MULTIPLE SHOOTING FOR GRADIENT-BASED PLANNING WITH WORLD MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Planning with world models is typically framed as a trajectory optimization problem where the goal is to find an action sequence that drives a system from an initial to a target state. Since world models are trained purely for prediction, the task-specific objective is specified only at inference, making the planner a key determinant of success. Standard approaches formulate this as a single shooting problem, where actions are the only optimization variables and the objective is evaluated by unrolling a learned latent dynamics model. However, optimizing through these rollouts compounds model errors, exposes the optimizer to many local minima, and is inherently sequential. Multiple shooting formulations sidestep these issues by treating states as additional optimization variables but traditionally rely on second-order methods that scale poorly to high-dimensional latent spaces. We propose PALM, a first-order multiple shooting planner for learned latent spaces. PALM treats latent states as explicit optimization variables and enforces dynamics consistency through an augmented Lagrangian. By relaxing dynamics constraints during optimization, PALM can move through infeasible regions and escape local minima where single shooting methods typically get stuck. We evaluate PALM on navigation (Wall, PointMaze) and manipulation (PushT) tasks. On Wall and PointMaze, PALM outperforms gradient-based baselines on average and degrades more slowly as the goal gets further: on the longest Wall crossing tasks, it succeeds on 82% of tasks, against at most 31% for other gradient-based planners. CEM achieves higher final success, but PALM reaches competitive success in less wall-clock time. On PushT, all planners reach similar success rates.

1 INTRODUCTION

Learning predictive models of dynamical systems directly from sensory data is a central challenge in robotics and reinforcement learning. World models (Ha & Schmidhuber, 2018; Hafner et al., 2019) address this by jointly learning an encoder to map high-dimensional observations onto compact latent features (LeCun, 2022), and a predictor that models the dynamics of these features, conditioned on control inputs (which may themselves be encoded). These models can be rolled out without executing actions in the real world and thus allow planners to optimize trajectories entirely in latent space, offering a promising path toward data-efficient planning.

Trained on a predictive objective alone, without any task-specific reward, world models frame planning as an inference-time trajectory optimization problem: given initial and target embeddings of observations, the planner must find an action sequence that drives one to the other. Because the task is specified only at inference, a single world model can be used for arbitrary downstream tasks; but for the same reason, the quality of the planner directly determines what the world model can accomplish.

Existing methods formulate planning as a single shooting problem where actions are the only decision variables, and the objective, a terminal cost on the latent reached by rolling out the predictor over a horizon T, is minimized. This formulation, however, suffers from several limitations. First, prediction errors compound along the horizon: because the predictor is learned, each prediction is imperfect, and the autoregressive rollout feeds those errors back into the predictor. Second, the resulting objective is non-convex with potentially many local minima. Third, the rollout is inherently sequential: latent states must be computed step by step, which prevents parallelization across

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.

timesteps within a candidate trajectory. Sampling-based planners, such as the cross-entropy method (CEM) (Rubinstein & Kroese, 2004) and model predictive path integral (MPPI) control (Williams et al., 2017), mitigate the non-convexity of the objective with a population of candidates, but every candidate is evaluated with an autoregressive rollout, so prediction errors still compound, and evaluating a population at every step is expensive. Gradient-based planners are a natural choice for differentiable world models and avoid the cost of evaluating on a population, but in the single shooting formulation, gradients with respect to early actions must propagate through T Jacobian-vector products of the predictor, which may lead to vanishing or exploding gradients at long horizons. Multiple shooting offers a principled alternative. Rather than optimizing over actions alone and obtaining the states from an autoregressive rollout, it optimizes over both states and actions and enforces the dynamics as constraints. When every time step is a shooting node, as in this work, multiple shooting and collocation are equivalent (Betts, 2010; Kelly, 2017). Because each constraint depends only on adjacent states and a single action, the rollout limitations are addressed by design: the predictor is only called one step ahead, so prediction errors do not compound during optimization; gradients no longer propagate through T compositions of the predictor; and the constraints at each step can be evaluated in parallel. While it has been the dominant approach to trajectory optimization in optimal control for decades Bock & Plitt (1984); Betts (2010), the resulting constrained problem is typically solved by second-order methods that scale cubically in the state dimension (Wächter & Biegler, 2006; Tassa et al., 2014). So far, multiple shooting has been applied to learned latent dynamics either with second-order solvers with specialized preconditioning (Rybkin et al., 2021), which remain limited to latent spaces of a few hundred dimensions, or with penalty methods (Psenka et al., 2026), which need several auxiliary heuristics (state noise, stop-gradients, periodic serial rollouts) to handle the resulting optimization landscape.

In this work, we formulate planning in learned latent spaces as a multiple shooting problem and solve it with PALM (Planning with Augmented-Lagrangian Multiple shooting), a first-order instantiation of the classical method of multipliers (Hestenes, 1969; Powell, 1969) tailored to learned latent dynamics. PALM only requires a differentiable one-step dynamics function and handles latent states that combine multiple modalities. Our contributions are:

- We formulate planning in learned latent spaces as a multiple shooting problem and propose PALM, a first-order augmented Lagrangian solver that treats latent states as additional optimization variables alongside actions and enforces dynamics consistency through per-timestep multipliers and adaptive penalty coefficients, requiring only first order steps on the inner subproblem.
- We show that the combination of the augmented search space and the dynamics relaxation lets the optimizer move through infeasible regions of the joint state–action space and escape local minima where single shooting methods get stuck. Multiplier updates then progressively tighten the constraint violations, returning a dynamically feasible trajectory at convergence.
- We evaluate PALM on three environments (PushT, PointMaze, Wall) on tasks of increasing difficulty. On Wall and PointMaze, PALM achieves higher success rates than gradient-based planners. CEM achieves higher final success on the most challenging tasks, but PALM reaches comparable success in less wall-clock planning time. On PushT, all planners achieve similar success rates.

2 Related work

Planning with learned world models. A long line of work learns latent dynamics from high-dimensional observations and uses them to plan. World Models (Ha & Schmidhuber, 2018), PlaNet (Hafner et al., 2019), and the Dreamer family (Hafner et al., 2020; 2021; 2025) learn reconstruction-based latent dynamics. TD-MPC (Hansen et al., 2022) and TD-MPC2 (Hansen et al., 2024) drop reconstruction, instead learning deterministic latent models with a self-consistency objective and coupling them with online planning. A promising recent line builds latent dynamics on top of joint-embedding predictive architectures (JEPAs) (LeCun, 2022; Assran et al., 2023; Bardes et al., 2024; Assran et al., 2025), training entirely on a predictive objective with no reward signal or goal conditioning, so that the task is specified only at inference through the planner. DINO-WM (Zhou et al., 2025) builds a predictor on frozen DINOv2 patch tokens and evaluates planning on PushT, PointMaze, and Wall, following the reward-free setup of Sobal et al. (2025), while LeWorld-Model (Maes et al., 2026) trains a JEPA end-to-end from pixels into a compact latent. We use the same environments and offline data, and build our own evaluation sets. Among these, methods that

train a task-specific policy on top of the world model (Hafner et al., 2020; 2021) must be retrained for each new objective. The JEPA-based models we build on avoid this by specifying the task only at inference, allowing a single world model to serve arbitrary goals. The resulting trajectory optimization problem is solved by a planner. PALM, which we introduce in Section 3.3, is one such planner and is in principle compatible with any of these architectures.

Single-shooting planners. Existing planners nearly all adopt a single shooting formulation: actions are the only optimization variables, and the latent trajectory is obtained by autoregressive rollout of the predictor. Sampling-based methods are widely used: CEM (Rubinstein & Kroese, 2004), MPPI (Williams et al., 2017) and variants such as iCEM (Pinneri et al., 2020) mitigate the non-convex landscape with a population of candidates and are the planners of choice in Chua et al. (2018); Hafner et al. (2019); Hansen et al. (2024); Maes et al. (2026). Gradient-based alternatives (Universal Planning Networks (Srinivas et al., 2018), Grad-CEM (Bharadhwaj et al., 2020)) exploit the differentiability of the predictor but must propagate gradients through T Jacobian-vector products, which may become unstable at long horizons. Compounding prediction error has historically been managed at the cost of expressiveness, through short planning horizons (Chua et al., 2018; Ebert et al., 2018) or model ensembles (Janner et al., 2019). A complementary line of work addresses the problem from the *model* side, smoothing the planning landscape via curvature regularization (Wang et al., 2026) or adversarial finetuning (Parthasarathy et al., 2025). Our contribution is orthogonal: we change the optimization *formulation* so that compounding does not arise in the inner loop in the first place, and PALM can in principle be combined with any of these model-side improvements.

Multiple shooting, collocation, and augmented Lagrangians. Multiple shooting (Bock & Plitt, 1984) and direct collocation (Hargraves & Paris, 1987; Betts, 2010; Becerra, 2012; Kelly, 2017) are standard approaches to trajectory optimization in classical optimal control: states are treated as optimization variables alongside actions, and dynamics are imposed as equality constraints. The resulting nonlinear programs are typically solved with second-order methods such as sequential quadratic programming, interior point methods (Wächter & Biegler, 2006), or differential dynamic programming (Mayne, 1966; 1973; Tassa et al., 2014). These solvers exploit the banded structure of the KKT system to achieve $\mathcal{O}(T \cdot d^3)$ per-iteration cost, but remain prohibitive when the state dimension d is large, as it is for modern world models. Augmented Lagrangian methods (Hestenes, 1969; Powell, 1969; Bertsekas, 2014; Boyd et al., 2010) decouple the constrained problem into subproblems amenable to first-order solvers, avoiding the need for Hessian factorizations. Their key advantage over pure penalty methods (Nocedal & Wright, 2006) is that the multiplier updates drive constraint violations to zero with a bounded penalty coefficient (Bertsekas, 2014), keeping the inner subproblem well-conditioned. Augmented Lagrangian formulations have been used in adjacent settings, including large-scale PDE-constrained control (Fang et al., 2022), multiple-shooting Neural ODE training (Turan & Jäschke, 2021; Prabhu et al., 2025), and neural network training viewed as optimal control (Evens et al., 2021), which also makes the connection to target propagation (Lee et al., 2015) explicit. To our knowledge, however, first-order augmented Lagrangian methods have not been applied to inference-time planning in learned latent spaces.

