Pixels to Proofs: Probabilistically-Safe Latent World Model Control via Parallel Conformal Robust MPC

Devesh Nath*, Anutam Srinivasan*, Haoran Yin*, Ruitong Jiang, Jeffrey Fang, Glen Chou
{dnath7, asrinivasan350, hyin95, rjiang77, jfang301, chou}@gatech.edu
*Equal Contribution Georiga Institute of Technology **Code / Video / Website**

***Abstract*—We present SLS^2, a framework for feedback motion planning from pixels using learned latent world models. Our central premise is that world models for robotics should be evaluated not only by prediction accuracy, but by whether their latent dynamics support reliable, robust long-horizon planning. To this end, SLS^2 trains an action-conditioned joint-embedding world model with compact Markovian latent states, which we refer to as a Markovian world model. This representation enables efficient gradient-based MPC through learned latent dynamics. This Markov state abstraction substantially improves planning performance by making latent rollouts more predictive, locally optimizable, and compatible with receding-horizon control. To account for inevitable model error over imagined horizons, we combine conformal prediction with a GPU-parallel system level synthesis (SLS) robust MPC scheme, yielding calibrated latent error bounds and reachable tubes for closed-loop execution. We further learn and conformalize a latent constraint checker, allowing the planner to impose probabilistic safety constraints directly in latent space. Across vision-based control tasks, SLS^2 improves goal-reaching, robust task completion, and safety over latent world-model and safe-planning baselines, demonstrating that Markovian latent world models can serve as effective inductive biases for long-horizon robot planning from pixels.**

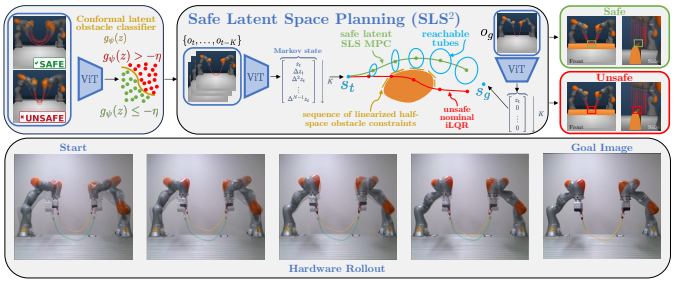

Fig. 1. **Method overview.** SLS^2 enables robustly safe, reachability-constrained latent planning from pixels. *Top row*: We summarize our method pipeline. *Bottom row*: We show a sample rollout generated by executing control actions computed by the nominal gradient-based planner on which the robust planner is built. We overlay the goal pose, shown with a green rope, on the rollout to highlight the system's progress toward the goal state.

I. Introduction

Model-based control relies on state representations that are predictive, approximately Markovian, and suitable for control. In visuomotor control, however, the robot only observes images, making it difficult to estimate or explicitly define task-relevant states. Learned world models replace hand-designed states with latent dynamics learned from data, but predictive latent representations are not necessarily useful for control. Long-horizon planning requires latent states that retain action-relevant information, support gradient-based planning, and admit uncertainty quantification. Without this, imagined trajectories can drift, planners can exploit model errors, and latent safety constraints may fail under closed-loop control.

To bridge the gap, we train a world model to directly exhibit these properties. We also propose SLS^2, which leverages these properties for robust latent space planning. We use Conformal Prediction (CP) to bound model error, and System Level Synthesis (SLS) to robustly account for these errors at planning time. We evaluate SLS^2 on navigation, manipulation, contact-rich pushing, and deformable rope-control tasks, demonstrating improved planning performance, robustness, and safety compared to latent-world-model and safe-planning baselines. *Our contributions are:*

- We introduce a *Markovian world model* for pixel-based robot planning, using visual embeddings and temporal differences to learn action-conditioned latent dynamics for safe gradient-based MPC.
- We develop a conformal robust MPC method that calibrates latent dynamics and constraint-checker errors, then propagates them with SLS to obtain reachable tubes for closed-loop constraint satisfaction.
- We show across vision-based control tasks that Markovian latent structure improves goal-reaching, robust task completion, and safety, and that removing this abstraction degrades planning performance.

II. Related Work

World models learn predictive representations for imagined rollouts, while JEPA methods predict future embeddings rather than pixels, enabling planning directly in latent space [7, 8, 3]. Recent robotics systems leverage these representations for reward-free and zero-shot planning from offline data [17, 25]. We build on Le-WM [10], an end-to-end JEPA world model for latent-space MPC, and introduce training modifications that improve predictive accuracy and gradient-based planning.

Safety filters enforce constraints by minimally modifying nominal actions. Classical methods [1, 12, 22] provide strong guarantees but typically require known dynamics and hand-designed safety constraints. More recent data-driven and latent-space methods learn barriers, safety certificates, or reachability-informed filters directly from data and high-dimensional observations [15, 5, 13, 16]. However, most operate as policy filters, safety predictors, or recovery mechanisms around a given policy [11]. In contrast, we directly synthesize safety-constrained feedback policies and motion plans through reachability-informed latent-space control, while providing calibrated probabilistic guarantees via SLS and CP.

III. Preliminaries and Problem Statement

We consider an unknown partially observed dynamical system described by an *unknown* state representation $x_k \in \mathcal{X}$

and control $u_k \in \mathcal{U} \subseteq \mathbb{R}^{n_u}$, and observations $o_k \in \mathcal{O} \subseteq \mathbb{R}^{n_o}$:

$$x_{k+1} = g(x_k, u_k), \qquad o_k = h(x_k), \tag{1}$$

where $o_k \in \mathcal{O}$ are RGB images. In our setting, the planner lacks access to the true state x_k and representation $\mathcal{X}$; g and h are unknown, and only images $o \in \mathcal{O}$ from (1) are observed.

We consider safe feedback motion planning from images using a learned latent world model. We are given an offline dataset of image transitions $\mathcal{D} := \{(o_t, u_t, o_{t+1})\}$ collected from task-relevant rollouts. We wish to learn a compact latent representation where prediction, planning, and reachability-based safety verification are tractable. We train a visual encoder $\mathrm{enc}_\theta : \mathcal{O} \to \mathbb{R}^{n_z}$, with $z_t = \mathrm{enc}_\theta(o_t)$. Using enc_θ, we construct a Markovian latent state $s_t^z = [z_t, \Delta z_t, \ldots, \Delta^K z_t] \in \mathcal{S} \subseteq \mathbb{R}^{n_s}$, where $\Delta z_t = z_t - z_{t-1}$, from a history of encoded images. The learned latent dynamics $f_\phi : \mathcal{S} \times \mathcal{U} \to \mathcal{S}$ predict the next Markovian state $\hat{s}_{t+1}^z = f_\phi(s_t^z, u_t)$. As f_ϕ can be inaccurate, we model its prediction error as $s_{t+1}^z = f_\phi(s_t^z, u_t) + e_t$, where $\mathcal{E} \subseteq \mathbb{R}^{n_s}$ is calibrated from a held-out subset of $\mathcal{D}$ using CP to contain the true prediction error e_t with a user-specified probability $1 - \delta \in (0, 1)$. Specifically, we solve:

Problem 1. *Learning a planning- and verification-friendly latent world model.* Using $\mathcal{D}$, learn enc_θ and f_ϕ such that the latent state s_t^z is predictive under actions, the learned dynamics are multi-step accurate, differentiable, and compatible with gradient-based trajectory optimization. Construct uncertainty sets $\mathcal{E}(s, u) \subseteq \mathbb{R}^{n_s}$ such that the prediction error e at a given (s, u) satisfies $e \in \mathcal{E}(s, u)$ with probability $1 - \delta$. Finally, learn a latent safety score $c_\psi : \mathbb{R}^{n_z} \to \mathbb{R}$, where larger values indicate safer states, and define the constraint score $g_\psi(s) := -c_\psi(\Pi_z s)$, where Π_z extracts the zeroth-order latent state.

Problem 2. *Robust latent MPC.* Given f_ϕ, $\mathcal{E}$, and a latent safe set $\mathcal{S}_{\text{safe}} := \{s \in \mathcal{S} \mid g_\psi(s) \leq -\eta\}$ for $\eta \in \mathbb{R}$, find a nominal length-T latent trajectory $\boldsymbol{\zeta} := \{\zeta_k\}_{k=0}^{T}$ and controls $\mathbf{v} := \{v_k\}_{k=0}^{T-1}$ satisfying $\zeta_{k+1} = f_\phi(\zeta_k, v_k)$, together with a causal state-feedback controller $\pi := (\pi_0, \ldots, \pi_{T-1})$, $\pi_k : \mathcal{S}^{k+1} \to \mathcal{U}$ stabilizing the true latent dynamics $s_{k+1}^z = f_\phi(s_k^z, \pi_k(s_{0:k}^z)) + e_k$ about $(\boldsymbol{\zeta}, \mathbf{v})$, where e_k is the true prediction error realized by executing on the true system (1). This MPC should produce reachable tubes $\mathcal{R}_k^s$ and $\mathcal{R}_k^u$ such that, with probability at least $1 - \delta$, $s_k^z \in \mathcal{R}_k^s \subseteq \mathcal{S}_{\text{safe}}$ for all $k = 0, \ldots, T$ and $\pi_k(s_{0:k}^z) \in \mathcal{R}_k^u \subseteq \mathcal{U}$ for all $k = 0, \ldots, T-1$.

IV. Methodology

Our method, SLS^2 (Fig. 1), trains a Markovian latent model and constraint classifier (Sec. IV-A), conformalizes its uncertainty (Sec. IV-B), and incorporates the resulting calibrated uncertainty sets into SLS-based robust MPC for probabilistically-safe feedback motion planning (Sec. IV-C).

A. Training World Models Friendly for Gradient-Based MPC

Given the offline dataset $\mathcal{D}$, we train an action-conditioned latent world model following a Le-WM-style [10] joint-embedding predictive architecture. Given an image observation o_t, an encoder enc_θ maps the image into a compact latent embedding $z_t = \mathrm{enc}_\theta(o_t)$. The encoder is implemented as a ViT backbone trained from scratch, followed by an MLP projector that maps the encoder output to the latent embedding space. Before encoding, image observations are resized to a fixed resolution, normalized, and organized into short temporal windows sampled from offline trajectories. Actions are normalized using the dataset mean and standard deviation. To model dynamics, we construct a Markov latent state by augmenting the current embedding with finite-difference latent features $s_t^z = \left[z_t,\ \Delta z_t,\ \ldots,\ \Delta^K z_t\right], \Delta z_t = z_t - z_{t-1}$, where K denotes the Markov order. The latent dynamics predictor f_ϕ is an MLP that takes the current Markov latent state and action as input and predicts the next Markov latent state:

$$\hat{s}_{t+1}^z = f_\phi(s_t^z, a_t). \tag{2}$$

For multi-step training, the predictor is rolled out autoregressively over a horizon H, where each predicted latent state is fed back into the model for the next prediction step $\hat{s}_{t+h+1}^z = f_\phi(\hat{s}_{t+h}^z, a_{t+h})$ for $h = 0, \ldots, H-1$. The main training loss is a multi-step latent prediction objective. At each prediction step, the predicted Markov latent state is compared against the target Markov latent state computed from the encoded ground-truth future observations $\mathcal{L}_{\text{pred}} = \frac{1}{H}\sum_{h=1}^{H} \left\|\hat{s}_{t+h}^z - s_{t+h}^z\right\|_2^2$. We follow Le-WM [10] and prevent representational collapse via the SIGReg loss, $\mathcal{L}_{\text{sig}}$, [4] and also encourage consecutive latent velocity vectors to align via temporal straightening, $\mathcal{L}_{\text{str}}$, [23]. The full objective is thus $\mathcal{L} = \mathcal{L}_{\text{pred}} + \lambda_{\text{sig}}\mathcal{L}_{\text{sig}} + \lambda_{\text{str}}\mathcal{L}_{\text{str}}$.

Obstacle Classifier. For constrained tasks, we train a latent obstacle classifier to detect constraint violations. For each task, we construct a balanced dataset of obstacle and non-obstacle observations by sampling task-relevant configurations, labeling them with the known simulator constraint, and rendering RGB images. The dataset has the common form $\mathcal{D}_{\text{obs}} = \{(o_i, y_i)\}_{i=1}^{N}$, with $y_i \in \{0, 1\}$, where $y_i = 0$ denotes a constraint-violating image and $y_i = 1$ denotes a non-violating image. We use the frozen encoder to map each image to $z_i = \mathrm{enc}_\theta(o_i)$, and train a small MLP classifier $c_\psi : \mathbb{R}^{n_z} \to \mathbb{R}$ that outputs a signed score $c_\psi(z_i)$, with positive scores indicating non-violating states and negative scores indicating violations. Binary labels are mapped to signed labels $\tilde{y}_i \in \{-1, +1\}$, with $y_i = 0$ mapped to -1 and $y_i = 1$ mapped to $+1$, and the classifier is trained with the hinge loss, $\mathcal{L}_{\text{obs}} = \frac{1}{N}\sum_{i=1}^{N} \max\left(0,\ m - \tilde{y}_i c_\psi(z_i)\right)$, where m is the classification margin. We normalize latent features using the training-split mean and standard deviation, then calibrate the classifier threshold on a held-out split. Let $\mathcal{I}_{\text{cal}}^- := \{i \in \mathcal{I}_{\text{cal}} : y_i = 0\}$ denote the violating calibration indices. For each $i \in \mathcal{I}_{\text{cal}}^-$, we compute the nonconformity score $r_i = \max\left(0,\ c_\psi(z_i)\right)$ and choose η as the empirical $(1 - \delta)$ quantile of these scores. At planning time, a latent Markov state s is treated as safe if

$$c_\psi(\Pi_z s) \geq \eta, \quad \Longleftrightarrow \quad g_\psi(s) := -c_\psi(\Pi_z s) \leq -\eta. \tag{3}$$

The resulting calibrated classifier is used during constrained planning to detect if predicted latent rollouts are unsafe.

