SDTA-MDP: Symbolic Distributed Time Aggregation for Continuous Control

Anonymous authors
Paper under double-blind review

Abstract

Existing methods address continuous-state Markov decision processes (MDPs) with continuous actions, but balancing control performance, computational cost, and interpretability remains challenging. We propose Symbolic Distributed Time-Aggregated MDP solving (SDTA-MDP). Building on symbolic state partitioning, the framework combines time aggregation over semantic partitions with local control for finite and bounded continuous actions. The topology groups states by feasible program behavior and summarizes block interiors through absorption cost, duration, and successor probabilities. SDTA-MDP solves a frontier semi-Markov decision process with time aggregation and passes the resulting block actions to local controllers for action selection. We establish path signature preservation and bound the effect of absorption estimation error on the frontier Bellman update. Experiments show that SDTA-MDP achieves higher success rates and lower costs than the evaluated baselines on selected stochastic tasks. For environments supplying executable dynamics and branch predicates, the framework organizes continuous states into interpretable symbolic blocks and summarizes their temporal behavior for control.

1 Introduction

Many continuous state control systems combine continuous dynamics with guarded discrete modes that encode operating conditions and changes in dynamics (Henzinger, 1996; Bemporad & Morari, 1999). Standard controllers based on a dynamics model, such as MPC, predict trajectories and optimize costs over a finite horizon (Mayne et al., 2000); unless logic is modeled explicitly, program branches are not represented in the optimization (Bemporad & Morari, 1999). If this logic is retained in the control model, states with the same program behavior need not be treated separately, which can reduce model size while preserving interpretability (Abel et al., 2016; Bastani et al., 2018). We ask how executable environment semantics can guide state abstraction and how the resulting abstraction can support time aggregation for control.

Grid discretization replaces a continuous state space with finitely many representative states or cells, making dynamic programming tractable on the resulting finite model (Whitt, 1978). Tile coding instead represents continuous states with a finite collection of overlapping coarse coding features (Sutton, 1996). Both representations can grow with resolution even where states follow the same program behavior. Learned neural policies avoid explicit grids but generally do not expose decision rules that people can readily understand or verify (Verma et al., 2018; Bastani et al., 2018). Manually supplied state space decompositions can reduce a global MDP to smaller local problems, but they require a predefined partition into regions (Dean & Lin, 1995). In the DTPTA formulation, this supplied structure also specifies boundaries and remote states (Alexandre et al., 2025). State abstraction theory explains when compression can preserve decision structure (Puterman, 1994; Li et al., 2006). Recent work extends this foundation to approximate abstractions and learned representations based on reward and transition behavior (Abel et al., 2016; Gelada et al., 2019; Zhang et al., 2021). Symbolic execution and SMT solving expose feasible program paths (King, 1976; de Moura & Bjørner, 2008; Baldoni et al., 2018). SymPar provides the closest starting point. It uses symbolic execution to obtain path conditions for each action and SMT checks to construct

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

semantic state partitions for tabular reinforcement learning (Ghaffari et al., 2025). SymPar uses these partitions for tabular learning but does not organize them into a control model over time. DTPTA instead applies time aggregation to a decomposition chosen from problem structure and modeling judgment (Alexandre et al., 2025). SDTA-MDP connects these ideas by using semantic partitions to organize time aggregation and local control.

We introduce Symbolic Distributed Time-Aggregated MDP solving (SDTA-MDP).[1] Following SymPar, we treat supported disturbance realizations as symbolic execution alternatives (Ghaffari et al., 2025). SDTA-MDP then projects the disturbance by merging alternatives that induce the same program path, yielding state dependent possible path predicates for topology construction. Because several paths may then be possible at the same state, it forms disjoint blocks whose states share the same path signature. For tasks with piecewise affine successor fragments, exact-LRA enumerates the admissible successor blocks for each combination of source block and action before sampling. The absorption rollouts then estimate cost, duration, and probability only over this support. When exact-LRA is unavailable, those same rollouts identify empirical support from their observed first exits. The resulting quantities define a finite frontier SMDP whose actions at the block level guide local control over a short horizon.

The method uses different topology constructions for finite and bounded continuous actions. For finite actions, the topology covers the complete intended action set, so all states within a block have the same path signature under every available action. This reduces to a common program path when execution is deterministic. For bounded continuous actions, finite representatives of both intended actions and execution noise define the topology, while execution draws on adaptive semantic candidates. When a task provides a supported linear safety formula, the selected action is projected exactly onto the safe set defined by SMT constraints. Thus the finite action core gives the cleanest semantic guarantee, and the continuous action route gives a practical extension through representative actions. The framework organizes declared program predicates into a symbolic model and combines temporal summaries of block interiors with local control.

This work makes the following four contributions to symbolic continuous state control:

- We construct a branch predicate topology that preserves path signatures for finite action MDPs and extend it to bounded continuous actions through representative actions.
- We formulate time aggregation over symbolic blocks by summarizing their internal evolution through absorption costs, durations, and successor probabilities, yielding a frontier SMDP for block-level control.
- We connect frontier actions to continuous control over a short horizon using adaptive semantic candidates and exact SMT projection for supported linear safety formulas.
- We evaluate SDTA-MDP under matched stochastic action execution, reporting control quality, safety violations, model calls, running time, partition size, confidence intervals, and scaling behavior.

A more detailed review of related literature is provided in Appendix A.

2 Method

SDTA-MDP receives an MDP with a bounded continuous state space, an executable program for transitions and costs, and a symbolic predicate interface. It constructs a topology from the predicates, solves a frontier SMDP, and returns an executable policy with diagnostic information.

[1] Code and experiment scripts are publicly available at https://anonymous.4open.science/r/SDTA-MDP-anonymous-F334/.

2.1 Setting and Overview

We consider an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, f)$, where $\mathcal{M}$ denotes the complete control problem. The bounded set $\mathcal{S} \subseteq \mathbb{R}^d$ is the state space, and $f(s, a)$ is the stage cost. We write $\mathcal{A} = \mathcal{A}_f$ for a finite action set and $\mathcal{A} = \mathcal{A}_c$ for a bounded continuous action set. Let ξ denote the execution disturbance. For a fixed realization of ξ, the successor state is $T(s, a, \xi)$. The distribution of ξ induces the transition kernel $P(\cdot \mid s, a)$.

For an intended action a, let Ξ_a be the declared support of ξ. One execution of the transition program may encounter a sequence of branch outcomes, which we call a program path. Let $\psi_i(s, a, \xi)$ be the raw formula for the i-th path produced by symbolic execution. Following SymPar, we treat supported disturbance realizations as symbolic execution alternatives. SDTA-MDP then projects the disturbance by merging alternatives that induce the same program path, yielding a possible path predicate for each program path:

$$\phi_i^a(s) \equiv \exists \xi : \big(\xi \in \Xi_a \wedge \psi_i(s, a, \xi)\big). \tag{1}$$

Here, $\phi_i^a(s)$ states whether path i can occur under at least one disturbance in Ξ_a. We write $\mathcal{C}^a$ for the set of all such possible path predicates associated with intended action a. The formulas in $\mathcal{C}^a$ may overlap because different disturbances can lead to different paths at the same state. Finite action slip permits exact enumeration of the executed action support. For continuous Gaussian noise, the main experiments approximate the unbounded support with finitely many executed actions in a clipped interval. Equation 1 records path possibility rather than probability; the absorption rollouts estimate transition probabilities later.

For finite actions, the representative set contains every available action, so $\widehat{\mathcal{A}}_{\text{rep}} = \mathcal{A}_f$. For bounded continuous actions, $\widehat{\mathcal{A}}_{\text{rep}} \subset \mathcal{A}_c$ is a finite set used only to construct the symbolic topology. During execution, the controller generates a state dependent local candidate set around the block action and selects the executed action from that set.

The topology construction produces symbolic blocks B_ℓ. For each pair consisting of a block and an action, the method obtains support over successor blocks in one of two ways. When available, the exact-LRA backend returns the admissible support F_{LRA} before sampling. Otherwise, the absorption rollouts record the targets reached at their first exit. The same rollouts estimate the absorption cost h_f, absorption time h_1, and distribution $\widetilde{p}$ over successor blocks. Their support forms the directed frontier graph F. The frontier calculation returns a scalar cost rate η, block values V_F, and block actions. In an exact solution of the average-cost Bellman equation, V_F represents differential cost relative to η. Appendix B.2 states sufficient conditions for such a solution to exist. The predicate RemoteBlock(B_ℓ) identifies blocks assigned an edge or remote role and handled by a safety surrogate.

Algorithm 1 presents the complete procedure, while Figure 1 illustrates its workflow with five stages.

The exact-LRA procedure returns F_{LRA} when its assumptions hold and unavailable otherwise. In the exact route, $F = F_{\text{LRA}}$ is fixed before absorption estimation, and supported but unobserved edges may retain zero empirical probability. In the sampled route, F is assembled from the distinct first exits recorded by the same rollouts used to estimate the indexed absorption quantities.

