# JOINT WORLD ACTION MODELS THROUGH DIFFERENTIABLE TRAJECTORY OPTIMIZATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Joint world action models (WAMs) generate future actions and states together, modeling world-action trajectories rather than actions alone. However, trajectory imitation does not ensure that these trajectories are task-effective or consistent with dynamics. We present **world action trajectory optimization (WATO)**, which optimizes generated actions and future states jointly for control. WATO uses both as references in a dynamics-constrained trajectory optimization problem and differentiates the task loss through the optimized trajectory to train the generator. This makes generated trajectories task-aware and dynamics-consistent while using trajectory optimization only during training. At deployment, WATO directly generates candidate trajectories and ranks them with a world model, without iterative trajectory optimization. Across 7 visual control environments, WATO improves average success from $79.9\%$ for the imitation joint WAM to $91.7\%$, outperforming strong model-based, goal-conditioned RL, and generative control baselines. Compared with cross-entropy method (CEM), WATO is on average $11.5\times$ faster and uses $600.5\times$ fewer world model evaluations at deployment.

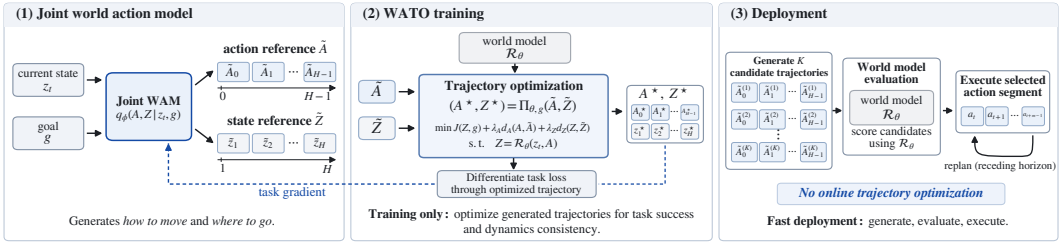


Figure 1: **WATO overview.** Joint world-action trajectories are optimized during training and amortized into the generator, enabling deployment with generation and world model evaluation only.

## 1 INTRODUCTION

Generative policies learn distributions over actions or action trajectories from data, reducing online planning (Janner et al., 2022; Chi et al., 2023; Park et al., 2025b). Recent world action models (WAMs) connect action generation with future prediction (Zhu et al., 2025; Ye et al., 2026). We focus on *joint* WAMs that generate future actions and states or observations together. We treat them as a model class over world-action trajectories, independent of architecture or scale. Unlike generative policies, joint WAMs specify both how the agent acts and where the trajectory is intended to go. This provides richer information for control, but does not make the trajectory task-effective.

A joint WAM trained by trajectory imitation mainly reproduces the offline data distribution. Its generated actions need not reach the goal, and its generated states need not match the consequences of those actions under the control dynamics. A trajectory can therefore be data-consistent yet task-ineffective or dynamics-inconsistent. The problem is not only to improve the actions, but to optimize the world-action trajectory as a whole. This leads to our central question:

*How should joint world-action trajectories be optimized for task performance while remaining consistent with action-conditioned dynamics?*



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.

Several existing approaches provide natural alternatives, but each uses only part of the world-action trajectory. Model-based planners optimize actions through world model rollouts (Chua et al., 2018; Maes et al., 2026), but omit generated future states and require online optimization. Task losses can be backpropagated through learned dynamics (Hafner et al., 2020), but supervise only the actions. Consistency objectives can align generated states with world model predictions (Huang et al., 2026), but use them as prediction targets rather than control references. Optimized trajectories can also be distilled into a policy (Pinneri et al., 2021), but detached targets discard how the solution depends on the generated trajectory. None of these approaches uses both generated actions and future states to parameterize trajectory optimization and learn through its optimized solution.

We introduce **world action trajectory optimization (WATO)**. WATO uses generated actions and future states as references in a dynamics-constrained trajectory optimization problem. The WAM proposes an intended trajectory, while a world model predicts the consequences of its actions (Zhou et al., 2025a; Sobal et al., 2025; Maes et al., 2026). The task objective drives the trajectory toward the goal. The action reference specifies how to move, and the state reference specifies where to go. Trajectory optimization combines both references with the world model to produce a task-directed, dynamics-consistent trajectory.

The generated world-action trajectory does not only initialize the optimizer, but it also parameterizes the optimization objective. Changing either the actions or future states can therefore change the optimized trajectory. WATO differentiates the task loss through this solution using implicit differentiation (Amos et al., 2018; Wan et al., 2024), providing task gradients to both generated actions and states. Flow matching provides offline trajectory imitation (Lipman et al., 2023), while trajectory optimization makes the joint WAM task-aware.

Trajectory optimization is used only during training. At deployment, WATO directly generates candidate world-action trajectories and evaluates their action sequences with the world model, i.e. action-conditioned dynamics. Deployment performs a lightweight best-of-$K$ selection over these proposals, but no iterative trajectory optimization is required.

We evaluate WATO on 7 visual control environments spanning navigation, reaching, contact-rich manipulation, and multi-stage goal-conditioned control. WATO improves average success from $79.9\%$ for the imitation-trained joint WAM to $91.7\%$, outperforming strong model-based, goal-conditioned RL, and generative control baselines. Compared with CEM, WATO is $11.5\times$ faster and uses $600.5\times$ fewer world model evaluations. We also show that differentiating through trajectory optimization outperforms direct rollout, consistency regularization, and distillation. Removing either the action or state reference reduces performance, showing that both are crucial for control.

Our contributions are threefold:

- We study joint WAMs as generative models over world-action trajectories and formulate task-aware world-action generation as dynamics-constrained trajectory optimization.
- We introduce **WATO**, which uses generated actions and future states as optimization references and differentiates the task loss through the optimized trajectory to train the generator.
- We show across diverse control tasks that WATO improves joint world-action generation, outperforms strong model-based and learning-based baselines, and amortizes trajectory optimization into the generator for efficient deployment.