State-space planning with learned dynamics. Latent-Space Collocation (LatCo; Rybkin et al., 2021) is the closest precedent: it optimizes latent states and actions and enforces dynamics through a Lagrangian, maximizing a learned reward. PALM differs in two aspects. First, LatCo constrains the squared norm of the dynamics residual, with one scalar multiplier per timestep, whereas PALM constrains the residual itself, with one multiplier per latent dimension per timestep. Second, LatCo takes Levenberg–Marquardt steps, whose cost grows cubically with the latent dimension, whereas PALM solves its augmented Lagrangian subproblem with a first-order method. GRASP (Psenka et al., 2026) also optimizes latent states, but enforces dynamics with a quadratic penalty and adds three mechanisms to handle the resulting landscape: noise on the states, stop-gradients on the state inputs of the predictor, and periodic single shooting refinements of the actions. We find that the augmented Lagrangian formulation alone achieves better performance on the same tasks (Sec. 4, Table 1).

3 METHOD

We formulate planning in learned latent spaces as a multiple shooting problem and solve it with a first-order augmented Lagrangian method. This section introduces the problem (Sec. 3.1), presents the multiple shooting reformulation (Sec. 3.2), and describes the PALM algorithm (Sec. 3.3).

3.1 PROBLEM FORMULATION

Let $\mathcal{Z} \subset \mathbb{R}^n$ be a state space, $\mathcal{U} \subset \mathbb{R}^d$ an action space, and $f : \mathcal{Z} \times \mathcal{U} \to \mathcal{Z}$ a differentiable dynamics function such that $z_{t+1} = f(z_t, u_t)$. Given an initial state z_0 and a goal state z_g, we seek a sequence of T actions $\boldsymbol{u} = [u_0, \dots, u_{T-1}]$ that brings $f^{[T]}(z_0, \boldsymbol{u})$ close to z_g, where $f^{[T]}$ denotes applying f autoregressively T times starting from z_0. In the world model setting we consider, $\mathcal{Z}$ is the latent space of a learned encoder Φ that maps observations to latent states, f is a learned predictor, and initial and goal states are encoded from observations as $z_0 = \Phi(o_0)$ and $z_g = \Phi(o_g)$. Reaching z_g in latent space serves as a proxy for bringing the system to the state observed in o_g.

Single shooting. Mathematically, the problem can be formalized as

$$\min_{\boldsymbol{u}} \ \|f^{[T]}(z_0, \boldsymbol{u}) - z_g\|_2^2 + \frac{\alpha_u}{Td} \sum_{t=0}^{T-1} \|u_t\|_2^2, \tag{1}$$

where the only optimization variables are the actions $\boldsymbol{u}$ and $\alpha_u \geq 0$ weights the action regularization.As discussed in Section 1, this formulation compounds prediction errors along the horizon, requires gradient propagation through T compositions of f, and prevents parallelization across timesteps.

3.2 MULTIPLE SHOOTING FORMULATION

We reformulate problem 1 by introducing the intermediate latent states $\boldsymbol{z} = [z_1, \dots, z_{T-1}]$ as additional optimization variables and imposing the dynamics as explicit constraints:

$$\min_{\boldsymbol{u}, \boldsymbol{z}} \ \frac{\alpha_g}{2}\|z_g - f(z_{T-1}, u_{T-1})\|_2^2 + \frac{\alpha_u}{Td} \sum_{t=0}^{T-1} \|u_t\|_2^2 \ \text{ s.t. } \ z_{t+1} = f(z_t, u_t), \quad u_t \in \mathcal{U}, \tag{2}$$

where $\alpha_g > 0$ and $\alpha_u \geq 0$ weight the goal term and the action regularization, and $\mathcal{U}$ is the set of admissible actions. The dynamics constraints hold for $t = 0, \dots, T-2$ and the action constraints for $t = 0, \dots, T-1$. The terminal condition $z_g = f(z_{T-1}, u_{T-1})$ appears as a quadratic penalty in the objective, and the action bounds are enforced by projection after each optimization step.

At optimality, problems 1 and 2 are equivalent, but solving problem 2 addresses the three limitations of problem 1: each constraint calls f only one step ahead, so errors do not compound during optimization, gradients no longer propagate through T compositions of f, and all constraints can be evaluated in parallel.

3.3 PALM: FIRST-ORDER AUGMENTED LAGRANGIAN SOLVER

We solve the constrained problem 2 using the method of multipliers (Hestenes, 1969; Powell, 1969). The formulation itself is standard (Bertsekas, 2014; Nocedal & Wright, 2006); our contribution is showing that it can be applied directly to learned latent dynamics with a first-order inner solver. Defining the dynamics residual $c_t = z_{t+1} - f(z_t, u_t)$ for $t = 0, \dots, T-2$, the augmented Lagrangian of the problem is

$$\mathcal{L}(\boldsymbol{u}, \boldsymbol{z}, \boldsymbol{\lambda}) = \frac{\alpha_g}{2}\|z_g - f(z_{T-1}, u_{T-1})\|_2^2 + \frac{\alpha_u}{Td} \sum_{t=0}^{T-1} \|u_t\|_2^2 + \sum_{t=0}^{T-2} \lambda_t^\top c_t + \sum_{t=0}^{T-2} \frac{\rho_t}{2}\|c_t\|_2^2, \tag{3}$$

where $\lambda_t \in \mathbb{R}^n$ is the Lagrange multiplier for constraint c_t and $\rho_t > 0$ is its penalty coefficient. We use per-timestep penalties $\boldsymbol{\rho} = [\rho_0, \dots, \rho_{T-2}]$ because constraint violations at different timesteps can differ in scale.

When the latent state concatenates multiple modalities, we write z_t^m for the features of modality m in z_t and define one residual per modality, $c_t^m = S_m\big(z_{t+1}^m - f(z_t, u_t)^m\big)$, with its own multiplier λ_t^m and penalty ρ_t^m, where S_m is a scaling matrix. The goal term is scaled the same way (Appendix A.2.4). The augmented Lagrangian combines the advantages of penalty methods and Lagrangian duality: the quadratic penalty makes the inner subproblem well-conditioned for first-order optimization, while the multiplier updates drive constraint violations to zero without requiring the penalty coefficients in $\boldsymbol{\rho}$ to grow unboundedly (Bertsekas, 2014).

Primal–dual algorithm. PALM alternates between a *primal step* (inner loop), which minimizes Eq. 3 over $(\boldsymbol{z}, \boldsymbol{u})$ with $\boldsymbol{\lambda}$ fixed, and a *dual step* (outer loop), which updates the multipliers and penalty coefficients. The primal step runs K iterations of Adam (Kingma & Ba, 2015) on $\mathcal{L}$.

We do not solve the primal step to convergence: the method of multipliers allows inexact minimization of the augmented Lagrangian (Bertsekas, 2014), and we use a small fixed number of inner steps K. Figure 2 shows the dynamics residual during optimization.

Outer-loop schedule. We adopt the adaptive penalty and multiplier schedule of Algorithm 17.4 in Nocedal & Wright (2006). After each primal step, we evaluate the dynamics residuals and update each timestep and modality independently. We compare the squared residual $\|c_t^m\|_2^2$ with its tolerance τ_t^m. When the residual is above its tolerance and the penalty is below its cap, we increase ρ_t^m and skip the multiplier update. Otherwise, if the residual is above the tolerance floor $\tau_{\min}^m$, we update the multiplier

$$\lambda_t^m \leftarrow \lambda_t^m + \eta \rho_t^m c_t^m. \tag{4}$$

Finally, we update the tolerance: if the residual is within tolerance, we tighten it to $\beta_- \tau_t^m$; otherwise, we loosen it to $\beta_+ \tau_t^m$. The tolerance never goes below the floor $\tau_{\min}^m$. The terminal weight α_g stays fixed. The full procedure (Algorithm 1) and hyperparameter values are given in Appendix A.2.4.

Stopping criterion. The algorithm stops when all intermediate tolerances reach their floors, $\tau_t^m = \tau_{\min}^m$ for all $t \le T-2$ and all modalities m, or when $\max_{0 \le t \le T-1} \sum_m \|c_t^m\|_2^2 < \tau_{\text{conv}}$, where c_{T-1}^m is the terminal residual. Otherwise, it stops after $K_{\max}$ outer iterations.

4 Experiments

We evaluate PALM along three axes: (1) success rate compared with gradient-based and sampling-based planners, and how it changes with task difficulty (Sec. 4.2); (2) how PALM's goal loss and dynamics residual change during optimization, and how the dynamics relaxation helps it escape local minima (Sec. 4.3 and 4.4); and (3) success rate as a function of wall-clock planning time (Sec. 4.5).

4.1 Setup

Environments. We evaluate PALM on three environments spanning navigation (Wall and PointMaze) and manipulation (PushT) (Appendix, Fig. 5) **PointMaze** requires a point-mass agent to navigate from a start to a goal location through a maze. **Wall** requires the agent to reach a goal in one of two rooms separated by a wall with a door. **PushT** requires the agent to push a T-shaped block to a target pose. All three environments use 2-dimensional continuous actions (Appendix A.2.3).