2.2 Semantic Topology and Frontier Support

For a state s and a topology action a, define the per-action path signature $\Sigma_a(s) = \{i \mid \phi_i^a(s)\}$ as the set of program paths that can occur at s. The complete path signature $\Sigma(s) = (\Sigma_a(s))_{a \in \widehat{\mathcal{A}}_{\text{rep}}}$ collects these sets over all representative actions. The projected formulas may overlap, so the implementation first splits the state space on each formula and then joins cells with the same path signature. Each state is characterized by its complete path signature. We retain every signature realized by at least one state and index the resulting signatures as $\sigma_1, \ldots, \sigma_L$. Each σ_ℓ defines the symbolic block $B_\ell = \{s \in \mathcal{S} \mid \Sigma(s) = \sigma_\ell\}$, and these blocks form the partition $\mathcal{P} = \{B_\ell\}_{\ell=1}^{L}$. Here and below, ℓ and m index arbitrary blocks in $\mathcal{P}$, and σ_ℓ is the path signature associated with B_ℓ. The corresponding SMT model returns a satisfying assignment to the state variables, denoted by w_ℓ. This witness certifies

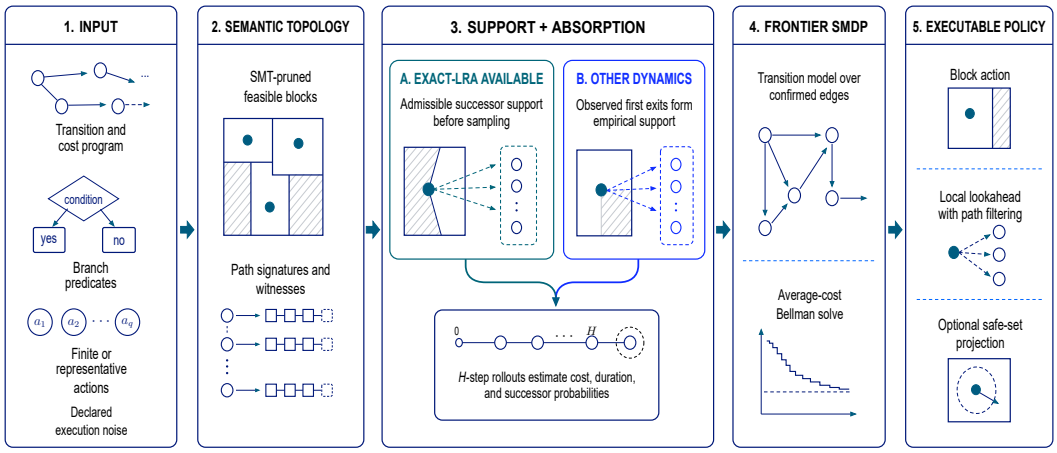


Figure 1: The five stages of SDTA-MDP. Declared execution noise enters the semantic topology through path signatures; successor support and absorption quantities then define the frontier SMDP used by local control.

Algorithm 1: SDTA-MDP

Input: $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, f)$; raw path formulas $\{\psi_i(s, a, \xi)\}$ and disturbance supports $\{\Xi_a\}$; horizon H, rollouts N, lookahead depth K

Output: π: executable policy; $\mathcal{P}, F, V_F, \eta$: partition, frontier graph, frontier values, and average cost

$\widehat{\mathcal{A}}_{\text{rep}} \leftarrow \text{Procedure(RepresentativeActions)}(\mathcal{A})$;

$\{\mathcal{C}^a\} \leftarrow \text{Procedure(ProjectPathSupport)}(\{\psi_i\}, \{\Xi_a\}, \widehat{\mathcal{A}}_{\text{rep}})$;

$\mathcal{P} \leftarrow \text{Procedure(AtomizeAndIntersect)}(\{\mathcal{C}^a\})$;

$F_{\text{LRA}} \leftarrow \text{Procedure(ExactLRASupport)}(\mathcal{P}, \widehat{\mathcal{A}}_{\text{rep}}, P)$;

if $F_{\text{LRA}} = \text{unavailable}$ then
  $F \leftarrow \varnothing$;
else
  $F \leftarrow F_{\text{LRA}}$;

foreach $B_\ell \in \mathcal{P}$ do
  if RemoteBlock(B_ℓ) then
    Procedure(AssignSafetySurrogate)(B_ℓ);
  else
    foreach $a \in \widehat{\mathcal{A}}_{\text{rep}}$ do
      $(\widehat{h}_f(B_\ell, a), \widehat{h}_1(B_\ell, a), \widehat{p}(\cdot \mid B_\ell, a)) \leftarrow$ Procedure(EstimateFirstExitAbsorption)($B_\ell, a, H, N, F_{\text{LRA}}$);
      if $F_{\text{LRA}} = \text{unavailable}$ then
        $F \leftarrow F \cup \text{Procedure(ObservedFirstExits)}(B_\ell, a)$;

$\eta, V_F, a_F \leftarrow \text{Procedure(SolveFrontierSMDP)}(F, \widehat{h}_f, \widehat{h}_1, \widehat{p})$;

$\pi \leftarrow \text{Procedure(BuildSupportAwareLocalControllers)}(a_F, \mathcal{P}, K)$;

return $\pi, \mathcal{P}, F, V_F, \eta$;

that the block is nonempty and provides a concrete state for later role assignment; it is not assumed to summarize all states in the block.

Definition 1 (Symbolic block defined by a path signature). Two states are equivalent with respect to the path signature when they have the same $\Sigma(s)$. The nonempty equivalence classes form the symbolic blocks.

This grouping preserves possible program paths, but does not require equal transition probabilities, costs, or values within a block. The absorption model summarizes the dynamics through a reference initial-state distribution for each block, as specified in Section 2.3.

For finite actions, $\widehat{\mathcal{A}}_{\text{rep}} = \mathcal{A}_f$, so the topology covers every intended action. For continuous actions, the partition is constructed only from $\widehat{\mathcal{A}}_{\text{rep}}$, and local search selects the final action. Proposition 1 proves path signature preservation over all finite actions and over the declared representative actions in the continuous construction.

For any state s, let $B(s) \in \mathcal{P}$ denote the unique symbolic block containing s. Given a source block B_ℓ and block action a, let $\tau_\ell^{\text{exit}} = \inf\{t \geq 1 \mid B(s_t) \neq B_\ell\}$ be the first time the trajectory leaves B_ℓ. We define F_H as the set of action-labeled edges representing first exits from a source block within H steps. An edge (B_ℓ, a, B_m) belongs to F_H if, under fixed intended action a, the trajectory first leaves B_ℓ and enters B_m within H steps with positive probability. The frontier relation reachable within H steps is the support of these first exits:

$$F_H = \left\{ (B_\ell, a, B_m) \;\middle|\; B_\ell \neq B_m,\ \Pr\Big(B(s_{\tau_\ell^{\text{exit}}}) = B_m,\ \tau_\ell^{\text{exit}} \leq H \mid s_0 \in B_\ell, a\Big) > 0 \right\}. \tag{2}$$

Figure 4 illustrates the path-signature blocks and directed first-exit support defined above; the support is obtained exactly by symbolic analysis or empirically from absorption rollouts.

When the transition program exposes piecewise affine successor branches, exact-LRA constructs an admissible support F_{LRA} through satisfiability checks in linear real arithmetic without quantifiers (QF-LRA). This step is separate from the disturbance projection in Equation 1: disturbance projection defines the blocks, whereas exact-LRA determines which successor blocks an affine transition may reach. For branch r, $g_r(s)$ is its guard and the successor state is

$$T_{a,r}(s) = A_{a,r}s + b_{a,r}, \tag{3}$$

where $A_{a,r}$ describes the linear state dependence and $b_{a,r}$ is the affine offset. For every source block, action, affine branch, and target block, exact-LRA asks whether the following condition for crossing between blocks is satisfiable:

$$F_{\text{LRA}} = \{(B_\ell, a, B_m) \mid B_\ell \neq B_m,\ \exists r, s:\ s \in B_\ell,\ g_r(s),\ A_{a,r}s + b_{a,r} \in B_m\}. \tag{4}$$

Under these assumptions, Proposition 3 establishes that F_{LRA} contains exactly the feasible declared affine crossings and that every realized first exit satisfies $F_H \subseteq F_{\text{LRA}}$. The backend therefore passes F_{LRA} to absorption estimation as strict successor support: observed exits outside it are treated as a model inconsistency, while supported but unobserved edges retain zero empirical probability.

When exact-LRA is unavailable, SDTA-MDP constructs an empirical frontier graph, denoted by $\widehat{F}_H$, from the absorption rollouts. Each absorption rollout is limited to H steps and continues until it leaves the source block, the task terminates, or the horizon is reached. Whenever a rollout first reaches a target different from the source block, the corresponding edge is added to $\widehat{F}_H$. The same observations are used to estimate the distribution $\widetilde{p}$ over successor blocks. Thus $\widehat{F}_H \subseteq F_H$ contains the observed support; finite N may miss rare exits.

2.3 Absorption SMDP and Local Control

The frontier SMDP follows the DTPTA principle of time aggregation (Alexandre et al., 2025). It summarizes an interior excursion by its cost, duration, and distribution over the next frontier. Let s_t denote the state at time t, let τ_{term} be the task termination time, and define the stopping time $\tau_H = \min(\tau_\ell^{\text{exit}}, \tau_{\text{term}}, H)$. For each symbolic block B_ℓ, let μ_ℓ denote the reference distribution used to initialize absorption rollouts within the block. Each rollout starts from $s_0 \sim \mu_\ell$ and holds the intended action a fixed until τ_H. The three

absorption quantities average over both the initial state and the stochastic transitions:

$$h_f(B_\ell, a) = \mathbb{E}_{s_0 \sim \mu_\ell}\left[\sum_{t=0}^{\tau_H - 1} f(s_t, a)\right], \tag{5}$$

$$h_1(B_\ell, a) = \mathbb{E}_{s_0 \sim \mu_\ell}[\tau_H], \tag{6}$$

$$\widetilde{p}(B_m \mid B_\ell, a) = \Pr_{s_0 \sim \mu_\ell}\left(B(s_{\tau_H}) = B_m\right). \tag{7}$$

The implementation draws candidate states uniformly from the block's sampling box and accepts those satisfying its predicate; if no candidate is accepted within the sampling budget, it uses the SMT witness. Equation 5 is the cumulative cost at the stopping time. Equation 6 is the holding time of the macro transition. Equation 7 is the distribution over terminal blocks at the stopping time. Under exact-LRA, the support of $\widetilde{p}$ for transitions to other blocks is restricted to F_{LRA}. Under sampled fallback, the observed transitions to other blocks define $\widehat{F}_H$. Appendix C gives the DTPTA source form and the variable substitution leading to these equations.

With the reference distributions μ_ℓ fixed, the absorption quantities define a finite frontier model on the symbolic blocks. Each abstract transition summarizes an excursion initialized from the reference distribution of its source block. For the fixed reference frontier model, the average-cost optimality equation takes the following form when a common optimal gain and finite relative values exist:

$$V_F(B_\ell) = \min_a \left[h_f(B_\ell, a) - \eta h_1(B_\ell, a) + \sum_{B_m} \widetilde{p}(B_m \mid B_\ell, a) V_F(B_m)\right]. \tag{8}$$

Proposition 2 derives the Bellman expression from the absorption quantities of the reference model. Theorem 1 bounds the empirical absorption errors and their resulting perturbation of this Bellman update. For a concrete state s, a stopped-cost representation connects its initial state to the prescribed frontier values. Here, a_t denotes the action taken at time t under a fixed local policy:

$$V(s) = \mathbb{E}\left[\sum_{t=0}^{\tau_H - 1} \left(f(s_t, a_t) - \eta\right) + V_F(B(s_{\tau_H})) \,\middle|\, s_0 = s\right]. \tag{9}$$

Equation 9 provides an interpretation of time aggregation; the implemented controller uses the local objective in Equation 13. Using numerical estimates of η and V_F, the implementation selects a block action based on the following minimization:

$$a_F(B_\ell) \in \arg\min_a \left[h_f(B_\ell, a) - \eta h_1(B_\ell, a) + \sum_{B_m} \widetilde{p}(B_m \mid B_\ell, a) V_F(B_m)\right]. \tag{10}$$

The frontier solver passes $a_F(B_\ell)$, rather than V_F, to the local controller. For continuous actions, the incumbent anchors the candidate refinements in Equation 11. For finite actions, the candidate set contains all available actions and is independent of the incumbent.