## 2 Preliminaries

**Visual offline goal-conditioned control.** We consider a reward-free offline dataset of $N$ visual trajectories, $\mathcal{D} = \{\tau^{(n)}\}_{n=1}^{N}$, where $\tau^{(n)} = (o_0^{(n)}, a_0^{(n)}, \dots, o_{T_n}^{(n)})$, with visual observations $o_t \in \mathcal{O}$ and primitive actions $a_t \in \mathcal{A}$. We group $B$ consecutive actions into an action block $A_h = (a_{t+hB}, \dots, a_{t+(h+1)B-1}) \in \mathcal{U}$, where $\mathcal{U} := \mathcal{A}^B$. An action trajectory $A = (A_0, \dots, A_{H-1}) \in \mathcal{U}^H$ spans $L = BH$ environment steps.

We use pretrained latent world models following LeWM (Maes et al., 2026). The encoder maps an observation to a latent state $z_t = E_\theta(o_t)$, and the action-conditioned predictor rolls out future latent states from an action trajectory. Given $z_t$ and $A$, we write

$$\hat{Z}(A) = \mathcal{R}_\theta(z_t, A) = (\hat{z}_1, \dots, \hat{z}_H). \tag{1}$$

A goal observation $o_g$ is encoded as $g = E_\theta(o_g)$. For a latent trajectory $Z = (z_1, \dots, z_H)$, we use the terminal goal cost $J(Z, g) = \|z_H - g\|_2^2$. Control proceeds in receding horizon by repeatedly generating an action trajectory, executing a segment, and planning from the new observation.

**Generative policies and joint WAMs.** Let $c = (z_t, g)$ denote the context that includes the current-state and goal condition. An action-only generative policy models $\tilde{A} \sim q_\phi^A(A \mid c)$. We focus on *joint* WAMs, which instead model future actions and states together:

$$(\tilde{A}, \tilde{Z}) \sim q_\phi(A, Z \mid c), \qquad \tilde{Z} = (\tilde{z}_1, \dots, \tilde{z}_H). \tag{2}$$

A joint WAM therefore induces an action-only distribution through the marginal $q_{\phi,A}(A \mid c) = \int q_\phi(A, Z \mid c)\, dZ$. The distinction is the generated object: an action-only policy generates how to act, while a joint WAM additionally generates an intended future trajectory.

The generated future $\tilde{Z}$ is distinct from the world model prediction $\hat{Z}(\tilde{A}) = \mathcal{R}_\theta(z_t, \tilde{A})$. The joint WAM proposes both actions and future states, while the pretrained world model predicts the states induced by those actions. Throughout the paper, $\tilde{\cdot}$ denotes a generated proposal, $\hat{\cdot}$ a world model prediction, and $\cdot^\star$ an optimized trajectory.

We parameterize the joint WAM with conditional flow matching (Lipman et al., 2023). For data $x$, noise $\epsilon \sim \mathcal{N}(0, I)$, condition $c$, and flow time $s \sim \mathrm{Unif}[0, 1]$, we use the linear path $x_s = (1 - s)\epsilon + sx$ and train $v_\phi(x_s, s; c)$ toward $x - \epsilon$. Section 3 applies this objective to joint action-state trajectories and makes the resulting generator task-aware through trajectory optimization.

# 3 World Action Trajectory Optimization

## 3.1 Imitating World-Action Trajectories Is Not Enough

We first train a joint world action flow (WAF) to imitate state-action trajectories from the offline dataset. For a local trajectory, let $X = (A, Z)$ and condition the generator on the current state and goal, $c = (z_t, g)$. For each training window, we use its final observation as the goal. The WAF generates actions and future states jointly as $(\tilde{A}, \tilde{Z}) \sim q_\phi(A, Z \mid z_t, g)$.

We parameterize $q_\phi$ with conditional flow matching. Let $\epsilon = (\epsilon_A, \epsilon_Z) \sim \mathcal{N}(0, I)$, $s \sim \mathrm{Unif}[0, 1]$, and $X_s = (1 - s)\epsilon + sX$. The joint vector field $v_\phi = (v_\phi^A, v_\phi^Z)$ is trained with

$$\mathcal{L}_{\mathrm{WAF}} = \mathbb{E}\left[\mathrm{MSE}\left(v_\phi^A(X_s, s; c), A - \epsilon_A\right) + \lambda_{\mathrm{WAF}}\, \mathrm{MSE}\left(v_\phi^Z(X_s, s; c), Z - \epsilon_Z\right)\right]. \tag{3}$$

This objective imitates the joint state-action distribution in the offline data, but does not optimize trajectories for control. The generated actions need not reach the goal under the pretrained dynamics, so $J(\mathcal{R}_\theta(z_t, \tilde{A}), g)$ can remain large. The generated states also need not match the consequences of those actions, so $\tilde{Z}$ need not equal $\mathcal{R}_\theta(z_t, \tilde{A})$. Imitation alone therefore provides neither task optimality nor per-sample dynamics-consistency.

## 3.2 Why Not Optimize the Generator Directly?

A simple way to provide task-awareness is to evaluate its generated actions through the world model. Direct rollout training minimizes $\mathcal{L}_{\mathrm{rollout}} = J(\mathcal{R}_\theta(z_t, \tilde{A}), g)$. This provides a task gradient to $\tilde{A}$, but $\tilde{Z}$ does not affect action optimization. The joint WAF is thus used only as an action generator.

