45	47.0	0.53 ± 0.44
	Hard	48.0	0.53 ± 0.44	37.0	0.65 ± 0.43	45.0	0.60 ± 0.44	37.0	0.68 ± 0.42
	All	55.3	0.45 ± 0.44	51.7	0.49 ± 0.44	51.0	0.52 ± 0.45	48.7	0.54 ± 0.44
Medium-Stitch	Easy	81.0	0.21 ± 0.37	80.0	0.23 ± 0.39	75.0	0.28 ± 0.42	82.0	0.20 ± 0.34
	Medium	85.0	0.17 ± 0.34	81.0	0.19 ± 0.35	77.0	0.25 ± 0.40	69.0	0.32 ± 0.43
	Hard	71.0	0.30 ± 0.41	65.0	0.35 ± 0.44	56.0	0.46 ± 0.47	61.0	0.40 ± 0.44
	All	79.0	0.23 ± 0.38	75.3	0.26 ± 0.40	69.3	0.33 ± 0.44	70.7	0.31 ± 0.41
Large-Stitch	Easy	74.0	0.27 ± 0.39	73.0	0.28 ± 0.40	65.0	0.38 ± 0.45	67.0	0.34 ± 0.43
	Medium	81.0	0.21 ± 0.35	79.0	0.23 ± 0.35	73.0	0.29 ± 0.40	61.0	0.40 ± 0.45
	Hard	62.0	0.41 ± 0.43	49.0	0.50 ± 0.45	52.0	0.50 ± 0.45	45.0	0.57 ± 0.45
	All	72.3	0.30 ± 0.40	67.0	0.33 ± 0.42	63.3	0.39 ± 0.44	57.7	0.43 ± 0.45
Giant-Stitch	Easy	61.0	0.43 ± 0.43	56.0	0.49 ± 0.43	60.0	0.46 ± 0.44	20.0	0.82 ± 0.36
	Medium	56.0	0.46 ± 0.43	50.0	0.53 ± 0.43	45.0	0.58 ± 0.44	10.0	0.91 ± 0.27
	Hard	54.0	0.50 ± 0.42	41.0	0.64 ± 0.42	35.0	0.68 ± 0.41	8.0	0.93 ± 0.23
	All	57.0	0.46 ± 0.43	49.0	0.55 ± 0.43	46.7	0.57 ± 0.44	12.7	0.89 ± 0.29
Medium-Explore	Easy	92.0	0.11 ± 0.27	92.0	0.11 ± 0.25	81.0	0.21 ± 0.38	53.0	0.51 ± 0.46
	Medium	90.0	0.13 ± 0.29	82.0	0.19 ± 0.36	73.0	0.28 ± 0.43	63.0	0.42 ± 0.44
	Hard	85.0	0.18 ± 0.35	80.0	0.24 ± 0.38	74.0	0.28 ± 0.42	44.0	0.60 ± 0.45
	All	89.0	0.14 ± 0.30	84.7	0.18 ± 0.34	76.0	0.26 ± 0.41	53.3	0.51 ± 0.46
Large-Explore	Easy	83.0	0.20 ± 0.35	79.0	0.22 ± 0.36	74.0	0.29 ± 0.42	17.0	0.83 ± 0.37
	Medium	86.0	0.16 ± 0.31	83.0	0.20 ± 0.34	80.0	0.22 ± 0.37	15.0	0.84 ± 0.35
	Hard	80.0	0.26 ± 0.36	66.0	0.38 ± 0.43	68.0	0.36 ± 0.43	11.0	0.90 ± 0.28
	All	83.0	0.21 ± 0.35	76.0	0.26 ± 0.39	74.0	0.29 ± 0.41	14.3	0.86 ± 0.34
Overall	*All*	71.8	0.30 ± 0.41	67.0	0.35 ± 0.43	62.3	0.40 ± 0.45	48.8	0.53 ± 0.46

Table F.2: HumanoidMaze LTL benchmark results. Each environment–difficulty setting contains 100 test cases. SR is finite-lasso execution success rate; NCC is normalized capped cost for prefix completion plus one suffix traversal. Dark/light gray shading marks the best/second-best method within each setting; higher SR and lower NCC are better.

Environment	Difficulty	SAGAS		LF-GAS		GA-GAS		AP-HIQL	
		SR (%)	NCC	SR (%)	NCC	SR (%)	NCC	SR (%)	NCC
Medium-Navigate	Easy	81.0	0.24 ± 0.36	78.0	0.24 ± 0.35	76.0	0.29 ± 0.39	75.0	0.29 ± 0.38
	Medium	66.0	0.36 ± 0.41	66.0	0.38 ± 0.41	55.0	0.48 ± 0.45	77.0	0.30 ± 0.37
	Hard	67.0	0.36 ± 0.40	57.0	0.45 ± 0.42	55.0	0.52 ± 0.44	61.0	0.48 ± 0.42
	All	71.3	0.32 ± 0.39	67.0	0.35 ± 0.40	62.0	0.43 ± 0.44	71.0	0.36 ± 0.40
Medium-Stitch	Easy	68.0	0.31 ± 0.41	72.0	0.30 ± 0.39	69.0	0.33 ± 0.42	74.0	0.30 ± 0.39
	Medium	70.0	0.32 ± 0.39	74.0	0.31 ± 0.38	61.0	0.42 ± 0.44	67.0	0.37 ± 0.40
	Hard	64.0	0.39 ± 0.41	54.0	0.50 ± 0.43	56.0	0.50 ± 0.44	51.0	0.53 ± 0.42
	All	67.3	0.34 ± 0.40	66.7	0.37 ± 0.41	62.0	0.42 ± 0.44	64.0	0.40 ± 0.41
Large-Navigate	Easy	69.0	0.35 ± 0.42	73.0	0.32 ± 0.39	66.0	0.39 ± 0.43	69.0	0.35 ± 0.40
	Medium	66.0	0.37 ± 0.41	61.0	0.43 ± 0.43	55.0	0.48 ± 0.45	62.0	0.42 ± 0.42
	Hard	57.0	0.51 ± 0.43	44.0	0.59 ± 0.43	45.0	0.58 ± 0.43	48.0	0.56 ± 0.43
	All	64.0	0.41 ± 0.42	59.3	0.44 ± 0.43	55.3	0.48 ± 0.44	59.7	0.44 ± 0.43
Large-Stitch	Easy	75.0	0.30 ± 0.39	66.0	0.39 ± 0.43	61.0	0.43 ± 0.44	74.0	0.37 ± 0.38
	Medium	64.0	0.42 ± 0.43	60.0	0.42 ± 0.43	55.0	0.50 ± 0.45	67.0	0.44 ± 0.39
	Hard	49.0	0.56 ± 0.43	50.0	0.58 ± 0.42	41.0	0.65 ± 0.42	45.0	0.63 ± 0.38
	All	62.7	0.42 ± 0.43	58.7	0.46 ± 0.43	52.3	0.53 ± 0.45	62.0	0.48 ± 0.40
Giant-Navigate	Easy	44.0	0.53 ± 0.46	47.0	0.57 ± 0.46	47.0	0.56 ± 0.46	26.0	0.75 ± 0.38
	Medium	37.0	0.66 ± 0.43	39.0	0.64 ± 0.43	40.0	0.63 ± 0.44	22.0	0.79 ± 0.36
	Hard	23.0	0.77 ± 0.39	20.0	0.81 ± 0.35	18.0	0.84 ± 0.34	12.0	0.89 ± 0.28
	All	34.7	0.65 ± 0.43	35.3	0.67 ± 0.43	35.0	0.68 ± 0.43	20.0	0.81 ± 0.35
Giant-Stitch	Easy	52.0	0.48 ± 0.45	42.0	0.57 ± 0.45	38.0	0.61 ± 0.46	16.0	0.86 ± 0.32
	Medium	44.0	0.57 ± 0.44	40.0	0.60 ± 0.44	35.0	0.65 ± 0.43	10.0	0.91 ± 0.27
	Hard	37.0	0.67 ± 0.43	34.0	0.68 ± 0.42	32.0	0.68 ± 0.42	9.0	0.92 ± 0.26
	All	44.3	0.57 ± 0.45	38.7	0.62 ± 0.44	35.0	0.65 ± 0.44	11.7	0.89 ± 0.28
Overall	*All*	57.4	0.45 ± 0.44	54.3	0.49 ± 0.44	50.3	0.53 ± 0.45	48.1	0.56 ± 0.43

F.2 Paired comparisons and confidence intervals

We compare methods on matched tasks to separate method differences from variation across settings. The navigation benchmark contains 4,200 tasks shared by four methods, giving 16,800 method–task outcomes. Each environment–difficulty group has 100 tasks. Domain-level means average these equally sized groups, while their standard deviations describe cross-setting variation. Pairing outcomes for the same formula and initial state within each setting and difficulty yields 2,400 AntMaze pairs and 1,800 HumanoidMaze pairs for each method comparison.