For continuous execution, let a^- denote the previously applied action; when local search is initialized in block B_ℓ, its value is the incumbent $a_F(B_\ell)$. The local candidate set combines default actions, critical actions supplied by the environment, semantic actions that depend on the state, and refinements around a^-:

$$\widehat{\mathcal{A}}_{\mathrm{loc}}(s; a^-) = \Pi_{\mathcal{A}}\left(\mathcal{A}_{\mathrm{base}}(a^-) \cup \mathcal{A}_{\mathrm{crit}}(s)\right). \tag{11}$$

Here, $\mathcal{A}_{\mathrm{base}}(a^-)$ provides default and refinement actions around a^-, while $\mathcal{A}_{\mathrm{crit}}(s)$ provides semantic actions for state s. The operator $\Pi_{\mathcal{A}}$ maps all candidates into the allowed action set $\mathcal{A}$. If the environment declares unsafe program paths, the controller removes candidates whose nearest topology action has a path signature containing any declared unsafe path. The filter is applied only when at least one candidate remains; otherwise the original candidate set is retained. Denote the resulting set by $\widehat{\mathcal{A}}_{\mathrm{eff}}(s; a^-)$.

When the environment supplies a safety formula, SDTA-MDP projects it onto the action coordinate to obtain the feasible interval

$$\mathcal{A}_{\text{safe}}(s) = \{a \in [\underline{a}, \overline{a}] \mid \phi_{\text{safe}}(s, a)\}. \tag{12}$$

Here, $\phi_{\text{safe}}(s, a)$ is the declared safety formula, and $[\underline{a}, \overline{a}]$ gives the action bounds. If no supported safety formula is supplied, we set $\mathcal{A}_{\text{safe}}(s) = \mathcal{A}$.

SDTA-MDP adapts the standard objective used in receding horizon control over a finite horizon (Mayne et al., 2000; Rawlings et al., 2017). For a current state s and initial action a, $J_K(s, a)$ measures the total cost over the next K steps. The calculation applies a first and then chooses the remaining actions optimally from the local candidate sets, with a violation penalty and task terminal potential:

$$J_K(s, a) = f(s, a) + \lambda \mathbf{1}_{\text{viol}}(s, a) + \min_{\substack{a_t \in \widehat{\mathcal{A}}_{\text{loc}}(s_t; a_{t-1}) \\ t=1,\dots,K-1}} \left[\sum_{t=1}^{K-1} \left(f(s_t, a_t) + \lambda \mathbf{1}_{\text{viol}}(s_t, a_t) \right) + \rho \Phi(s_K) \right], \tag{13}$$

Here, $\mathbf{1}_{\text{viol}}(s, a)$ is the violation indicator and λ is its penalty weight. Φ distinguishes concrete states within a block. It is a heuristic score predefined in the task code, rather than a value obtained from the frontier solver. Here, ρ is its weight, and K is the lookahead depth introduced in Algorithm 1. Appendix D.1 specifies how execution noise is treated during local scoring. For $s \in B_\ell$, the executable policy is

$$\pi(s) = \arg \min_{a \in \widehat{\mathcal{A}}_{\text{eff}}(s; a_F(B_\ell)) \cap \mathcal{A}_{\text{safe}}(s)} J_K(s, a). \tag{14}$$

2.4 Safety Projection and Remote Blocks

The intersection with $\mathcal{A}_{\text{safe}}(s)$ in Equation 14 restricts local lookahead to the declared feasible interval. This projection onto the action space is separate from the construction of successor support by exact-LRA. The projection restricts the action chosen at the current state, whereas exact-LRA restricts the successor blocks admitted during absorption estimation. Both operations are separate from the disturbance projection in Equation 1, which determines possible program paths before the blocks are constructed. For example, a vehicle traveling at velocity v brakes with a constant acceleration $a < 0$ and must stop within a time window of length $\Delta t = 2$. The corresponding safe action set is $\mathcal{A}_{\text{safe}}(v) = \{a \in [-10, -0.001] \mid v \leq -2a\}$.

Finally, the environment supplies a role mapping R_{role} that evaluates the witness w_ℓ and path signature σ_ℓ defined above. The implementation identifies edge and remote roles with the single predicate

$$\text{RemoteBlock}(B_\ell) = \mathbf{1}\{R_{\text{role}}(w_\ell, \sigma_\ell) \in \{\textsf{edge}, \textsf{remote}\}\}. \tag{15}$$

Here, $\mathbf{1}\{\cdot\}$ is the indicator function. Such blocks receive a fixed safety policy or surrogate outcome instead of a full absorption solve, and diagnostics report which blocks were truncated. Appendix B states and proves the structural and finite sample properties used by the method.

3 Experiments

We evaluate the control quality and computational requirements of combining semantic topology, time aggregation, and local control under stochastic execution. MPC provides a planning reference, while SymPar+Q evaluates tabular learning on symbolic partitions. For continuous actions, we additionally compare with SMoSE, which learns an interpretable policy through shallow experts and a router (Vincze et al., 2025), and SAC, a neural policy baseline based on maximum entropy reinforcement learning (Haarnoja et al., 2018). Experimental settings and implementation details are provided in Appendix D.

3.1 Main Results: Stochastic Control

Tables 1 and 2 report mean cost and success over ten seeds and twenty evaluation episodes per seed. Within each task and seed, all methods use the same initial states and execution disturbances. SMoSE and SAC are each trained for 100,000 environment interactions per seed and evaluated using their final policies. Appendix E reports complete results, including violation rates, confidence intervals, and resource measurements.

Comparison with MPC and SymPar+Q. SDTA-MDP matches or exceeds the MPC success rate on five of seven tasks and uses fewer model evaluations on three. This comparison evaluates the practical control quality and model use of the symbolic pipeline. On all three finite action tasks, SDTA-MDP has lower cost and fewer model evaluations than the evaluated SymPar+Q configuration. Success increases from 0.880 to 1.000 on PointMass and from 0.475 to 0.930 on DoubleIntegrator, while BrakingCar success is slightly lower (0.835 versus 0.845). These are comparisons of complete controllers combining abstraction and control. On PointMass, the SDTA-MDP violation rate is 0.100, compared with 0.075 for SymPar+Q and 0.040 for MPC.

Bounded continuous actions. SDTA-MDP retains an explicit symbolic representation of program behavior while achieving unit observed success on Pendulum, Continuous PointMass, and Continuous DoubleIntegrator. On Pendulum and Continuous PointMass, it obtains lower mean cost and higher success than SMoSE. On Pendulum, SDTA-MDP achieves higher success than SAC (1.000 versus 0.945), with a slightly higher mean cost (48.585 versus 47.355). These results illustrate effective continuous control through interpretable state organization, time aggregation, and local refinement. Figures 2 and 3 compare cost and success for finite and continuous actions, respectively: bars show means over ten seeds, and error bars indicate bootstrap 95% confidence intervals computed over seeds.

3.2 Local Control Analysis

Local lookahead. We compare the complete method with a variant that removes local lookahead. Removing lookahead reduces success to 0.120 on Pendulum and 0.150 on Continuous DoubleIntegrator, while leaving ContinuousBrakingCar and Continuous PointMass essentially unchanged (Appendix Table 11). Thus, state dependent short horizon refinement is critical when the frontier action alone does not determine a successful trajectory, but it need not change tasks already resolved by the incumbent block action.

3.3 Topology Compactness

Partition structure. Appendix Table 7 summarizes the topologies obtained in the main experiments. At the main resolution, six tasks contain 5–26 symbolic blocks, compared with 64 FineGrid cells. Continuous PointMass is the exception: preserving its noise dependent path signatures produces 135 blocks and 3964.9 empirical frontier edges, whereas the other tasks contain 26.9–294.7 edges. The topology groups states by program behavior; its size depends on the semantic complexity of the task.

Grid resolution. SDTA-MDP used 5–135 symbolic blocks, compared with 64–4,096 grid states across the tested resolutions. On Pendulum, five blocks achieved unit observed success at a lower cost than the 4,096-state grid. Grouping states by program behavior can thus retain useful control information without representing each geometric subdivision separately (Appendix Tables 9 and 10).

Computational tradeoff. Complete counts include preparation and online predictions, excluding actual evaluation transitions. SDTA-MDP uses fewer evaluations than MPC on DoubleIntegrator, Pendulum, and Continuous PointMass, but more on the other four tasks. Appendix B analyzes absorption rollout complexity; Appendix E reports full results and diagnostics, with running times in Table 6.

Table 1: Stochastic control with finite actions.

	SDTA-MDP		MPC		SymPar+Q	
Benchmark	Cost	Succ.	Cost	Succ.	Cost	Succ.
BrakingCar	20.689	0.835	19.117	0.850	21.968	0.845
PointMass	14.730	1.000	13.936	1.000	24.034	0.880
DoubleIntegrator	4.920	0.930	6.809	0.885	91.855	0.475

Table 2: Stochastic control with continuous actions.