A stronger variant can also match the generated states to the world model rollout. For example, one can minimize $\mathcal{L}_{\mathrm{direct+cons}} = J(\mathcal{R}_\theta(z_t, \tilde{A}), g) + \beta d_Z(\mathcal{R}_\theta(z_t, \tilde{A}), \tilde{Z})$. This encourages dynamics-consistency, but still trains $\tilde{Z}$ to follow the rollout rather than to guide trajectory optimization.

Another alternative is optimization distillation. Given a generated proposal $\tilde{\tau} = (\tilde{A}, \tilde{Z})$, a trajectory optimizer produces an optimized trajectory $\tau^\star$, which is then treated as a detached regression target. This transfers optimized trajectories to the generator, but discards how $\tau^\star$ changes with $\tilde{\tau}$. The optimizer provides targets rather than gradients through the optimized solution.

These alternatives either leave the generated future outside trajectory optimization or use optimization only to produce detached targets. WATO instead uses the full generated state-action trajectory to define the optimization problem and learns through its optimized solution.

### 3.3 Dynamics-Constrained Trajectory Projection

WATO uses the generated trajectory as a reference for trajectory optimization rather than executing it directly. For the current state $z_t$, define the set of trajectories consistent with the pretrained world model as $\mathcal{F}_\theta(z_t) = \{(A, Z) : A \in \mathcal{U}^H,\ Z = \mathcal{R}_\theta(z_t, A)\}$. Given a generated reference $\tilde{\tau} = (\tilde{A}, \tilde{Z})$, we define the *dynamics-constrained trajectory projection*

$$\Pi_{\theta,g}(\tilde{\tau}) = \operatorname*{arg\,min}_{(A,Z)\in\mathcal{F}_\theta(z_t)} J(Z, g) + \lambda_A d_A(A, \tilde{A}) + \lambda_Z d_Z(Z, \tilde{Z}) \tag{4}$$

The task cost drives the trajectory toward the goal. The action reference preserves how the WAM proposes to move, while the state reference preserves where it proposes to go. The world model constrains these references through action-conditioned dynamics.

We use $d_A(A, \tilde{A}) = H^{-1}\sum_{h=0}^{H-1} \|A_h - \tilde{A}_h\|_2^2$ and $d_Z(Z, \tilde{Z}) = H^{-1}\sum_{h=1}^{H} \|\mathcal{N}_Z(z_h) - \mathcal{N}_Z(\tilde{z}_h)\|_2^2$, where $\mathcal{N}_Z$ is a fixed latent standardization. We solve Equation (4) in reduced space, with the action trajectory as the only optimization variable and $Z(A) = \mathcal{R}_\theta(z_t, A)$. The optimization becomes

$$A^\star = \operatorname*{arg\,min}_{A\in\mathcal{U}^H} J(Z(A), g) + \lambda_A d_A(A, \tilde{A}) + \lambda_Z d_Z(Z(A), \tilde{Z}), \qquad Z^\star = Z(A^\star). \tag{5}$$

The optimized trajectory is task-directed and dynamics-consistent while remaining close to the generated state-action trajectory. Importantly, $\tilde{Z}$ is a reference, not an independently optimized state trajectory. Dynamics-consistency follows from the world model rollout rather than an additional penalty. Because both $\tilde{A}$ and $\tilde{Z}$ parameterize the optimization objective, changing either reference can change the optimized trajectory. We next characterize this dependence and show how generated future states shape action optimization through the learned dynamics.

### 3.4 Learning Through the Optimized Trajectory

The generated trajectory parameterizes the trajectory optimization problem, not only its initialization. Define the task objective $F(A) = J(\mathcal{R}_\theta(z_t, A), g)$ and the standardized rollout $r(A) = \operatorname{vec}\mathcal{N}_Z(\mathcal{R}_\theta(z_t, A))$. We similarly write $\tilde{r} = \operatorname{vec}\mathcal{N}_Z(\tilde{Z})$. The reduced objective in Equation (5) is $\Psi(A; \tilde{A}, \tilde{r}) = F(A) + \lambda_A H^{-1}\|A - \tilde{A}\|_2^2 + \lambda_Z H^{-1}\|r(A) - \tilde{r}\|_2^2$.

Let $S_\star = \partial r(A)/\partial A|_{A=A^\star}$ denote the standardized world model rollout Jacobian. For an interior local solution, the first-order condition is

$$\nabla F(A^\star) + \frac{2\lambda_A}{H}(A^\star - \tilde{A}) + \frac{2\lambda_Z}{H} S_\star^\top \big(r(A^\star) - \tilde{r}\big) = 0. \tag{6}$$

The action reference acts directly in action space. The state reference acts through $S_\star^\top$, which maps a future-state error into an action-space optimization direction. Generated future states therefore affect trajectory optimization through the action-conditioned world dynamics.

**Proposition 1 (Dynamics-aware reference sensitivity).** Suppose $A^\star$ is an interior local solution, $\Psi$ is twice continuously differentiable near $A^\star$, and $\mathcal{H}_\star := \nabla^2_{AA}\Psi(A^\star; \tilde{A}, \tilde{r})$ is nonsingular. Then

$$\frac{\partial A^\star}{\partial \tilde{A}} = \frac{2\lambda_A}{H}\mathcal{H}_\star^{-1}, \qquad \frac{\partial A^\star}{\partial \tilde{r}} = \frac{2\lambda_Z}{H}\mathcal{H}_\star^{-1} S_\star^\top \tag{7}$$

Actions affect the optimized trajectory through the local optimization Hessian. Future states additionally pass through the rollout Jacobian, so their influence depends on how actions change the predicted future. Both references can thus receive task gradients through the optimized solution.