We obtain paired-bootstrap intervals by resampling matched records within each fixed environment stratum and taking the 2.5% and 97.5% percentiles. For SR, two-sided exact McNemar tests use the discordant outcomes, with Holm correction across 24 comparisons: three baselines, two domains, and four task groups (all difficulties combined, Easy, Medium, and Hard). The six aggregate comparisons in Table F.3 belong to this family. Paired-bootstrap intervals also quantify NCC differences, while Wilson intervals summarize SR within each setting and difficulty.

Table F.3: Aggregate paired effects (SAGAS minus baseline). Brackets are 95% paired-bootstrap intervals; p denotes the Holm-corrected p-value from a two-sided exact McNemar test for SR.

Domain	Baseline	ΔSR (pp)	ΔNCC	p
AntMaze	LF-GAS	$+4.9\ [3.1, 6.7]$	$-0.046\ [-0.061, -0.030]$	2.2×10^{-6}
	GA-GAS	$+9.5\ [7.7, 11.5]$	$-0.101\ [-0.118, -0.083]$	2.8×10^{-21}
	AP-HIQL	$+23.0\ [20.7, 25.3]$	$-0.224\ [-0.245, -0.204]$	5.3×10^{-67}
HumanoidMaze	LF-GAS	$+3.1\ [1.0, 5.3]$	$-0.034\ [-0.052, -0.016]$	0.032
	GA-GAS	$+7.1\ [5.1, 9.2]$	$-0.076\ [-0.095, -0.058]$	9.8×10^{-10}
	AP-HIQL	$+9.3\ [6.7, 12.0]$	$-0.111\ [-0.133, -0.089]$	4.5×10^{-10}

All six aggregate comparisons favor SAGAS: its SR intervals lie above zero and its NCC intervals below zero. For LF-GAS and GA-GAS, the graph and low-level policy are held fixed, so the paired differences reflect high-level planning choices under a common executor, including how reliably the selected waypoint sequences can be realized.

Aggregate SAGAS SR is $1724/2400 = 71.83\%$ in AntMaze (Wilson 95% CI $[70.0\%, 73.6\%]$) and $1033/1800 = 57.39\%$ in HumanoidMaze ($[55.1\%, 59.7\%]$). Table F.4 reports Wilson intervals for each method, environment setting, and difficulty.

Table F.4: Per-setting finite-lasso SR (%) with Wilson 95% intervals. Each method–difficulty cell contains 100 tasks.

Setting	Difficulty	SAGAS	LF-GAS	GA-GAS	AP-HIQL
Ant giant-nav.	Easy	61 [51.2, 70.0]	62 [52.2, 70.9]	55 [45.2, 64.4]	62 [52.2, 70.9]
Ant giant-nav.	Medium	57 [47.2, 66.3]	56 [46.2, 65.3]	53 [43.3, 62.5]	47 [37.5, 56.7]
Ant giant-nav.	Hard	48 [38.5, 57.7]	37 [28.2, 46.8]	45 [35.6, 54.8]	37 [28.2, 46.8]
Ant giant-stitch	Easy	61 [51.2, 70.0]	56 [46.2, 65.3]	60 [50.2, 69.1]	20 [13.3, 28.9]
Ant giant-stitch	Medium	56 [46.2, 65.3]	50 [40.4, 59.6]	45 [35.6, 54.8]	10 [5.5, 17.4]
Ant giant-stitch	Hard	54 [44.3, 63.4]	41 [31.9, 50.8]	35 [26.4, 44.7]	8 [4.1, 15.0]
Ant large-expl.	Easy	83 [74.5, 89.1]	79 [70.0, 85.8]	74 [64.6, 81.6]	17 [10.9, 25.5]
Ant large-expl.	Medium	86 [77.9, 91.5]	83 [74.5, 89.1]	80 [71.1, 86.7]	15 [9.3, 23.3]
Ant large-expl.	Hard	80 [71.1, 86.7]	66 [56.3, 74.5]	68 [58.3, 76.3]	11 [6.3, 18.6]
Ant large-nav.	Easy	73 [63.6, 80.7]	67 [57.3, 75.4]	59 [49.2, 68.1]	77 [67.8, 84.2]
Ant large-nav.	Medium	63 [53.2, 71.8]	67 [57.3, 75.4]	68 [58.3, 76.3]	71 [61.5, 79.0]
Ant large-nav.	Hard	55 [45.2, 64.4]	49 [39.4, 58.7]	45 [35.6, 54.8]	51 [41.3, 60.6]
Ant large-stitch	Easy	74 [64.6, 81.6]	73 [63.6, 80.7]	65 [55.3, 73.6]	67 [57.3, 75.4]
Ant large-stitch	Medium	81 [72.2, 87.5]	79 [70.0, 85.8]	73 [63.6, 80.7]	61 [51.2, 70.0]
Ant large-stitch	Hard	62 [52.2, 70.9]	49 [39.4, 58.7]	52 [42.3, 61.5]	45 [35.6, 54.8]
Ant medium-expl.	Easy	92 [85.0, 95.9]	92 [85.0, 95.9]	81 [72.2, 87.5]	53 [43.3, 62.5]
Ant medium-expl.	Medium	90 [82.6, 94.5]	82 [73.3, 88.3]	73 [63.6, 80.7]	63 [53.2, 71.8]
Ant medium-expl.	Hard	85 [76.7, 90.7]	80 [71.1, 86.7]	74 [64.6, 81.6]	44 [34.7, 53.8]
Ant medium-nav.	Easy	78 [68.9, 85.0]	79 [70.0, 85.8]	68 [58.3, 76.3]	70 [60.4, 78.1]
Ant medium-nav.	Medium	83 [74.5, 89.1]	74 [64.6, 81.6]	60 [50.2, 69.1]	70 [60.4, 78.1]
Ant medium-nav.	Hard	65 [55.3, 73.6]	60 [50.2, 69.1]	54 [44.3, 63.4]	61 [51.2, 70.0]
Ant medium-stitch	Easy	81 [72.2, 87.5]	80 [71.1, 86.7]	75 [65.7, 82.5]	82 [73.3, 88.3]
Ant medium-stitch	Medium	85 [76.7, 90.7]	81 [72.2, 87.5]	77 [67.8, 84.2]	69 [59.4, 77.2]
Ant medium-stitch	Hard	71 [61.5, 79.0]	65 [55.3, 73.6]	56 [46.2, 65.3]	61 [51.2, 70.0]
Humanoid giant-nav.	Easy	44 [34.7, 53.8]	47 [37.5, 56.7]	47 [37.5, 56.7]	26 [18.4, 35.4]
Humanoid giant-nav.	Medium	37 [28.2, 46.8]	39 [30.0, 48.8]	40 [30.9, 49.8]	22 [15.0, 31.1]
Humanoid giant-nav.	Hard	23 [15.8, 32.2]	20 [13.3, 28.9]	18 [11.7, 26.7]	12 [7.0, 19.8]
Humanoid giant-stitch	Easy	52 [42.3, 61.5]	42 [32.8, 51.8]	38 [29.1, 47.8]	16 [10.1, 24.4]
Humanoid giant-stitch	Medium	44 [34.7, 53.8]	40 [30.9, 49.8]	35 [26.4, 44.7]	10 [5.5, 17.4]
Humanoid giant-stitch	Hard	37 [28.2, 46.8]	34 [25.5, 43.7]	32 [23.7, 41.7]	9 [4.8, 16.2]
Humanoid large-nav.	Easy	69 [59.4, 77.2]	73 [63.6, 80.7]	66 [56.3, 74.5]	69 [59.4, 77.2]
Humanoid large-nav.	Medium	66 [56.3, 74.5]	61 [51.2, 70.0]	55 [45.2, 64.4]	62 [52.2, 70.9]
Humanoid large-nav.	Hard	57 [47.2, 66.3]	44 [34.7, 53.8]	45 [35.6, 54.8]	48 [38.5, 57.7]
Humanoid large-stitch	Easy	75 [65.7, 82.5]	66 [56.3, 74.5]	61 [51.2, 70.0]	74 [64.6, 81.6]
Humanoid large-stitch	Medium	64 [54.2, 72.7]	60 [50.2, 69.1]	55 [45.2, 64.4]	67 [57.3, 75.4]
Humanoid large-stitch	Hard	49 [39.4, 58.7]	50 [40.4, 59.6]	41 [31.9, 50.8]	45 [35.6, 54.8]
Humanoid medium-nav.	Easy	81 [72.2, 87.5]	78 [68.9, 85.0]	76 [66.8, 83.3]	75 [65.7, 82.5]
Humanoid medium-nav.	Medium	66 [56.3, 74.5]	66 [56.3, 74.5]	55 [45.2, 64.4]	77 [67.8, 84.2]
Humanoid medium-nav.	Hard	67 [57.3, 75.4]	57 [47.2, 66.3]	55 [45.2, 64.4]	61 [51.2, 70.0]
Humanoid medium-stitch	Easy	68 [58.3, 76.3]	72 [62.5, 79.9]	69 [59.4, 77.2]	74 [64.6, 81.6]
Humanoid medium-stitch	Medium	70 [60.4, 78.1]	74 [64.6, 81.6]	61 [51.2, 70.0]	67 [57.3, 75.4]
Humanoid medium-stitch	Hard	64 [54.2, 72.7]	54 [44.3, 63.4]	56 [46.2, 65.3]	51 [41.3, 60.6]