	SDTA-MDP		MPC		SMoSE		SAC	
Benchmark	Cost	Succ.	Cost	Succ.	Cost	Succ.	Cost	Succ.
Cont. BrakingCar	13.952	0.905	13.884	0.910	14.160	0.910	12.662	0.915
Pendulum	48.585	1.000	47.965	1.000	49.037	0.995	47.355	0.945
Cont. PointMass	9.950	1.000	9.692	1.000	22.504	0.495	8.391	1.000
Cont. DoubleIntegrator	3.714	1.000	4.231	1.000	3.652	1.000	3.606	1.000

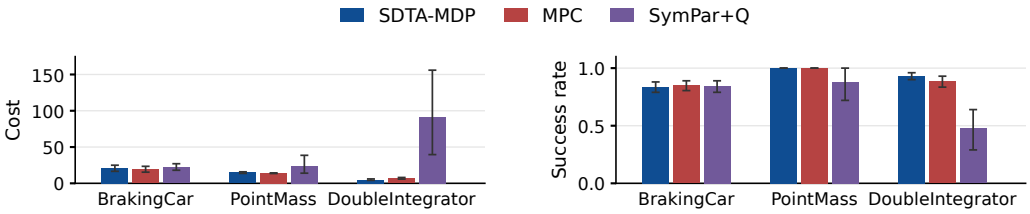


Figure 2: Control performance on finite action tasks.

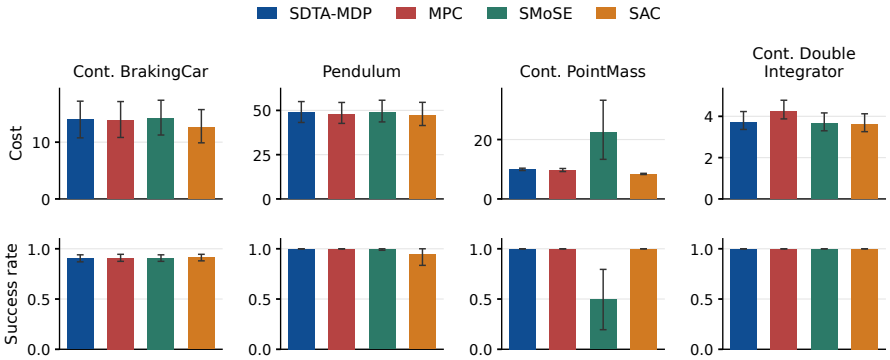


Figure 3: Control performance on continuous action tasks.

4 Conclusion

SDTA-MDP organizes program behavior in continuous state MDPs into an interpretable symbolic topology. Time aggregation summarizes within block trajectories through absorption cost, duration, and successor distributions, forming a reusable block decision model. Local control adapts block decisions to concrete states, supporting finite and bounded continuous actions without uniform state discretization. Under the stated assumptions, the analysis establishes path signature preservation and successor support correctness, and bounds the effect of finite sampling on the frontier Bellman update. Experiments show competitive control performance on both finite and bounded continuous action tasks. On selected tasks, SDTA-MDP achieves favorable tradeoffs between control quality and the number of model evaluations.

Reproducibility Statement

Appendix B states the assumptions and proofs supporting the theoretical results. Appendix D specifies the benchmarks, stochastic execution models, baselines, implementation settings, and evaluation protocol, while Appendix E reports complete results and diagnostic analyses.

References

David Abel, David Hershkowitz, and Michael L. Littman. Near optimal behavior via approximate state abstraction. In Proceedings of the 33rd International Conference on Machine Learning, volume 48 of Proceedings of Machine Learning Research, pp. 2915–2923. PMLR, 2016. URL https://proceedings.mlr.press/v48/abel16.html.

Rodrigo e Alvim Alexandre, Marcelo D. Fragoso, Virgilio J. M. Ferreira Filho, and Edilson F. Arruda. Solving markov decision processes via state space decomposition and time aggregation. European Journal of Operational Research, 324(1):155–167, 2025. doi: 10.1016/j.ejor.2025.01.037. URL https://www.sciencedirect.com/science/article/pii/S0377221725000803.

Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, pp. 2669–2678. AAAI Press, 2018. doi: 10.1609/aaai.v32i1.11797. URL https://ojs.aaai.org/index.php/AAAI/article/view/11797.

Aaron D. Ames, Xiangru Xu, Jessy W. Grizzle, and Paulo Tabuada. Control barrier function based quadratic programs for safety critical systems. IEEE Transactions on Automatic Control, 62(8):3861–3876, 2017. doi: 10.1109/TAC.2016.2638961. URL https://doi.org/10.1109/TAC.2016.2638961.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, pp. 1726–1734. AAAI Press, 2017. doi: 10.1609/aaai.v31i1.10916. URL https://ojs.aaai.org/index.php/AAAI/article/view/10916.

Roberto Baldoni, Emilio Coppa, Daniele Cono D'Elia, Camil Demetrescu, and Irene Finocchi. A survey of symbolic execution techniques. ACM Computing Surveys, 51(3):1–39, 2018. doi: 10.1145/3182657. URL https://dl.acm.org/doi/10.1145/3182657.

Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In Advances in Neural Information Processing Systems, volume 31, pp. 2499–2509. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/hash/e6d8545daa42d5ced125a4bf747b3688-Abstract.html.

Alberto Bemporad and Manfred Morari. Control of systems integrating logic, dynamics, and constraints. Automatica, 35(3):407–427, 1999. doi: 10.1016/S0005-1098(98)00178-2. URL https://www.sciencedirect.com/science/article/pii/S0005109898001782.

Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In Tools and Algorithms for the Construction and Analysis of Systems, volume 4963 of Lecture Notes in Computer Science, pp. 337–340. Springer, 2008. doi: 10.1007/978-3-540-78800-3_24. URL https://link.springer.com/chapter/10.1007/978-3-540-78800-3_24.

Thomas Dean and Shieu-Hong Lin. Decomposition techniques for planning in stochastic domains. In Proceedings of the 14th International Joint Conference on Artificial Intelligence, pp. 1121–1127. Morgan Kaufmann, 1995. URL https://www.ijcai.org/Proceedings/95-2/Papers/014.pdf.

Awi Federgruen, A. Hordijk, and H. C. Tijms. Denumerable state semi-Markov decision processes with unbounded costs, average cost criterion. Stochastic Processes and their Applications, 9(2):223–235, 1979. doi: 10.1016/0304-4149(79)90034-6. URL https://doi.org/10.1016/0304-4149(79)90034-6.

Sicun Gao, Soonho Kong, and Edmund M. Clarke. dReal: An SMT solver for nonlinear theories over the reals. In Automated Deduction, volume 7898 of Lecture Notes in Computer Science, pp. 208–214. Springer, 2013. doi: 10.1007/978-3-642-38574-2_14. URL https://link.springer.com/chapter/10.1007/978-3-642-38574-2_14.

Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G. Bellemare. DeepMDP: Learning continuous latent space models for representation learning. In Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, pp. 2170–2179. PMLR, 2019. URL https://proceedings.mlr.press/v97/gelada19a.html.

Mohsen Ghaffari, Mahsa Varshosaz, Einar Broch Johnsen, and Andrzej Wasowski. Symbolic state partitioning for reinforcement learning. In Fundamental Approaches to Software Engineering, volume 15693 of Lecture Notes in Computer Science, pp. 122–144. Springer, 2025. doi: 10.1007/978-3-031-90900-9_7. URL https://link.springer.com/chapter/10.1007/978-3-031-90900-9_7.

Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in Markov decision processes. Artificial Intelligence, 147(1–2):163–223, 2003. doi: 10.1016/S0004-3702(02)00376-4. URL https://www.sciencedirect.com/science/article/pii/S0004370202003764.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pp. 1861–1870, 2018. URL https://proceedings.mlr.press/v80/haarnoja18b.html.

Thomas A. Henzinger. The theory of hybrid automata. In Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science, pp. 278–292. IEEE, 1996. doi: 10.1109/LICS.1996.561342. URL https://doi.org/10.1109/LICS.1996.561342.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association, 58(301):13–30, 1963. doi: 10.1080/01621459.1963.10500830. URL https://doi.org/10.1080/01621459.1963.10500830.

Khimya Khetarpal, Martin Klissarov, Maxime Chevalier-Boisvert, Pierre-Luc Bacon, and Doina Precup. Options of interest: Temporal abstraction with interest functions. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, pp. 4444–4451, 2020. doi: 10.1609/aaai.v34i04.5871. URL https://ojs.aaai.org/index.php/AAAI/article/view/5871.

James C. King. Symbolic execution and program testing. Communications of the ACM, 19(7):385–394, 1976. doi: 10.1145/360248.360252. URL https://doi.org/10.1145/360248.360252.

Lihong Li, Thomas J. Walsh, and Michael L. Littman. Towards a unified theory of state abstraction for MDPs. In Proceedings of the International Symposium on Artificial Intelligence and Mathematics, 2006. URL https://www.cs.rutgers.edu/~mlittman/papers/abstraction-isaim06.pdf.

David Q. Mayne, James B. Rawlings, Christopher V. Rao, and Pierre O. M. Scokaert. Constrained model predictive control: Stability and optimality. Automatica, 36(6):789–814, 2000. doi: 10.1016/S0005-1098(99)00214-9. URL https://www.sciencedirect.com/science/article/pii/S0005109899002149.

Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. Machine Learning, 13(1):103–130, 1993. doi: 10.1007/BF00993104. URL https://link.springer.com/article/10.1007/BF00993104.

Martin L. Puterman. Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, 1994. ISBN 9780471619772. doi: 10.1002/9780470316887. URL https://doi.org/10.1002/9780470316887.

James B. Rawlings, David Q. Mayne, and Moritz M. Diehl. Model Predictive Control: Theory, Computation, and Design. Nob Hill Publishing, 2 edition, 2017. URL https://sites.engineering.ucsb.edu/~jbraw/mpc/MPC-book-2nd-edition-5th-printing.pdf. Fifth printing, September 2024.

Reuven Y. Rubinstein and Dirk P. Kroese. Simulation and the Monte Carlo Method. Wiley, 3 edition, 2016. doi: 10.1002/9781118631980. URL https://onlinelibrary.wiley.com/doi/book/10.1002/9781118631980.

Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo (eds.), Advances in Neural Information Processing Systems 8, pp. 1038–1044. MIT Press, 1996. URL https://proceedings.neurips.cc/paper/1995/hash/8f1d43620bc6bb580df6e80b0dc05c48-Abstract.html.

Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. MIT Press, 2 edition, 2018. ISBN 9780262039246. URL https://mitpress.mit.edu/9780262039246/reinforcement-learning/.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence, 112(1–2):181–211, 1999. doi: 10.1016/S0004-3702(99)00052-1. URL https://www.sciencedirect.com/science/article/pii/S0004370299000521.