This structure also changes the local geometry of action optimization. The state-reference term contributes $2S_\star^\top S_\star$ plus a residual-weighted second-order term from the rollout. When the state-reference residual is small, the local Hessian is approximated by

$$\mathcal{H}_\star \approx \nabla^2 F(A^\star) + \frac{2\lambda_A}{H} I + \frac{2\lambda_Z}{H} S_\star^\top S_\star. \tag{8}$$

**Corollary 1 (Dynamics-induced local geometry).** The action reference contributes isotropic curvature in action space, while the state reference contributes the positive-semidefinite term $S_\star^\top S_\star$. Action perturbations are therefore weighted by how strongly they change the predicted future. Generated states thus shape the local geometry of trajectory optimization through the learned dynamics.

We compute these sensitivities by implicit differentiation of the trajectory optimality conditions. Further derivations and local analysis are given in Appendix C.

### 3.5 The WATO Objective

Let $\tilde{\tau}_\phi = (\tilde{A}, \tilde{Z}) \sim q_\phi(A, Z \mid z_t, g)$ denote a generated trajectory and $\tau^\star = (A^\star, Z^\star) = \Pi_{\theta,g}(\tilde{\tau}_\phi)$ its optimized trajectory. WATO trains the same joint WAF with both imitation and task supervision:

$$\mathcal{L}_{\text{WATO}} = \mathcal{L}_{\text{WAF}} + \lambda_{\text{task}} \mathbb{E}\left[J(Z^\star, g)\right] \tag{9}$$

The flow matching term continues to imitate offline state-action trajectories, while the task term favors generated references that lead to better optimized trajectories.

The task gradient passes through the optimized trajectory as

$$\frac{dJ(Z^\star, g)}{d\phi} = \frac{\partial J}{\partial \tau^\star} \underbrace{\frac{\partial \tau^\star}{\partial \tilde{\tau}_\phi}}_{\text{solution sensitivity}} \frac{\partial \tilde{\tau}_\phi}{\partial \phi}. \tag{10}$$

By Proposition 1, action gradients pass through the local optimization Hessian, while state gradients additionally pass through the rollout Jacobian. The world model is differentiable when computing these gradients, but its parameters $\theta$ remain fixed. WATO updates only the generator parameters $\phi$.

### 3.6 Training and Deployment

WATO uses two training stages. We first pretrain the joint WAF with $\mathcal{L}_{\text{WAF}}$ to imitate offline state-action trajectories. We then continue training the same generator with $\mathcal{L}_{\text{WATO}}$, differentiating through trajectory optimization to provide task supervision.

Trajectory optimization is removed after training. At deployment, WATO generates $K$ independent proposals $(\tilde{A}^{(k)}, \tilde{Z}^{(k)}) \sim q_\phi(A, Z \mid z_t, g)$. Each action proposal is rolled out with the pretrained world model as $\hat{Z}^{(k)} = \mathcal{R}_\theta(z_t, \tilde{A}^{(k)})$. The controller selects $k^\star = \arg\min_k J(\hat{Z}^{(k)}, g)$ and executes the selected action segment in receding horizon. The generated state trajectory is not directly scored at deployment. It contributes through joint generation and trajectory-optimization training.

Optimization is used to train the generator, *not* to operate the deployed controller. Deployment requires only trajectory generation and world model evaluation. Full algorithm is provided in Algorithm 1.

**Remark: Optimization amortization.** Under local regularity assumptions, small projection residuals bound the task-cost gap between generated and optimized trajectories and the generated trajectory's inconsistency with the world model (see Appendix C.4).

## 4 Related Work

**Generative policies and WAMs.** Generative policies learn distributions over actions or trajectories with multimodality (Janner et al., 2022; Ajay et al., 2023; Chi et al., 2023; Park et al., 2025b). WAMs extend action generation by jointly modeling future actions and states or observations (Zhu et al., 2025; Cen et al., 2025; Ye et al., 2026). Recent work also studies how generated futures should be used at inference time (Yuan et al., 2026; Zhao et al., 2026). WATO instead optimizes the generated world-action trajectory for control. Generated actions and future states jointly define a dynamics-constrained optimization problem, and the generator is trained through its solution.

**World models and model-based planning.** World models learn action-conditioned dynamics and use predicted futures to evaluate and/or optimize actions (Hafner et al., 2019; Hansen et al., 2024;

**Algorithm 1** World Action Trajectory Optimization (WATO)

**Require:** Offline dataset $\mathcal{D}$, pretrained world model $\mathcal{R}_\theta$ (Equation (1))
▷ WAF pretraining
Pretrain joint WAF $q_\phi(A, Z \mid z_t, g)$ with flow matching (Equation (3))
▷ WATO training loop
**while** not converged **do**
  Sample $(z_t, g, A, Z) \sim \mathcal{D}$
  Generate $(\tilde{A}, \tilde{Z}) \sim q_\phi(A, Z \mid z_t, g)$ (Equation (2))
  ▷ Dynamics-constrained trajectory optimization
  Compute $(A^\star, Z^\star) = \Pi_{\theta,g}(\tilde{A}, \tilde{Z})$ (Equation (4))
  ▷ Learning through the optimized trajectory
  Compute task loss $J(Z^\star, g)$
  Differentiate through the optimized trajectory (Equation (10))
  Update $\phi$ with $\mathcal{L}_{\text{WATO}}$ (Equation (9))
▷ Deployment
Generate $K$ proposals $(\tilde{A}^{(k)}, \tilde{Z}^{(k)}) \sim q_\phi(A, Z \mid z_t, g)$
Roll out $\hat{Z}^{(k)} = \mathcal{R}_\theta(z_t, \tilde{A}^{(k)})$ for each proposal
Select $k^\star = \arg\min_k J(\hat{Z}^{(k)}, g)$
Execute the selected action segment and replan

Sobal et al., 2025; Maes et al., 2026). Model-based planners commonly optimize action sequences online using sampling or gradient-based methods (Power & Berenson, 2022; Zhou et al., 2025b). WATO uses a pretrained world model to constrain trajectory optimization during training. Deployment only requires trajectory generation and world model evaluation, without online optimization.