Evaluation sets. For each environment, we generate sets of start–goal pairs of increasing difficulty. Each goal is the state reached by an expert trajectory k environment steps after the start (details in Appendix A.1). Unlike the protocol of prior world model papers (Zhou et al., 2025; Psenka et al., 2026; Maes et al., 2026), which samples goals k steps ahead along non-optimal dataset trajectories, this makes k a measure of task difficulty. On Wall, tasks differ in whether the agent is on the same side of the wall in start and goal, or on a different side (the agent has to cross the door to reach the goal). On PointMaze, the agent has to reach the goal either with a certain velocity (*passthrough*) or with low velocity (*rest*). On PushT, tasks differ in how much the T translates and rotates between start and goal. We use $k \in \{25, 30, 40, 50\}$ on Wall, $\{10, 15, 20, 25\}$ for PointMaze passthrough goals, $\{25, 30, 35, 40\}$ for rest goals, and $k = 25$ on PushT, with 9 translation–rotation sets (Appendix, Table 2). All environments use a frameskip of 5: each planned action concatenates 5 environment actions, so a plan of T actions covers $5T$ environment steps.

World model. For each environment, we train a world model on the offline trajectories from DINO-WM (Zhou et al., 2025): a frozen DINOv2 backbone encodes observations into patch features, and we train a CNN projector that compresses them to a $14\times14\times8$ latent, jointly with a latent predictor that evolves the projected features conditioned on actions (Appendix A.2.1). All planners share the same trained world model per environment.

Baselines. We compare PALM against two single shooting and two multiple shooting planners. **SS-Grad** optimizes actions alone with Adam on the single shooting objective (Eq. 1), with latent states obtained by autoregressive rollouts of f. **CEM** (Rubinstein & Kroese, 2004) optimizes actions on the same single shooting objective by iteratively sampling and refitting a distribution over action sequences. **LatCo** (Rybkin et al., 2021) jointly optimizes latent states and actions under dynamics constraints and bounds on the actions. Like PALM, it alternates primal and dual steps, but its primal step is a single damped Gauss–Newton step on states and actions, and its dual step updates one scalar multiplier per timestep, applied to the squared dynamics residual. **GRASP** (Psenka et al., 2026) is a pure penalty method: it lifts latent states as optimization variables and enforces dynamics using a quadratic penalty, without Lagrange multipliers. It combines this penalty with state-noise injection, state-input stop-gradients, and periodic single shooting gradient refinements of the actions. All planners use the same trained world model, planning instances, and planning horizons. We adapt LatCo and GRASP to our goal-reaching, open-loop setting (Appendix A.2.6 and A.2.5). In particular, the original GRASP protocol periodically executes intermediate plans in the simulator and reports an instance as successful if one of these executions succeeds. This uses simulator feedback that is unavailable in our setting, so we run GRASP for a fixed number of refinement phases and return its final actions. Hyperparameters for all methods are in Table 4.

Evaluation protocol. Each method is evaluated on every set of Table 2. The planner runs its full optimization loop; the resulting action sequence is executed in the simulator once, without replanning. No planner uses the simulator during optimization. PALM stops when its stopping criterion is met (Section 3.3) or after a maximum number of iterations; baselines run for a fixed budget. We report success rate using the per-environment criteria of DINO-WM (Zhou et al., 2025) on Wall and PointMaze; on PushT, success only requires the T-shaped block to reach its target pose (Appendix A.2.9). We report the mean and standard deviation of the success rate over 6 groups of 50 start–goal pairs. Wall-clock times are measured on one NVIDIA A100 and include only the planner's optimization. PALM-specific optimization details are in Appendix A.2.4.

4.2 Planning performance

Table 1 reports success rates on all evaluation sets. PALM has a higher success rate than the other gradient-based planners (SS-Grad, GRASP and LatCo) on Wall crossing tasks and on PointMaze. On Wall same-side tasks, LatCo reaches 99.7% and PALM 98.4%. Unlike GRASP, PALM needs no state noise, stop-gradients or single shooting refinements. The gap is largest on the tasks that require a long detour: on Wall crossing tasks, PALM reaches 86.9% against 61.2% for SS-Grad, 51.9% for GRASP and 27.2% for LatCo, and on PointMaze rest goals, 59.6% against 32.9%, 37.4% and 15.9%. CEM has the highest success rate on Wall crossing tasks and on PointMaze rest goals; PALM matches it on PointMaze passthrough goals. On Wall same-side tasks, CEM, LatCo and PALM all exceed 98%. On PushT, CEM, SS-Grad, GRASP and PALM reach between 37% and 41%, within one standard deviation of each other.

Figure 1 shows success rate as a function of goal distance. The success rate of SS-Grad and GRASP drops as the goal gets further, while PALM's decreases more slowly. At the largest distance on Wall crossing tasks ($k = 50$), PALM succeeds on 82.3% of the pairs, against 31.0% for SS-Grad and 22.7% for GRASP. On PointMaze passthrough goals at $k = 25$, PALM has the highest success rate of all planners, with 80.3% against 74.7% for CEM. On PointMaze rest goals, GRASP matches CEM at $k = 25$ (98.3% and 99.7%) but drops to 6.3% at $k = 35$ and 3.7% at $k = 40$. LatCo, which also optimizes states and actions but takes Gauss–Newton steps with one scalar multiplier per timestep, reaches 100% on the shortest PointMaze passthrough goals ($k = 10$), like GRASP and slightly above PALM, but drops quickly as the goal gets further: on PointMaze passthrough goals to 8.0% at $k = 25$, on PointMaze rest goals from 52.0% at $k = 25$ to 0% at $k = 40$, and on Wall crossing tasks at $k = 50$, it succeeds on 7.3% of the pairs against 82.3% for PALM. Results on Wall same-side tasks and on each PushT set are in Appendix A.3.

Table 1: Success rate (%) of each planner. Each column averages the sets of Table 2 for that task type. CEM is sampling-based; the other planners are gradient-based. Bold: best gradient-based planner. Mean ± std over 6 groups of 50 start–goal pairs.

	Wall		PointMaze		PushT
Planner	same-side	crossing	passthrough	rest	all sets
CEM	99.6 ± 0.4	97.2 ± 1.4	89.8 ± 2.2	87.3 ± 1.6	40.4 ± 2.3
SS-Grad	83.4 ± 1.0	61.2 ± 2.7	57.5 ± 3.5	32.9 ± 2.3	$\mathbf{40.4 \pm 2.6}$
GRASP	75.4 ± 1.1	51.9 ± 2.8	71.7 ± 3.2	37.4 ± 2.4	38.2 ± 2.4
LatCo	$\mathbf{99.7 \pm 0.4}$	27.2 ± 3.1	63.7 ± 1.5	15.9 ± 1.9	36.5 ± 2.5
PALM	98.4 ± 1.0	$\mathbf{86.9 \pm 3.0}$	$\mathbf{88.6 \pm 0.9}$	$\mathbf{59.6 \pm 3.7}$	37.1 ± 4.0

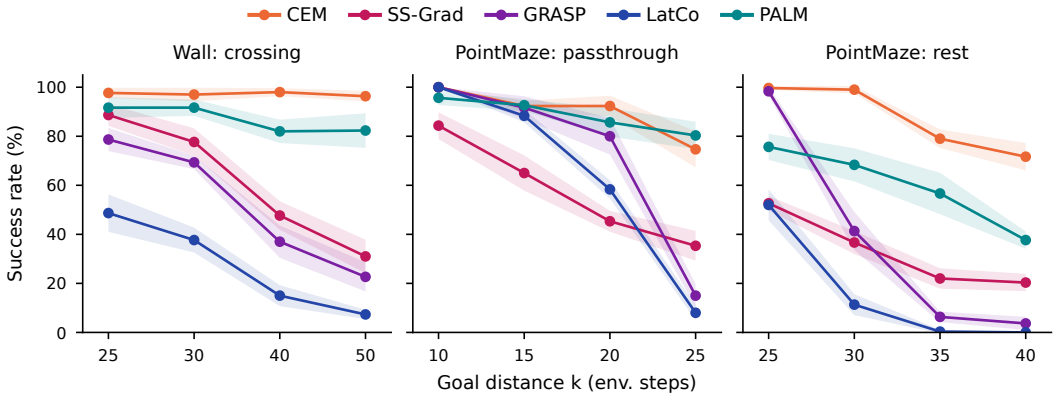


Figure 1: Success rate as a function of the goal distance k (environment steps), with planning horizon $T = k/5$. Wall crossing: start and goal are on opposite sides of the wall. PointMaze passthrough: the goal is a position and velocity reached by the expert. PointMaze rest: the goal is a position with near-zero velocity. Mean ± std over 6 groups of 50 start–goal pairs. Wall same-side tasks are in Appendix A.3.

4.3 Understanding PALM's optimization

4.3.1 Goal loss and dynamics consistency

Figure 2 shows how PALM's goal loss and dynamics residual evolve during optimization. Early in the optimization, the goal loss on the optimized states decreases quickly while the dynamics residual is still large: the states move toward the goal without being consistent with the dynamics. As the penalty and the multipliers grow, the residual decreases, and the goal loss of the rollout of PALM's actions approaches the goal loss on its optimized states.

PALM ends with a lower rollout goal loss than SS-Grad on PointMaze (0.021 vs. 0.337) and Wall (0.0095 vs. 0.056); on PushT, SS-Grad ends lower (0.024 vs. 0.033).

4.3.2 Qualitative analysis on Wall

Figure 3 shows how PALM and SS-Grad plans develop on one Wall crossing task ($k = 50, T = 10$) where PALM succeeds and SS-Grad fails. Appendix A.4 shows the same behavior on a pendulum swing-up with known dynamics. A failure case of PALM is shown in Figure 9, where the world model lets the plan pass through the wall.