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pp. 5045–5054. PMLR, 2018. URL https://proceedings.mlr.press/v80/verma18a.html.

Mátyás Vincze, Laura Ferrarotti, Leonardo Lucio Custode, Bruno Lepri, and Giovanni Iacca. SMoSE: Sparse mixture of shallow experts for interpretable reinforcement learning in continuous control tasks. Proceedings of the AAAI Conference on Artificial Intelligence, 39(20):20982–20990, 2025. doi: 10.1609/aaai.v39i20.35394. URL https://ojs.aaai.org/index.php/AAAI/article/view/35394.

Kim Peter Wabersich and Melanie N. Zeilinger. A predictive safety filter for learning-based control of constrained nonlinear dynamical systems. Automatica, 129:109597, 2021. doi: 10.1016/j.automatica.2021.109597. URL https://www.sciencedirect.com/science/article/pii/S0005109821001175.

Ward Whitt. Approximations of dynamic programs, i. Mathematics of Operations Research, 3(3):231–243, 1978. doi: 10.1287/moor.3.3.231. URL https://doi.org/10.1287/moor.3.3.231.

Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. In International Conference on Learning Representations, 2021. URL https://openreview.net/forum?id=-2FCwDKRREu.

A Related Work

State abstraction and aggregation. Value iteration and policy iteration provide the standard solution methods for finite MDPs (Puterman, 1994). Model minimization and bisimulation methods identify states with equivalent stochastic behavior (Givan et al., 2003), while general abstraction theory studies when aggregation preserves values or policies (Li et al., 2006). Recent work gives performance bounds for approximate abstractions and learns compact representations from reward and transition behavior (Abel et al., 2016; Gelada et al., 2019; Zhang et al., 2021). Grid value iteration and approximations based on tile coding provide geometric baselines for finite reductions (Sutton & Barto, 2018). They therefore

motivate the FineGridVI and TileQ comparisons used in our experiments. These geometric or learned reductions need not correspond to the control flow of an executable continuous control program. SDTA-MDP instead builds its partition from supplied branch predicates, grouping states by program behavior rather than geometric proximity alone.

Symbolic program structure. Symbolic execution describes feasible program paths through constraints on symbolic inputs (King, 1976; Baldoni et al., 2018). SMT solvers such as Z3 make arithmetic and Boolean feasibility checks practical (de Moura & Bjørner, 2008), while dReal extends this approach to nonlinear constraints (Gao et al., 2013). Programmatic policy learning and decision tree extraction produce structured policies that can be inspected or verified (Verma et al., 2018; Bastani et al., 2018). These methods organize the policy representation, whereas SymPar uses program path conditions to construct semantic partitions for tabular reinforcement learning (Ghaffari et al., 2025). SymPar demonstrates that a finitely constrained disturbance can be treated as a symbolic input in its Random Walk example. SDTA-MDP projects supported disturbance alternatives into possible path predicates and uses the resulting blocks to define absorption quantities and a frontier control model.

Temporal abstraction and continuous control. Temporal abstraction represents extended actions through SMDP Bellman updates (Sutton et al., 1999). Later work learns options and their termination or initiation behavior directly from experience (Bacon et al., 2017; Khetarpal et al., 2020). DTPTA uses a related decomposition into boundaries and interiors, solves a reduced boundary problem, and distributes local computation across submodels (Alexandre et al., 2025). In SDTA-MDP, the decomposition is derived from executable branch predicates rather than supplied by the user or defined through options designed by hand. MPC provides a strong comparison because it repeatedly solves a problem over a finite horizon using the same transition interface (Rawlings et al., 2017). SAC learns continuous control policies through off-policy actor-critic updates under a maximum entropy objective (Haarnoja et al., 2018). SMoSE represents the policy through a sparse mixture of shallow experts, with an interpretable router selecting one expert at each decision (Vincze et al., 2025). Prioritized sweeping also uses a known or learned transition model, but allocates computation to selected value updates rather than repeatedly planning over a moving horizon (Moore & Atkeson, 1993). MPC and prioritized sweeping measure online planning strength and update efficiency, respectively. SDTA-MDP instead constructs a reusable offline topology that makes the influence of program structure explicit.

Safety filters and action correction. Formal shields and predictive safety filters monitor candidate actions and replace them when they violate a specification or model constraint (Alshiekh et al., 2018; Wabersich & Zeilinger, 2021). Control barrier functions similarly impose action constraints through online optimization (Ames et al., 2017). SDTA-MDP takes a complementary approach by using the same symbolic interface for topology construction and action correction. For supported piecewise linear formulas, SMT projection maps each selected local action exactly onto the feasible interval at the current state. This integration allows semantic action selection and safety correction to operate within one symbolic control pipeline, without requiring a separate policy model.

B Theoretical Properties and Proofs

This section states and proves the structural properties of the predicate topology and frontier SMDP. The results establish path signature preservation, a Bellman representation of the reference frontier model, exact-LRA support correctness, and finite sample error propagation under their stated assumptions.

B.1 Formal Statements

Assumptions. For programs with loops, we analyze the execution paths described by the supplied path formulas. For finite actions, the projected predicate interface is complete over $\mathcal{A}_f$: every covered state has at least one feasible path on the declared disturbance support

for each intended action. For continuous actions, the same assumption holds only on $\widehat{\mathcal{A}}_{\text{rep}}$. Exact finite slip support includes every executed action with positive probability. An exact bounded disturbance projection requires the stated disturbance set and path formulas to admit exact elimination; the finite representative construction for Gaussian noise is only an approximation. SMT checks are assumed sound for the supplied formulas. Exact-LRA support construction additionally requires QF-LRA block formulas, guards for each affine piece, and exhaustive affine successor pieces. Claims about finite samples assume independent absorption rollouts, each limited to H steps, for every untruncated pair of a block and an action.

B.2 Topology and Frontier Guarantees

Proposition 1 (Path signature preservation). Let $\mathcal{P}$ be the predicate partition defined in Section 2.2. For any block $B_\ell \in \mathcal{P}$, states $s, s' \in B_\ell$, and intended action $a \in \mathcal{A}_f$, $\Sigma_a(s) = \Sigma_a(s')$. Thus the two states have the same path signature over the declared execution disturbances. For the continuous construction, the same equality holds for every $a \in \widehat{\mathcal{A}}_{\text{rep}}$ used to construct the topology.

Proof. Let B_ℓ be a block in the predicate partition defined in Section 2.2. For each finite intended action $a \in \mathcal{A}_f$, the construction splits the state domain on every possible path predicate ϕ_i^a and labels each satisfiable cell by the set of predicates that hold there. Cells with the same label are merged, while cells with different labels remain separate. Hence any two states $s, s' \in B_\ell$ satisfy the same possible path predicates and have $\Sigma_a(s) = \Sigma_a(s')$. Applying the argument to every finite action proves the claim. Restricting the same construction to $\widehat{\mathcal{A}}_{\text{rep}}$ proves the representative action statement for continuous actions. □

Proposition 2 (Bellman Representation of the Reference Frontier Model). Fix the symbolic partition, representative actions, reference distributions μ_ℓ, and stopping rule. For any scalar η and finite block value function V_F, the expected adjusted cost of a stopped excursion plus its terminal block value equals the corresponding absorption expression. Minimizing this expression over a gives the right-hand side of Equation 8.

Proof. For a fixed pair (B_ℓ, a), the definitions of the absorption quantities give

$$\mathbb{E}_{s_0 \sim \mu_\ell}\left[\sum_{t=0}^{\tau_H - 1} \big(f(s_t, a) - \eta\big) + V_F(B(s_{\tau_H}))\right] = h_f(B_\ell, a) - \eta h_1(B_\ell, a) + \sum_{B_m} \widetilde{p}(B_m \mid B_\ell, a) V_F(B_m).$$

Linearity of expectation gives the cumulative term $h_f - \eta h_1$. Conditioning the terminal value on the reached block gives the remaining sum. Minimizing over a yields the right-hand side of Equation 8. □

For the finite reference SMDP, assume finite expected costs and positive finite expected holding times. Also assume that every assignment of one fixed action to each block has the following property: from any initial block, the process eventually reaches, with probability one, the same set of mutually reachable blocks and remains there. This set may differ between action assignments. Under these conditions, a common optimal average-cost rate and finite relative values exist (Federgruen et al., 1979).

Proposition 3 (Exact-LRA support correctness). Assume block membership, affine piece guards, and target block membership are expressible in QF-LRA; the declared pieces exhaust all realized successors; and the SMT procedure is sound and complete for QF-LRA. Then every edge in F_{LRA} represents a feasible declared affine crossing, every feasible declared affine crossing is included in F_{LRA}, and every realized first exit satisfies $F_H \subseteq F_{\text{LRA}}$.

Proof. For a fixed source block, action, affine piece, and target block, the exact-LRA backend asks whether the conjunction

$$s \in B_\ell, \qquad g_r(s), \qquad A_{a,r} s + b_{a,r} \in B_m$$

is satisfiable. Under the stated QF-LRA assumptions, this conjunction is satisfiable exactly when some state in the source block can cross to the target block through declared affine piece r. Soundness follows because every support edge has a satisfying model for that conjunction. Completeness follows because every feasible witness comprising a source, an affine piece, and a target is included in the exhaustive piece enumeration and is accepted by the complete QF-LRA feasibility procedure. Immediately before any realized first exit from B_ℓ to B_m, the trajectory is still in B_ℓ and its next state lies in B_m. The exit therefore satisfies one of the exhaustive crossing queries, which proves $F_H \subseteq F_{\mathrm{LRA}}$. □

B.3 Absorption Error with Finite Samples

The simultaneous bounds below apply Hoeffding's inequality to the bounded rollout statistics and then take a union bound over pairs of blocks and actions and over candidate successor blocks (Hoeffding, 1963). For each untruncated pair (B_ℓ, a), $\widehat{h}_f$, $\widehat{h}_1$, and $\widehat{p}$ denote the empirical estimates from N independent absorption rollouts. Within this subsection, arguments (B_ℓ, a) are suppressed in the scalar absorption quantities, and $\widehat{p}(B_m)$ abbreviates $\widehat{p}(B_m \mid B_\ell, a)$ for the fixed pair under consideration. Let D bound the number of candidate successor blocks for any such pair. Under exact-LRA, these candidates are the admissible targets, including B_ℓ when the rollout does not exit; under sampled fallback, take $D = |\mathcal{P}|$ before observing the rollouts.