**Differentiable optimization and generative control.** Differentiable optimization exposes how an optimized trajectory changes with model, cost, or policy parameters (Amos et al., 2018; Amos & Yarats, 2020). Prior work uses this principle to train policies through optimization or predictive control (Xu & She, 2024; Xu et al., 2025; Wan et al., 2024). Other methods combine generative models with trajectory optimization, model-based projections, or MPC (Römer et al., 2024; Mizuta & Leung, 2026; Idoko et al., 2026; Li et al., 2026). WATO uses both actions and future states to define the optimization objective, then differentiates task performance through its solution. Thus, differentiable optimization serves as a learning mechanism for WAMs rather than refinement only.

**Offline goal-conditioned control.** Offline goal-conditioned RL (GCRL) learns to reach goals from fixed trajectory data. Prior methods include goal-conditioned behavior cloning, value learning, quasimetric learning, and hierarchical RL (Ding et al., 2019; Kostrikov et al., 2022; Wang et al., 2023; Park et al., 2023; 2025a). WATO addresses the same downstream goal-reaching problem from a different direction. It starts from a joint WAF and makes its trajectories task-aware using a world model and differentiable trajectory optimization. We compare against these methods as goal-conditioned control baselines.

## 5 Experiments

We evaluate WATO along four questions. First, does world-action trajectory optimization improve goal-conditioned control over model-based, generative, and RL methods? Second, is differentiating through trajectory optimization more effective than direct task training or optimization distillation? Third, do generated future states improve control beyond action generation alone? Finally, is trajectory optimization successfully transferred into the generator during training?

### 5.1 Experimental Setup

**Environments.** We evaluate WATO on 7 environments covering distinct control challenges. `TwoRooms` tests visual navigation, `Reacher` tests continuous reaching, and `PushT` tests contact-rich pushing (Maes et al., 2026). We additionally use four visual manipulation tasks from OGBench: `cube-{single, double}`, `scene-play`, and `puzzle-3x3` (Park et al., 2025a). Together, these environments cover navigation, reaching, manipulation, and multi-stage goal-conditioned control.

**Evaluation.** We evaluate goal reaching from offline data. Each initial observation is paired with a goal observation $L = 25$ primitive steps later in a evaluation trajectory, and the controller receives a 50-step execution budget. Unless otherwise stated, WATO generates $K = 8$ candidates using 8 flow integration steps (NFE $= 8$) at deployment. Success rate is the primary metric. We also report final goal error, latency, and world model evaluations in the appendix.

**Baselines.** We compare against three groups of methods. For *goal-conditioned learning*, we use GCBC (Ding et al., 2019), GCIVL (Kostrikov et al., 2022), GCIQL (Park et al., 2025a), and QRL (Wang et al., 2023). For *generative control*, we compare with an flow policy, a joint WAF trained only by flow matching, and generative predictive control (GPC) (Kurtz & Burdick, 2026), adapted to our offline setting. For *model-based control*, we use CEM (Maes et al., 2026) and gradient-based MPC. GPC, CEM, gradient-based MPC, and WATO use the same pretrained LeWM. Model-based planners also use the same task cost. All methods use the same data, goal distribution, and evaluation protocol. Training details are given in Appendix F.2.

### 5.2 How Good is WATO for Visual Offline Goal-Conditioned Control?

We first evaluate WATO across all 7 environments. Table 1 reports the success rates. WATO achieves the highest macro-average success of 91.7% and is best or tied for best on six of 7 environments, while GPC performs best on `Reacher`.

Compared with the joint WAF, WATO improves average success from 79.9% to 91.7%. This 11.8 percentage points (pp) gain shows that trajectory imitation alone does not fully exploit the generated state-action trajectory. Optimizing it during training makes the generator more effective for control.

WATO also outperforms online model-based planning. CEM and gradient-based MPC optimize actions at deployment using the same world model, achieving average success rates of 65.0% and 76.7%, respectively. WATO uses optimization only during training and deploys the generator without online optimization. Its gains therefore come from improving the generator rather than additional extensive test-time search. Section 5.5 examines this transfer of optimization into the generator.

Table 1: **Overall control performance.** Success rates across **7** visual control environments.

| | Goal-conditioned learning | | | | Generative control | | | Model-based control | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Environment | GCBC | GCIVL | GCIQL | QRL | Flow Policy | Joint WAF | GPC | CEM | Grad. MPC | **WATO** |
| `TwoRooms` | 81 ±2 | 77 ±3 | 80 ±5 | 85 ±2 | 79 ±2 | 83 ±1 | 87 ±2 | 86 ±3 | **92** ±3 | **93** ±0 |
| `Reacher` | 74 ±3 | 65 ±3 | 71 ±4 | 73 ±3 | 64 ±5 | 67 ±3 | **91** ±2 | 70 ±2 | 81 ±4 | 85 ±2 |
| `PushT` | 66 ±1 | 45 ±2 | 39 ±0 | 58 ±1 | 77 ±2 | 84 ±4 | **99** ±1 | 82 ±2 | 97 ±3 | **100** ±0 |
| `cube-single` | 75 ±0 | 45 ±3 | 52 ±1 | 72 ±3 | 81 ±3 | 88 ±1 | 78 ±3 | 68 ±8 | 76 ±2 | **99** ±1 |
| `cube-double` | 63 ±1 | 21 ±4 | 37 ±5 | 60 ±1 | 62 ±4 | **89** ±2 | 64 ±7 | 58 ±3 | 71 ±1 | **90** ±3 |
| `scene-play` | 55 ±2 | 32 ±2 | 40 ±4 | 58 ±0 | 68 ±3 | 78 ±3 | 37 ±2 | 42 ±11 | 59 ±5 | **94** ±1 |
| `puzzle-3x3` | 51 ±2 | 49 ±8 | 37 ±6 | 45 ±1 | 54 ±3 | 70 ±3 | 39 ±5 | 49 ±4 | 61 ±2 | **81** ±2 |
| Average | 66.4 | 47.7 | 50.9 | 64.4 | 69.3 | 79.9 | 70.7 | 65.0 | 76.7 | **91.7** |

[1] Environments except for the top three are from OGBench (Park et al., 2025a) `visual-v0`.
[2] Success rates are mean ± std (%) over **4** seeds, with **100** tests per seed. Bold indicates the best mean and results within 2 pp of it.