F.3 Execution length on common-success tasks

NCC summarizes execution cost over all tasks, including failures. To compare how many steps the methods take when they all succeed, we restrict Tables F.5 and F.6 to tasks on which every method shown meets the finite-lasso criterion. Each row reports the number of such tasks and the mean number of environment steps each method takes to complete the prefix and first suffix traversal.

Table F.5: AntMaze common-success execution-length results. Each row is restricted to test cases solved by all displayed methods. N is the number of shared successful cases; entries report mean$\pm$standard deviation of execution length in environment steps for prefix completion plus one suffix traversal. Dark/light gray shading marks the shortest/second-shortest method within each setting.

Environment	Difficulty	N	SAGAS	LF-GAS	GA-GAS	AP-HIQL
Medium-Navigate	Easy	47	527 ± 342	587 ± 367	543 ± 357	617 ± 352
	Medium	46	515 ± 325	557 ± 335	529 ± 325	628 ± 440
	Hard	33	661 ± 476	731 ± 470	697 ± 522	886 ± 584
	All	126	557 ± 378	614 ± 389	578 ± 400	692 ± 465
Large-Navigate	Easy	45	919 ± 1306	843 ± 746	796 ± 733	761 ± 564
	Medium	39	865 ± 556	994 ± 660	862 ± 602	859 ± 495
	Hard	21	1165 ± 1004	1325 ± 1006	1267 ± 956	1119 ± 649
	All	105	948 ± 1020	995 ± 788	915 ± 753	869 ± 568
Giant-Navigate	Easy	30	905 ± 631	1033 ± 636	940 ± 674	1235 ± 868
	Medium	24	1454 ± 654	1548 ± 637	1490 ± 706	1727 ± 884
	Hard	15	1214 ± 971	1415 ± 1291	1289 ± 1201	1770 ± 1602
	All	69	1163 ± 754	1295 ± 842	1208 ± 849	1522 ± 1086
Medium-Stitch	Easy	59	544 ± 357	558 ± 317	524 ± 333	733 ± 492
	Medium	50	505 ± 258	543 ± 261	549 ± 334	832 ± 501
	Hard	31	681 ± 372	793 ± 476	712 ± 397	950 ± 572
	All	140	561 ± 333	605 ± 353	575 ± 354	816 ± 517
Large-Stitch	Easy	44	758 ± 637	856 ± 649	767 ± 641	852 ± 733
	Medium	44	763 ± 459	888 ± 495	788 ± 472	951 ± 574
	Hard	21	1189 ± 590	1289 ± 632	1302 ± 744	1498 ± 719
	All	109	843 ± 582	953 ± 606	878 ± 631	1017 ± 706
Giant-Stitch	Easy	12	1105 ± 1173	1458 ± 1307	1056 ± 1205	1600 ± 1241
	Medium	4	1413 ± 1612	1530 ± 1536	1413 ± 1612	1788 ± 1382
	Hard	4	664 ± 366	673 ± 376	673 ± 376	1419 ± 891
	All	20	1079 ± 1135	1316 ± 1222	1051 ± 1154	1601 ± 1155
Medium-Explore	Easy	47	315 ± 242	396 ± 303	380 ± 392	1298 ± 1413
	Medium	50	407 ± 220	433 ± 228	433 ± 269	1619 ± 1882
	Hard	33	456 ± 267	592 ± 389	545 ± 386	1544 ± 1131
	All	130	386 ± 246	460 ± 310	442 ± 351	1484 ± 1548
Large-Explore	Easy	15	233 ± 266	257 ± 258	232 ± 265	420 ± 304
	Medium	15	225 ± 174	263 ± 194	271 ± 258	1012 ± 1420
	Hard	8	1014 ± 1666	1004 ± 1658	977 ± 1671	859 ± 613
	All	38	394 ± 818	416 ± 809	404 ± 819	746 ± 972
Overall	*All*	737	688 ± 661	762 ± 651	709 ± 637	1034 ± 960

Table F.6: HumanoidMaze common-success execution-length results. Each row is restricted to test cases solved by all displayed methods. N is the number of shared successful cases; entries report mean±standard deviation of execution length in environment steps for prefix completion plus one suffix traversal. Dark/light gray shading marks the shortest/second-shortest method within each setting.