Theorem 1 (Finite sample absorption and Bellman error). Assume the stopped cumulative cost lies in $[0, C_{\max}]$ and $\tau_H \in [0, H]$. Let $M = |\mathcal{P}|\,|\widehat{\mathcal{A}}_{\mathrm{rep}}|$, choose $\delta \in (0,1)$, and let $\mathcal{T}$ and $\widehat{\mathcal{T}}$ denote the exact and empirical Bellman operators associated with Equation 8. With probability at least $1-\delta$, the following bounds hold simultaneously:

$$|\widehat{h}_f - h_f| \le \epsilon_f = C_{\max}\sqrt{\frac{\log(6M/\delta)}{2N}}, \tag{16}$$

$$|\widehat{h}_1 - h_1| \le \epsilon_1 = H\sqrt{\frac{\log(6M/\delta)}{2N}}, \tag{17}$$

$$|\widehat{p}(B_m) - \widetilde{p}(B_m)| \le \epsilon_p = \sqrt{\frac{\log(6MD/\delta)}{2N}}. \tag{18}$$

If $\|V_F\|_\infty \le V_{\max}$, the same event also gives

$$\left\|\widehat{\mathcal{T}}V_F - \mathcal{T}V_F\right\|_\infty \le \epsilon_B = \epsilon_f + |\eta|\epsilon_1 + V_{\max}D\epsilon_p. \tag{19}$$

Proof. For one pair of a block and an action, the two-sided Hoeffding inequality (Hoeffding, 1963) gives

$$\Pr\Big(|\widehat{h}_f - h_f| > \epsilon\Big) \le 2\exp\big(-2N\epsilon^2/C_{\max}^2\big).$$

Allocate failure probability $\delta/3$ to each of the cost, time, and successor probability families. There are at most M cost estimates, so requiring the right-hand side above to be at most $\delta/(3M)$ gives

$$2\exp\big(-2N\epsilon^2/C_{\max}^2\big) \le \frac{\delta}{3M} \quad\Longleftrightarrow\quad \epsilon^2 \ge \frac{C_{\max}^2}{2N}\log\frac{6M}{\delta}.$$

Taking the positive square root gives ϵ_f; replacing the range $C_{\max}$ by H gives ϵ_1. Each successor estimate is an average of Bernoulli indicators in $[0,1]$. There are at most MD such estimates, and requiring

$$2\exp\big(-2N\epsilon^2\big) \le \frac{\delta}{3MD}$$

gives $\epsilon_p = \sqrt{\log(6MD/\delta)/(2N)}$. A union bound over the three families makes their total failure probability at most δ.

On this event, subtracting the exact Bellman term from the empirical term for a fixed (B_ℓ, a) and applying the triangle inequality gives

$$|\widehat{h}_f - h_f| + |\eta|\,|\widehat{h}_1 - h_1| + \left|\sum_{B_m}(\widehat{p}(B_m) - \widetilde{p}(B_m))V_F(B_m)\right|.$$

The first two terms are bounded by ϵ_f and $|\eta|\epsilon_1$. For the probability term, $|V_F(B_m)| \le V_{\max}$ and there are at most D candidate successors, so

$$\left|\sum_{B_m}(\widehat{p}(B_m) - \widetilde{p}(B_m))V_F(B_m)\right| \le V_{\max} D \epsilon_p.$$

Adding the three bounds gives ϵ_B for one pair. The difference between the exact and empirical minima is no larger than the largest per-action difference, and maximizing over source blocks yields Equation 19. □

The dominant implementation cost is the absorption pass, roughly $|\mathcal{P}|\,|\widehat{\mathcal{A}}_{\text{rep}}|NH$, plus the smaller cost of solving the finite frontier model. The method is most useful when the predicate topology is much smaller than a grid that achieves comparable accuracy.

C Additional Method Details

C.1 DTPTA Source Form for Absorption Quantities

Equations 5–7 are obtained from the standard time aggregation form based on boundaries and interiors. If C_i is an interior component, ∂C is the boundary set, and

$$\tau_\partial = \inf\{t \ge 1 \mid X_t \in \partial C\},$$

then the local component used by DTPTA is summarized by

$$\begin{aligned} H_f(i,a) &= \mathbb{E}\left[\sum_{t=0}^{\tau_\partial - 1} f(X_t, a) \,\middle|\, X_0 \in C_i\right], \\ H_1(i,a) &= \mathbb{E}[\tau_\partial \mid X_0 \in C_i], \\ P_\partial(j \mid i,a) &= \Pr(X_{\tau_\partial} = j \mid X_0 \in C_i), \qquad j \in \partial C. \end{aligned}$$

The SDTA-MDP construction applies the substitutions $C_i \mapsto B_\ell$, $X_t \mapsto s_t$, $j \mapsto B_m$, and $\tau_\partial \mapsto \tau_H = \min(\tau_\ell^{\text{exit}}, \tau_{\text{term}}, H)$, while replacing the boundary set ∂C with reachable successor blocks. Under this mapping, the DTPTA quantities H_f, H_1, and P_∂ become the absorption cost h_f in Equation 5, the holding time h_1 in Equation 6, and the terminal block distribution $\widetilde{p}$ in Equation 7, respectively. Figure 4 illustrates this replacement through predicate-defined blocks and their possible targets at the first exit. The distribution $\widetilde{p}$ is defined by the reference initial-state distribution, transition kernel, and stopping rule. Its empirical estimate $\widehat{p}$ assigns frequencies to observed outcomes; an unobserved successor can have zero empirical frequency while retaining positive reference probability. The implementation estimates these expectations and probabilities with Monte Carlo stopped rollouts (Rubinstein & Kroese, 2016).

C.2 Symbolic Block Illustration

Figure 4 illustrates a two dimensional continuous state space partitioned into symbolic blocks. States in the same block share the same path signature over representative actions. Dark short segments mark representative contacts between blocks rather than complete geometric boundaries, and curved arrows indicate possible targets at the first exit, as represented by the support relation in Equation 2. The hatched and dashed regions denote the remote and violation roles supplied by the environment predicates.

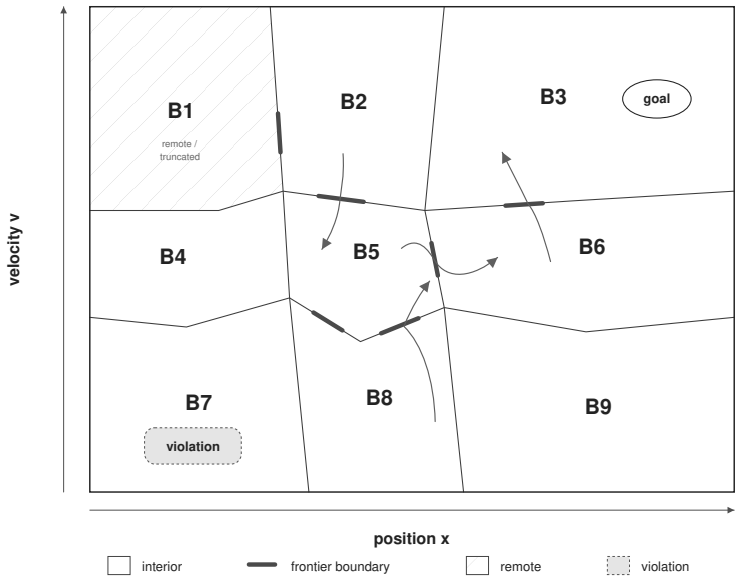


Figure 4: Illustrative symbolic block partition.

D Details on Experimental Setups

Benchmarks and stochastic execution. The finite action benchmarks are BrakingCar, PointMass, and DoubleIntegrator; the continuous action benchmarks are ContinuousBrakingCar, Pendulum, Continuous PointMass, and Continuous DoubleIntegrator. All methods access the same executable interface for transitions and costs. For finite actions, the intended action is replaced uniformly by another available action with probability 0.10. For continuous actions, execution adds clipped Gaussian noise with standard deviation equal to 5% of the action range.

Noise aware topology. For finite actions, topology construction includes every action with positive execution probability. For continuous actions, it uses three executed action representatives in an interval extending two noise standard deviations from the mean; this interval contains Gaussian probability mass 0.9545 before clipping. Execution disturbances are projected into the path formulas, and states are partitioned by the resulting path signatures. When a task declares unsafe path labels, local control removes candidates containing those labels whenever an admissible candidate remains.

Baselines and implementation. SDTA-MDP uses up to $N = 32$ absorption rollouts per block-action pair, horizon $H = 80$, and the sampled frontier backend. Lookahead depths are 2 for both BrakingCar variants, 3 for finite-action PointMass, 1 for continuous PointMass, 4 for both DoubleIntegrator variants, and 2 for Pendulum. FineGridVI uses eight bins per dimension, discount 0.98, and either the exact transition expectation for finite actions or seven node Gauss–Hermite quadrature. MPC evaluates eight common scenarios per candidate and holds each candidate action over a horizon-6 rollout. For finite actions, SymPar+Q uses the authors' Q-learning backend (Ghaffari et al., 2025) for 20,000 episodes on supplied blocks refined for control; this comparison excludes JPF/SPF partition discovery.

SMoSE and SAC. We adapt the authors' SMoSE implementation (Vincze et al., 2025) to the four continuous action environments, retaining its actor, critic, and SAC update code. SMoSE uses eight shallow experts with one selected at each decision; the neural SAC baseline (Haarnoja et al., 2018) uses two hidden layers of 256 units.

Evaluation protocol. Planning and execution use separate environments, and all methods receive disturbances generated from matching seeds. We use ten seeds and twenty evaluation episodes per seed, yielding 390 records: four methods on all seven tasks, SymPar+Q on the three finite action tasks, and SMoSE and SAC on the four continuous action tasks. We report mean cost, success rate, violation rate, and model evaluations; Appendix E specifies the reported bootstrap 95% intervals. SDTA-MDP calls come from a complete-count replay including recursive predictions; control statistics and running times retain the original measurements. Baseline planning and training counts are retained, with SymPar+Q's separately logged evaluation transitions excluded. For SMoSE and SAC, the 100,000 training interactions in each run are recorded separately. During evaluation, the policy directly outputs actions, so the number of online model predictions is zero. Actual evaluation environment transitions caused by action execution are excluded from model evaluations. Paired permutation tests use random sign flips over the ten matched seeds. All experiments ran on CPU under 64-bit Windows 11 Pro using Python 3.12.10, an Intel Core i5-10300H processor, and 16 GB RAM; CUDA was not used.