### 5.3 What Does WATO Generate?

Figure 2 separates single-step candidate quality from full closed-loop behavior. A good local proposal does not necessarily yield reliable control over repeated planning.

At one planning step, WATO generates only $K = 8$ concentrated, low-error candidates. CEM uses 300 rollouts but produces a broader candidate set, while the flow policy includes several large-error proposals. GCIQL and GCBC provide only a single proposal without candidate selection. WATO therefore achieves high-quality candidates with little test-time search.

Across eight heterogeneous episodes, WATO succeeds in $8/8$, compared with $7/8$ for the joint WAF, $6/8$ for CEM, and $5/8$ for the flow policy. Relative to the joint WAF, WATO yields a tighter candidate distribution and more reliable closed-loop trajectories. CEM also shows that more online search does not necessarily improve reliability. GCBC further illustrates the gap between local proposal quality and closed-loop behavior: its proposal in Fig. 2(a) can lie near the goal, yet only $5/8$ episodes succeed in Fig. 2(b). Overall, WATO combines sample-efficient candidate generation with stable repeated planning.

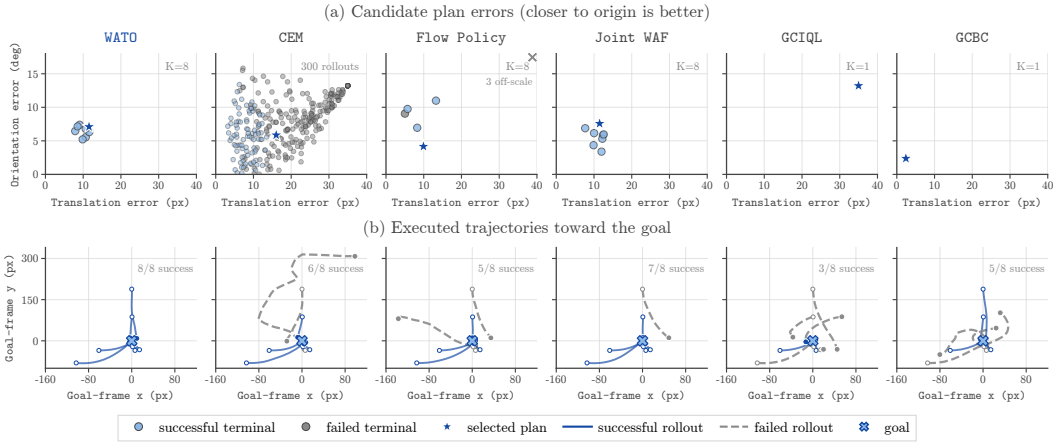


Figure 2: **Visualizing control behavior in `PushT`.** *(a)* Terminal object-position and orientation errors for candidates from a *single planning step* in one representative episode. Each candidate is replayed from the same state. *(b)* Goal-aligned object trajectories over 8 *full episodes* with heterogeneous initializations, with the goal at the origin.

### 5.4 How Beneficial is Differentiation Through Trajectory Optimization?

We next ask whether WATO benefits specifically from differentiating through trajectory optimization. All variants use the same joint WAF, offline data, and world model. Only the training objective changes. *Joint WAF* uses flow matching alone. *Direct rollout* adds the goal-reaching loss through the world model rollout, supervising the generated actions. *Direct rollout + consistency* also matches states to the rollout. *Distillation* trains the generator toward detached trajectories produced by the same optimizer. WATO instead differentiates through the optimized solution, allowing both action and state references to receive gradients according to how they affect that solution.

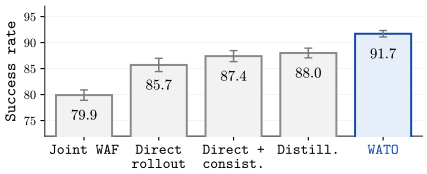


Figure 3: **Training objective ablation.** Average success across all environments.

Figure 3 shows a steady improvement across these objectives. Joint WAF achieves 79.9%, direct rollout 85.7%, direct rollout with consistency 87.4%, and distillation 88.0%. WATO reaches 91.7%, indicating that differentiating through optimization provides more effective supervision than direct task losses, consistency regularization, or detached optimized targets.

### 5.5 Is Trajectory Optimization Well-Amortized through WATO?

WATO uses trajectory optimization during training but removes it at deployment. We test whether this optimization is transferred into the generator by applying the same trajectory optimization layer before and after WATO training.

Table 2 reports the additional success obtained from online trajectory optimization. Before WATO training, optimization improves success by 10.4 pp on average. After WATO training, the remaining gain drops to 1.3 pp. Thus, WATO learns to generate goal-reaching and dynamics-consistent trajectories that already capture most of the benefit of trajectory optimization.

| Environment | Before WATO | After WATO |
|---|---|---|
| `TwoRooms` | +8.0 | +1.0 |
| `Reacher` | +15.0 | +3.0 |
| `PushT` | +14.0 | +0.0 |
| `cube-single` | +9.0 | +1.0 |
| `cube-double` | +3.0 | +1.0 |
| `scene-play` | +14.0 | +1.0 |
| `puzzle-3x3` | +10.0 | +2.0 |
| Average | **+10.4** | **+1.3** |

Table 2: **Remaining benefit of online trajectory optimization is small.** Additional success from the same optimizer before and after WATO training, in percentage points.