B. Latent Uncertainty Quantification via CP

To ensure closed-loop safety for planning, we need two additional sources of uncertainty to be bounded: 1) latent

dynamics error and 2) in-distribution domain calibration. First, to quantify latent prediction error, we use split CP [21, 9]. Let $\mathcal{D}_{\text{cal}} = \{(s_i, u_i, s_i^+)\}_{i=1}^n$ be a held-out set of latent transitions obtained by encoding a subset of $\mathcal{D}$, and define residuals $e_i := s_i^+ - f_\phi(s_i, u_i)$. To capture state-action-dependent error geometry, we train an uncertainty model $\Sigma_\psi : \mathcal{S} \times \mathcal{U} \to \mathbb{S}_{++}^{n_s}$ using the multivariate Gaussian negative log-likelihood loss

$$\mathcal{L}_{\text{MGNLL}} = \tfrac{1}{2}\left(e_i^\top \Sigma_\psi(s_i, u_i)^{-1} e_i + \log\det \Sigma_\psi(s_i, u_i)\right). \quad (4)$$

Σ_ψ is not calibrated; it only provides a local ellipsoid shape for the prediction error. We calibrate its scale using CP with scores $r_i := e_i^\top \Sigma_\psi(s_i, u_i)^{-1} e_i$. Let $r_{(1)} \le \cdots \le r_{(n)}$ be the sorted scores. For trajectory-level failure probability δ over horizon T, set $\bar{\delta} := \delta/T$ and $q := r_{(\lceil (n+1)(1-\bar{\delta}) \rceil)}$, with $q = +\infty$ if the index exceeds n. For any latent state-action pair (s, u), the calibrated one-step error set is

$$\mathcal{E}(s, u) := \{e \mid e^\top \Sigma_\psi(s, u)^{-1} e \le q\}. \quad (5)$$

Writing $\Sigma_\psi(s, u) = L_\psi(s, u) L_\psi(s, u)^\top$ using its Cholesky factor L_ψ, this is equivalently $\mathcal{E}(s, u) = \sqrt{q}\, L_\psi(s, u) \mathcal{B}^{n_s}$, where $\mathcal{B}^{n_s} := \{s \in \mathbb{R}^{n_s} \mid \|s\|_2 \le 1\}$.

Second, we conformalize an in-domain latent region to keep the planner within the support of the calibration data. Since the SIGReg loss encourages the embeddings to follow a Gaussian distribution [4], we define the in-domain set as a calibrated Gaussian density sublevel set. Concretely, we fit an ellipsoid to the calibration embeddings and choose its conformal threshold so that it contains most calibration latents with the desired coverage. Let μ and Σ_{ID} be the empirical mean and covariance of calibration latents in $\mathcal{D}_{\text{cal}}$, and define $d_i := (s_i - \mu)^\top \Sigma_{\text{ID}}^{-1} (s_i - \mu)$. With q_{ID} the $(1 - \alpha_{\text{ID}})$ split conformal quantile of $\{d_i\}$, the calibrated in-domain set is

$$\mathcal{I} := \{s \mid (s - \mu)^\top \Sigma_{\text{ID}}^{-1} (s - \mu) \le q_{\text{ID}}\}. \quad (6)$$

During planning, we require $\mathcal{R}_k^s \subseteq \mathcal{I}$ for all k. This controls support mismatch, while the CP guarantee itself assumes that single step residual scores are exchangeable with the calibration scores. In practice, we support this assumption by collecting calibration transitions from representative randomized rollouts that vary initial states and goals.

C. Conformalized Robust MPC via SLS

To synthesize robust latent feedback policies, we use SLS [2]. SLS enables efficient robust MPC by optimizing over closed-loop system responses rather than feedback gains directly, and admits fast GPU-parallel implementations for high-dimensional systems [6]. This is critical for latent-space planning, where the latent state dimension can be high. We build on [18], using the conformal calibration in Sec. IV-B to define the disturbance sets to be propagated by SLS. We model the learned latent dynamics with calibrated additive error,

$$s_{k+1} = f_\phi(s_k, u_k) + E_k w_k, \qquad w_k \in \mathcal{B}^{n_s}, \quad (7)$$

where $E_k = \sqrt{q} L_\psi(s_k, u_k)$, recovering $E_k \mathcal{B}^{n_s}$ as the CP-calibrated, ellipsoidal one-step latent prediction-error set defined in (5). Given the current latent $\bar{s}_0$, SLS finds a nominal trajectory $\boldsymbol{\zeta} := \{\zeta_k\}_{k=0}^T$, controls $\mathbf{v} := \{v_k\}_{k=0}^{T-1}$, and closed-loop response matrices Φ^s, Φ^u mapping disturbances to deviations from the nominal state and control trajectories. Around (ζ_k, v_k), let $A_k := \nabla_s f_\phi(\zeta_k, v_k)$ and $B_k := \nabla_u f_\phi(\zeta_k, v_k)$. From Sec. IV-B, if $\Sigma_\psi(\zeta_k, v_k) = L_\psi(\zeta_k, v_k) L_\psi(\zeta_k, v_k)^\top$, we set

$$E_k := \sqrt{q}\, L_\psi(\zeta_k, v_k). \quad (8)$$

Thus SLS propagates the CP-calibrated ellipsoidal model errors over time to construct reachable tubes and constraint tightenings. The resulting CP-calibrated SLS MPC problem is

$$\begin{aligned}
\min_{\boldsymbol{\zeta}, \mathbf{v}, \Phi^s, \Phi^u} \quad & J(\boldsymbol{\zeta}, \mathbf{v}) + J_f(\zeta_T) + H(\Phi^s, \Phi^u) && \text{(9a)}\\
\text{s.t.} \quad & k = 0, \ldots, T-1, \quad j < k, \quad i = 1, \ldots, n_c \\
& \zeta_{k+1} = f_\phi(\zeta_k, v_k), \quad \zeta_0 = \bar{s}_0, && \text{(9b)}\\
& \Phi^s_{k+1,j} = A_k \Phi^s_{k,j} + B_k \Phi^u_{k,j}, && \text{(9c)}\\
& \Phi^s_{j+1,j} = E_j, && \text{(9d)}\\
& g_i(\zeta_k, v_k) + b_i + \tau_{i,k} \le 0, && \text{(9e)}\\
& \mathcal{R}_k^s \subseteq \mathcal{I}, && \text{(9f)}
\end{aligned}$$

where $\tau_{i,k} := \sum_{j=0}^{k-1} \|\nabla_s g_i(\zeta_k, v_k) \Phi^s_{k,j} + \nabla_u g_i(\zeta_k, v_k) \Phi^u_{k,j}\|_2$, and $\mathcal{R}_k^s := \zeta_k \oplus \bigoplus_{j=0}^{k-1} \Phi^s_{k,j} \mathcal{B}^{n_s}$, $\mathcal{R}_k^u := v_k \oplus \bigoplus_{j=0}^{k-1} \Phi^u_{k,j} \mathcal{B}^{n_s}$ are the state and control reachable tubes, respectively. Here, $\bigoplus$ and $\oplus$ denote Minkowski sums, and b_i is a constraint-index-dependent bias equal to η for obstacle constraints and zero for control constraints. For sets $\mathcal{A}$ and $\mathcal{B}$, $\mathcal{A} \oplus \mathcal{B} = \{a + b \mid a \in \mathcal{A}, b \in \mathcal{B}\}$. The objective (9a) combines nominal task cost, terminal cost, and a tube-size regularizer $H(\Phi^s, \Phi^u)$ (see App. B for details). The constraint (9b) enforces nominal feasibility under the latent dynamics, and (9c)-(9d) enforce that SLS correctly propagates disturbance, with the CP-calibrated disturbance set injected through (9d). The constraints (9e) are tightened by reachable tube margins $\tau_{i,k}$, enforcing robust satisfaction of the learned latent safety constraints over the full reachable tube, and (9f) keeps the reachable tube inside the conformalized in-domain latent region (6). For instance, with $g_\psi(s) := -c_\psi(\Pi_z s)$, the obstacle constraint $g_\psi(s_k) \le -\eta$ becomes $g_\psi(\zeta_k) + \eta + \tau_k \le 0$ at each step k to enforce *robust* constraint satisfaction. This CP-informed SLS scheme guarantees safety with $1 - \delta$ probability. We formally state Theorem A.1 in App. C). Beyond robust MPC, the latent dynamics (2), learned via Sec. IV-A, can also be used for gradient-based nominal trajectory optimization by removing the reachability-related terms in (9) and retaining only the nominal latent space dynamics constraints in (9b).

V. Experiments

We evaluate on four pixel-based continuous-control tasks: Reacher [19], OGBench Cube [14], Push-T [25], and a bimanual rope manipulation task. We study 1) nominal gradient-based planning, which tests if our Markov latent dynamics enable long-horizon planning, and 2) robust constrained planning, which tests if SLS^2 improves safety using calibrated latent constraints and robust tubes. We compare against the corresponding world-model planning and safe latent-space planning baselines in Secs. V-A and V-B. For nominal tasks, we report success rate and minimum distance to the goal;

TABLE I

ABLATION STUDY ON REACHER. THE REFERENCE SETTING IS THE FULL METHOD DESCRIBED IN SEC. IV. WORLD-MODEL ABLATIONS KEEP THE PLANNER FIXED AND CHANGE ONLY THE LATENT MODEL/TRAINING COMPONENT. THE PLANNER ABLATION KEEPS THE WORLD MODEL FIXED AND CHANGES ONLY THE MPC OPTIMIZER.

Ablation	Variant	**Success (%)** ↑	Min. qpos dist. ↓	Final qpos dist. ↓	Solve time / step (ms) ↓
Reference	Full model + iLQR	$\mathbf{83.50 \pm 4.06}$	0.377 ± 0.119	0.497 ± 0.152	426.35 ± 17.52
World model	Non-Markov history-2 + iLQR	1.00 ± 1.22	1.786 ± 0.088	2.080 ± 0.087	67.00 ± 3.01
	Single-step loss + iLQR	24.50 ± 4.58	1.283 ± 0.138	1.746 ± 0.205	578.33 ± 16.56
Planner	Full model + CEM	57.00 ± 1.87	0.380 ± 0.100	0.511 ± 0.106	67.00 ± 0.94

TABLE II

NOMINAL PLANNING PERFORMANCE ACROSS BENCHMARK TASKS. SUCCESS IS REPORTED IN PERCENTAGE, AND DIST. DENOTES MINIMUM DISTANCE TO THE GOAL.

Task	Method	**Success (%)** ↑	Dist. ↓	Task	Method	**Success (%)** ↑	Dist. ↓
Reacher	Ours	$\mathbf{83.5 \pm 4.06}$	0.377 ± 0.119	Rope	Ours	$\mathbf{93.75 \pm 2.71}$	0.020 ± 0.002
	Le-WM	17.0 ± 7.6	1.305 ± 1.109		Le-WM	13.0 ± 4.3	0.093 ± 0.015
	DINO-WM	24.5 ± 6.2	0.533 ± 0.115		DINO-WM	25.0 ± 7.91	0.058 ± 0.002
	PLDM	40.5 ± 9.9	0.821 ± 0.158		PLDM	7.0 ± 5.34	0.130 ± 0.019
Cube	Ours	$\mathbf{91.5 \pm 4.2}$	0.051 ± 0.055	Push-T	Ours	$\mathbf{51.54 \pm 4.40}$	29.86 ± 35.55
	Le-WM	28.5 ± 3.8	0.133 ± 0.023		Le-WM	1.5 ± 2.00	113.61 ± 10.25
	DINO-WM	69.0 ± 2.95	0.054 ± 0.048		DINO-WM	2.52 ± 0.04	112.58 ± 3.36
	PLDM	28.75 ± 6.62	0.145 ± 0.005		PLDM	1.50 ± 1.37	126.07 ± 5.26

for constrained tasks, we also report safety rate and robust success rate. See App. D for further implementation details.