D.1 Local Control and Absorption Sampling

Local action candidates. For continuous actions, the initial candidates comprise the action bounds, the clipped zero action, the interval midpoint, and the task-specific critical values in Table 3. Two refinement rounds add the reference action and offsets equal to one quarter and one eighth of the action-range width. Candidates are clipped, rounded to twelve decimal places, sorted, and deduplicated; if the base set exceeds its task-specific cap, evenly spaced indices of the sorted set are retained. State-dependent semantic actions are appended after this cap, followed by clipping and deduplication. Finite-action tasks enumerate all available actions.

Table 3: Continuous action candidate settings.

Task	Action bounds	Additional critical values	Base cap
BrakingCar	$[-10, -0.001]$	$-5, -2.5, -0.5, -0.05, -0.01$	13
Pendulum	$[-2, 2]$	$-1, 1$	9
PointMass	$[-\pi, \pi]$	$-\pi/2, \pi/4, \pi/2$	11
DoubleIntegrator	$[-1, 1]$	$-0.5, 0.5$	9

Continuous PointMass adds headings toward the goal and four fixed obstacle-avoidance waypoints. They are computed with atan2 from the current position to $(0.9, 0.9)$, $(0.40, 0.70)$, $(0.70, 0.40)$, $(0.40, 0.66)$, and $(0.66, 0.40)$. Thus, the cap of eleven applies to its base candidates, not to the final merged set, which can contain up to sixteen candidates before path filtering. These waypoints are supplied by the task, not learned by the controller. The other three continuous tasks add no state-dependent semantic actions.

Absorption sampling. We sample states within bounds derived from each block's predicates and retain those satisfying the block formula. Up to 32 accepted states initialize the absorption rollouts; if none is accepted, an SMT witness is used. Estimates are normalized by the actual number of rollouts. The sampling budget therefore need not equal the realized sample count; applying the finite-sample bounds in Appendix B requires the actual sample counts and the stated sampling assumptions.

Noise in local predictions. When the environment declares unsafe path labels, local scoring averages the first transition using the exact finite action-slip distribution or seven-point Gauss–Hermite quadrature. Subsequent lookahead steps predict state transitions without execution noise. For tasks without declared unsafe path labels, this approximation is used throughout local action scoring. Absorption rollouts and policy execution still include the prescribed noise. The local score adds a penalty of 100 for each predicted constraint violation and weights the task terminal potential by 5.

D.2 Frontier Solver and Symbolic Construction

Numerical solution. We use policy iteration, initialized by minimizing estimated cumulative cost divided by estimated holding time in each block. Policy evaluation solves the stationary equations by least squares, clips negative components of the resulting vector to zero, and normalizes it; a nonpositive total triggers a uniform-vector fallback. The average-cost estimate is the weighted cost divided by the weighted holding time using this vector. Relative values are obtained by a second least-squares solve, replacing the first value equation by a zero-reference constraint. Policy improvement minimizes the frontier Bellman expression, retaining the current action when its score is within 10^{-7} of the minimum. Iteration stops when the policy is unchanged or after 500 improvement steps; this stopping rule does not itself certify a small Bellman residual. Holding times are floored at 10^{-9} for numerical calculations, and the stationary weighted holding-time denominator is floored at 10^{-12}. The block action returned by this procedure anchors continuous local candidates, as described in Section 2.

Symbolic preprocessing. Symbolic construction checks the feasibility of candidate path combinations and reuses cached satisfiability results. With overlapping path predicates, refinement explores positive and negative branches, so the number of intermediate cells can grow exponentially with the number of predicates in the worst case. SMT pruning and merging equal path signatures reduce the explored set but do not establish a polynomial bound. Witness extraction and coordinate-bound optimization incur additional solver calls. These calls are distinct from the dynamics evaluations reported in the experiment tables; the number of final blocks is not a count of SMT queries. If an SMT query remains unresolved, partition construction retries it with a longer timeout. If the retry is also inconclusive, construction stops and reports an error. An unresolved query does not prove that a path is infeasible, so discarding it could remove a valid execution path.

E Additional Empirical Audits

This appendix reports diagnostics about individual mechanisms that support, but are not required for, the main claims about finite actions, continuous actions, and topology in Section 3.

E.1 Complete Stochastic Results

Tables 4 and 5 report all main baselines. The finite action results use ten seeds and twenty episodes per seed, with each intended action replaced by another action with probability 0.10. Methods share seed-matched disturbances; FineGridVI uses the exact action-slip expectation, and MPC averages eight common scenarios per decision. SymPar+Q uses its Q-learning backend with the supplied symbolic partition refined for control. The continuous action results use the same seeds and episode counts with clipped Gaussian actuator noise whose standard deviation is 5% of the action range. For continuous actions, FineGridVI uses seven-point Gauss–Hermite transition expectations, and MPC again averages eight common scenarios per decision. Both tables report mean cost with bootstrap 95% confidence intervals; the continuous action table also gives intervals for success and violation. Intervals are computed by resampling the ten seed means 10,000 times; an interval that collapses at zero or one does not establish population certainty. Lower cost and violation are better.

Table 4: Finite action stochastic results.

Benchmark	Method	Cost ↓ [95% CI]	Succ. ↑	Viol. ↓	Calls ↓
BrakingCar	SDTA	20.689 [16.555, 24.990]	0.835	0.165	4.5k
BrakingCar	MPC	19.117 [15.496, 23.556]	0.850	0.150	2.1k
BrakingCar	FineGridVI	23.402 [19.750, 27.229]	0.815	0.185	3.1k
BrakingCar	TileQ	27.414 [23.050, 32.143]	0.780	0.220	442.2
BrakingCar	SymPar+Q	21.968 [18.200, 27.329]	0.845	0.155	19.4k
PointMass	SDTA	14.730 [13.686, 15.884]	1.000	0.100	343.3k
PointMass	MPC	13.936 [13.480, 14.378]	1.000	0.040	104.8k
PointMass	FineGridVI	15.298 [14.525, 16.071]	0.970	0.030	1.6k
PointMass	TileQ	23.300 [21.690, 25.040]	0.580	0.000	20.2k
PointMass	SymPar+Q	24.034 [14.078, 39.414]	0.880	0.075	2066.4k
DoubleIntegrator	SDTA	4.920 [4.156, 6.152]	0.930	0.000	67.5k
DoubleIntegrator	MPC	6.809 [5.857, 8.113]	0.885	0.000	86.1k
DoubleIntegrator	FineGridVI	36.930 [32.711, 43.059]	0.230	0.000	576.0
DoubleIntegrator	TileQ	32.872 [15.479, 52.696]	0.705	0.000	30.5k
DoubleIntegrator	SymPar+Q	91.855 [39.268, 156.631]	0.475	0.000	1683.8k

Table 5: Continuous action stochastic results.

Method	Cost ↓ [95% CI]	Succ. ↑ [95% CI]	Viol. ↓ [95% CI]	Calls ↓
Cont. BrakingCar				
SDTA-MDP	13.952 [10.757, 17.216]	0.905 [0.870, 0.940]	0.095 [0.060, 0.130]	10.3k
MPC	13.884 [10.828, 17.150]	0.910 [0.875, 0.945]	0.090 [0.055, 0.125]	1.8k
SMoSE	14.160 [11.258, 17.383]	0.910 [0.875, 0.940]	0.090 [0.060, 0.125]	100.0k
SAC	12.662 [9.882, 15.736]	0.915 [0.880, 0.945]	0.085 [0.055, 0.120]	100.0k
FineGridVI	15.574 [12.414, 18.797]	0.905 [0.870, 0.940]	0.095 [0.060, 0.130]	3.6k
TileQ	20.520 [16.316, 24.794]	0.855 [0.810, 0.900]	0.145 [0.100, 0.190]	434.3
Pendulum				
SDTA-MDP	48.585 [43.187, 54.989]	1.000 [1.000, 1.000]	0.000 [0.000, 0.000]	59.8k
MPC	47.965 [42.665, 54.486]	1.000 [1.000, 1.000]	0.000 [0.000, 0.000]	229.4k
SMoSE	49.037 [43.497, 55.741]	0.995 [0.985, 1.000]	0.000 [0.000, 0.000]	100.0k
SAC	47.355 [41.468, 54.551]	0.945 [0.835, 1.000]	0.000 [0.000, 0.000]	100.0k
FineGridVI	66.093 [57.780, 75.470]	0.280 [0.225, 0.330]	0.000 [0.000, 0.000]	2.2k
TileQ	157.754 [108.209, 226.715]	0.290 [0.165, 0.440]	0.000 [0.000, 0.000]	44.0k
Cont. PointMass				
SDTA-MDP	9.950 [9.570, 10.357]	1.000 [1.000, 1.000]	0.025 [0.010, 0.040]	66.5k
MPC	9.692 [9.221, 10.250]	1.000 [1.000, 1.000]	0.030 [0.010, 0.050]	160.4k
SMoSE	22.504 [13.332, 33.234]	0.495 [0.195, 0.795]	0.000 [0.000, 0.000]	100.0k
SAC	8.391 [8.222, 8.622]	1.000 [1.000, 1.000]	0.005 [0.000, 0.015]	100.0k
FineGridVI	19.583 [18.722, 20.462]	1.000 [1.000, 1.000]	0.360 [0.320, 0.405]	2.7k
TileQ	26.146 [19.337, 33.448]	0.420 [0.220, 0.630]	0.070 [0.000, 0.210]	14.9k
Cont. DoubleIntegrator				
SDTA-MDP	3.714 [3.365, 4.232]	1.000 [1.000, 1.000]	0.000 [0.000, 0.000]	476.7k
MPC	4.231 [3.876, 4.779]	1.000 [1.000, 1.000]	0.000 [0.000, 0.000]	72.1k
SMoSE	3.652 [3.299, 4.165]	1.000 [1.000, 1.000]	0.000 [0.000, 0.000]	100.0k
SAC	3.606 [3.257, 4.123]	1.000 [1.000, 1.000]	0.000 [0.000, 0.000]	100.0k
FineGridVI	11.181 [8.301, 15.469]	0.995 [0.985, 1.000]	0.000 [0.000, 0.000]	2.2k
TileQ	14.553 [9.750, 20.568]	0.865 [0.700, 0.990]	0.000 [0.000, 0.000]	23.1k

Running time. Table 6 reports ten-seed mean running times on the common Windows CPU environment, with twenty evaluation episodes per seed. This retained timing table covers the original baseline set; it excludes SMoSE and SAC, whose separate training runs included intermediate evaluations and checkpoint overhead. The SDTA-MDP columns separate topology and frontier preparation from policy evaluation; the other method columns report their complete preparation and evaluation time. SymPar+Q applies only to finite actions, and all values are implementation measurements rather than complexity guarantees.