Environment	Difficulty	N	SAGAS	LF-GAS	GA-GAS	AP-HIQL
Medium-Navigate	Easy	56	1752 ± 2487	1687 ± 2323	1704 ± 2371	1925 ± 1332
	Medium	42	2103 ± 2222	2634 ± 2515	2388 ± 2546	2343 ± 1453
	Hard	34	1953 ± 1457	2251 ± 1648	2096 ± 1611	2527 ± 1421
	All	132	1916 ± 2168	2134 ± 2258	2023 ± 2265	2213 ± 1407
Medium-Stitch	Easy	43	1576 ± 1340	1590 ± 1230	1667 ± 1318	1894 ± 1665
	Medium	41	1805 ± 1204	1981 ± 1396	1938 ± 1775	2270 ± 1350
	Hard	26	2049 ± 1274	2359 ± 1521	2218 ± 1536	3738 ± 3205
	All	110	1773 ± 1277	1917 ± 1385	1898 ± 1553	2470 ± 2150
Large-Navigate	Easy	44	2529 ± 2615	2453 ± 2369	2489 ± 2543	2968 ± 2519
	Medium	33	2993 ± 3156	3305 ± 3103	2893 ± 3113	2911 ± 2443
	Hard	23	3352 ± 2115	3362 ± 2001	3391 ± 2005	4496 ± 3116
	All	100	2872 ± 2701	2943 ± 2575	2830 ± 2638	3301 ± 2697
Large-Stitch	Easy	51	2745 ± 2161	3432 ± 2979	3009 ± 2674	4673 ± 2983
	Medium	35	2711 ± 2243	3054 ± 2556	3117 ± 2807	5122 ± 3650
	Hard	16	3582 ± 1777	4844 ± 3596	3710 ± 1892	5459 ± 2820
	All	102	2864 ± 2138	3524 ± 2978	3156 ± 2605	4950 ± 3188
Giant-Navigate	Easy	14	1935 ± 2252	1650 ± 1577	1935 ± 2252	4651 ± 4257
	Medium	10	3526 ± 4211	4120 ± 4593	4120 ± 4593	5211 ± 2607
	Hard	3	1232 ± 1320	1143 ± 584	1232 ± 1320	2356 ± 2733
	All	27	2446 ± 3093	2508 ± 3191	2666 ± 3363	4604 ± 3566
Giant-Stitch	Easy	10	1696 ± 1485	1956 ± 1414	1708 ± 1476	4088 ± 4014
	Medium	7	2738 ± 2147	3031 ± 2046	2687 ± 2131	7303 ± 7000
	Hard	6	1981 ± 2216	2024 ± 2183	1981 ± 2216	2510 ± 1764
	All	23	2087 ± 1866	2301 ± 1813	2077 ± 1852	4655 ± 4929
Overall	*All*	494	2310 ± 2215	2565 ± 2445	2430 ± 2386	3300 ± 2838

We also form a separate common-success subset for each SAGAS–baseline pair and compare execution lengths on that subset. SAGAS uses on average 103.3 fewer environment steps than LF-GAS in AntMaze (95% CI $[81.6, 126.5]$) and 210.0 fewer in HumanoidMaze ($[102.1, 313.8]$), again measured over the prefix and first suffix traversal.

F.4 SAGAS and AP-HIQL across AntMaze datasets

To examine how relative performance varies across offline datasets, we compare SAGAS and AP-HIQL separately on Navigate, Stitch, and Explore using the 2,400 matched AntMaze tasks from Appendix F.2. AP-HIQL is most competitive on Navigate, whose offline data contain longer, more

Table F.7: Success rates by AntMaze dataset type. ΔSR is the difference in percentage points (SAGAS minus AP-HIQL). Brackets give 95% confidence intervals from 20,000 paired bootstrap replicates stratified by environment.

Dataset type	Tasks	SAGAS SR	AP-HIQL SR	ΔSR (pp)
Navigate	900	64.78%	60.67%	$+4.11\ [0.22, 8.11]$
Stitch	900	69.44%	47.00%	$+22.44\ [18.56, 26.33]$
Explore	600	86.00%	33.83%	$+52.17\ [48.17, 56.33]$
All	2,400	71.83%	48.83%	$+23.00\ [20.67, 25.33]$

coherent trajectories. SAGAS's SR advantage increases from 4.11 percentage points on Navigate to 22.44 on Stitch and 52.17 on Explore, where the data consist of fragmented or exploratory trajectories.

Table F.8: Predicate generalization. SR is finite-lasso execution success rate, determined by independent post-rollout verification. The suites use their stated suffix criteria and are reported separately. Per-family results are in Table F.9.

Setting / task construction	Tasks	Accepting lassos	SR
(a) AntMaze: overlapping regions, $M = 2$			
Giant-Stitch	160	149/160	130/160 (81.3%)
Large-Explore	160	143/160	130/160 (81.3%)
All overlapping-region tasks	320	292/320	260/320 (81.3%)
(b) OGBench Scene: joint object configurations, $M = 1$			
24 formulas $\times$ five initial states	120	120/120	81/120 (67.5%)

F.5 Overlapping-region and Scene manipulation results

The two predicate suites test simultaneous proposition assignments and joint object configurations. Their results use the respective protocols in Appendices E.6 and E.7.

Overlapping regions. SAGAS finds accepting lassos for 292 of the 320 tasks across AntMaze Giant-Stitch and Large-Explore. Independent post-rollout verification of the saved state trajectories confirms finite-lasso success on 260 of the 320 tasks, giving 81.3% SR (Wilson 95% CI $[76.6\%, 85.1\%]$). Both environments achieve 130/160 successes, but performance varies by task family: ordered overlap has the lowest SR in Giant-Stitch, whereas negative-until has the lowest SR in Large-Explore (Table F.9).

Joint object configurations. SAGAS finds accepting lassos for all 120 Scene tasks, and independent post-rollout verification confirms finite-lasso success on 81, giving 67.5% SR (Wilson 95% CI $[58.7\%, 75.2\%]$). Table F.9 breaks these results down by task family. Of the 39 failed executions, 25 fail to reach a target and 11 exceed the local step budget. The remaining failures are two negative-literal violations and one segment whose observed labels admit no automaton run between its planned source and target states.

Table F.9: Per-family predicate-generalization results. SR is computed over all tasks in each family using independent post-rollout finite-lasso verification.

Task family	Tasks	Accepting lassos	SR
AntMaze Giant-Stitch ($M = 2$)			
Overlap event	40	36/40	33/40 (82.5%)
Negative-until overlap	40	37/40	34/40 (85.0%)
Recurrent overlap	40	37/40	34/40 (85.0%)
Ordered overlap	40	39/40	29/40 (72.5%)
AntMaze Large-Explore ($M = 2$)			
Overlap event	40	38/40	36/40 (90.0%)
Negative-until overlap	40	31/40	28/40 (70.0%)
Recurrent overlap	40	37/40	32/40 (80.0%)
Ordered overlap	40	37/40	34/40 (85.0%)
OGBench Scene ($M = 1$)			
Single-stage configuration	20	20/20	17/20 (85.0%)
Ordered two-stage manipulation	20	20/20	14/20 (70.0%)
Ordered three-stage manipulation	20	20/20	11/20 (55.0%)
Choice, then configuration	20	20/20	13/20 (65.0%)
Negative-until	20	20/20	11/20 (55.0%)
Recurrent suffix	20	20/20	15/20 (75.0%)

G Failure-mode analysis

To identify where tasks fail, we group the SAGAS outcomes from the navigation benchmark in Tables F.1 and F.2. Of the 4,200 cases, 2757 succeed and 1443 fail, giving an overall SR of 65.6%.

Table G.1 assigns each failure to the first stage at which progress toward the finite-lasso criterion stops.

Table G.1: Aggregate SAGAS failure modes over 4,200 LTL test cases. Percentages are reported relative to failed cases and to all cases.

Failure mode	Count	Failed cases (%)	All cases (%)
Prefix execution or semantic realization failure	930	64.4	22.1
Suffix-cycle or finite-lasso realization failure	423	29.3	10.1
No accepting prefix or suffix found	90	6.2	2.1

More than 93% of failures occur during physical execution after a candidate prefix or suffix has been found. Prefix-stage locomotion failure, such as stalling or overturning, is the most frequent cause (748 cases). The remaining execution failures involve incomplete first and second suffix traversals (201 and 136), incomplete prefix tracking (100), or failed semantic checks during the suffix and prefix (86 and 82). Search finds no accepting prefix in 69 cases and no suffix for the retained prefixes in 21 cases.

Table G.2: Domain-wise SAGAS failure breakdown. Percentages in the last three columns are relative to failures within the corresponding domain.