A. Nominal Planning

Table II summarizes nominal planning results on Reacher, Cube, Push-T, and Rope. Across all four tasks, SLS^{2}achieves the highest success rate among the evaluated methods: 83.5% on Reacher, 91.5% on Cube, 51.54% on Push-T, and 93.75% on Rope. The gains are especially large on Reacher and Rope, where the best baseline reaches only 40.5% and 25.0% success, respectively. On Cube, SLS^2 also improves over the strongest baseline, DINO-WM, which achieves 69.0% success. Push-T remains the most challenging nominal task due to long-horizon contact-rich manipulation, but SLS^2 still obtains the best success rate and the lowest minimum distance to the goal.

The Reacher ablation in Table I further shows that both the Markovian state and the gradient-based planner are important. Replacing the Markov latent dynamics with a non-Markov history-based predictor reduces success to 1.0%, removing the multi-step dynamics loss reduces success to 24.5%, and replacing iLQR with CEM reduces success to 57.0%. This suggests that unifying Markov latent dynamics, multi-step training, and gradient-based planning is necessary for reliable long-horizon nominal planning from pixels. Qualitative visualizations and hardware planning figures are in App. G.

B. Robustly-Constrained Safe Latent Space MPC

Rope. For Rope, we introduce an ellipsoidal obstacle that the rope must avoid while moving from start to goal (details in App. D). Table III shows that SLS^2 is the only method that achieves 100% safety, and it obtains the highest robust success rate of 88.57%. In contrast, HJ-filtered, LPB, nominal iLQR, and the constrained iLQR variants all incur safety failures, showing the importance of accounting for latent model error. Qualitative tube and t-SNE visualizations are in Figs. 3, 16.

Reacher. For Reacher, we impose joint-angle constraints on the second joint, requiring it to remain between -2.88 and -2.45 radians while completing the task. We show that SLS^2 remains in the computed tubes in Fig. **??**. Table III shows that both SLS^2 and HJ-filtered achieve 100% robust success, while the constrained iLQR variants are safe but can

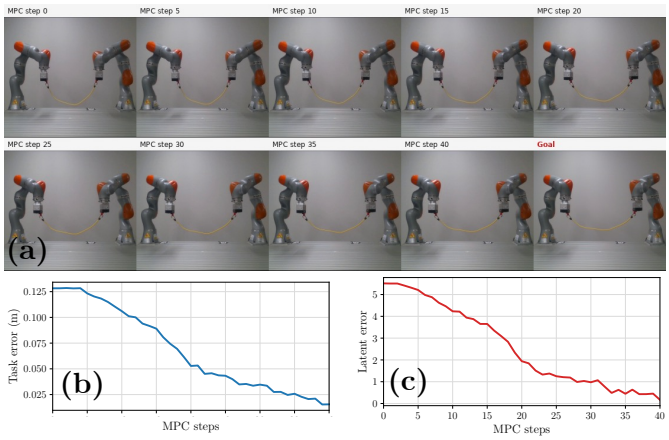

Fig. 2. **Hardware Rope.** (a) Time-lapse of iLQR-based long-horizon planning on real hardware directly from pixels without state estimation. In this example, the bimanual setup has to stretch the rope to reach the goal image. (b) Task and (c) latent space error plots demonstrating convergence to the desired goal state on hardware.

TABLE III

WE EVALUATE SLS^2 AGAINST BASELINES APPROACHES (DETAILS IN APPENDIX E) FOR ROBUST TASK COMPLETION. FOR ROBUST TASK COMPLETION WHERE TRAJECTORIES ARE SAFE AND REACH THE GOAL, SLS^2 PERFORMS THE BEST COMPARED TO ALL BASELINES.

Task	Method	Safety (%) ↑	Success (%) ↑	**Robust Success (%)** ↑	Solve time/step (s) ↓
Reacher	Ours (SLS^2)	100.00	100.00	**100.00**	0.360 ± 0.002
	Constrained iLQR w/ latent ID	100.00	97.22	97.22	0.360 ± 0.002
	Constrained iLQR w/o latent ID	100.00	97.22	97.22	0.363 ± 0.003
	HJ-filtered	100.00	100.0	**100.0**	–
	LPB	0.00	100.00	0.00	–
	Nominal iLQR	0.00	100.0	0.00	–
OGBench (Cube)	Ours (SLS^2)	94.29	97.14	**94.29**	0.359 ± 0.016
	Constrained iLQR w/ latent ID	30.56	86.11	27.78	0.345 ± 0.016
	Constrained iLQR w/o latent ID	25.00	91.67	22.22	0.349 ± 0.013
	HJ-filtered	0.00	88.57	0.00	–
	LPB	0.00	85.71	0.00	–
	Nominal iLQR	0.00	82.86	0.00	–
Rope	Ours (SLS^2)	100.00	88.57	**88.57**	0.650 ± 0.042
	Constrained iLQR w/ latent ID	69.44	91.67	61.11	0.597 ± 0.026
	Constrained iLQR w/o latent ID	77.78	91.67	69.44	0.593 ± 0.019
	HJ-filtered	37.14	100.00	37.14	–
	LPB	57.15	100.00	57.15	–
	Nominal iLQR	54.29	100.00	54.29	–

fail to complete the task. SLS^2 also has nearly the same solve time as constrained iLQR, indicating that the robust tightenings in (9e) add limited computational overhead.

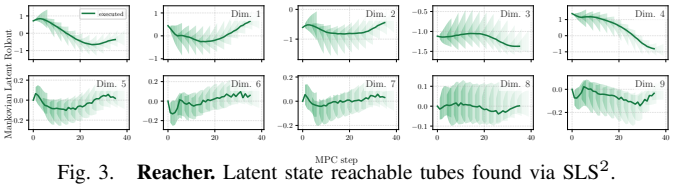

Fig. 3. **Reacher.** Latent state reachable tubes found via SLS^2.

Cube. For OGBench Cube, we constrain the gripper height to remain below 9 cm after grasping the cube. Table III shows that SLS^2 achieves the highest robust success rate, 94.29%, while all baselines have substantially lower safety and robust success. This result shows that SLS^2 can robustly alter the behavior of a learned world-model planner, preventing the nominal tendency to lift the cube unnecessarily high.

VI. CONCLUSION

We propose a Markovian latent space that is amenable to gradient-based planning. We also propose SLS^2, a method for probabilistically-safe latent space planning that uses CP and SLS for uncertainty quantification and propagation. We improve task and robust task success relative to baselines, while ensuring scalability via GPU parallelization.

References

[1] 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, 2016.

[2] James Anderson, John C Doyle, Steven H Low, and Nikolai Matni. System level synthesis. *Annual Reviews in Control*, 47:364–393, 2019.

[3] Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, et al. V-jepa 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025.

[4] Randall Balestriero and Yann LeCun. Lejepa: Provable and scalable self-supervised learning without the heuristics. *arXiv preprint arXiv:2511.08544*, 2025.

[5] Fernando Castaneda, Haruki Nishimura, Rowan Thomas McAllister, Koushil Sreenath, and Adrien Gaidon. In-distribution barrier functions: Self-supervised policy filters that avoid out-of-distribution states. In *Learning for Dynamics and Control Conference*, pages 286–299. PMLR, 2023.

[6] Jeffrey Fang and Glen Chou. Safe large-scale robust nonlinear mpc in milliseconds via reachability-constrained system level synthesis on the gpu. *arXiv preprint arXiv:2604.07644*, 2026.

[7] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

[8] Yann LeCun et al. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, 62 (1):1–62, 2022.

[9] Jing Lei and Larry Wasserman. Distribution-free prediction bands for non-parametric regression. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 76(1):71–96, 2014.

[10] Lucas Maes, Quentin Le Lidec, Damien Scieur, Yann LeCun, and Randall Balestriero. Leworldmodel: Stable end-to-end joint-embedding predictive architecture from pixels. *arXiv preprint arXiv:2603.19312*, 2026.

[11] Zhenjiang Mao, Carson Sobolewski, and Ivan Ruchkin. How safe am i given what i see? calibrated prediction of safety chances for image-controlled autonomy. In *6th Annual Learning for Dynamics & Control Conference*, pages 1370–1387. PMLR, 2024.

[12] Ian M Mitchell, Alexandre M Bayen, and Claire J Tomlin. A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games. *IEEE Transactions on automatic control*, 50(7):947–957, 2005.

[13] Kensuke Nakamura, Lasse Peters, and Andrea Bajcsy. Generalizing safety beyond collision-avoidance via latent-space reachability analysis. *arXiv preprint arXiv:2502.00935*, 2025.

[14] Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. Ogbench: Benchmarking offline goal-conditioned rl. In *International Conference on Learning Representations*, volume 2025, pages 94937–94982, 2025.

[15] Alexander Robey, Haimin Hu, Lars Lindemann, Hanwen Zhang, Dimos V Dimarogonas, Stephen Tu, and Nikolai Matni. Learning control barrier functions from expert demonstrations. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 3717–3724. Ieee, 2020.

[16] Junwon Seo, Kensuke Nakamura, and Andrea Bajcsy. Uncertainty-aware latent safety filters for avoiding out-of-distribution failures. *arXiv preprint arXiv:2505.00779*, 2025.

[17] Uladzislau Sobal, Wancong Zhang, Kyunghyun Cho, Randall Balestriero, Tim GJ Rudner, and Yann LeCun. Learning from reward-free offline data: A case for planning with latent dynamics models. *Advances in Neural Information Processing Systems*, 38:43905–43941, 2026.

[18] Anutam Srinivasan, Antoine Leeman, and Glen Chou. Safety beyond the training data: Robust out-of-distribution mpc via conformalized system level synthesis. In *8th Annual Learning for Dynamics and Control Conference*, 2026.

[19] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[20] Russ Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics, 2019. URL https://drake.mit.edu.

[21] Ryan J Tibshirani, Rina Foygel Barber, Emmanuel Candes, and Aaditya Ramdas. Conformal prediction under covariate shift. *Advances in neural information processing systems*, 32, 2019.

[22] Kim Peter Wabersich and Melanie N Zeilinger. A predictive safety filter for learning-based control of constrained nonlinear dynamical systems. *Automatica*, 129:109597, 2021.

[23] Ying Wang, Oumayma Bounou, Gaoyue Zhou, Randall Balestriero, Tim GJ Rudner, Yann LeCun, and Mengye Ren. Temporal straightening for latent planning. *arXiv preprint arXiv:2603.12231*, 2026.

[24] Grady Williams, Andrew Aldrich, and Evangelos A Theodorou. Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, 40(2):344–357, 2017.

[25] Gaoyue Zhou, Hengkai Pan, Yann Lecun, and Lerrel Pinto. DINO-WM: World models on pre-trained visual features enable zero-shot planning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 79115–79135. PMLR, 13–19 Jul 2025.

APPENDIX

In the following, we provide an overview of our appendices. In App. A, we provide training details for the latent world models, including the offline data collection procedure, shared HDF5 dataset format, optimization settings, and task-specific latent dynamics network architectures. In App. B, we review the system level synthesis (SLS) formulation used to compute robust feedback policies and reachable tubes for the learned latent dynamics, and describe how nominal constraints are tightened to account for calibrated model error. In App. C, we prove the probabilistic containment guarantee for the conformal SLS tubes. In App. D, we provide experimental implementation details, including the benchmark setup, nominal planning objectives, world-model baselines, implementation of Problem (9), calibrated model-error bounds, latent constraint classifiers, in-distribution latent support constraints, safety baselines, and the real-world rope manipulation setup. In App. G, we provide additional experimental results, including nominal trajectory time-lapses, safe versus unsafe trajectory comparisons, latent reachable-tube visualizations, and expanded quantitative tables for nominal planning and robust task-completion baselines.

A. Training Details

a) Offline Dataset: We train the latent world model from offline trajectories collected independently for each task. Each trajectory consists of a sequence of image observations and actions,

$$\tau = \{(o_t, a_t)\}_{t=0}^{T-1}, \tag{10}$$

where o_t is a rendered RGB observation and a_t is the action applied between consecutive observations. The datasets are collected before world model training and no online interaction is used during model learning. The world model is trained only from the observation-action sequences, while additional simulator states, rewards, and task-specific quantities are stored for analysis, visualization, and evaluation.

Across environments, we use task-specific data-generation policies to produce trajectories with sufficient interaction coverage. For expert-control tasks, such as Reacher, trajectories are collected by rolling out a trained expert policy. For manipulation tasks that require structured behavior, such as OGBench Cube, trajectories are generated using an oracle planner that constructs task-space keyframes and executes the corresponding control sequence. For the two-manipulator rope environment, we use randomized task-space trajectory generators, including spline- and waypoint-based policies, to cover a diverse range of rope configurations and manipulator motions. This produces datasets that are not tied to a single evaluation goal, but instead expose the world model to local dynamics over a broad set of feasible interactions.