4.4 Analysis: escaping local minima

We investigate how the dynamics relaxation helps in escaping local minima. For every start–goal pair, we record the final goal loss $\|f^{[T]}(z_0, \boldsymbol{u}) - z_g\|_2^2$ of the actions returned by PALM and SS-

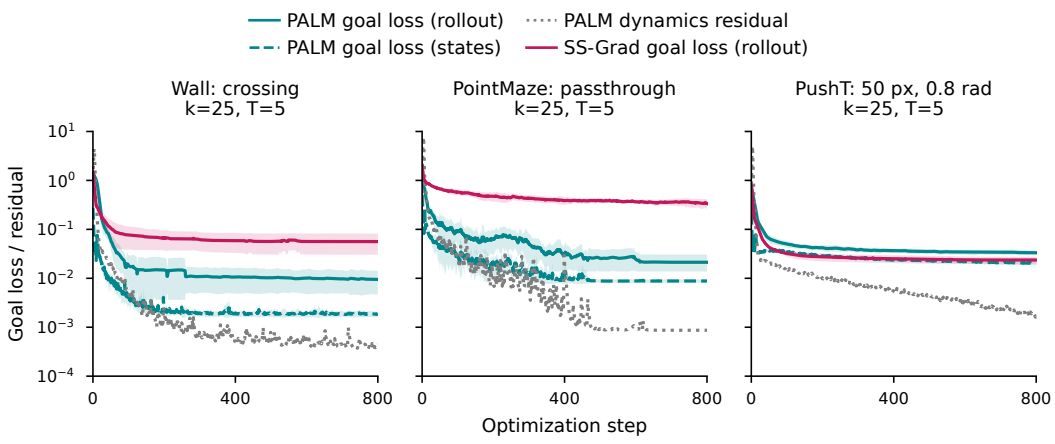


Figure 2: Goal loss and dynamics residual during optimization. We show PALM's goal loss evaluated on its optimized states and on a rollout of its actions, together with its mean squared dynamics residual. SS-Grad's forward-rollout goal loss is shown for comparison. All tasks use a planning horizon of $T = 5$, corresponding to $k = 25$ environment steps. Each set contains 284 to 300 start–goal pairs.

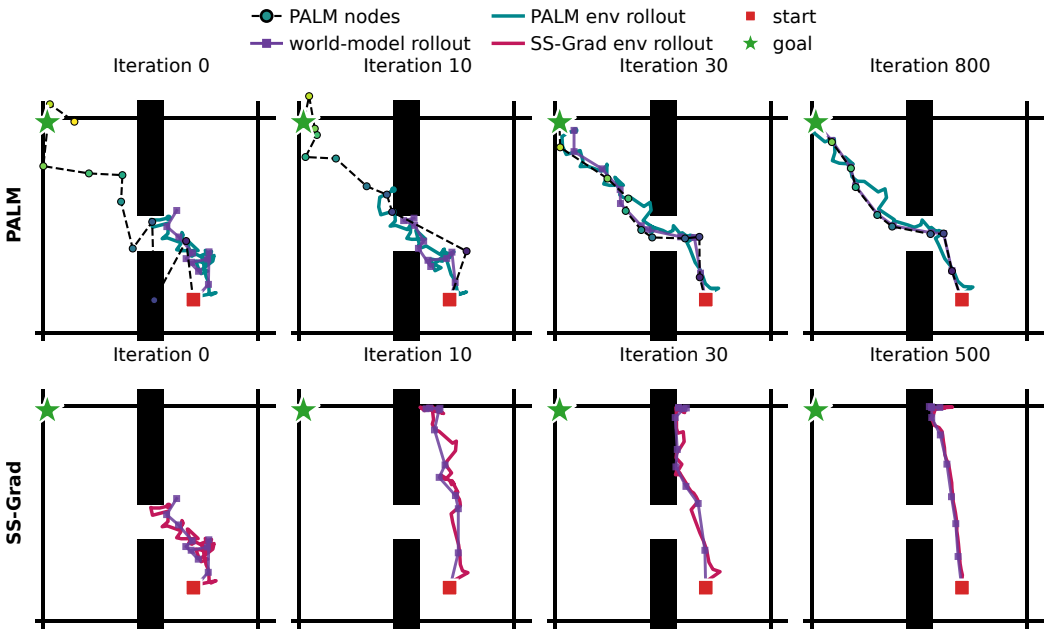


Figure 3: PALM vs. single shooting (SS-Grad) on a Wall crossing task. PALM optimizes intermediate states (nodes, colored by timestep) together with actions. Early iterations allow infeasible trajectories, such as nodes inside the wall, which lets the plan find the door. Multiplier updates then enforce the dynamics, and both the final world-model and environment rollouts reach the goal. SS-Grad optimizes actions through the rollout and gets stuck against the wall.

Grad. Appendix Figure 7 plots one against the other. On Wall, SS-Grad ends with a goal loss above 0.1 while PALM ends below 0.1 on 47.3% of the pairs; the reverse happens on 3.3%, and both end above 0.1 on 10.7%. On PointMaze, these fractions are 54.7%, 1.0% and 0.7%. These pairs are local minima of the single shooting objective that PALM avoids by optimizing the states jointly with the actions. On PushT, PALM ends with a slightly higher goal loss than SS-Grad on most pairs (Appendix Figure 8).

4.5 COMPUTATIONAL EFFICIENCY

Figure 4 shows success rate as a function of wall-clock planning time. At each time, we execute the actions the planner holds at that time. We use Wall crossing tasks at $k = 50$ and PointMaze passthrough goals at $k = 25$. The time counts only the planner's optimization, for a batch of 50 start–goal pairs on one NVIDIA A100. PALM reaches most of its final success rate early. On PointMaze passthrough goals, it reaches 71.0% after 10 s, while CEM needs 26 s to reach 70%. On Wall crossing tasks, PALM reaches 70% after 20 s, before CEM has any success; CEM then reaches a higher final success rate (96.3%). From about 10 s on Wall and 1 s on PointMaze, PALM stays above the other gradient-based planners, which remain below 40%. PALM also stops on its own once its stopping criterion is met, whereas the baselines run for a fixed number of iterations. On PointMaze rest goals, PALM reaches 41.7% at 2.1 s but ends below CEM and GRASP; on PushT, all planners reach a similar final success rate and SS-Grad is the fastest (Appendix A.3).

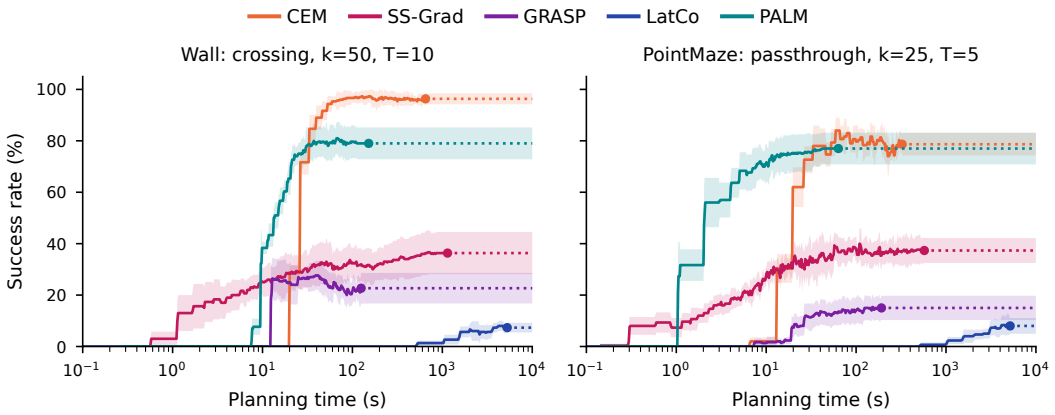


Figure 4: Success rate as a function of wall-clock planning time, measured for a batch of 50 start–goal pairs on one NVIDIA A100. Mean ± std over 6 groups of 50 pairs. After a planner stops, its final success rate is held constant (dotted line). Results on PushT and PointMaze rest goals are in Appendix A.3.

5 CONCLUSION

We have introduced PALM, a first-order augmented Lagrangian planner for multiple shooting in learned latent spaces. On Wall and PointMaze, PALM outperforms gradient-based baselines in open-loop planning, with the gap widening at longer horizons. CEM still reaches a higher final success rate, but PALM reaches high success earlier on the longest Wall crossing tasks. On PushT, all planners reach similar success rates. The per-instance goal loss comparison with the single shooting baseline shows that, on Wall and PointMaze, PALM avoids local minima. Our results focus on open-loop planning. Future work should consider extending PALM to closed-loop MPC and combining it with sampling-based initialization or multiple PALM solves. Constraining shooting nodes to remain near the world model's training distribution to limit model exploitation is another promising direction. Our evaluation is limited to 2D environments with relatively low-dimensional action spaces; scaling to contact-rich 3D manipulation with longer horizons will test the limits of the current penalty and multiplier schedules. We evaluate PALM with a single world model architecture (DINO-WM); while the formulation is model-agnostic, its sensitivity to model fidelity and latent dimensionality remains to be characterized. All experiments are conducted in simulation; deploying PALM on a physical robot, where model mismatch is more severe, is a natural next step.

AI USE STATEMENT

In this work, we used generative AI tools to assist with writing and debugging code. We have not used generative AI tools for any other task with required disclosure under the ICLR 2027 AI policy. Additionally, we used generative AI tools to create and modify figures, and to rephrase text and fix

LaTeX syntax in the paper. All AI-assisted code was reviewed and tested by the authors, and all text was checked and revised by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Ethics statement

This work does not involve human subjects or personal data. All experiments are run in simulation, and the world models are trained on publicly available offline datasets. PALM is a general planning method for learned world models; like other planners, it could be used in robotic systems, where plans computed with an inaccurate world model can fail in the real environment.

Reproducibility statement

Appendix A.2 describes the world model, the planners and their hyperparameters, Appendix A.1 the generation of the evaluation sets, and Appendix A.2.10 the hardware and evaluation setup. Code, configuration files and trained world-model checkpoints will be released upon publication.

References

Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture. *arXiv preprint arXiv:2301.08243*, 2023.