Domain	Cases	Successes	SR (%)	Prefix failure	Suffix failure	Planning failure
AntMaze	2400	1724	71.8	417 (61.7%)	206 (30.5%)	53 (7.8%)
HumanoidMaze	1800	1033	57.4	513 (66.9%)	217 (28.3%)	37 (4.8%)

Execution failures are more frequent in HumanoidMaze, while planning failures remain uncommon in both domains (Table G.2). Giant mazes have the lowest SR, at 34.7% for `humanoidmaze-giant-navigate`, 44.3% for `humanoidmaze-giant-stitch`, 55.3% for `antmaze-giant-navigate`, and 57.0% for `antmaze-giant-stitch`. Longer waypoint chains expose the frozen executor to repeated tracking and suffix-stability challenges. Appendix F.2 compares SAGAS with the baselines on matched tasks.

Although every sampled proposition has connected anchors, prefix or suffix search fails in 90 cases under the chosen graph scale H_{TD} and candidate budget, particularly for formulas with more labels and negative or until constraints.

H Component and sensitivity analysis

The following analyses examine how semantic screening, execution steering, and learned graph costs contribute to planning and execution. We then vary graph resolution, data availability, and predicate constraints to measure their effects on plan feasibility and computation.

H.1 Soft-label screening and guard-aware steering

The paired ablations use the same 100 tasks per difficulty as the main benchmark on `antmaze-large-explore` and `humanoidmaze-medium-stitch`. The former tests semantic filtering under low-quality, broad-coverage data; the latter adds more demanding locomotion dynamics. Tasks, initial states, graph, and frozen policy are held fixed.

The **w/o Soft** variant removes negative-literal soft-label screening from product search while retaining anchors and execution rules. The **w/o Steering** variant keeps the selected plan and removes only the guard-aware steering term during execution. The two comparisons thus separate planning-time screening from execution-time adjustment.

Table H.1: Soft-label screening and guard-aware steering ablations. Each setting contains 100 tasks per difficulty; Average rows pool all 300 tasks. SR and NCC follow Appendix E.4, with NCC reported as mean±standard error. Higher SR and lower NCC are better.

Environment	Difficulty	SAGAS		w/o soft		w/o steering	
		SR (%)	NCC	SR (%)	NCC	SR (%)	NCC
Ant Large-Explore	Easy	83.0	0.198 ± 0.035	82.0	0.208 ± 0.036	82.0	0.209 ± 0.036
	Medium	86.0	0.162 ± 0.031	82.0	0.200 ± 0.035	85.0	0.174 ± 0.032
	Hard	80.0	0.263 ± 0.036	77.0	0.289 ± 0.038	78.0	0.269 ± 0.037
	Average	83.0	0.208 ± 0.020	80.3	0.232 ± 0.021	81.7	0.217 ± 0.020
Humanoid Medium-Stitch	Easy	68.0	0.312 ± 0.041	68.0	0.312 ± 0.041	68.0	0.314 ± 0.041
	Medium	70.0	0.316 ± 0.039	69.0	0.325 ± 0.040	70.0	0.317 ± 0.039
	Hard	64.0	0.387 ± 0.041	63.0	0.396 ± 0.041	60.0	0.426 ± 0.043
	Average	67.3	0.338 ± 0.023	66.7	0.344 ± 0.023	66.0	0.352 ± 0.024

Removing soft-label screening lowers average AntMaze Large-Explore SR from 83.0% to 80.3% and raises NCC from 0.208 to 0.232, with the largest differences on Medium and Hard tasks (Table H.1). Removing steering has a smaller effect in the same direction. In HumanoidMaze Medium-Stitch, soft labels mainly improve NCC, while steering is most useful on Hard tasks: without it, SR falls from 64.0% to 60.0% and NCC rises from 0.387 to 0.426. The full method achieves higher aggregate SR and lower NCC than either ablated variant in both environments.

H.2 Graph cost and realized execution length

SAGAS ranks candidate plans using learned temporal-distance edge costs. We examine whether the resulting graph cost reflects the number of environment steps needed to execute a plan.

The analysis uses 140 successful SAGAS rollouts, with 20 difficulty-stratified tasks from each of seven settings. For each rollout, we pair the planned graph cost with the realized number of environment steps for the same prefix and first suffix traversal. We compute a Spearman rank correlation separately for each setting.

Table H.2: Across-task association between graph cost and realized steps on successful executions. Each setting contributes 20 trajectories.

Setting	Spearman ρ
AntMaze Giant-Navigate	0.974
AntMaze Giant-Stitch	0.887
AntMaze Large-Explore	0.746
AntMaze Large-Navigate	0.973
AntMaze Large-Stitch	0.979
HumanoidMaze Giant-Stitch	0.893
HumanoidMaze Medium-Stitch	0.752

Correlations range from 0.746 to 0.979, with a median of 0.893 and a stratified-bootstrap 95% interval of $[0.823, 0.968]$ for the median. Within each setting, higher graph costs generally correspond to more execution steps, supporting the use of learned temporal-distance costs as a proxy for execution length.

H.3 Graph resolution and data subsampling

We examine how graph resolution and the amount of offline data used for graph construction affect graph size, accepting-lasso discovery, and planning time. For graph resolution, we compare temporal-distance scales at 0.75, 1, and 1.25 times the domain defaults: $H_{\mathrm{TD}} \in \{6, 8, 10\}$ in AntMaze Large-Stitch and Large-Explore, and $H_{\mathrm{TD}} \in \{24, 32, 40\}$ in HumanoidMaze Medium-Stitch. Each setting uses the same 30 Medium and 30 Hard tasks across scales, with initial states, learned representation, semantic thresholds, and search budget held fixed.

For data subsampling, we use a fixed seed to randomly select half of the complete trajectories in AntMaze Large-Explore, retaining 2.5 million of the five million states. We build an additional graph at the default scale from this subset, which also supplies node support and anchor candidates. The representation remains the one trained on the full dataset. We evaluate the graph on the same 60 tasks as the full-data graphs.

Table H.3: Graph resolution and data subsampling. Data denotes the fraction of offline trajectories used for graph construction, node support, and anchor retrieval. Accepting-lasso counts cover all 30 tasks per difficulty in each condition. Time is median total planning time in seconds, including semantic augmentation, automaton construction, graph copying, and search.

H_{TD}	Data	Nodes	Edges	Accepting lassos		Time (s)	
				Medium	Hard	Medium	Hard
AntMaze Large-Stitch (30 tasks per difficulty)							
6	100%	3,500	86,194	30/30	27/30	3.80	10.08
8	100%	1,563	29,166	30/30	28/30	1.81	4.10
10	100%	855	10,710	30/30	27/30	1.13	2.01
AntMaze Large-Explore (30 tasks per difficulty)							
6	100%	4,986	77,078	30/30	30/30	5.01	9.97
8	100%	2,525	34,774	30/30	29/30	3.61	5.97
10	100%	1,567	24,954	30/30	29/30	3.29	5.01
8	50%	1,798	22,402	30/30	29/30	1.89	3.37
HumanoidMaze Medium-Stitch (30 tasks per difficulty)							
24	100%	6,169	299,600	29/30	29/30	10.12	30.59
32	100%	2,188	107,362	30/30	29/30	4.51	11.43
40	100%	1,043	48,778	29/30	29/30	2.85	6.00

Increasing H_{TD} produces coarser graphs with fewer nodes and edges and reduces median planning time in all three settings (Table H.3). Accepting-lasso counts vary by only one of the 60 tasks across scales within each setting. All requested propositions retain connected anchors in every tested condition.

Halving the graph-construction data in Large-Explore reduces the node count from 2,525 to 1,798 and median planning time from 4.45 to 2.45 s over the 60 tasks. The half-data and full-data default graphs agree on whether an accepting lasso is found for every task, finding one for 59 of the 60. For these tasks, the smaller graph and support pool therefore reduce planning time while preserving the ability to find accepting lassos.