All datasets are stored in a common HDF5 format. For each episode, we store the rendered pixel observations, applied actions, episode length, episode offset, episode seed, and per-step episode/step indices. Actions are padded with a final invalid entry so that the action array has the same temporal length as the observation array; this makes it straightforward to sample fixed-length training windows while ignoring the last action of each episode. Episodes shorter than a minimum length are discarded. The collection loop either runs for a fixed number of episodes or until a specified target number of transitions is reached. This common storage format allows the same world-model training pipeline to be reused across all environments.

b) Training the World Model: Across experiments, we train the model using AdamW with learning rate 5×10^{-5}, weight decay 10^{-3}, mixed-precision training, and gradient clipping with maximum norm 1.0. Model checkpoints are saved after each epoch, and the same general training pipeline is used across all tasks, with only task-dependent choices such as latent dimension, Markov order, rollout horizon, and loss weights adjusted for each environment.

c) MLP Latent Dynamics Models: All four tasks use the same vision backbone: a tiny ViT encoder on 224×224 RGB images with patch size 14. The encoder output is projected to a task-specific latent $z_t \in \mathbb{R}^{n_z}$. The dynamics model then forms a Markov latent state

$$s_t = [z_t, \Delta z_t, \ldots, \Delta^K z_t] \in \mathbb{R}^{(K+1)n_z},$$

where K is the desired Markov order. A feed-forward MLP predicts the next Markov state from the current Markov state and action,

$$\hat{s}_{t+1} = f_\phi(s_t, u_t).$$

Here the listed action dimension is the number of control channels concatenated into the predictor input; since all runs use frameskip 1, this is also the effective action dimension. Reacher and Push-T use two-dimensional actions, Rope uses a three-dimensional endpoint action, and OGBench Cube uses the five-dimensional environment action vector. The predictor is trained by autoregressively rolling this one-step MLP for `num_preds`=5 future steps.

TABLE IV
MODEL ARCHITECTURE AND STATE DIMENSIONS ACROSS TASKS.

Task	n_z	K	$\dim(s_t)$	$\dim(v_t)$	MLP Size	Straightening
OGBench-Cube	12	1	24	5	512×2	yes
Reacher	5	1	10	2	512×2	no
Push-T	48	2	144	2	512×3	no
Rope	12	1	24	3	512×2	yes

The MLP column reports hidden width times number of hidden layers, with GELU nonlinearities and a linear output layer. The MLP size column only considers the latent dynamics predictor, excluding the ViT encoder and latent projector. The SIGReg term uses the same sketch size in all runs: 17 quadrature knots and 1024 random projections, weighted by $\lambda_{\text{SIG}} = 0.005$. Temporal straightening is active only for the OGBench Cube and Rope runs.

B. System Level Synthesis

a) System Level Synthesis.: We use state-feedback SLS to compute robust feedback policies and reachable tubes for the learned latent dynamics. Let $s_k \in \mathbb{R}^{n_s}$ be the Markov

latent state and $u_k \in \mathbb{R}^{n_u}$ the control. Around a nominal trajectory $(\zeta_{0:T}, v_{0:T-1})$ satisfying $\zeta_{k+1} = f_\phi(\zeta_k, v_k)$, we linearize

$$\Delta s_{k+1} = A_k \Delta s_k + B_k \Delta u_k + E_k w_k, \qquad w_k \in \mathcal{B}^{n_s},$$

where $A_k = \nabla_s f_\phi(\zeta_k, v_k)$, $B_k = \nabla_u f_\phi(\zeta_k, v_k)$, and $E_k \mathcal{B}^{n_s}$ is a calibrated latent prediction-error set.

SLS parameterizes the closed-loop system by response matrices $\Phi^s_{k,j}$ and $\Phi^u_{k,j}$ such that

$$s_k - \zeta_k = \sum_{j=0}^{k-1} \Phi^s_{k,j} w_j, \qquad u_k - v_k = \sum_{j=0}^{k-1} \Phi^u_{k,j} w_j.$$

These responses are valid if they satisfy the finite-horizon SLS constraints

$$\Phi^s_{k+1,j} = A_k \Phi^s_{k,j} + B_k \Phi^u_{k,j}, \quad j < k, \qquad \Phi^s_{k+1,k} = E_k.$$

Then defining $\oplus$ and $\bigoplus$ as the Minkowski sum of two sets (i.e., $\mathcal{A} \oplus \mathcal{B} = \{a + b \mid a \in \mathcal{A}, b \in \mathcal{B}\}$ for sets, $\mathcal{A}$ and $\mathcal{B}$), the corresponding reachable tubes are

$$\mathcal{R}^s_k = \zeta_k \oplus \bigoplus_{j=0}^{k-1} \Phi^s_{k,j} \mathcal{B}^{n_s}, \qquad \mathcal{R}^u_k = v_k \oplus \bigoplus_{j=0}^{k-1} \Phi^u_{k,j} \mathcal{B}^{n_s}.$$

For constraints $\mathcal{F} := \{(s,u) \mid g_i(s,u) + b_i \le 0,\ i = 1, \dots, n_c\}$, we impose robust satisfaction by tightening the nominal constraints. Linearizing g_i around (ζ_k, v_k) gives the sufficient condition

$$g_i(\zeta_k, v_k) + b_i + \sum_{j=0}^{k-1} \left\| \nabla_s g_i(\zeta_k, v_k) \Phi^s_{k,j} + \nabla_u g_i(\zeta_k, v_k) \Phi^u_{k,j} \right\|_2 \le 0,$$

for all i and k. The SLS MPC problem jointly optimizes $(\zeta, v, \Phi^s, \Phi^u)$ subject to nominal dynamics, the SLS response constraints, and these tightened constraints.

C. Proofs

Theorem A.1 (Probabilistic containment of conformal SLS tubes)**.** *Consider the latent closed-loop dynamics $s_{k+1} = f_\phi(s_k, u_k) + e_k$, for $k = 0, \dots, T-1$. Let q be the split conformal quantile of the calibration scores $r_i = e_i^\top \Sigma_\psi(s_i, u_i)^{-1} e_i$ and $e_i := s_i^+ - f_\phi(s_i, u_i)$, computed at per-step miscoverage $\bar{\delta} = \delta/T$. For each closed-loop transition, define the test score $r_k^{\text{test}} := e_k^\top \Sigma_\psi(s_k, u_k)^{-1} e_k$. Assume that, for each k, r_k^{test} is exchangeable with the calibration scores. Let $\Sigma_\psi(s,u) = L_\psi(s,u) L_\psi(s,u)^\top$ and define the local conformal error set $\mathcal{E}(s,u) := \sqrt{q}\, L_\psi(s,u) \mathcal{B}^{n_s}$ as defined in* (5)*. Suppose the SLS controller is synthesized with disturbance matrices E_k such that the realized conformal error sets are contained in the SLS disturbance sets, $\mathcal{E}(s_k, u_k) \subseteq E_k \mathcal{B}^{n_s}$, for $k = 0, \dots, T-1$. If the SLS controller is robustly correct for all disturbances $w_k \in \mathcal{B}^{n_s}$ in $\Delta s_{k+1} = A_k \Delta s_k + B_k \Delta u_k + E_k w_k$, where $\Delta s_k := s_k - \zeta_k$ and $\Delta u_k := u_k - v_k$, then $\mathbb{P}\left[(s_k \in \mathcal{R}^s_k,\ \forall k = 0, \dots, T) \wedge (u_k \in \mathcal{R}^u_k,\ \forall k = 0, \dots, T-1)\right] \ge 1 - \delta$.*

Proof: By split CP, exchangeability gives

$$\mathbb{P}\left[r_k^{\text{test}} \le q\right] \ge 1 - \bar{\delta}$$

for each k. Since

$$r_k^{\text{test}} \le q \quad \Longleftrightarrow \quad e_k \in \sqrt{q}\, L_\psi(s_k, u_k) \mathcal{B}^{n_s} = \mathcal{E}(s_k, u_k),$$

a union bound over the T transitions yields

$$\mathbb{P}\left[e_k \in \mathcal{E}(s_k, u_k),\ \forall k = 0, \dots, T-1\right] \ge 1 - T\bar{\delta} = 1 - \delta.$$

On this event, the assumed containment $\mathcal{E}(s_k, u_k) \subseteq E_k \mathcal{B}^{n_s}$ implies that each realized prediction error can be written as

$$e_k = E_k w_k, \qquad w_k \in \mathcal{B}^{n_s}.$$

Thus the realized closed-loop trajectory is generated by a disturbance sequence inside the uncertainty set used by SLS. Robust correctness of SLS then implies $s_k \in \mathcal{R}^s_k$ for $k = 0, \dots, T$ and $u_k \in \mathcal{R}^u_k$ for $k = 0, \dots, T-1$. Therefore the tube-containment event holds with probability at least $1 - \delta$. ■

D. Experimental Implementation

In the following sections, we discuss the experimental implementations for two of our central contributions: our ***nominal planner*** and our ***robust planner***. Section D1 discusses the experimental setup for our nominal experiments. Then, Section D2 discusses the implementation details of our **novel** gradient-based iLQR planner, which leverages the Markovian latent space to enable high task success rates (cf. Table I for the ablation study). Next, Section D3 details the baseline implementations used to evaluate our nominal planner against. For the safety experiments, Section D4 provides details on how our robust planner, SLS^2, is implemented, and Section E provides details on the safety baselines we compare our approach to. Lastly, Section F discusses our real-world hardware experiment.

1) Experiment Setup: All experiments are run on an Intel i9-14900K CPU, 64 GB RAM, and an NVIDIA RTX 4090 GPU. Nominal planning is evaluated over five random seeds with 40 held-out episodes per seed, while constrained planning is evaluated on 35 randomized start-goal conditions per task. Planners are evaluated over the full task horizon rather than short local segments.

We evaluate on four continuous-control benchmarks spanning low-dimensional reaching, rigid-object manipulation, planar pushing, and deformable-object manipulation. For each task, we train from expert demonstrations and evaluate planning using task-specific goal-reaching criteria.

a) Reacher.: Reacher is the hard variant of the DeepMind Control Suite Reacher task, where a two-joint planar arm must reach a target position. We render 224×224 RGB observations with the target hidden from the image observation. The dataset contains 24,650 expert episodes collected with a Soft Actor-Critic policy. Episodes are collected at 50 Hz for up to 100 control steps and terminate early when the end effector reaches within 0.03 of the target.

b) Cube.: Cube is the OGBench `cube-single-v0` task, where a robot arm must grasp and move a cube to a target 3D position and yaw. We use front-camera pixel observations and continuous 5D actions. The dataset contains 11,227 oracle-planner episodes recorded at 20 Hz for up to 100 steps. Episodes terminate when the cube is within 0.04 m of the target position and within 0.2 rad of the target yaw.

c) Push-T.: Push-T is a 2D planar manipulation task where a circular pusher moves a T-shaped block to a target pose. We use `gym-pusht/PushT-v0` and store RGB observations, pusher positions, and continuous 2D actions. The dataset contains 25,000 expert episodes: 20,000 nominal diffusion-policy rollouts and 5,000 edge-case rollouts. Episodes contain up to 167 recorded steps.

d) Rope.: Rope is a bimanual MuJoCo manipulation environment where two robot arms control the endpoints of a deformable rope. The task exposes the model to nonlinear rope dynamics by moving the rope attachments along randomized task-space trajectories. We collect 15,000 episodes using a cubic-spline policy with six randomized waypoints. Episodes are recorded at 20 Hz, average approximately 80 steps, and are rendered as 224×224 RGB observations.

e) Planning success criteria.: For Reacher, success is measured by the wrapped joint-position distance between the current and goal configurations. The primary threshold is 0.1 in qpos distance, and we also report a stricter full-observation threshold of 0.05 as a diagnostic. For Cube, success requires the cube position to be within 0.04 m of the goal position. For Push-T, success requires the T-block to be within 20 pixels of the goal position with yaw error at most 0.35 rad. For Rope, success requires the task-target distance to reach the dataset goal tolerance of 0.05.

2) Nominal Planning Setup: The nominal MPC planners optimize in the learned Markov latent state. For Reacher, OGBench-Cube, and Rope, the nominal planner is receding-horizon iLQR with horizon $T = 15$, but the quadratic cost matrices are task-specific (where weights scale the identity matrix $I_n \in \mathbb{R}^{n\times n}$):

TABLE V
REACHER, OGBENCH-CUBE, ROPE: NOMINAL iLQR PLANNING HORIZONS AND COST WEIGHTS.

Task	T	Q_T	Q	R
Reacher	15	$5.0I_{10}$	$0.05I_{10}$	$0.1I_2$
OGBench-Cube	15	$15.0I_{24}$	$0.05I_{24}$	$0.5I_5$
Rope	15	$15.0I_{24}$	$0.005I_{24}$	$0.01I_3$

For these iLQR-only planners, the rollout is generated by the learned one-step dynamics $\zeta_{t+1} = f_\phi(\zeta_t, v_t)$, and the open-loop control sequence $\mathbf{v}$ is optimized with the finite-horizon cost

$$J_{\mathrm{iLQR}}(\boldsymbol{\zeta}, \mathbf{v}) = \sum_{t=0}^{T-1} \left(\|\zeta_t - \zeta_g\|_Q^2 + \|v_t\|_R^2 \right) + \|\zeta_T - \zeta_g\|_{Q_T}^2, \tag{11}$$

where ζ_g is the goal latent state. In Reacher, the same expression is used but the cost is computed only on the zeroth-order latent coordinates (i.e., z), so the cost does not directly penalize the finite-difference components of the Markov state. At each environment step, iLQR linearizes the learned dynamics along the current rollout, performs a Riccati-style backward pass to compute local feedback/feedforward updates, line-searches the updated control sequence, executes the first action, and replans from the newly observed image.