Mahmoud Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Mojtaba Komeili, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, Sergio Arnaud, Abha Gejji, Ada Martin, Francois Robert Hogan, Daniel Dugas, Piotr Bojanowski, Vasil Khalidov, Patrick Labatut, Francisco Massa, Marc Szafraniec, Kapil Krishnakumar, Yong Li, Xiaodong Ma, Sarath Chandar, Franziska Meier, Yann LeCun, Michael Rabbat, and Nicolas Ballas. V-jepa 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025.

Adrien Bardes, Quentin Garrido, Jean Ponce, Michael Rabbat, Yann LeCun, Mahmoud Assran, and Nicolas Ballas. Revisiting feature prediction for learning visual representations from video. *arXiv:2404.08471*, 2024.

Victor M Becerra. Practical direct collocation methods for computational optimal control. In *Modeling and optimization in space engineering*, pp. 33–60. Springer, 2012.

Dimitri P Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

John T Betts. *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010.

Homanga Bharadhwaj, Kevin Xie, and Florian Shkurti. Model-predictive control via cross-entropy and gradient-based optimization. In *Learning for Dynamics and Control*, pp. 277–286. PMLR, 2020.

Hans Georg Bock and Karl-Josef Plitt. A multiple shooting algorithm for direct solution of optimal control problems. *IFAC Proceedings Volumes*, 17(2):1603–1608, 1984.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2010. doi: 10.1561/2200000016.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/3de568f8597b94bda53149c7d7f5958c-Paper.pdf.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

Brecht Evens, Puya Latafat, Andreas Themelis, Johan Suykens, and Panagiotis Patrinos. Neural network training as an optimal control problem:—an augmented lagrangian approach—. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pp. 5136–5143. IEEE, 2021.

Liang Fang, Stefan Vandewalle, and Johan Meyers. A parallel-in-time multiple shooting algorithm for large-scale pde-constrained optimal control problems. *Journal of Computational Physics*, 452: 110926, 2022.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2(3):440, 2018.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pp. 2555–2565. PMLR, 2019.

Danijar Hafner, Timothy P. Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=S1lOTC4tDS.

Danijar Hafner, Timothy P. Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=0oabwyZbOu.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640(8059):647–653, 2025.

Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control. In *ICML*, 2022.

Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control, 2024.

Charles R Hargraves and Stephen W Paris. Direct trajectory optimization using nonlinear programming and collocation. *Journal of guidance, control, and dynamics*, 10(4):338–342, 1987.

Magnus R Hestenes. Multiplier and gradient methods. *Journal of optimization theory and applications*, 4(5):303–320, 1969.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, 2019.

Matthew Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM review*, 59(4):849–904, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

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

Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Joint european conference on machine learning and knowledge discovery in databases*, pp. 498–515. Springer, 2015.

Lucas Maes, Quentin Le Lidec, Damien Scieur, Yann LeCun, and Randall Balestriero. Leworld-model: Stable end-to-end joint-embedding predictive architecture from pixels. 2026.

David Mayne. A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems. *International Journal of Control*, 3(1):85–95, 1966.

David Q Mayne. Differential dynamic programming–a unified approach to the optimization of dynamic systems. In *Control and dynamic systems*, volume 10, pp. 179–254. Elsevier, 1973.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2006. Chapter 17.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu Xu, Jégou Hervé, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=a68SUt6zFt.

Arjun Parthasarathy, Nimit Karla, Rohun Agrawal, Yann LeCun, Oumayma Bounou, Pavel Izmailov, and Micah Goldblum. Closing the train–test gap in world models for gradient-based planning. *arXiv preprint arXiv:2512.09929*, 2025. URL https://arxiv.org/abs/2512.09929.

Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, Jan Achterhold, Joerg Stueckler, Michal Rolinek, and Georg Martius. Sample-efficient cross-entropy method for real-time planning. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2020. URL https://corlconf.github.io/paper_217.

Michael JD Powell. A method for nonlinear constraints in minimization problems. *Optimization*, pp. 283–298, 1969.

Siddharth Prabhu, Srinivas Rangarajan, and Mayuresh Kothare. A condensing approach to multiple shooting neural ordinary differential equation. *arXiv preprint arXiv:2506.00724*, 2025.

Michael Psenka, Michael Rabbat, Aditi Krishnapriyan, Yann LeCun, and Amir Bar. Parallel stochastic gradient-based planning for world models. 2026. URL https://arxiv.org/abs/2602.00475.

Reuven Y Rubinstein and Dirk P Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning*, volume 133. Springer, 2004.

Oleh Rybkin, Chuning Zhu, Anusha Nagabandi, Kostas Daniilidis, Igor Mordatch, and Sergey Levine. Model-based reinforcement learning via latent-space collocation, 2021.

Uladzislau Sobal, Wancong Zhang, Kyunghyun Cho, Randall Balestriero, Tim G. J. Rudner, and Yann LeCun. Learning from reward-free offline data: A case for planning with latent dynamics models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4732–4741. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/srinivas18b.html.

Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1168–1175. IEEE, 2014.

Evren Mert Turan and Johannes Jäschke. Multiple shooting for training neural differential equations on time series. *IEEE Control Systems Letters*, 6:1897–1902, 2021.

Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1): 25–57, 2006.

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.

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.

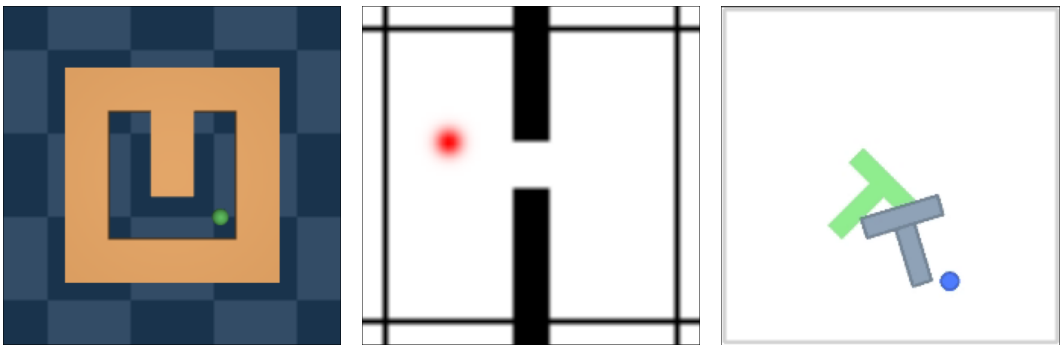
Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. Dino-wm: World models on pre-trained visual features enable zero-shot planning. In *International Conference on Machine Learning (ICML)*, 2025. URL https://arxiv.org/abs/2411.04983.

A Appendix

A.1 Evaluation sets

Table 2: Evaluation sets. k is the number of environment steps the expert takes from start to goal.

Env	Difficulty axes	k	Pairs per set
Wall	same side / crossing	$25, 30, 40, 50$	300
PointMaze	passthrough	$10, 15, 20, 25$	300
	rest	$25, 30, 35, 40$	300
PushT	translation $\{25, 50, 100\}$ px $\times$ rotation $\{0.2, 0.8, 1.5\}$ rad	25	284–300



(a) PointMaze (b) Wall (c) PushT

Figure 5: Evaluation environments. **PointMaze**: a point-mass agent navigates through a maze to a goal location. **Wall**: an agent must reach a goal on the other side of a door. **PushT**: an agent must push a T-shaped block to a target pose.

Comparison with the standard protocol. In the protocol used by prior world model papers, the goal is the state reached k steps after the start along a trajectory of the training dataset. These trajectories are not optimal: the agent may move back and forth or stay in place, so the goal can be close to the start even for large k. As a result, larger k does not mean a harder task. In our sets, goals come from expert trajectories that move toward the goal at every step, so the start–goal distance grows with k.

Wall. The expert follows the shortest path at a constant speed of 1.2 units per step, so the goal lies at a path distance of $1.2k$ from the start. Starts are sampled on a grid, among points that allow a path of this length, with a jitter of ± 0.5. The goal is placed along the shortest path toward a random point that satisfies the task condition (same side or crossing).

PointMaze. The expert follows the shortest path through the maze cells at its maximum velocity. Starts are sampled uniformly in free space, with the ball at rest, and the expert heads to the farthest cell. For *passthrough* goals, the goal is the state at step k, and pairs where the expert is braking at step k are rejected. For *rest* goals, the expert comes to rest at its target between steps $k-5$ and k.

PushT. The expert is a CEM planner with access to the true simulator, which searches $k = 25$ steps toward a sampled target pose. Starts follow the environment's reset distribution. The goal is the pose reached by the expert, kept only if its translation and rotation fall within the set's ranges. Some sets therefore contain fewer than 300 pairs (284 to 300).

Validation and splits. Each set is split into 6 disjoint subsets of 50 pairs; for PushT sets with fewer than 300 pairs, the last subset is smaller (34 to 46 pairs). Reported success rates are the mean over the 6 subsets, and $\pm$ is the standard deviation over them.

A.2 Implementation details

A.2.1 World model architecture

Our world model has four components: a frozen DINOv2-S visual encoder, a trainable CNN projector that compresses the visual features, proprioception and action encoders, and a latent predictor. The encoders and predictor architecture follow DINO-WM (Zhou et al., 2025).

Visual encoder. A frozen DINOv2-S backbone (Oquab et al., 2024) (`dinov2_vits14`) maps 224×224 RGB observations to a 14×14 grid of 384-dimensional patch tokens.

Projector. A two-layer 2-D CNN with layer norm (3×3 kernels, stride 1, padding 1) reduces the channel dimension $384\to192\to8$, producing a $14\times14\times8$ latent. Spatial resolution is preserved.

Proprioception and action encoders. Each is a per-timestep linear projection (implemented as a 1×1 Conv1d) that maps its input vector to a 10-dimensional embedding. The resulting embeddings are concatenated to the visual tokens along the channel dimension.