H.4 Predicate constraints and support subsampling

We examine how adding object-state constraints and reducing offline support affect the availability of connected anchors and predicate grounding time. Starting from the six Scene configurations $D_r, D_o, W_r, W_o, C_{CD}, C_{CW}$ in Table E.4, we first constrain only the cube's position (three coordinates), then add the drawer and window positions (five), and finally one button state (six). The graph, learned representation, and target values and thresholds for the retained constraints remain fixed. Removing constraints makes some configurations identical, leaving two, five, and six distinct predicates at the respective levels. Each distinct predicate is evaluated once per support subset.

The full support pool contains 2,864 distinct offline states, with at most four retained per graph node. Using three random seeds, we draw nested subsets containing 25% and 50% of the pool and compare them with the full pool, evaluated once. Each subset supplies both node support and anchor candidates. Grounding time covers evaluator initialization, predicate evaluation, joint-witness retrieval, and anchor insertion.

Table H.4: Scene predicate grounding as object constraints and support size vary. Coordinates gives the number of constrained object-state coordinates. Connected reports how many of the six configurations have a connected anchor under those constraints. Matching-state counts and grounding times are medians over distinct predicates. Ranges give the minimum and maximum of these medians across the three subset-sampling seeds.

Coordinates	Support	Predicates	Connected	Matching states	Time (s)
3	25%	2	6/6	234.0–238.5	2.33–2.51
3	50%	2	6/6	462.5–474.0	4.56–5.26
3	100%	2	6/6	939.0	9.30
5	25%	5	6/6	33.0–46.0	4.48–4.73
5	50%	5	6/6	71.0–82.0	8.64–9.01
5	100%	5	6/6	149.0	17.54
6	25%	6	6/6	27.0–30.5	6.58–6.81
6	50%	6	6/6	58.5–67.5	12.78–13.23
6	100%	6	6/6	134.5	25.93

More specific predicates have fewer matching support states and take longer to ground (Table H.4). With full support, median grounding time rises from 9.30 s for cube position alone to 17.54 s with drawer and window constraints, and to 25.93 s when a button state is added.

Reducing support lowers grounding time at every constraint level. For the six-coordinate predicates, the median falls from 25.93 s at full support to 6.58–6.81 s at 25% support. All six configurations retain connected anchors at this lowest support level for each of the three seeds.

I Planning time and search complexity

We report task-time planning costs for navigation and Scene, then examine the graph and automaton sizes that determine the product search space. Controlled comparisons on fixed tasks measure the effects of search strategy and candidate budget.

I.1 Task-time planning

Navigation. We measure task-time planning as the combined time spent on semantic graph augmentation and product search before execution. Table I.1 summarizes this time for tasks that SAGAS successfully executes, grouped by domain, task difficulty, and maze size.

Table I.1: Navigation task-time planning in seconds (mean±standard deviation over successful SAGAS executions). Times include semantic graph augmentation and product search; the Medium, Large, and Giant columns denote maze size.

Domain	Difficulty	Medium	Large	Giant
AntMaze	Easy	1.67 ± 2.04	1.86 ± 2.45	1.40 ± 1.85
	Medium	1.87 ± 1.96	1.82 ± 2.05	1.58 ± 1.94
	Hard	6.10 ± 10.99	11.84 ± 27.70	6.81 ± 14.29
HumanoidMaze	Easy	4.88 ± 7.36	6.73 ± 9.44	11.45 ± 19.57
	Medium	6.53 ± 6.78	6.90 ± 10.13	12.23 ± 11.72
	Hard	20.05 ± 39.82	26.04 ± 55.74	52.58 ± 214.41

Hard tasks take longer to plan and show greater variation in planning time than Easy and Medium tasks in both domains. HumanoidMaze has longer average planning times than AntMaze at every difficulty and maze size, with the highest mean on Giant Hard (52.58 s).

Scene. Table I.2 summarizes planning time over all 120 Scene instances under the protocol in Appendix E.7. Semantic graph augmentation accounts for most of the computation, while product

search takes a median of 0.51 s. Appendix H.4 examines how predicate constraints and support size affect grounding time.

Table I.2: Scene task-time planning over all 120 instances. Times are in seconds; p95 denotes the 95th percentile.

Stage	Median	p95	Maximum
Semantic graph augmentation	39.61	54.07	55.34
Product search	0.51	1.75	1.99
Total planning	40.63	55.38	57.09

I.2 Offline graph and automaton sizes

Each navigation setting uses one offline graph for all 300 tasks, while each task formula is translated into its own Büchi automaton. We report their sizes separately to characterize the two components of product planning.

The 14 navigation graphs contain 944–7,950 nodes and 7,522–349,626 directed edges, with one connected component per graph (Table I.3). The table also reports estimated dataset sizes, compression ratios, and the number of original states retained per node for semantic evaluation. Compression compares the estimated number of dataset states with the number of graph nodes; retained support determines the states available for task-time predicate evaluation.

Table I.3: Frozen navigation graphs. Dataset scale is estimated as $\widehat{N}_{\mathcal{D}} = 1 + i_{\max}$, where $i_{\max}$ is the largest stored zero-based dataset index. Compression is $\widehat{N}_{\mathcal{D}}/|\mathcal{V}|$. Support entries give the median and 95th percentile of preserved states per node. All graphs have one connected component; each symmetric connection is counted as two directed edges.

Setting	Est. states	Nodes	Edges	Compression	Support
AntMaze (29D state)					
Giant-Navigate	1,000,000	978	7,522	1,022×	27 / 244.8
Giant-Stitch	1,000,000	2,080	35,900	481×	22 / 123.0
Large-Explore	5,000,000	2,514	34,390	1,989×	18.5 / 129.0
Large-Navigate	1,000,000	1,092	12,310	916×	22 / 196.9
Large-Stitch	1,000,000	1,563	29,166	640×	24 / 169.0
Medium-Explore	5,000,000	1,879	25,668	2,661×	17 / 193.1
Medium-Navigate	1,000,000	944	12,792	1,059×	21 / 237.1
Medium-Stitch	1,000,000	1,184	23,190	845×	25 / 257.4
HumanoidMaze (69D state)					
Giant-Navigate	4,000,000	5,982	198,296	669×	61 / 415.8
Giant-Stitch	4,000,000	7,950	349,626	503×	83 / 428.0
Large-Navigate	2,000,000	2,175	61,162	920×	28 / 133.3
Large-Stitch	2,000,000	3,185	138,408	628×	50 / 214.0
Medium-Navigate	2,000,000	1,771	58,180	1,129×	33 / 115.0
Medium-Stitch	2,000,000	2,190	107,818	913×	65 / 289.0

For the automata, we translate all 4,200 formulas with `ltl2ba` and count states and transitions before support-aware preprocessing. Each conjunctive term in a DNF guard is counted as a separate transition, matching the guard terms evaluated by the product successor. Table I.4 reports these sizes by task difficulty and across the full benchmark.

Table I.4: Automaton sizes before support-aware preprocessing for 4,200 formulas, with 1,400 per difficulty. Entries are median / p95 / maximum; transitions count guard-normalized DNF terms.

Difficulty	NBA states	DNF transitions
Easy	5 / 32 / 32	15 / 243 / 243
Medium	6 / 29 / 33	19 / 243 / 340
Hard	16 / 217 / 1,520	81 / 5,265 / 80,379
All	7 / 90 / 1,520	27 / 1,299 / 80,379

Hard tasks have larger median automata and a longer tail of large instances, while Easy and Medium tasks have similar sizes. The difficulty categories are defined by template composition and proposition count, with logical simplification performed during translation, so their automaton-size ranges overlap.

At task time, support-aware preprocessing removes unsupported guards and irrelevant states. The reachable product then depends on these retained automaton transitions, the semantic graph, and the initial state. The next section measures this task-specific product directly.

I.3 Product-graph sizes and reachability

To measure the search space induced by each task, we explicitly construct the reachable product for 120 fixed tasks. These come from AntMaze Giant-Stitch and Large-Explore, and HumanoidMaze Giant-Stitch and Medium-Stitch. Each setting contributes ten tasks at each of the three difficulty levels, giving 40 tasks per difficulty.