For Push-T, the nominal planner uses MPPI to produce a long-horizon warm-start and then uses a short-horizon iLQR tracker to refine the first segment. The MPPI horizon is $H^{\mathrm{mppi}} = 45$, while the iLQR tracking horizon is $T = 15$. The MPPI cost matrices are $Q_T^{\mathrm{mppi}} = 10.0I_{144}$, $Q^{\mathrm{mppi}} = 0.05I_{144}$, and $R^{\mathrm{mppi}} = 0.01I_2$; the iLQR tracking matrices are $Q_T = 10.0I_{144}$, $Q = 1.0I_{144}$, and $R = 0.01I_2$. MPPI samples noisy control sequences, rolls them out through the same latent dynamics, and scores each sampled trajectory with

$$\begin{aligned} J_{\mathrm{MPPI}}(\boldsymbol{\zeta}, \mathbf{v}) = &\sum_{t=0}^{H^{\mathrm{mppi}}-1} \left(\|\zeta_t - \zeta_g\|_{Q^{\mathrm{mppi}}}^2 + \|v_t\|_{R^{\mathrm{mppi}}}^2 \right) \\ &+ \|\zeta_{H^{\mathrm{mppi}}} - \zeta_g\|_{Q_T^{\mathrm{mppi}}}^2. \end{aligned} \tag{12}$$

The sampled costs are converted to rewards by negation and weighted by the MPPI reward temperature when updating the nominal control sequence. The iLQR tracker then optimizes over the shorter horizon $T = 15$ against the MPPI reference trajectory $\bar{\boldsymbol{\zeta}}$ rather than a fixed goal:

$$J_{\mathrm{track}}(\boldsymbol{\zeta}, \mathbf{v}) = \sum_{t=0}^{T-1} \left(\|\zeta_t - \bar{\zeta}_t\|_Q^2 + \|v_t\|_R^2 \right) + \|\zeta_T - \bar{\zeta}_T\|_{Q_T}^2. \tag{13}$$

Thus the MPPI stage adds a sampling-based global search over long control sequences, while the iLQR stage locally smooths and tracks the selected latent trajectory before receding-horizon execution.

3) World Model Baseline Implementation:

a) Training and evaluation protocol.: This section describes the implementation details for the world-model baselines and our planning pipeline. Across all tasks, we follow the training and evaluation workflow of the `stable-worldmodel` codebase whenever possible. We compare against PLDM, DINO-WM, and Le-WM. Unless otherwise specified, each baseline is trained for 10 epochs following the Le-WM training protocol [10], where this training length was reported to provide strong planning performance. When task-compatible pretrained or reusable checkpoints are available, we initialize from those checkpoints and fine-tune on our dataset. In particular, due to the size of the Push-T dataset and the cost of training from scratch, the Le-WM model for Push-T is fine-tuned from a pretrained Push-T object checkpoint. For the customized Rope task, no pretrained checkpoints are available, so all world models are trained from scratch using the same offline dataset.

b) Shared preprocessing and architecture.: All baselines are trained from image observations resized to 224×224

with patch size 14. We use a history length of 3 and a one-step prediction horizon. During planning, the learned one-step latent dynamics are rolled out autoregressively to evaluate multi-step action sequences. The dataset is split into 90% training and 10% validation, and we apply gradient clipping with threshold 1.0. Unless otherwise stated, models are trained with batch size 128. We use frame skip 5 for Reacher, Cube, and Push-T, and frame skip 1 for Rope.

c) PLDM.: For PLDM, we train with AdamW using learning rate 5×10^{-5} and weight decay 10^{-3}. The encoder is a ViT-Tiny model with embedding dimension 192. The predictor has depth 6, 16 attention heads, MLP dimension 2048, head dimension 64, and dropout 0.1. Proprioceptive inputs are disabled. The active loss weights are 18 for the standard-deviation loss, 0.7 for the temporal standard-deviation loss, 12 for the covariance loss, and 0.2 for the temporal alignment loss.

d) DINO-WM.: For DINO-WM, we use a frozen DINOv2-Small visual backbone and train the latent predictor with AdamW using learning rate 5×10^{-4} and no weight decay. The predictor uses the same architecture as PLDM: depth 6, 16 attention heads, MLP dimension 2048, head dimension 64, and dropout 0.1. Actions are encoded with dimension 10. Reacher, Cube, and Rope are trained without proprioceptive encoding, while Push-T uses proprioceptive encoding.

e) Le-WM.: For Le-WM, we train with AdamW using learning rate 5×10^{-5} and weight decay 10^{-3}. The model uses a ViT-Tiny encoder with image size 224, patch size 14, history size 3, prediction horizon 1, and embedding dimension 192. The latent predictor uses the same configuration as the other baselines: depth 6, 16 attention heads, MLP dimension 2048, head dimension 64, and dropout 0.1. For Rope, Le-WM is trained from scratch with batch size 128, frame skip 1, and SigReg regularization with weight 0.09, 17 knots, and 1024 random projections.

For Push-T, we fine-tune the pretrained Le-WM Push-T object checkpoint on the combined Push-T dataset. The Push-T model uses frame skip 5, action dimension 2, and effective action dimension 10. We train for 10 epochs with batch size 110, bfloat16 mixed precision, gradient clipping at 1.0, and the Le-WM latent prediction objective. During fine-tuning, we use SigReg regularization with weight 0.005, 17 knots, and 1024 random projections, together with temporal straightening loss with weight 0.01.

f) Planning evaluation.: For nominal world-model planning baselines, PLDM, DINO-WM, and Le-WM use the standard CEM planner provided by the `stable-worldmodel` implementation. CEM optimizes action sequences in the learned latent space and executes the selected actions in a receding-horizon manner. For our method, we use the same learned latent dynamics as the planning model but replace CEM with our gradient-based latent planner. In the robust setting, this planner is augmented with the SLS-based tube propagation and conformalized latent constraints described in the main text. This setup ensures that baseline performance reflects the default planning pipeline associated with each learned world model, while our method differs only in the planning and robustness layer built on top of the learned latent dynamics.

4) Implementing SLS^2 (Problem (9)) : Below we discuss our implementation of Problem (9). For self-containment, we restate the problem:

$$\min_{\boldsymbol{\zeta},\mathbf{v},\Phi^s,\Phi^u} \quad J(\boldsymbol{\zeta},\mathbf{v}) + J_f(\zeta_T) + H(\Phi^s,\Phi^u) \tag{9a}$$

$$\text{s.t.} \quad \zeta_{k+1} = f_\phi(\zeta_k, v_k), \quad \zeta_0 = \bar{s}_0 \quad k = 0,\dots,T-1, \tag{9b}$$

$$\Phi^s_{k+1,j} = A_k\Phi^s_{k,j} + B_k\Phi^u_{k,j} \quad 0 \le j < k, \quad k = 0,\dots,T-1, \tag{9c}$$

$$\Phi^s_{j+1,j} = E_j \quad j = 0,\dots,T-1, \tag{9d}$$

$$g_i(\zeta_k, v_k) + b_i + \tau_{i,k} \le 0 \quad i = 1,\dots,n_c, \quad k = 0,\dots,T, \tag{9e}$$

$$\mathcal{R}^s_k \subseteq \mathcal{I} \quad k = 0,\dots,T. \tag{9f}$$

In Section D5 we discuss the cost and reference trajectory generation used for the objective in (9a). Next, in Section D6 we discuss how the model error is calibrated. The calibrated error is used to initialize the system level constraints (SLC) in (9c)-(9d). Details on the obstacles and how we train a latent classifier and conformalize it are available in Section D7. In the optimization problem, the classifier is linearized and tightened in (9e). Lastly, we discuss the in-domain ellipsoid constraint, (9f), and how it is formed in Section D8.

5) Defining the Objective: To implement Problem (9), we follow [6, 18] and use a real-time-iteration (RTI) scheme which solves a single SQP (instead of solving several successive SQPs until convergence) to enable faster solve times. To avoid ill-conditioned problems and improve numerical stability, we use the MPPI algorithm [24] to warm-start the MPC solve at each time step. Below, we describe the MPPI warm-start objective, the nominal SLS objective, the associated cost weights, and the modifications used for the Constraint-iLQR baselines.

Let $\zeta_t \in \mathbb{R}^{n_s}$ denote the latent state and ζ_g the goal latent state. MPPI first generates a reference trajectory $\bar{\boldsymbol{\zeta}} := \{\bar{\zeta}_k\}_{k=0}^{T}$ and $\bar{\mathbf{v}} := \{\bar{v}_k\}_{k=0}^{T-1}$ by minimizing the sampling objective

$$J_{\mathrm{MPPI}}(\boldsymbol{\zeta},\mathbf{v}) = \sum_{t=0}^{H^{\mathrm{mppi}}-1} \Big(\|\zeta_t - \zeta_g\|^2_{Q^{\mathrm{mppi}}} + \|v_t\|^2_{R^{\mathrm{mppi}}} + J_{\mathrm{soft}}(\zeta_t) \Big) + \|\zeta_{H^{\mathrm{mppi}}} - \zeta_g\|^2_{Q_T^{\mathrm{mppi}}}. \tag{15}$$

where J_{soft} denotes task-dependent soft penalties used during MPPI sampling. The MPPI horizon H^{mppi} can be greater than T (the tracking horizon used by SLS) to encourage smaller actions at each step. Furthermore, we define $\|x\|^2_M := x^\top M x$ for $x \in \mathbb{R}^n$ and $M \in \mathbb{R}^{n\times n}$. The nominal RTI-SQP/SLS objective then tracks the MPPI reference trajectory $\bar{\boldsymbol{\zeta}}$ and $\bar{\mathbf{v}}$

TABLE VI
MPPI WARM-START AND NOMINAL SLS TRACKING WEIGHTS USED IN OUR EXPERIMENTS.

Task	Objective	Horizon	State Weights	Control Weight
Rope	MPPI	15	$Q^{\text{mppi}} = \text{blkdiag}(0.005I_{12}, 1.0I_{12})$ $Q_T^{\text{mppi}} = \text{blkdiag}(10I_{12}, 1.0I_{12})$	$R^{\text{mppi}} = 1.0I_3$
	SLS	10	$Q = 0.1I_{24}, \quad Q_T = 5.0I_{24}$	$R = 0.2I_3$
Reacher	MPPI	60	$Q^{\text{mppi}} = \text{blkdiag}(0.005I_5, 1.0I_5)$ $Q_T^{\text{mppi}} = \text{blkdiag}(100I_5, 1.0I_5)$	$R^{\text{mppi}} = 0.01I_2$
	SLS	5	$Q = \text{blkdiag}(0.005I_5, 1.0I_5)$ $Q_T = \text{blkdiag}(5.0I_5, 0.9I_5)$	$R = 0.1I_2$
OGBench-Cube	MPPI	10	$Q^{\text{mppi}} = \text{blkdiag}(0.05I_{12}, 0.10I_{12})$ $Q_T^{\text{mppi}} = \text{blkdiag}(15I_{12}, 1.0I_{12})$	$R^{\text{mppi}} = 1.0I_5$
	SLS	10	$Q = \text{blkdiag}(0.05I_{12}, 0.05I_{12})$ $Q_T = \text{blkdiag}(1.0I_{12}, 1.0I_{12})$	$R = 0.5I_5$

TABLE VII
MPPI HYPERPARAMETERS AND SOFT-PENALTY WEIGHTS USED FOR WARM-START TRAJECTORY GENERATION.

Task	Samples	Updates	Reward Weight	Noise Scale	β	λ_{ID}	λ_{obs}
Rope	2048	6	25	0.20	0.65	20	0
Reacher	512	5	20	0.15	0.70	0	1000
OGBench-Cube	512	5	30	0.20	0.60	10	100

over the MPC horizon T:

$$J_{\text{SLS}}(\boldsymbol{\zeta}, \mathbf{v}) = \sum_{t=0}^{T-1} \left(\|\zeta_t - \bar{\zeta}_t\|_Q^2 + \|v_t - \bar{v}_t\|_R^2 \right) + \|\zeta_T - \bar{\zeta}_T\|_{Q_T}^2. \tag{16}$$

Our cost matrices penalizing state tracking $Q_T, Q, Q_T^{\text{mppi}}, Q^{\text{mppi}} \succeq 0$ and control penalties $R, R^{\text{mppi}} \succ 0$ are summarized in Table VI. While interpretability is minimal in the latent space, we leverage the Markovian structure to apply different penalties to position states (corresponding to the zeroth-order state) and velocity states (corresponding to the first-order state). Compactly, we use $I_n \in \mathbb{R}^{n\times n}$ as the n-dimensional identity matrix, and for $A \in \mathbb{R}^{n\times n}, B \in \mathbb{R}^{m\times m}$ we denote $\text{blkdiag}(A, B)$ as the block diagonal matrix.