Latent predictor. The predictor f is a ViT-style Transformer (Dosovitskiy et al., 2021) with depth 6, 16 heads of dimension 64, MLP dimension 2048, and dropout 0.1. It uses learned temporal and spatial positional embeddings and applies causal masking across $H{=}3$ history frames. At each frame, the token concatenates the projected visual features, the proprioception embedding, and the action embedding; given the last H frames, f predicts the next-step latent as the last temporal token of its output.

Latent state. The latent state z_t used for planning concatenates the flattened projected visual features ($14 \times 14 \times 8 = 1568$ dimensions) and the proprioception embedding (10 dimensions), so $n = 1578$.

A.2.2 World model training

We train the projector, predictor, action encoder, and proprioception encoder jointly while keeping the DINOv2 backbone frozen. We use Adam (Kingma & Ba, 2015) with separate learning rates per component: 5×10^{-4} for the predictor, 5×10^{-4} for the action and proprioception encoders, 3×10^{-4} for the decoder (used only for visualisation; not involved in projector or predictor training), and 1×10^{-6} for the projector. No learning-rate scheduler is used. We train with batch size 32 on 224×224 images at frameskip 5, using the offline trajectories from (Zhou et al., 2025) with a 0.9:0.1 train/validation split. PushT is trained for 2 epochs given the larger dataset; PointMaze and Wall are trained for 100 epochs. Actions are normalised before being passed to the action encoder.

A.2.3 Action spaces

All environments have 2-dimensional continuous actions. Since each planned action concatenates 5 environment actions (frameskip 5), planners optimize 10-dimensional actions.

Wall Displacement $(\Delta x, \Delta y)$ of the agent; the environment applies $\mathbf{p}_{t+1} = \mathbf{p}_t + 2\,\mathbf{a}_t$, with collisions resolved geometrically. Bounds: $[-1.8, 1.8]$ per coordinate, the range of the dataset actions.

PointMaze Force (u_x, u_y) applied to the point mass. Bounds: $[-1, 1]$ per coordinate, the environment limits.

PushT Relative target displacement $(\Delta x, \Delta y)$ of the agent in pixels, divided by 100; the environment moves the agent toward the target with PD control. Bounds: $[-0.34, 0.35]$ per coordinate, the 95% quantile range of the dataset actions.

A.2.4 PALM PLANNER

Our world model has two modalities, visual (v) and proprioceptive (p), with dimensions $n_v = 1568$ and $n_p = 10$. We set $S_m = I/\sqrt{n_m}$, so that $\|c_t^m\|_2^2$ is the mean squared prediction error per coordinate of modality m. PALM minimizes the augmented Lagrangian of Eq. 3, with the goal term scaled like the residuals, $\frac{\alpha_g}{2}\sum_m \|c_{T-1}^m\|_2^2$, where c_{T-1}^m is the terminal residual of Algorithm 1. The goal weight grows with the horizon, $\alpha_g = 10^4 \cdot \max(1, T-4)$, and is fixed during optimization. Hyperparameters used in the reported runs are listed in Table 3.

Algorithm 1 PALM

Require: initial latent z_0, goal latent z_g, predictor f, horizon T, inner-step count K, maximum outer iterations $K_{\max}$

Initialize $\boldsymbol{u}$ with Gaussian noise and $\boldsymbol{z} = [z_1, \dots, z_{T-1}]$ by perturbed interpolation between z_0 and z_g
Initialize $\lambda_t^m \leftarrow 0$, $\rho_t^m \leftarrow \rho_{\text{init}}$ for $t = 0, \dots, T-2$ and all m
for $k = 0, \dots, K_{\max} - 1$ **do**
  Run K Adam steps on $\mathcal{L}$ over $(\boldsymbol{z}, \boldsymbol{u})$, clipping actions to their bounds after each step
  Update the learning rates according to their schedule
  if $k = K_{\max} - 1$ **then**
    return $\boldsymbol{u}$
  end if
  Compute $c_t^m \leftarrow S_m\big(z_{t+1}^m - f(z_t, u_t)^m\big)$ for $t = 0, \dots, T-2$ and all m
  Compute the terminal residual $c_{T-1}^m \leftarrow S_m\big(z_g^m - f(z_{T-1}, u_{T-1})^m\big)$ for all m
  if $k = 0$ **then**
    $\tau_t^m \leftarrow \max(r\|c_t^m\|_2^2, \tau_{\min}^m)$ for $t = 0, \dots, T-2$ and all m ▷ calibrate tolerances
  end if
  for each $t = 0, \dots, T-2$ and all m **do**
    if $\|c_t^m\|_2^2 > \tau_t^m$ and $\rho_t^m < \rho_{\text{limit}}$ **then**
      $\rho_t^m \leftarrow \gamma \rho_t^m$
    else if $\|c_t^m\|_2^2 \geq \tau_{\min}^m$ **then**
      $\lambda_t^m \leftarrow \lambda_t^m + \eta \rho_t^m c_t^m$
    end if
    if $\|c_t^m\|_2^2 > \tau_t^m$ **then**
      $\tau_t^m \leftarrow \max(\beta_+ \tau_t^m, \tau_{\min}^m)$
    else
      $\tau_t^m \leftarrow \max(\beta_- \tau_t^m, \tau_{\min}^m)$
    end if
  end for
  if all intermediate tolerances are at their floors **or** $\max_{0 \leq t < T} \sum_m \|c_t^m\|_2^2 < \tau_{\text{conv}}$ **then**
    return $\boldsymbol{u}$
  end if
end for

Tolerance floor $\tau_{\min}$. We set $\tau_{\min}^m$ to half the median one-step prediction error of the world model on held-out transitions, measured as the mean squared error per coordinate of modality m.

Parameter	Wall	PointMaze	PushT
Action learning rate lr_a	10^{-2}	2×10^{-1}	2×10^{-2}
Environment action bounds	$[-1.8, 1.8]$	$[-1, 1]$	$[-0.34, 0.35]$
Visual tolerance floor $\tau^v_{\min}$	6×10^{-4}	6×10^{-4}	1.95×10^{-3}
Proprioceptive tolerance floor $\tau^p_{\min}$	7.4×10^{-5}	1.1×10^{-3}	4.45×10^{-5}
Latent learning rate lr_z	1		
Inner Adam steps K	5		
Maximum outer iterations $K_{\max}$	800		
Learning-rate decay	$\times 0.99$ every 10 outer iterations		
Action initialization	$\mathcal{N}(0, I)$		
Latent initialization	Interpolation with orthogonal Gaussian noise		
Latent initialization noise scale	0.95		
Goal weight α_g	$10^4 \cdot \max(1, T-4)$		
Action regularization α_u	0		
Initial penalty ρ_{init}	1		
Penalty growth factor γ	5		
Penalty cap ρ_{limit}	2×10^4		
Multiplier step η	0.1		
Initial multipliers	0		
Tolerance calibration ratio r	0.95		
Tolerance decrease / increase β_- / β_+	$\times 0.5$ / $\times 1.1$		
Stopping threshold τ_{conv}	10^{-3}		

Table 3: PALM hyperparameters. Action bounds are per coordinate, before normalization; on PushT they are the 95% quantile range of the dataset actions.

Tolerances. The tolerances are initialized from the residuals after the first inner loop: $\tau^m_t = \max(r\|c^m_t\|^2_2, \tau^m_{\min})$ for all t and m (Algorithm 1). The tolerance floor also gates the multiplier update: when the penalty is not increased, λ^m_t is updated only if $\|c^m_t\|^2_2 \geq \tau^m_{\min}$.

Penalty coefficients. Penalty coefficients are initialized at $\rho_{\mathrm{init}} = 1$ for all t and m and multiplied by $\gamma = 5$ when the growth condition is met, up to $\rho_{\mathrm{limit}} = 2 \times 10^4$.

Learning rates. We use two separate Adam optimizers, one for the states $\boldsymbol{z}$ and one for the actions $\boldsymbol{u}$, with learning rates lr_z and lr_a (Table 3). Both are decayed by a factor of 0.99 every 10 outer iterations.

A.2.5 GRASP PLANNER

GRASP fixes z_0 to the initial latent and $z_T = z_g$ to the goal, and optimizes the actions $\boldsymbol{u} = [u_0, \ldots, u_{T-1}]$ and intermediate lifted states $\boldsymbol{z} = [z_1, \ldots, z_{T-1}]$. Let

$$\hat{z}_{t+1} = f\big(\mathrm{sg}(z_t), u_t\big), \tag{5}$$

where sg denotes the stop-gradient operator. Under the supplied GRASP configuration, the lifted objective is

$$\min_{\boldsymbol{u}, \boldsymbol{z}} \frac{1}{T} \sum_{t=0}^{T-1} \|\hat{z}_{t+1} - z_{t+1}\|^2_2 + \frac{1}{T-1} \sum_{t=0}^{T-2} \|\hat{z}_{t+1} - z_g\|^2_2 . \tag{6}$$

The predictor receives $\mathrm{sg}(z_t)$ as its state input, so gradients do not propagate through z_t. GRASP additionally injects noise into the lifted states during optimization.

Every $K_{\mathrm{sync}} = 100$ lifted updates, GRASP refines the current action sequence using 25 steps of single-shooting Adam. Starting from the current actions, it autoregressively rolls out the predictor from z_0 and minimizes the terminal latent distance

$$\min_{\boldsymbol{u}} \left\| f^{[T]}(z_0, \boldsymbol{u}) - z_g \right\|^2_2 . \tag{7}$$

The lifted states are held fixed and do not enter this refinement objective. The refined actions are kept only if they lower the terminal latent loss; otherwise, the actions preceding the refinement are

restored. We terminate GRASP immediately after the last scheduled refinement: the tenth on Wall and PushT (1,001 lifted updates and 10×25 refinement updates), and the thirtieth on PointMaze (3,001 lifted updates and 30×25 refinement updates). The original GRASP evaluation protocol periodically executes intermediate plans in the simulator and counts an instance as successful if any of these plans succeeds. This uses the simulator for evaluation, which is unavailable to the planner in the open-loop setting. We therefore query the simulator only once, after optimization, to evaluate the final retained action sequence. In our experiments, allowing additional refinement phases caused the actions to leave the training distribution and substantially reduced success.