Table 6: Running time by method in seconds.

	SDTA-MDP		MPC	FineGridVI	TileQ	SymPar+Q
Benchmark	Prep.	Eval.	Total	Total	Total	Total
BrakingCar	8.17	0.14	0.07	0.12	0.04	2.93
PointMass	6.29	17.33	5.88	0.16	1.98	13.58
DoubleIntegrator	3.40	2.13	3.40	0.17	2.35	7.48
Cont. BrakingCar	42.21	0.34	0.11	0.26	0.06	–
Pendulum	2.23	4.17	12.67	0.37	4.41	–
Cont. PointMass	230.11	3.71	12.69	0.32	1.97	–
Cont. DoubleIntegrator	6.03	29.64	4.11	0.26	2.28	–

Partition scaling. Table 7 reports SDTA blocks and empirical first-exit support averaged over ten seeds, together with an independent reconstruction of the symbolic partition. FineGrid uses eight bins per dimension, Tile uses four bins per dimension, and Random uses sixteen prototypes. A Grid/SDTA ratio below one means that SDTA contains more partition elements than FineGrid.

Table 7: Partition size under stochastic path projection.

Benchmark	Projection	SDTA blocks	Check	FineGrid	Tile	Random	Grid/SDTA	Support edges
BrakingCar	exact slip	6	6	64	16	16	10.67	29.1
PointMass	exact slip	5	5	64	16	16	12.80	34.8
DoubleIntegrator	exact slip	6	6	64	16	16	10.67	26.9
Cont. BrakingCar	3 noise reps.	26	26	64	16	16	2.46	294.7
Pendulum	3 noise reps.	5	5	64	16	16	12.80	44.9
Cont. PointMass	3 noise reps.	135	135	64	16	16	0.47	3964.9
Cont. DoubleIntegrator	3 noise reps.	6	6	64	16	16	10.67	40.5

Table 8 compares geometric partition growth with the stochastic SDTA topology. Ratios are averaged over three finite action or four continuous action benchmarks.

Table 8: Geometric partition scaling relative to the stochastic SDTA topology.

Regime	Bins/dim	FineGrid cells	Tile cells	Mean SDTA blocks	Mean Grid/SDTA	Mean Tile/SDTA
finite	8	64	16	5.67	11.4×	2.8×
finite	16	256	64	5.67	45.5×	11.4×
finite	32	1024	256	5.67	182.0×	45.5×
finite	64	4096	1024	5.67	728.2×	182.0×
finite	128	16384	4096	5.67	2912.7×	728.2×
continuous	8	64	16	43.00	6.6×	1.7×
continuous	16	256	64	43.00	26.4×	6.6×
continuous	32	1024	256	43.00	105.6×	26.4×
continuous	64	4096	1024	43.00	422.4×	105.6×
continuous	128	16384	4096	43.00	1689.7×	422.4×

State-grid resolution. We evaluated FineGridVI with 8, 16, 32, and 64 grid points per state dimension, corresponding to 64, 256, 1,024, and 4,096 grid states. Its action set, discount factor of 0.98, and stochastic transition expectations were held fixed. Each configuration was evaluated over ten paired seeds and twenty episodes per seed, with a limit of 120 steps per episode. SDTA-MDP and all grid configurations used identical initial states and execution disturbances within each seed; this additional evaluation used a separate set of initial states and disturbances from the main tables. Tables 9 and 10 report mean cost with bootstrap 95% confidence intervals over seeds, together with mean success, violation, and state representation size. The States column gives the number of symbolic blocks for SDTA-MDP and grid states for FineGridVI.

Symbolic grouping produced compact representations while retaining useful control performance. On Pendulum, SDTA-MDP used five blocks and achieved unit observed success with mean cost 52.214, compared with 4,096 grid states and cost 54.472 at the same success rate. On ContinuousBrakingCar, 26 blocks yielded cost 14.685 and success 0.895, compared with cost 14.685 and success 0.900 for 1,024 grid states. These results illustrate that fine geometric subdivision is not necessary for useful control on these tasks: SDTA-MDP organizes the state space by program behavior instead of assigning a separate state to each grid location. Finer grids nevertheless obtained lower costs on both PointMass variants and Continuous DoubleIntegrator, and lower violation rates on the PointMass tasks. This comparison measures representation size and control quality for complete controllers with different planning objectives, rather than the isolated effect of partitioning.

Table 9: State-grid resolution on finite-action tasks.

Benchmark	Method	Cost [95% CI]	Success	Violation	States
BrakingCar	SDTA-MDP	21.527 [16.884, 26.409]	0.825	0.175	6
	FineGridVI 8	24.372 [20.698, 28.290]	0.805	0.195	64
	FineGridVI 16	21.684 [18.794, 24.394]	0.830	0.170	256
	FineGridVI 32	19.124 [15.740, 22.244]	0.850	0.150	1,024
	FineGridVI 64	18.041 [14.792, 21.317]	0.860	0.140	4,096
PointMass	SDTA-MDP	15.205 [14.227, 16.411]	1.000	0.120	5
	FineGridVI 8	15.332 [14.452, 16.490]	0.975	0.020	64
	FineGridVI 16	13.397 [12.694, 14.188]	1.000	0.015	256
	FineGridVI 32	12.890 [12.495, 13.323]	1.000	0.005	1,024
	FineGridVI 64	12.890 [12.507, 13.272]	1.000	0.000	4,096
DoubleIntegrator	SDTA-MDP	5.485 [4.975, 5.995]	0.880	0.000	6
	FineGridVI 8	36.921 [33.273, 41.977]	0.225	0.000	64
	FineGridVI 16	5.761 [5.168, 6.395]	0.900	0.000	256
	FineGridVI 32	5.597 [5.072, 6.208]	0.805	0.000	1,024
	FineGridVI 64	5.193 [4.710, 5.736]	0.820	0.000	4,096

Table 10: State-grid resolution on continuous-action tasks.

Benchmark	Method	Cost [95% CI]	Success	Violation	States
Cont. BrakingCar	SDTA-MDP	14.685 [11.453, 18.455]	0.895	0.105	26
	FineGridVI 8	16.060 [12.858, 19.974]	0.900	0.100	64
	FineGridVI 16	16.565 [12.818, 20.412]	0.885	0.115	256
	FineGridVI 32	14.685 [11.386, 18.576]	0.900	0.100	1,024
	FineGridVI 64	14.833 [11.550, 18.780]	0.900	0.100	4,096
Pendulum	SDTA-MDP	52.214 [47.431, 58.733]	1.000	0.000	5
	FineGridVI 8	73.914 [67.956, 81.424]	0.280	0.000	64
	FineGridVI 16	59.143 [53.781, 66.023]	0.325	0.000	256
	FineGridVI 32	57.714 [52.577, 64.482]	0.845	0.000	1,024
	FineGridVI 64	54.472 [49.483, 61.184]	1.000	0.000	4,096
Cont. PointMass	SDTA-MDP	9.900 [9.434, 10.374]	1.000	0.020	135
	FineGridVI 8	18.753 [17.384, 20.242]	1.000	0.315	64
	FineGridVI 16	9.326 [9.001, 9.702]	1.000	0.015	256
	FineGridVI 32	8.903 [8.710, 9.162]	1.000	0.005	1,024
	FineGridVI 64	8.711 [8.642, 8.776]	1.000	0.000	4,096
Cont. DoubleIntegrator	SDTA-MDP	4.236 [3.833, 4.634]	1.000	0.000	6
	FineGridVI 8	10.987 [9.348, 13.161]	1.000	0.000	64
	FineGridVI 16	4.505 [4.083, 4.922]	1.000	0.000	256
	FineGridVI 32	4.414 [4.004, 4.814]	1.000	0.000	1,024
	FineGridVI 64	4.177 [3.776, 4.572]	1.000	0.000	4,096

Local lookahead ablation. Table 11 compares the complete method with the variant that removes local lookahead under the same stochastic execution protocol. NoLook removes the local refinement while retaining the same noise model, projected topology, sampled frontier, seeds, and evaluation episodes. Lower cost and violation are better.

Table 11: Local lookahead ablation under stochastic execution.

Benchmark	Variant	Cost ↓ [95% CI]	Succ. ↑	Viol. ↓
Cont. BrakingCar	SDTA	13.952 [10.810, 17.351]	0.905	0.095
Cont. BrakingCar	NoLook	13.940 [10.810, 17.351]	0.905	0.095
Pendulum	SDTA	48.585 [43.137, 55.905]	1.000	0.000
Pendulum	NoLook	265.862 [232.068, 295.696]	0.120	0.000
Cont. PointMass	SDTA	9.950 [9.570, 10.356]	1.000	0.025
Cont. PointMass	NoLook	9.950 [9.570, 10.356]	1.000	0.025
Cont. DoubleIntegrator	SDTA	3.714 [3.362, 4.290]	1.000	0.000
Cont. DoubleIntegrator	NoLook	53.346 [43.081, 64.515]	0.150	0.000

F Limitations

SDTA-MDP assumes that the environment supplies symbolic predicates or path conditions; it does not yet extract them automatically from arbitrary Python programs. Continuous action topology uses finite action and disturbance representatives, empirical first exit sampling can miss rare transitions, exact successor claims apply only to declared piecewise affine fragments, and remote blocks use surrogate truncation.

The theory bounds the induced frontier SMDP and Monte Carlo absorption error, but not the end to end performance gap to an optimal continuous action policy. Exact SMT projection certifies the declared action formula rather than stochastic successor safety; higher dimensional scaling and automatic predicate extraction remain open.

G Use of Large Language Models

Large language models were used to assist with writing and language polishing, retrieve and identify related work, and generate synthetic datasets.