In (15), we also add a soft penalty, J_{soft}, defined as

$$\begin{aligned} J_{\text{soft}}(\zeta) = \lambda_{\text{ID}} & \underbrace{\left[\max\{\rho_{\text{ID}}(\zeta) - 1, 0\}\right]^2}_{\text{Latent Ellipsoid}} \\ & + \lambda_{\text{obs}} \underbrace{\left[\text{softplus}\big(\eta - c_\psi(\Pi_z \zeta)\big)\right]^2}_{\text{Obstacle}}. \end{aligned} \tag{17}$$

where $\rho_{\text{ID}}(\zeta)$ is the conformal latent ellipsoid defined in (6) and $c_\psi(\Pi_z\zeta)$ is the obstacle classifier score evaluated on the zeroth-order latent component. λ_{obs} and λ_{ID} are hyperparameters controlling the strength of the respective penalties. The soft penalty in (17) encourages the reference trajectory to satisfy the constraints. Notably, these references are not guaranteed to robustly satisfy the constraints, and thus SLS is still required to enforce robust constraint satisfaction. The remaining MPPI hyperparameters alongside the values of λ_{obs} and λ_{ID} used for each task are summarized in Table VII.

6) Calibrated Error Bounds: To calibrate the error bounds we split our dataset $\mathcal{D}_{\text{cal}}$ in half (i.e., $\mathcal{D}_{\text{cal}}^{(1)}$ and $\mathcal{D}_{\text{cal}}^{(2)}$ such that $|\mathcal{D}_{\text{cal}}^{(1)}| = |\mathcal{D}_{\text{cal}}^{(2)}|$. For our experiments, $\mathcal{D}_{\text{cal}}$ is constructed using the same expert rollouts used to train the models, as the transitions encountered in $\mathcal{D}_{\text{cal}}$ are distributionally similar to the transitions encountered during planning time, preserving the exchangeability assumption. For Reacher, Rope, and OGBench-Cube, $|\mathcal{D}_{\text{cal}}| = 200,000$, $|\mathcal{D}_{\text{cal}}| = 80,000$, and $|\mathcal{D}_{\text{cal}}| = 98,800$ respectively. The first partition, $\mathcal{D}_{\text{cal}}^{(1)}$, is used to learn the error covariance, $\Sigma_\psi(s, u)$, structure by minimizing the MGNLL loss (4), while the second half is used to perform the conformal calibration described in Section IV-B. For the Reacher task, we trained $\Sigma_\psi(s, u)$ as a neural network with 2 hidden layers (128 neurons per hidden layer) at a learning rate of 0.0003 using a cosine annealing learning rate scheduler to stabilize training. The model directly outputs the Cholesky factor to avoid computing it in real time. For the Rope and OGBench-Cube tasks, we defined $\Sigma_\psi(s, u) := \Sigma_{\text{cov}}$ as a fixed parameter. In this case, the loss is minimized by computing $\Sigma_{\text{cov}} = \frac{1}{|\mathcal{D}_{\text{cal}}^{(1)}|} \sum_{(s_i, u_i, s_i^+) \in \mathcal{D}_{\text{cal}}^{(1)}} (s_i^+ - f_\phi(s_i, u_i))(s_i^+ - f_\phi(s_i, u_i))^\top$ as the sample error covariance. For the conformal calibration we set $\alpha = 0.1, 0.4,$ and 0.1 for Reacher, Rope, and OGBench-Cube respectively.

7) Training and Conformalizing the Constraint Classifier: For each constrained task, we train a latent-space classifier from a balanced dataset of rendered constraint-violating and non-violating observations. Let

$$\mathcal{D}_{\text{obs}} = \{(o_i, y_i)\}_{i=1}^N, \qquad y_i \in \{0, 1\},$$

where, following the notation in the main text, $y_i = 0$ denotes a constraint-violating observation and $y_i = 1$ denotes a non-violating observation. The task-specific geometric rule used to assign y_i is described below. In the implementation, the stored source label is the complementary indicator $\tilde{b}_i = 1 - y_i$, so $\tilde{b}_i = 1$ marks the violating class used for conformal calibration. Each image is encoded with the frozen world-model encoder $z_i = \text{enc}_\theta(o_i) \in \mathbb{R}^{n_z}$, and the latent features are standardized using the training-split mean and standard deviation. Let $\mathcal{I}_{\text{tr}}$, $\mathcal{I}_{\text{val}}$, and $\mathcal{I}_{\text{cal}}$ denote the index sets for the training, validation, and conformal calibration splits of $\mathcal{D}_{\text{obs}}$, respectively. We then train a small MLP score model $c_\psi : \mathbb{R}^{n_z} \to \mathbb{R}$ with the convention that larger scores indicate safer states. Equivalently, the signed labels are

$$\tilde{y}_i = \begin{cases} -1, & y_i = 0, \\ +1, & y_i = 1, \end{cases}$$

and the classifier is optimized with the hinge loss

$$\mathcal{L}_{\text{obs}}(\psi) = \frac{1}{N_{\text{tr}}} \sum_{i \in \mathcal{I}_{\text{tr}}} \max\big(0, m - \tilde{y}_i c_\psi(z_i)\big).$$

Thus, before conformal calibration, $c_\psi(z) > 0$ is classified as non-violating and $c_\psi(z) \le 0$ is classified as violating.

We calibrate the classifier using only held-out violating examples. Let $\mathcal{I}_{\text{cal}}^- = \{i \in \mathcal{I}_{\text{cal}} : y_i = 0\}$ and define the nonconformity score

$$r_i = \max\{0, c_\psi(z_i)\}, \qquad i \in \mathcal{I}_{\text{cal}}^-.$$

This score is zero for a violating calibration point that is already on the violating side of the nominal classifier and

positive only when the point is incorrectly placed on the safe side. For miscoverage level δ, we choose

$$\eta = \text{Quantile}_{\lceil (n_{\text{cal}}^- + 1)(1-\delta) \rceil} \left(\{ r_i : i \in \mathcal{I}_{\text{cal}}^- \} \cup \{\infty\} \right),$$

where $n_{\text{cal}}^- = |\mathcal{I}_{\text{cal}}^-|$. The conformalized violating set is

$$\mathcal{O}_{\text{conf}} = \{ z : c_\psi(z) < \eta \}.$$

Under the exchangeability assumption for future violating observations and the calibration split, this construction covers a future violating latent with probability at least $1 - \delta$.

To match the SLS notation that is used in the main text, we convert the classifier score into a constraint function of the following form:

$$\begin{aligned} g_\psi(s) &= -c_\psi(z), \qquad z = \Pi_z s, \\ \mathcal{S}_{\text{safe}} &= \{ s : g_\psi(s) \le -\eta \} \\ &= \{ s : -c_\psi(\Pi_z s) \le -\eta \}. \end{aligned} \tag{18}$$

Here Π_z extracts the zeroth-order latent embedding from the Markov latent state $s = [z, \Delta z, \ldots, \Delta^K z]$. This is the sign conversion used by the SLS planner: the raw network score c_ψ is larger for safe states, while the MPC constraint is written in the paper's standard form $g_\psi(s) \le 0$. The same threshold also appears in the MPPI warm-start soft penalty as a smooth relaxation of $\eta - c_\psi(z) \le 0$.

a) Translating Constraints for SLS: Using the obstacle constraint we derived, it can be plugged into SLS and robustly satisfied by tightening the constraint. For nominal constraint satisfaction at step k, in the trajectory ($0 \le k \le T$) we require, $\bar{g}_\psi(\zeta_k, v_k) \le -\eta$, where we define $\bar{g}_\psi(\zeta_k, v_k) = g(z_k)$ such that z_k is the zeroth-order component of the Markov state ζ_k. We then apply the constraint tightening as a function of the linearized constraint and tube (as discussed in Appendix B) to robustly satisfy the constraint for all Markov states, $s_k \in \mathcal{R}_k^s$ and control inputs $u_k \in \mathcal{R}_k^u$ in the tube. In our implementation we also have box constraints for the control input (i.e., $u_k \le u_{\text{max}}$ and $-u_k \le -u_{\text{min}}, \forall\, 0 \le k \le T-1$). Accounting for both upper and lower bound control constraints, this results in a total of $2n_u + 1 = n_c$ constraints in our problem. For succinctness, we collect all the constraints for each time step as follows:

$$g_i(\zeta_k, v_k) + b_i + \tau_{i,k} \le 0, \qquad i = 1, \ldots, n_c,\ k = 0, \ldots, T, \tag{19}$$

where $g_i(\zeta_k, v_k)$ is the constraint function, b_i is the nominal bound (adjusted by η for conformalized obstacle constraint indices), and $\tau_{i,k}$ is the constraint tightening.

b) Task-specific label rules and classifier sizes:

Reacher: For Reacher, the violating set is the joint-space box

$$\mathcal{Q}_{\text{obs}} = \{ q = (q_1, q_2) : 0 \le q_1 \le 3.1415,\ -2.88 \le q_2 \le -2.45 \}.$$

This region corresponds to configurations where the second link folds back very close to the first link. In the simulator this is a pose/joint-space obstacle constraint, while in planning it is enforced as a learned latent-space constraint from rendered images. We sample configurations inside $\mathcal{Q}_{\text{obs}}$ as violating examples and configurations outside the box as non-violating examples, then render each configuration from the fixed planner camera to obtain the balanced classifier dataset. The Reacher classifier uses a 5-dimensional latent input and an MLP with one hidden layer of width 6 followed by a scalar output, using GELU nonlinearities.

OGBench: For Cube, the classifier enforces a height constraint on the grasped cube/effector state. We synthesize grasped cube configurations, render front-camera observations with the target hidden, and label an observation as violating when the measured gripper height is above the configured threshold

$$h_{\text{grip}} > h_{\max}, \qquad h_{\max} = 0.09.$$

Samples with $h_{\text{grip}} \le 0.09$ are treated as non-violating. The sampled cube positions use the task-space bounds $x \in [0.30, 0.50]$, $y \in [-0.25, 0.25]$, and $z \in [0.02, 0.30]$. If precomputed dataset splits are unavailable, the script creates stratified source-training and calibration splits before carving out a validation subset. The Cube height classifier uses a 12-dimensional latent input and a linear score model $c_\psi(z) = w_\psi^\top z + b_\psi$, i.e., a single learned separating vector and bias.

Rope: For Rope, each sampled task state is $x = (r, h, w)$, where r is the endpoint reach coordinate, h is the endpoint height, and w is the rope width. The task-space bounds are

$$r \in [-0.10, 0.30], \qquad h \in [1.16, 1.35], \qquad w \in [0.20, 0.75].$$

The obstacle is a speedbump-shaped barrier in the (r, h) task-space slice, placed on the tabletop over the active reach interval $[0.05, 0.15]$. Formally, it is the upper half of an ellipse with center reach $r_c = 0.10$, half-width $a = 0.05$, base height $z_{\text{base}} = 0.75$, and peak height $z_{\text{peak}} = 0.91$:

$$b(r) = z_{\text{base}} + (z_{\text{peak}} - z_{\text{base}}) \sqrt{\max\left\{ 0, 1 - \left(\frac{r - r_c}{a} \right)^2 \right\}}.$$

Thus the bump rises 0.16 above the table at its midpoint and returns to the table height at $r = 0.05$ and $r = 0.15$. Because the rope sags as a function of width, we estimate a sag profile $s(w)$ from simulated proxy-rope configurations and use

$$\ell(x) = h - s(w)$$

as the approximate lowest rope height. A rope state is labeled violating when

$$r \in [0.05, 0.15] \quad \text{and} \quad \ell(x) \le b(r),$$

and non-violating otherwise. This gives a formal task-space obstacle that visually behaves like a smooth speedbump the rope must clear; the classifier then learns the corresponding latent-space constraint from fixed-camera renderings. The Rope classifier uses a 12-dimensional latent input and an MLP with one hidden layer of width 12 followed by a scalar output, using GELU nonlinearities.