We run GRASP with the same trained world model, action representation, planning instances, and planning horizons as the other planners.

For the main comparison, we use the supplied GRASP configuration without action projection.

A.2.6 LatCo planner

We adapt the deterministic LatCo planner (Rybkin et al., 2021) by replacing its learned reward objective with our terminal latent goal loss. With $\hat{z}_0 = z_0$ fixed, it optimizes the lifted states $\hat{z}_{1:T}$ and actions $u_{0:T-1}$ using the weighted least-squares objective

$$\Phi = \kappa^2 \left[\|S(\hat{z}_T - z_g)\|_2^2 + \sum_{t=0}^{T-1} \left(\mu_t \|S(\hat{z}_{t+1} - f(\hat{z}_t, u_t))\|_2^2 + \nu_t \|[|\tilde{u}_t| - 1]_+\|_2^2 \right) \right], \tag{8}$$

where S scales the visual and proprioceptive residuals by the inverse square roots of their respective dimensions, $\tilde{u}_t$ maps the action bounds to $[-1, 1]$, μ_t and ν_t are scalar dynamics and action multipliers, and $\kappa = (1 + T^{-1} \sum_t \sqrt{\mu_t} + T^{-1} \sum_t \sqrt{\nu_t})^{-1}$. Following LatCo, each iteration takes one damped Gauss–Newton step on states and actions, then updates each multiplier with the log-ratio rule

$$\mu_t \leftarrow \mu_t \left(1 + \eta \log_{10} \frac{r_t + 0.1\,\tau}{\tau} \right), \tag{9}$$

where r_t is the corresponding squared residual and τ its threshold (τ_{dyn} for μ_t, τ_{act} for ν_t). The original LatCo uses the latent model of PlaNet (Hafner et al., 2019), whose state has 230 dimensions (200 deterministic and 30 stochastic); our latent state has 1578 dimensions (1568 visual and 10 proprioceptive). We use matrix-free conjugate gradients to solve the Gauss–Newton system for this larger latent state. We run the planner for 50 iterations and execute its final action sequence open loop. Hyperparameters are in Table 4.

A.2.7 Baseline hyperparameters

All baselines share the same planning horizon and the same trained world model. For GRASP, we use the hyperparameters of the reference implementation; a small learning-rate sweep on PointMaze did not improve over them. For LatCo, we use the damping of the original implementation.

A.2.8 Planning protocol

Each planning episode begins from a latent z_0 encoded from the start observation and a goal latent z_g encoded from the goal observation of a start–goal pair (Appendix A.1). The optimized action sequence $u_{0:T-1}$ is executed in the simulator without replanning. . Each planned action concatenates 5 environment actions, executed one per step, so a plan of T actions covers $5T$ environment steps. Episodes are optimized in batches of 50. For PALM, an episode that meets the stopping criterion is frozen, and the remaining episodes of the batch continue until they converge or reach the maximum number of iterations. LatCo solves one episode at a time.

A.2.9 Success criteria

On Wall and PointMaze, we use the success criteria of DINO-WM (Zhou et al., 2025). On PushT, only the pose of the T-shaped block is scored:

- **PointMaze**: the agent's coordinates are within 0.5 environment units of the goal.
- **Wall**: the agent's coordinates are within 4.5 pixels of the goal in a 65×65 rendering.

Planner	Parameter	Value
SS-Grad	Optimizer	Adam
	LR (Wall, PushT / PointMaze)	0.1 / 0.5
	Action penalty α_u (Wall, PushT / PointMaze)	0.01 / 0.1
	Iterations	500
CEM	Population / elites	300 / 30
	Initial mean / std	0 / 1 (normalized actions)
	Update	mean and std of the elites
	Iterations	100
LatCo	Primal step	damped Gauss–Newton (damping 10^{-3})
	Linear solver	CG, max. 50 iterations, rel. tol. 10^{-6}
	Multiplier step η	0.1
	Thresholds τ_{dyn} / τ_{act}	10^{-2} / 5×10^{-3}
	Initial multipliers μ_t / ν_t	1 / 1
	State / action init.	linear interpolation / $\mathcal{N}(0, 0.1^2)$
	Iterations	50
	Action bounds	same as PALM
GRASP	Optimizer	Adam (joint over z, u)
	LR (z / u)	0.1 / 10^{-3}
	Dynamics penalty weight	1
	State-noise scale / max	0.5 / 1.0
	State / action init.	cylinder (0.95) / $\mathcal{N}(0, 0.01^2)$
	Refinement interval	100 updates
	Refinement steps / LR	25 / 0.1
	Lifted updates (Wall, PushT / PointMaze)	1,001 / 3,001
	Action bounds (Wall, PushT / PointMaze)	none / environment bounds

Table 4: Baseline planner hyperparameters.

- **PushT**: the position error of the T-block $\|\Delta \mathrm{T}_{xy}\|_2$ is within 20 pixels in a 512×512 frame, *and* the orientation error of the T-block is within $\pi/9$ rad.

A.2.10 Compute and reproducibility

All planning experiments were run on a single NVIDIA A100 GPU per evaluation, with batches of 50 episodes in bf16 precision. Each planner evaluates all start–goal pairs of each set (284 to 300), split into 6 groups of at most 50; success rate is averaged over these groups. Wall-clock times include only the planner's optimization, not model loading, environment construction, or plan execution. Code, configuration files, and trained world-model checkpoints will be released upon publication.

A.3 Additional results

Figure 6 (left) shows Wall tasks where start and goal are on the same side of the wall, and Figure 6 (right) the average over both same-side and crossing tasks. On same-side tasks, PALM and CEM stay above 96% at all distances, while SS-Grad and GRASP drop to 71.7% and 62.0% at $k = 40$. On PushT (Table 5), success rate decreases with both translation and rotation for all planners; PALM goes from 70.0% at 0.2 rad and 25 px to 10.0% at 1.5 rad and 100 px. All planners reach similar success rates on each set. PALM is 0.3 to 11.4 points below the best baseline; the gap is at most 3.7 points on seven of the nine sets, and larger on the 25 px sets at 0.8 rad (6.5 points) and 1.5 rad (11.4 points). Figure 8 compares the final goal loss of PALM and SS-Grad on one PushT set: unlike on Wall and PointMaze (Figure 7), SS-Grad rarely ends in a high-loss local minimum, and PALM ends with a slightly higher goal loss on most pairs. Figure 10 shows success rate as a function of planning time on PushT and PointMaze rest goals. On PushT, all planners reach a similar final success rate and SS-Grad is the fastest. On PointMaze rest goals, CEM and GRASP reach a higher final success rate than PALM.

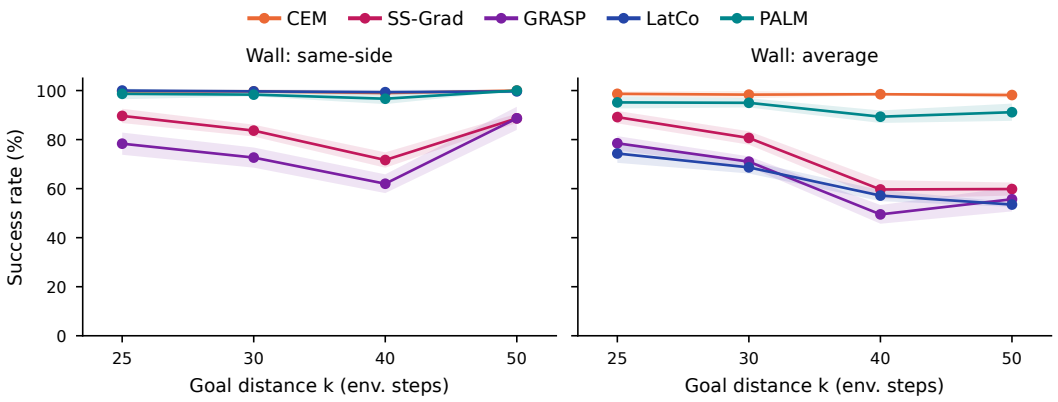


Figure 6: Success rate on Wall as a function of the goal distance k (environment steps), with planning horizon $T = k/5$. Left: start and goal are on the same side of the wall. Right: average of same-side and crossing tasks. Mean $\pm$ std over 6 groups of 50 start–goal pairs.

Table 5: Success rate (%) on each PushT set ($k = 25$, $T = 5$). CEM is sampling-based; the other planners are gradient-based. Bold: best gradient-based planner. Mean $\pm$ std over 6 groups of 50 start–goal pairs (284–300 pairs for the 1.5 rad sets). LatCo was run on one set only.

Rotation	Translation	CEM	SS-Grad	GRASP	LatCo	PALM
0.2 rad	25 px	73.3 ± 5.8	$\mathbf{71.7 \pm 5.2}$	71.0 ± 6.0	60.0 ± 6.3	70.0 ± 6.5
	50 px	51.7 ± 5.7	$\mathbf{53.3 \pm 6.4}$	50.0 ± 6.1	–	53.0 ± 8.7
	100 px	40.7 ± 6.4	$\mathbf{40.7 \pm 2.5}$	37.7 ± 4.2	–	40.0 ± 10.3
0.8 rad	25 px	52.6 ± 11.0	52.3 ± 13.3	$\mathbf{52.9 \pm 10.0}$	–	46.4 ± 9.0
	50 px	46.7 ± 5.8	$\mathbf{47.3 \pm 2.5}$	41.3 ± 5.8	–	44.7 ± 8.0
	100 px	30.3 ± 5.0	$\mathbf{31.7 \pm 3.1}$	27.7 ± 5.3	–	28.0 ± 3.5
1.5 rad	25 px	32.3 ± 8.6	$\mathbf{33.2 \pm 6.0}$	30.6 ± 7.5	–	21.8 ± 4.7
	50 px	23.6 ± 2.7	21.4 ± 3.2	$\mathbf{21.9 \pm 7.3}$	–	20.1 ± 6.4
	100 px	12.0 ± 4.2	$\mathbf{12.0 \pm 3.5}$	11.0 ± 3.8	–	10.0 ± 3.8

A.4 A CASE STUDY: THE INVERTED PENDULUM

We apply PALM to a pendulum swing-up with known dynamics to show how the shooting nodes and actions rollouts evolve during optimization.