All counts use the task-specific semantic graph and the automaton after support-aware preprocessing. The Cartesian state count $|\mathcal{V}_\phi||\widetilde{Q}|$ pairs every semantic-graph node with every retained automaton state and gives an upper bound on product size. To obtain the reachable subgraph, we start from the initial product states and exhaustively follow the movement transitions allowed by Eq. (4). Initialization follows the navigation protocol in Appendix E.4. Accepting self-loops at a fixed waypoint are tested separately as dwell suffixes and are excluded from the movement-edge counts.

Table I.5: Product sizes over 40 tasks per difficulty. Entries are median / p95 / maximum. Cartesian states gives $|\mathcal{V}_\phi||\widetilde{Q}|$ for the augmented semantic graph and support-restricted automaton; reachable states and edges describe the subgraph reached from the initial product states.

Difficulty	Cartesian states	Reachable states	Reachable edges
Easy	12,630 / 83,251 / 159,240	12,600 / 81,659 / 127,392	437,414 / 2,211,939 / 5,618,768
Medium	16,318 / 74,047 / 159,240	14,227 / 72,031 / 103,507	415,586 / 3,173,433 / 4,557,727
Hard	33,534 / 262,726 / 414,336	23,178 / 256,878 / 378,726	805,823 / 11,338,466 / 18,863,198

Hard tasks produce larger reachable products in both state and edge counts, while Easy and Medium have similar sizes (Table I.5). For each task, we also divide its reachable-state count by its Cartesian state count. The median of this fraction is 99.90% for Easy, 77.89% for Medium, and 79.85% for Hard, showing how much of the Cartesian product remains reachable under the task's transition constraints.

The semantic graphs themselves have comparable sizes across the three groups: median/p95 node counts are $2362/7962$, $2361/7962$, and $2367/7965$ for Easy, Medium, and Hard. The corresponding support-restricted automata have $5/20$, $5/24$, and $9/90$ states. The Hard group thus combines similarly sized semantic graphs with larger automata, consistent with its larger reachable products.

I.4 Search strategies and candidate budget

We compare search strategies and prefix-candidate budgets on the 40 Hard tasks from Appendix I.3. For each task, all variants use the same semantic graph, support-restricted automaton, initial product states, edge costs, and dwell-suffix test. An exhaustive reference computes the shortest prefix and return suffix for every reachable accepting endpoint and selects the pair with the lowest weighted lasso cost. It finds an accepting lasso for 39 tasks and establishes infeasibility for the remaining task in the fixed finite product.

Search strategy. We compare explicit-product Dijkstra, on-demand Dijkstra, and on-demand A*, all with $K = 5$. Explicit-product Dijkstra constructs the entire start-reachable product before searching. The on-demand variants generate successors as needed, and their comparison measures the effect of adding the A* heuristic.

Table I.6: Controlled search comparison on 40 Hard tasks. Times cover product construction and search, in seconds.

Variant	Median (s)	p95 (s)
On-demand A*, $K = 5$	4.28	58.22
On-demand Dijkstra, $K = 5$	4.58	62.73
Explicit-product Dijkstra, $K = 5$	5.02	74.32
Exhaustive accepting-endpoint reference	5.33	77.59

On-demand Dijkstra reduces both median and p95 runtime relative to explicit-product Dijkstra, and on-demand A* reduces them further (Table I.6).

Candidate budget. The budget K controls how many accepting prefix endpoints receive a suffix search. We vary K in the on-demand A* planner and compare the resulting lasso costs and feasibility outcomes with the exhaustive reference.

Table I.7: Candidate-budget sensitivity for on-demand A*. Agreement is with the exhaustive reference: feasibility is evaluated on all 40 tasks, including the one infeasible task, and exact cost on the 39 feasible tasks.

K	Feasibility agreement	Exact-cost agreement	Median time (s)	p95 time (s)
1	37/40	34/39	2.72	24.92
3	40/40	37/39	3.53	36.84
5	40/40	39/39	4.28	58.22
10	40/40	39/39	4.31	108.93

At $K = 1$, the planner misses three feasible lassos. Increasing the budget to $K = 3$ finds a lasso for every feasible task, although two costs exceed their exact optima by at most 3.583%. Both $K = 5$ and $K = 10$ match the reference on feasibility for all 40 tasks and on graph cost for all 39 feasible tasks (Table I.7). The default $K = 5$ achieves this agreement with a lower p95 runtime than $K = 10$ (58.22 versus 108.93 s), balancing search effort and solution quality on this task set.

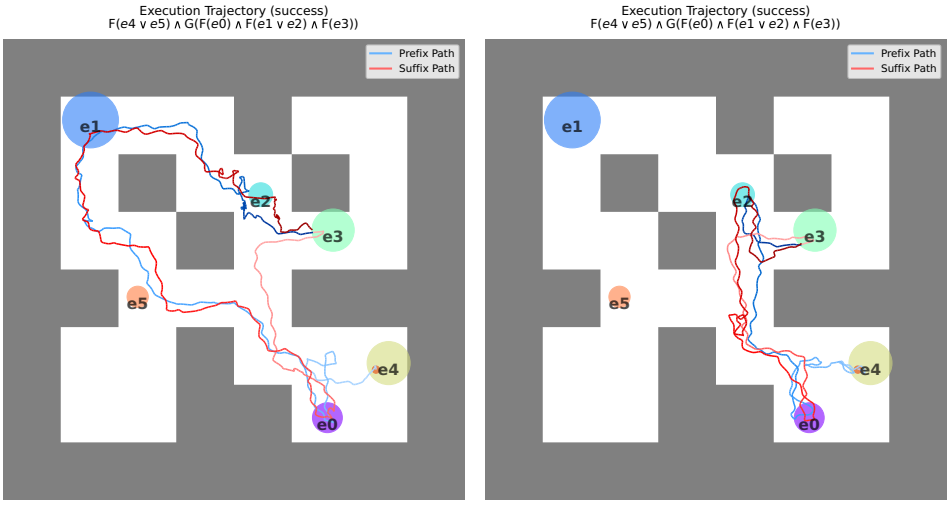
J Case studies

J.1 Product-space planning versus decoupled planning

The maze in Figure J.1 contains six labeled regions. The robot must visit either e_4 or e_5 at least once and repeatedly visit e_0, e_3, and either e_1 or e_2. This task is specified by

$$\phi_{\text{disj}} = \mathrm{F}(e_4 \vee e_5) \wedge \mathrm{G}\big(\mathrm{F}e_0 \wedge \mathrm{F}(e_1 \vee e_2) \wedge \mathrm{F}e_3\big). \tag{J.1}$$

The disjunctions allow several visiting patterns, whose costs depend on where the selected regions lie in the maze. SAGAS and LF-GAS use the same semantic graph and executor but choose these patterns differently. LF-GAS first selects an accepting automaton sequence with few logical transitions, then connects its witnesses by shortest graph paths. SAGAS selects the witnesses and visiting order jointly through cost-aware product search.



(a) LF-GAS (b) SAGAS

Figure J.1: Executions of the disjunctive task in Eq. (J.1). Blue and red curves show the prefix and suffix, respectively.

In Figure J.1, LF-GAS takes a detour through the upper-left region e_1. SAGAS instead uses e_2 for the recurrent disjunction, yielding a shorter route between the patrol targets. This example shows why the cost of reaching a region matters when choosing among logically valid alternatives.