8) Calibrating the Latent Support: To further validate the exchangeability assumption at planning time, we estimate and conformalize the support of our calibration dataset. SIGReg loss structures the latent space such that the raw latent vectors follow an isotropic multivariate Gaussian distribution with unit variance. Thus, the Markovian latent vector also inherits the Gaussian structure, but is not guaranteed to be isotropic due to correlation between successive latent states. Hence, we learn both the shape and size of the ellipsoidal geometry characterizing the support of our calibration dataset, and constrain our planner to remain within it. We refer to this constraint as the in-distribution (ID) latent ellipsoid. To obtain this constraint we use the same calibration dataset, $\mathcal{D}_{\text{cal}}$ and partitions, $\mathcal{D}_{\text{cal}}^{(1)}/\mathcal{D}_{\text{cal}}^{(2)}$from Section D6. We compute the sample covariance of the states, Σ_{ID}, using states, s_i, in $\mathcal{D}_{\text{cal}}^{(1)}$, and fix the mean $\mu = 0$. We then compute the Mahalanobis distance $d_i = s_i^\top \Sigma_{\text{ID}}^{-1} s_i$ for each state, $s_i \in \mathcal{D}_{\text{cal}}^{(2)}$. Lastly, we compute the $(1-\alpha_{\text{ID}})$-quantile q_{ID} of distances d_i to form the boundary of our support, as defined in (6). Formally, this provides a boundary between in-distribution points which we assume to be $(1-\alpha_{\text{ID}})\cdot 100\%$ and outlier points corresponding to $\alpha_{\text{ID}} \cdot 100\%$ of points in the calibration dataset, $\mathcal{D}_{\text{cal}}^{(2)}$. For each of our experiments we set $\alpha_{\text{ID}} = 0.1$.

E. Safety Baseline Implementation

a) HJ-filtered safety baseline.: We implement the HJ-filtered baseline as a latent safety shield applied on top of a nominal planner. We use the same encoder and latent dynamics notation as in the main method: an observation o_t is encoded as

$$z_t = \text{enc}_\theta(o_t),$$

and the planner state is the Markov latent state

$$s_t^z = [z_t, \Delta z_t, \ldots, \Delta^K z_t].$$

Actions are represented in the normalized action coordinates used by the learned world model:

$$\bar{u} = \text{clip}\left(\frac{u-\mu_u}{\sigma_u}, u_{\min}, u_{\max}\right), \qquad u_{\min} = -2, \quad u_{\max} = 2.$$

The HJ baseline rolls out the learned one-step latent dynamics

$$\hat{s}_{t+1}^z = f_\phi(s_t^z, \bar{u}_t).$$

The HJ baseline uses its own obstacle classifier convention, inherited from the HJ-filtered baseline implementation, which is distinct from the constraint classifier notation used by our SLS planner in the main body. To avoid overloading g_ψ, we denote the HJ classifier score by c_ω. Its signed safety margin is

$$\ell_{\text{HJ}}(s) = c_\omega(s) - \tau_{\text{HJ}},$$

where τ_{HJ} is the HJ classifier decision threshold, calibrated separately from the SLS constraint margin η. In our experiments, τ_{HJ} is selected using the same held-out calibration split used for conformal safety calibration. Under this baseline convention, positive values indicate states classified as safe and non-positive values indicate states classified as unsafe. Depending on the experiment configuration, the raw margin is optionally transformed by a monotone squashing function,

$$\ell_{\text{HJ}}(s) \leftarrow \tanh(\ell_{\text{HJ}}(s)) \quad \text{or} \quad \ell_{\text{HJ}}(s) \leftarrow \tanh(2\ell_{\text{HJ}}(s)).$$

We train a PyHJ/Avoid-DDPG recovery policy π_{HJ} and critic V_{HJ} in the learned latent dynamics. Episodes are initialized from cached latent states. For a sampled action $\bar{u}_t$, the imagined next latent state is

$$\tilde{s}_{t+1}^z = f_\phi(s_t^z, \bar{u}_t),$$

and the transition signal is the next-state safety margin:

$$r_t = \ell_{\text{HJ}}(\tilde{s}_{t+1}^z).$$

An episode terminates when $\ell_{\text{HJ}}(\tilde{s}_{t+1}^z) \le 0$. The learned critic therefore estimates recoverability under the latent recovery policy. At deployment, the HJ barrier score is

$$B_{\text{HJ}}(s) = \min\{\ell_{\text{HJ}}(s), V_{\text{HJ}}(s)\}.$$

Given a nominal raw action u_t^{nom} from the base planner, the filter first normalizes it to $\bar{u}_t^{\text{nom}}$ and predicts

$$\hat{s}_{t+1}^{z,\text{nom}} = f_\phi(s_t^z, \bar{u}_t^{\text{nom}}).$$

The nominal action is executed only if the current state is classified as safe and the predicted next state is accepted by the learned barrier. Otherwise, the filter replaces it with the HJ recovery action:

$$\bar{u}_t = \begin{cases} \pi_{\text{HJ}}(s_t^z), & \text{if } \ell_{\text{HJ}}(s_t^z) \le 0 \text{ or } B_{\text{HJ}}(\hat{s}_{t+1}^{z,\text{nom}}) \le \epsilon, \\ \bar{u}_t^{\text{nom}}, & \text{otherwise.} \end{cases}$$

We use $\epsilon = 0$ unless otherwise specified. The selected normalized action is then unnormalized before execution:

$$u_t = \bar{u}_t \sigma_u + \mu_u.$$

b) Latent Policy Barrier baseline.: We also evaluate an adapted Latent Policy Barrier (LPB) baseline, which incorporates a nearest-neighbor in-distribution barrier as a regularizer in the nominal iLQR planner. First, we build a bank of safe Markov latent states from the training dataset. Each image is encoded as $z_t = \text{enc}_\theta(o_t)$ and converted into the same Markov latent state s_t^z used by the planner. Unsafe states are removed using the analytical ground-truth obstacle check used to label obstacle data. Thus, the prototype bank contains only latent states whose corresponding environment configurations are collision-free under the geometry-based safety test.

The remaining safe states are whitened,

$$\tilde{s} = \frac{s - \mu_{\text{LPB}}}{\sigma_{\text{LPB}}},$$

and M prototype states $\mathcal{P} = \{p_j\}_{j=1}^M$ are sampled from the whitened safe set. We calibrate a distance threshold ρ as a chosen quantile of nearest-prototype distances on held-out non-prototype safe states. At planning time, each predicted latent state is scored by

$$d_{\text{LPB}}(s) = \min_{p_j \in \mathcal{P}} \left\| \frac{s - \mu_{\text{LPB}}}{\sigma_{\text{LPB}}} - p_j \right\|_2.$$

The LPB penalty is zero inside the calibrated safe tube and grows quadratically outside it:

$$c_{\mathrm{LPB}}(s) = [d_{\mathrm{LPB}}(s) - \alpha_{\mathrm{LPB}}\rho]_+^2 ,$$

where α_{LPB} is the threshold scale and $[\cdot]_+ = \max(\cdot, 0)$.

Unlike the HJ filter, LPB does not override actions after planning. Instead, it augments the iLQR trajectory objective. For a nominal latent trajectory $\boldsymbol{\zeta} = \{\zeta_k\}_{k=0}^{T}$ and normalized controls $\mathbf{v} = \{v_k\}_{k=0}^{T-1}$, the LPB-regularized objective is

$$\begin{aligned} J_{\mathrm{LPB}} = &\sum_{k=0}^{T-1} \left(q_s \|\zeta_k - \zeta_g\|_2^2 + q_u \|v_k\|_2^2 \right) \\ &+ q_f \|\zeta_T - \zeta_g\|_2^2 + \frac{\lambda_{\mathrm{LPB}}}{T} \sum_{k=0}^{T-1} c_{\mathrm{LPB}}(\zeta_k). \end{aligned} \tag{20}$$

where ζ_g is the goal Markov latent state. In our implementation, the LPB term is differentiable through the latent trajectory, and its gradient and Hessian are included in the iLQR backward pass. Thus, LPB biases the planner toward trajectories that remain close to the empirical distribution of safe latent states while still optimizing the original goal-reaching cost.

c) Constrained iLQR.: For our constrained iLQR implementation we solve Problem (9) without constraints (9c)-(9d), and remove the constraint tightening, $\tau_{i,k}$ in (9e). This results in a constrained safety-aware problem, but does not require a trajectory to robustly satisfy constraints–unlike SLS. Furthermore we have two variants, one with the in-domain latent ellipsoid constraint and one without. Outside of these changes, both of these approaches still use the same parameters and configurations as the SLS implementation.

F. Hardware Implementation

Our hardware experiment is the physical equivalent of our simulated bimanual rope task.

a) Hardware:: The bimanual setup consists of two KUKA iiwa arms placed 1.5 meters apart, each with a Schunk end-effector. The rope is attached to the arms via two 3D-printed clips that are screwed into the Schunk end-effector. Figures 8 and 9 provide visualizations of the hardware setup. To capture images, we used an Intel RealSense D415 camera (and use only the RGB images).

b) Data Collection:: We parameterize the rope manipulation task with a three-dimensional task space: reach, the shared forward/backward displacement of the end effectors; height, their shared vertical position; and width, their lateral separation. Each task-space command is converted to bimanual joint targets using Drake [20] inverse kinematics, with the two grippers moved symmetrically about the rope centerline. Data is collected in shards. At the start of each shard, the arms return to a fixed home pose, which defines the nominal task origin. We then sample 25 random task-space knots within the workspace bounds, enforcing a minimum separation between consecutive knots. We linearly interpolate between each pair to produce a dense sequence of intermediate task targets. This waypoint-based trajectory gives broad workspace coverage while keeping each low-level motion small and feasible for hardware execution. We considered smoother spline trajectories, as used in the sibling simulation experiment, but use piecewise-linear waypoint segments on the real system because they make the commanded task deltas explicit, easier to validate for collision and speed limits, and easier to execute with settling between samples. For each interpolated target, the corresponding joint command is executed, the arms are allowed to settle, and an RGB image is captured. We record the planned task-space delta together with the measured end-effector state and executed task-space delta, producing image-action pairs for training the dynamics model.

c) Model Training:: We collect our data with a data structure similar to that of all our data collection scripts for simulation. This enables us to swap in our hardware data directly with our simulation data. All the model parameters are equivalent to the corresponding simulation experiment for nominal planning.

d) Planning:: For nominal planning, we used MPPI to warm-start trajectory generation at each step and then iLQR to track it, similar to the Push-T approach described in Section D2. Since the low-level controller requires a minimal actuation in order to move, we define

$$J_{\mathrm{act}}(\mathbf{v}) = \lambda_{\mathrm{act}} \sum_{t=0}^{T-1} \left[\max\left(0, \frac{\delta_{\min} - \|v_t\|_\infty}{\delta_{\min}}\right) \right]^2 , \tag{21}$$

where $\delta_{\min}$ is the minimum actuation. Since the bimanual rope manipulation task has 3 actions controlling the distance between the two end-effectors (width), the height of the two end-effectors, and the depth of two end-effectors (reach), we only require one of the actions to meet the minimum actuation requirement. Thus, we compute the infinity norm of the nominal action $\|v_t\|_\infty$ to get the maximum value among the 3 actions and penalize it if it is below the minimum actuation threshold, $\delta_{\min}$. Thus, the cost used for the real-world experiment $J_{\mathrm{mppi}}^{\mathrm{real}}(\boldsymbol{\zeta}, \mathbf{v})$ is,

$$J_{\mathrm{MPPI}}^{\mathrm{real}}(\boldsymbol{\zeta}, \mathbf{v}) = \underbrace{J_{\mathrm{MPPI}}(\boldsymbol{\zeta}, \mathbf{v})}_{\text{Eq. (12)}} + J_{\mathrm{act}}(\mathbf{v}). \tag{22}$$

Similar to the simulation experiment (for Push-T), we use iLQR to track the MPPI-generated reference trajectory, and provide the next control input to apply at each MPC step. The MPPI parameters and iLQR parameters are available in Table VIII.

TABLE VIII
MPPI AND iLQR PLANNING PARAMETERS FOR THE REAL-WORLD ROPE MANIPULATION TASK.

Module	Parameter	Value	Parameter	Value
MPPI Warm-start	Horizon (H^{mppi})	60	Control Weight (R^{mppi})	$0.02I_3$
	Samples	512	Reward Weight	25.0
	Updates	6	Noise Scale	0.2
	State Stage Weight (Q^{mppi})	$1.5I_{24}$	Filter β	0.65
	State Terminal Weight (Q_T^{mppi})	$1000.0I_{24}$		
Actuation Penalty	Minimum Threshold ($\delta_{\min}$)	0.006	Penalty Weight (λ_{act})	10.0
iLQR Tracking	Horizon (T)	10	State Terminal Weight (Q_T)	$50.0I_{24}$
	State Stage Weight (Q)	$0.5I_{24}$	Control Weight (R)	$0.02I_3$

G. Extended Results

In this section, we provide additional experimental results beyond those reported in the main text. Figures 4 - 7 contain our nominal planning time-lapses where we visualize our goal-reaching performance. Figures 8 - 9 contain our nominal planning time-lapses for our bimanual rope manipulation hardware experiment and figure 10 validates convergence of our planner to the goal configuration in the task space for real hardware. Figures 11 - 13 contain time-lapse comparisons of safe vs. unsafe plans demonstrating SLS^2's ability to enforce safety. Figures 14 - 15 visualize the reachable tubes in the latent space. Lastly, we include the LPB safety baseline on Reacher, the LPB safety baseline on Cube, and the Le-WM nominal planning baseline on Push-T.