Problem. Given an initial state $(\theta_0, \dot{\theta}_0)$, we seek actions that bring the pendulum to the upright position $\theta = \pi$ with zero velocity. To avoid the discontinuity of the angle, the state is $z_t = (\sin\theta_t, \cos\theta_t, \dot{\theta}_t)$, and the goal is $z_g = (0, -1, 0)$. The action is a scalar torque $u_t \in \mathbb{R}$. We solve

$$\min_{\boldsymbol{u},\boldsymbol{z}} \sum_{t=0}^{T-1} \|u_t\|_2^2 + \frac{\alpha_g}{2}\|f(z_{T-1}, u_{T-1}) - z_g\|_2^2 \quad \text{s.t.} \quad z_{t+1} = f(z_t, u_t),\ t = 0, \dots, T-2, \tag{10}$$

with $T = 60$ steps (3 s).

Dynamics. The dynamics f are an Euler step with $\Delta t = 0.05\,\text{s}$, $g = 9.8\,\text{m/s}^2$ and rod length $L = 1\,\text{m}$:

$$\dot{\theta}_{t+1} = \dot{\theta}_t + \Delta t \left(-\frac{g}{L}\sin\theta_t + u_t\right), \tag{11}$$

$$\sin\theta_{t+1} = \sin\theta_t \cos(\Delta t\, \dot{\theta}_{t+1}) + \cos\theta_t \sin(\Delta t\, \dot{\theta}_{t+1}), \tag{12}$$

$$\cos\theta_{t+1} = \cos\theta_t \cos(\Delta t\, \dot{\theta}_{t+1}) - \sin\theta_t \sin(\Delta t\, \dot{\theta}_{t+1}). \tag{13}$$

The last two equations keep $(\sin\theta, \cos\theta)$ on the unit circle.

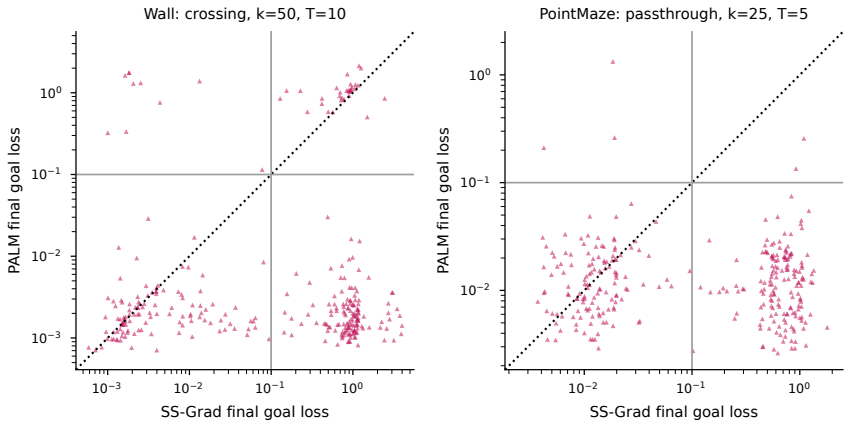


Figure 7: Final goal loss of PALM against SS-Grad, one point per start–goal pair (300 pairs per set). Points below the diagonal are pairs where PALM found actions with a lower goal loss. Grey lines mark a goal loss of 0.1.

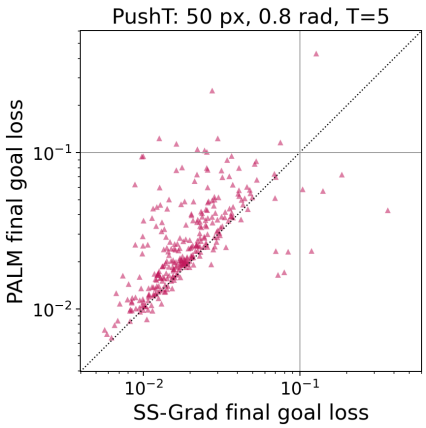


Figure 8: Final goal loss of PALM against SS-Grad on PushT (50 px, 0.8 rad, $T = 5$), one point per start–goal pair. Points below the diagonal are pairs where PALM found actions with a lower goal loss. Grey lines mark a goal loss of 0.1.

Initialization. Actions are initialized to zero. Shooting nodes are initialized by linear interpolation between z_0 and z_g, with Gaussian noise orthogonal to $z_g - z_0$, and their $(\sin\theta, \cos\theta)$ components are projected back to the unit circle.

Result. Figure 11 shows one solve, which illustrates how PALM works. Because the dynamics constraints are relaxed early in the optimization, the shooting nodes can violate them and move freely in state space: by iteration 200, they already form a swing-up trajectory that reaches the goal, while the rollout of the actions still oscillates around the start. As the penalties and multipliers grow, the dynamics residual decreases and the rollout moves closer to the nodes (iterations 340 and 440). At iteration 990, the rollout follows the nodes and reaches the goal: the trajectory found by relaxing the dynamics is now feasible. The same behavior appears on Wall (Figure 3).

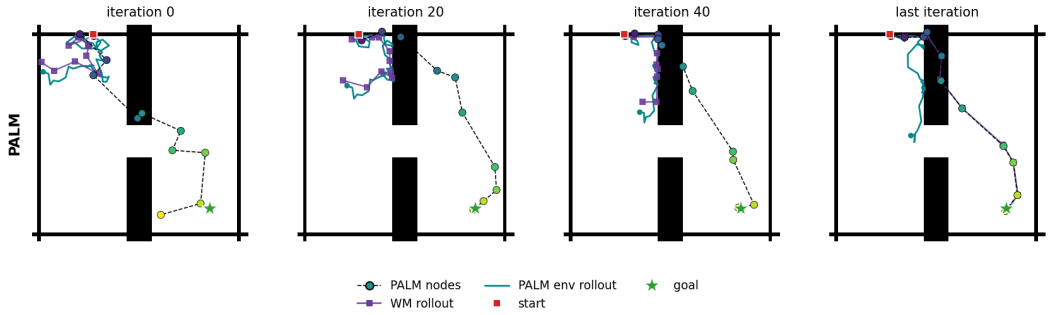


Figure 9: A Wall crossing task ($k = 50$, $T = 10$) where PALM fails. Same layout as the PALM row of Figure 3. At the last iteration, the nodes and the world-model rollout pass through the wall and reach the goal, but the environment rollout is blocked by the wall: the plan is consistent with the world model but not feasible in the environment.

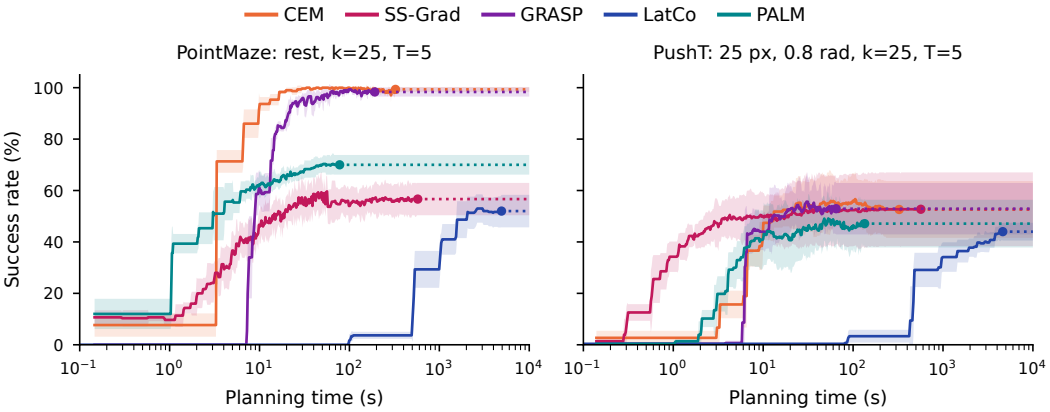


Figure 10: Success rate as a function of wall-clock planning time, for a batch of 50 start–goal pairs on one NVIDIA A100. Mean ± std over 6 groups of 50 pairs. After a planner stops, its final success rate is held (dotted).

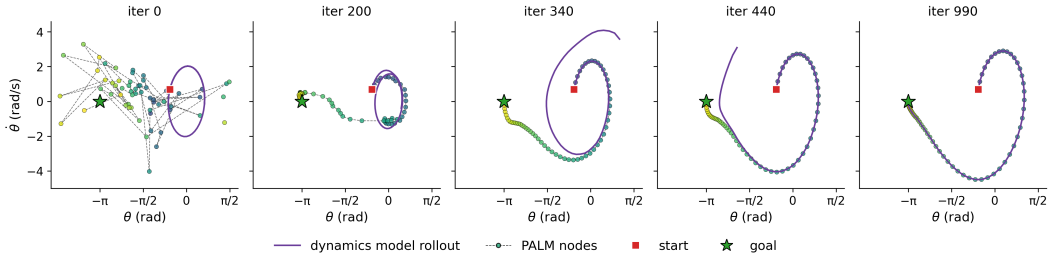


Figure 11: Five snapshots of one PALM solve on the pendulum swing-up, in the $(\theta, \dot{\theta})$ plane. Points are the shooting nodes, colored from start to end; the purple line is the rollout of the current actions from the start state.