J.2 Generalization across predicate geometries

The four maze tasks in Figure J.2 use new labeling functions with the same offline graph and frozen executor. Their regions vary in shape, size, and location, including circles, triangles, elongated strips, and non-convex polygons. Some regions are specified broadly enough to include walls or areas with sparse offline coverage. By selecting graph-connected anchors from predicate-satisfying offline states, SAGAS identifies data-supported targets within these regions, allowing users to specify predicates without precisely delineating their reachable portions. The tasks combine these geometries with the following temporal requirements:

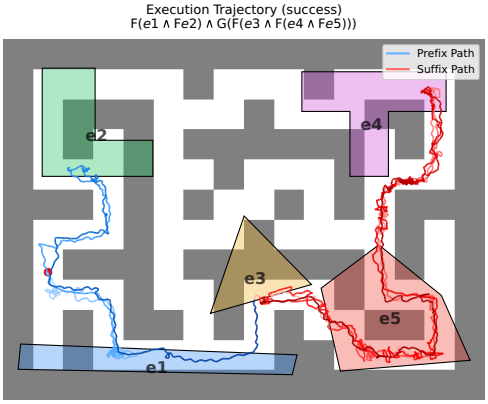
$$\begin{aligned}
\phi_{\text{rec}} &= \mathrm{F}(e_1 \wedge \mathrm{F}e_2) \wedge \mathrm{GF}(e_3 \wedge \mathrm{F}(e_4 \wedge \mathrm{F}e_5)),\\
\phi_{\text{neg}} &= \mathrm{G}\neg e_6 \wedge (\neg e_4 \,\mathrm{U}\, (e_1 \vee e_2)) \wedge \mathrm{F}e_4 \wedge \mathrm{G}(\mathrm{F}e_3 \wedge \mathrm{F}e_5),\\
\phi_{\text{mix}} &= \mathrm{G}\neg e_6 \wedge \mathrm{F}(e_1 \wedge \mathrm{F}(e_8 \wedge \mathrm{F}(e_2 \wedge \mathrm{F}e_7))) \wedge \mathrm{GF}(e_3 \wedge \mathrm{F}(e_4 \wedge \mathrm{F}e_5)),\\
\phi_{\text{mode}} &= (\neg e_1 \,\mathrm{U}\, (e_3 \vee e_4)) \wedge \mathrm{F}e_1 \wedge (\neg e_6 \,\mathrm{U}\, (e_7 \vee e_8))\\
&\quad \wedge \mathrm{F}e_6 \wedge \mathrm{GF}(e_2 \wedge \mathrm{F}e_5).
\end{aligned} \tag{J.2}$$

In Figure J.2(a), five polygonal regions define two ordered visits followed by a recurrent patrol. For ϕ_{rec}, the robot first visits the strip e_1 and then e_2 on the left of the maze. It then travels to the right-hand regions and follows the $e_3 \to e_4 \to e_5$ suffix loop.

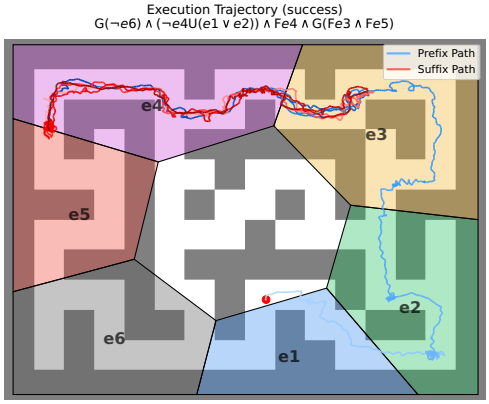
The six sectors in Figure J.2(b) surround an unlabeled starting area. Under ϕ_{neg}, e_6 is always forbidden, while entry into e_4 becomes permissible once e_1 or e_2 has been reached. The robot must also visit e_4 and repeatedly return to e_3 and e_5. The shown trajectory enters e_2 before e_4, then patrols between e_3 and e_5 while avoiding e_6.

Figure J.2(c) combines eight regions in a task with a longer ordered prefix. In the shown execution of ϕ_{mix}, the robot visits e_1, e_8, e_2, and e_7 in sequence before entering the $e_3 \to e_4 \to e_5$ patrol. The forbidden region e_6 must be avoided throughout both phases.

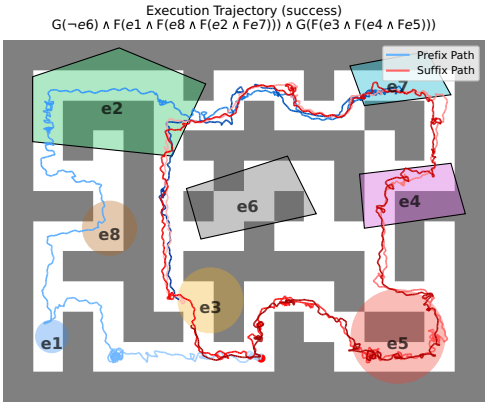
In Figure J.2(d), two elongated strips have independent entry conditions. The task ϕ_{mode} allows entry into the vertical strip e_1 once e_3 or e_4 has been visited, and into the horizontal strip e_6 once e_7 or e_8



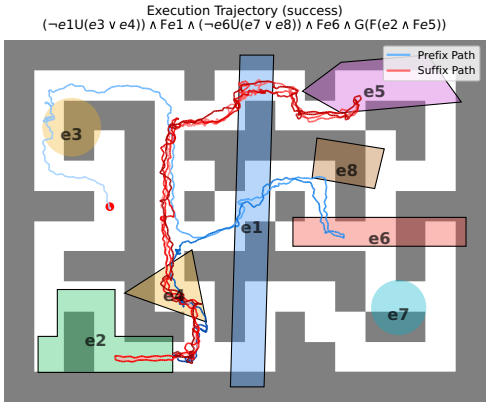
(a) Ordered visits followed by a three-region patrol (ϕ_{rec}).



(b) Conditional entry into e_4 and global avoidance of e_6 (ϕ_{neg}).



(c) A four-stage prefix and recurrent patrol under an avoidance constraint (ϕ_{mix}).



(d) Two conditionally accessible strips and a two-region patrol (ϕ_{mode}).

Figure J.2: Executions with heterogeneous predicate geometries for the four formulas in Eq. (J.2). Blue and red curves show the prefix and suffix, respectively. Each execution satisfies the finite-lasso criterion.

has been visited. The trajectory meets both conditions, visits the strips, and then patrols between e_2 and e_5.

Across these cases, SAGAS satisfies different visiting orders and avoidance requirements over new predicate geometries using the same learned graph and executor.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the offline, model-free, zero-shot LTL planning setting and the reusable graph, semantic augmentation, and product-search contributions. Appendix D states the main scope and conditional guarantee boundaries.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Appendix D discusses limitations from offline data support, learned reachability, negative constraints, recurrent evaluation, predicate scope, and deployment risk.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: Proposition 1 states conditional LTL satisfaction with a proof sketch in the main text; Appendix C defines execution realization and gives the complete soundness and satisfaction proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Section 4.1 and Appendix E describe the domains, task generation, baselines, metrics, evaluation protocol, and implementation parameters used for the main experiments.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The experiments use publicly available OGBench datasets. Code and reproduction instructions will be made publicly available through the project page: `https://cps-sjtu.github.io/SAGAS/`.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The main text summarizes the experimental protocol, while Appendices E, F, and E.5 provide task-generation, evaluation, platform, full-result, and parameter details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Table 1 identifies cross-setting standard deviations. Appendix F.2 reports paired-bootstrap confidence intervals, exact McNemar tests with Holm correction, and cell-level Wilson intervals; the additional suites report Wilson intervals for finite-lasso success.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix E describes the compute platform and Appendix I.1 reports planning-time statistics.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work uses public simulation benchmarks and does not involve human subjects, private data, or scraped personal content.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

Justification: The work studies simulated robotics planning benchmarks and does not directly involve social deployment or human-subject impact.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release high-risk pretrained models, image generators, or scraped datasets.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper credits the OGBench benchmark and prior methods used for comparison or implementation through citations, and the experiments use their publicly released assets according to their terms.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.

- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The submission included the navigation implementation and reproduction documentation. The appendices document the additional predicate suites and statistical analyses; these materials will be consolidated in the public release.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

Justification: The paper does not involve human subjects, so IRB approval or equivalent review is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs are not used as an important, original, or non-standard component of the core method.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.