H. Nominal Trajectory Time-lapses

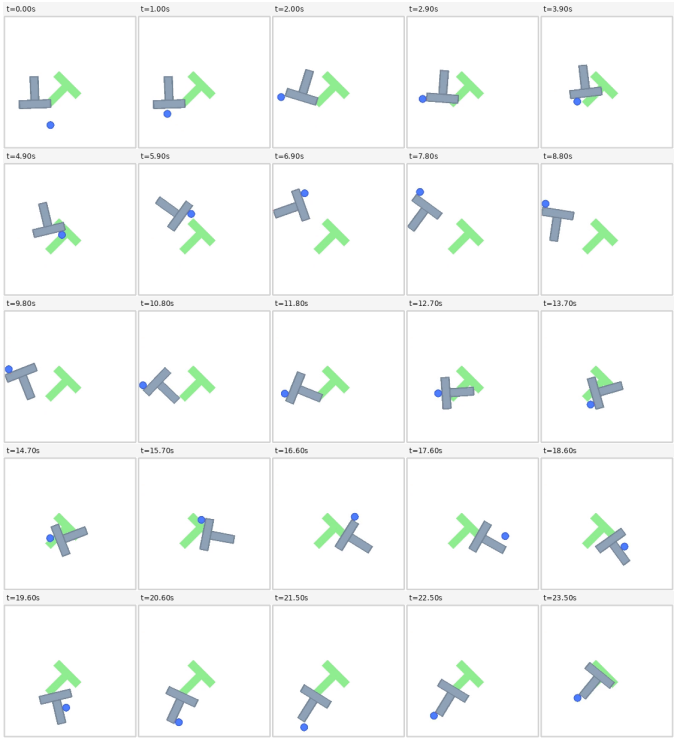

Fig. 4. **Push-T.** iLQR-based long-horizon planning, with MPPI warm-start. Our planner is able to successfully make and break contact with the T while executing complex maneuvers in order to push the T to the goal.

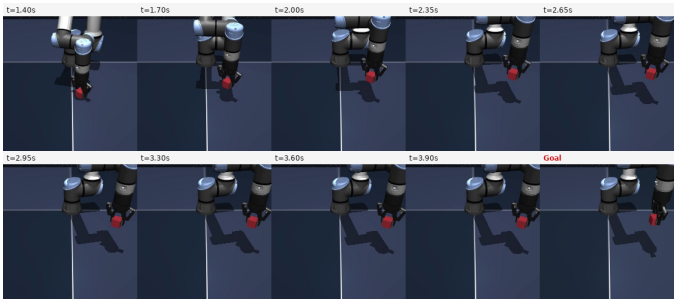

Fig. 5. **Cube.** iLQR-based long-horizon planning. The planner successfully moves the cube to the goal pose without dropping it, unlike the baseline approaches.

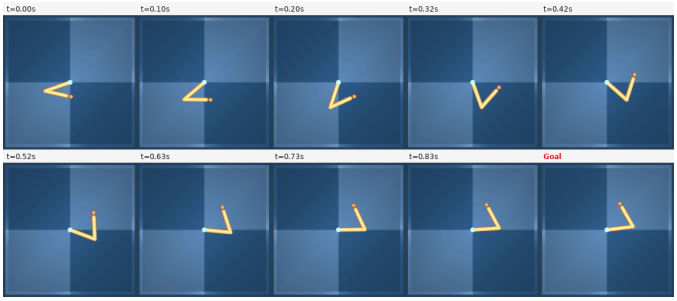

Fig. 6. **Reacher.** iLQR-based long-horizon planning. Similar to Figure 5, our gradient-based planner is able to successfully move to the goal in an efficient manner.

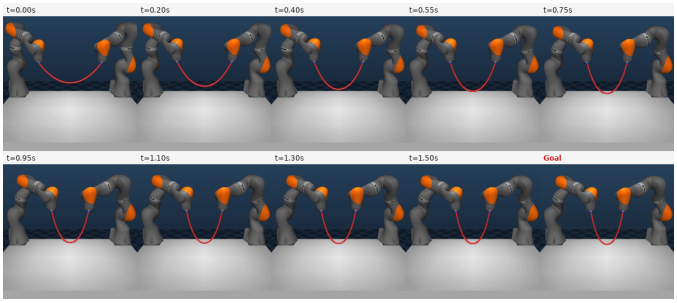

Fig. 7. **Rope.** iLQR-based long-horizon planning. The planner is able to correctly manipulate the deformable rope to reach the goal configuration.

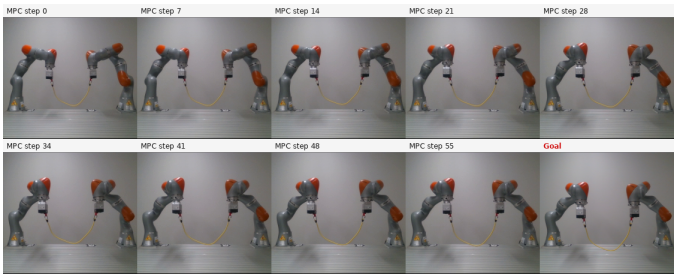

Fig. 8. **Real.** iLQR-based long-horizon planning on real hardware directly from pixels without state estimation. In this example, we demonstrate the arms moving forward to reach the goal image.

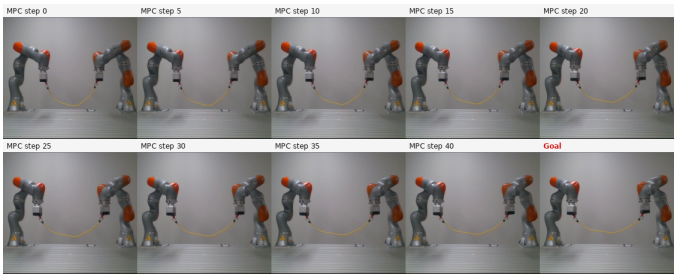

Fig. 9. **Real.** Another example of iLQR-based long-horizon planning on real hardware directly from pixels without state estimation. In this example, the bimanual setup has to stretch the rope to reach the goal image.

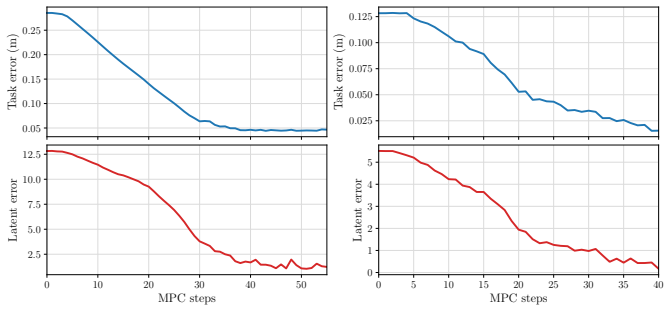

Fig. 10. **Real.** Task and latent space error plots demonstrating convergence to the desired goal state on hardware. Plot for Figure 8 on the left and plot for Figure 9 on the right.

I. Safe vs. Unsafe Time-lapses

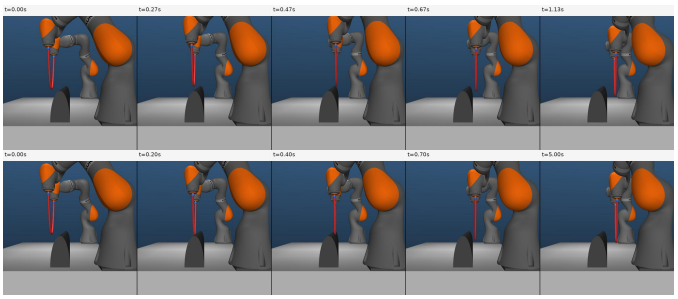

Fig. 11. **Rope.** Safe (top) plan vs. unsafe (bottom) plan time-lapse. The SLS^2 planner successfully maneuvers the rope over the obstacle while reaching the goal image, unlike the nominal planner which collides with the obstacle.

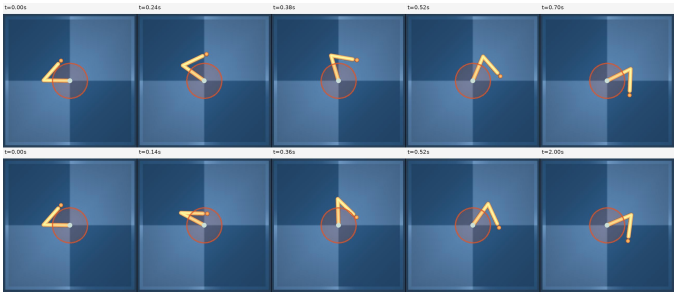

Fig. 12. **Reacher.** Safe (top) plan vs. unsafe (bottom) plan time-lapse. Similar to Figure 11, the nominal planner traverses through the "unsafe" region and violates the pose constraint, while SLS^2 avoids the unsafe region.

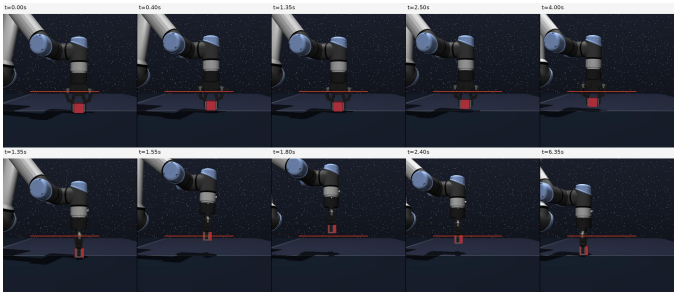

Fig. 13. **Cube.** Safe (top) plan vs. unsafe (bottom) plan time-lapse. SLS^2 maintains the cube below the height threshold while reaching the goal.

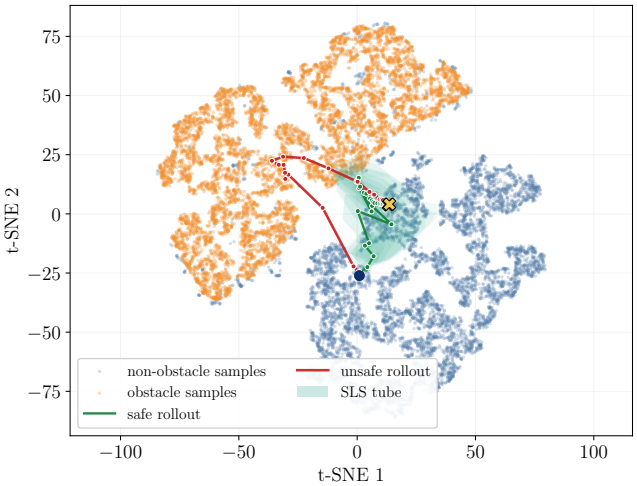

Fig. 16. **Robust planning; Rope.** t-SNE visualization of the rollout. Our tubes stay predominantly in the safe region; the small overlap with the obstacle region is due to the lossy 2D t-SNE projection.

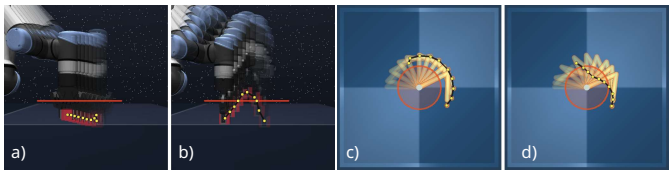

Fig. 17. **Robust planning**. **Cube (a-b) and Reacher (c-d)**. SLS^2 maps pixels directly to robustly-safe actions via latent robust MPC, ensuring robust constraint satisfaction (red) compared to unconstrained baselines. **a, b)** The Cube task with and without SLS^2 enforcing the height threshold. **c, d)** The Reacher task with (c) and without (d) SLS^2 enforcing the angle constraint on the second joint.

J. Latent Reachability Tubes

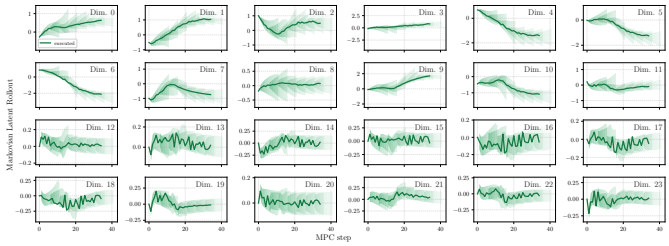

Fig. 14. **Rope.** Latent reachability tubes for the full Markov state computed via SLS^2.

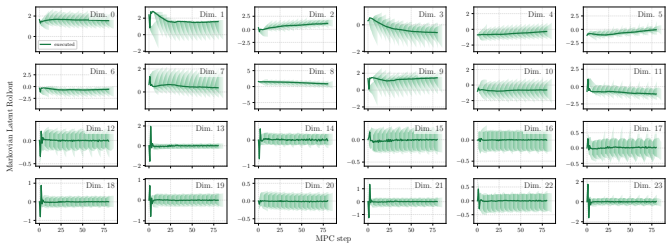

Fig. 15. **Cube.** Latent reachability tubes for the full Markov state computed via SLS^2.