### 5.6 Should We Generate Both States and Actions?

We next isolate the roles of generated actions and future states. The *action-only* variant removes $d_Z$, while the *state-only* variant removes $d_A$. They achieve average success rates of 85.7% and

85.0%, respectively, compared with 91.7% for full WATO. Thus, neither reference alone recovers the performance of their combination. Generated future states therefore contribute directly to trajectory optimization rather than serving only as auxiliary predictions. This result is consistent with Proposition 1: state references influence optimized actions through the world model rollout Jacobian. We further compare generated and rollout-derived state references to test whether generated futures provide benefits beyond dynamics-aware regularization (Appendix G.1).

### 5.7 How Does Trajectory Optimization Shape the Generator?

We next examine how the generator changes during WATO training on three environments: `PushT`, `cube-double`, and `scene-play`. For each generated world-action trajectory, we measure $d_A(A^\star, \tilde{A})$, $d_Z(Z^\star, \tilde{Z})$, and the task improvement $\Delta J = J(\mathcal{R}_\theta(z_t, \tilde{A}), g) - J(Z^\star, g)$ (see Sec. 3.3 for definitions).

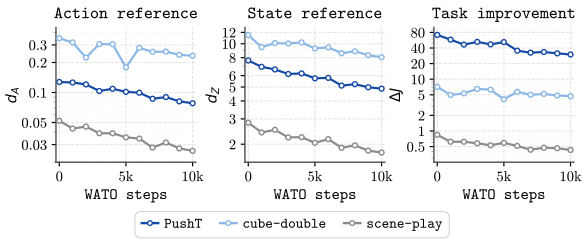


Figure 4: **Trajectory projection during WATO training.** Training dynamics on three representative environments, evaluated every 1,000 WATO steps.

Figure 4 shows that all three metrics decrease despite some fluctuations. From initialization to 10k WATO steps, the action-reference distance decreases by 39.3% on average across the three environments, the state-reference distance by 30.5%, and the remaining goal-reaching task improvement by 31.8%. The generator therefore moves closer to its optimized trajectories while requiring progressively less correction from trajectory optimization. This provides direct evidence that trajectory optimization is amortized into world-action generation.

### 5.8 How Efficient is WATO at Deployment?

Table 3 compares WATO with CEM at deployment. WATO reduces latency in every environment, with speedups from 5.8× to 25.0× and a geometric mean of 11.5×. World model evaluations decrease more sharply, from 375× to 1125×, with a geometric mean of 600.5×.

The larger reduction in world model evaluations than in latency shows that model evaluations alone do not determine wall-clock time, because trajectory generation and execution remain. WATO and the joint WAF use the same deployment procedure, so WATO's control gains require no additional test-time computation. Absolute latency, evaluation counts, and timing details are reported in Appendix H.

| Environment | Latency | WM evals |
|---|---|---|
| TwoRooms | 18.9× | 1125× |
| Reacher | 13.6× | 1125× |
| PushT | 25.0× | 1125× |
| cube-single | 5.8× | 375× |
| cube-double | 6.3× | 375× |
| scene-play | 12.2× | 375× |
| puzzle-3x3 | 9.2× | 375× |
| Geometric mean | **11.5×** | **600.5×** |

Table 3: **Efficiency relative to CEM at deployment.** Ratios are $T_{\text{CEM}}/T_{\text{WATO}}$ and $N_{\text{CEM}}/N_{\text{WATO}}$. Higher is better.

## 6 Discussion and Conclusion

WATO shows that trajectory optimization can train joint WAMs rather than only refine actions at deployment. Across 7 environments, it improves average success from 79.9% for the joint WAF to 91.7%, with ablations showing that both action and state references contribute. Decreasing projection residuals and optimization benefit further indicate that WATO transfers trajectory optimization into the generator. Generated future states also play a direct control role: as trajectory references, they shape optimized actions through the world model dynamics.

World models and joint WAMs therefore play complementary roles. The joint WAM proposes trajectories, while the pretrained world model provides action-conditioned dynamics during training and evaluates actions at deployment. WATO uses this model to improve generation without iterative online planning. Limitations include dependence on learned dynamics, the finite-horizon goal-conditioned setting, and evaluation only with flow matching. Extending WATO to robust longer-horizon control and other joint generative models remains open.

## AI USE STATEMENT

Generative AI was *only* used to assist with grammar checking and writing polish. *All ideas, scientific content, experimental results, and conclusions are the authors' own.*

## REPRODUCIBILITY STATEMENT

Implementation details, model hyperparameters, optimization procedures, and evaluation protocols are provided in the appendix. We also include anonymous code to reproduce the experiments.

## REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Anurag Ajay, Yilun Du, Abhi Gupta, Joshua B. Tenenbaum, Tommi S. Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision making? In *The Eleventh International Conference on Learning Representations*, 2023.

Brandon Amos and Denis Yarats. The differentiable cross-entropy method. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 291–302. PMLR, 2020.

Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J. Zico Kolter. Differentiable MPC for end-to-end planning and control. In *Advances in Neural Information Processing Systems*, volume 31, pp. 8289–8300, 2018.

Jun Cen, Chaohui Yu, Hangjie Yuan, Yuming Jiang, Siteng Huang, Jiayan Guo, Xin Li, Yibing Song, Hao Luo, Fan Wang, Deli Zhao, and Hao Chen. WorldVLA: Towards Autoregressive Action World Model, 2025.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin CM Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems*, 2023.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

Yiming Ding, Carlos Florensa, Pieter Abbeel, and Mariano Phielipp. Goal-conditioned imitation learning. *Advances in neural information processing systems*, 32, 2019.

Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Devin, Benjamin Eysenbach, and Sergey Levine. Learning to reach goals via iterated supervised learning. In *International Conference on Learning Representations*, 2021.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020.

Nick Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations*, volume 2024, pp. 47376–47405, 2024.

Hsiang-Wei Huang, Jianxu Shangguan, Junbin Lu, and Jenq-Neng Hwang. Leflow: Generative latent flow planning for world models. *arXiv preprint arXiv:2608.24855*, 2026.

Simon Idoko, Prajyot Jadhav, and Arun Kumar Singh. Flow-Opt: Scalable centralized multi-robot trajectory optimization with flow matching and differentiable optimization. *IEEE Transactions on Automation Science and Engineering*, 2026.

Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9902–9915. PMLR, 2022.

Daniel Kahneman, Patrick Egan, et al. *Thinking, fast and slow*, volume 1. Farrar, Straus and Giroux New York, 2011.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. 2022.

Vince Kurtz and Joel W. Burdick. Generative predictive control: Flow matching policies for dynamic and difficult-to-demonstrate tasks. In *2026 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2026.

Zeyang Li, Kaveh Alim, and Navid Azizan. Hardflow: Hard-constrained sampling for flow-matching models via trajectory optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023.

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.

Kazuki Mizuta and Karen Leung. Unified generation-refinement planning: Bridging guided flow matching and sampling-based MPC for social navigation. In *2026 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2026.

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

Seohong Park, Kevin Frans, Sergey Levine, and Aviral Kumar. Is value learning really the main bottleneck in offline RL? In *Advances in Neural Information Processing Systems*, volume 37, 2024.

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

Seohong Park, Qiyang Li, and Sergey Levine. Flow q-learning. In *International Conference on Machine Learning (ICML)*, 2025b.

Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, Jan Achterhold, Joerg Stueckler, Michal Rolinek, and Georg Martius. Sample-efficient cross-entropy method for real-time planning. *arXiv preprint arXiv:2008.06389*, 2020.

Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, and Georg Martius. Extracting strong policies for robotics tasks from zero-order trajectory optimizers. In *International Conference on Learning Representations*, 2021.

Thomas Power and Dmitry Berenson. Variational inference MPC using normalizing flows and out-of-distribution projection. In *Proceedings of Robotics: Science and Systems*, 2022.

Ralf Römer, Alexander von Rohr, and Angela P Schoellig. Diffusion predictive control with constraints. *arXiv preprint arXiv:2412.09342*, 2024.

Vlad 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*, volume 38, 2025.

Balakumar Sundaralingam, Siva Kumar Sastry Hari, Adam Fishman, Caelan Garrett, Karl Van Wyk, Valts Blukis, Alexander Millane, Helen Oleynikova, Ankur Handa, Fabio Ramos, Nathan Ratliff, and Dieter Fox. cuRobo: Parallelized collision-free minimum-jerk robot motion generation. *arXiv preprint arXiv:2310.17274*, 2023.

Weikang Wan, Ziyu Wang, Yufei Wang, Zackory Erickson, and David Held. DiffTORI: Differentiable trajectory optimization for deep reinforcement and imitation learning. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Tongzhou Wang, Antonio Torralba, Phillip Isola, and Amy Zhang. Optimal goal-reaching reinforcement learning via quasimetric learning. In *International Conference on Machine Learning*, pp. 36411–36430. PMLR, 2023.

Zhengtong Xu and Yu She. LeTO: Learning constrained visuomotor policy with differentiable trajectory optimization. *IEEE Transactions on Automation Science and Engineering*, 2024.

Zhengtong Xu, Zichen Miao, Qiang Qiu, Zhe Zhang, and Yu She. DiffOG: Differentiable policy trajectory optimization with generalizability. *IEEE Transactions on Robotics*, 2025.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, Ayaan Malik, Kyungmin Lee, William Liang, Nadun Ranawaka, Jiasheng Gu, Yinzhen Xu, Guanzhi Wang, Fengyuan Hu, Avnish Narayan, Johan Bjorck, Jing Wang, Gwanghyun Kim, Dantong Niu, Ruijie Zheng, Yuqi Xie, Jimmy Wu, Qi Wang, Ryan Julian, Danfei Xu, Yilun Du, Yevgen Chebotar, Scott Reed, Jan Kautz, Yuke Zhu, Linxi "Jim" Fan, and Joel Jang. World Action Models are Zero-shot Policies, 2026.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. MOPO: Model-based offline policy optimization. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Tianyuan Yuan, Zibin Dong, Yicheng Liu, and Hang Zhao. Fast-wam: Do world action models need test-time future imagination? *arXiv preprint arXiv:2603.16666*, 2026.

Weiheng Zhao, Haoyi Jiang, Xin Shi, Liu Liu, Fan Huang, Zhizhong Su, Wei Sui, and Xinggang Wang. Faster-WAM: Efficient Inference-Time Future Conditioning for Robust World Action Models, 2026.

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*, pp. 79115–79135. PMLR, 2025a.

Guangyao Zhou, Sivaramakrishnan Swaminathan, Rajkumar Vasudeva Raju, J. Swaroop Guntupalli, Wolfgang Lehrach, Joseph Ortiz, Antoine Dedieu, Miguel Lázaro-Gredilla, and Kevin Murphy. Diffusion model predictive control. *Transactions on Machine Learning Research*, 2025b.

Chuning Zhu, Raymond Yu, Siyuan Feng, Benjamin Burchfiel, Paarth Shah, and Abhishek Gupta. Unified world models: Coupling video and action diffusion for pretraining on large robotic datasets. In *Proceedings of Robotics: Science and Systems*, 2025.

# A World Action Models as a Generative Control Model Class

We make this *perspective manifest* by treating WAMs as a mathematical class of conditional trajectory models, independent of architecture or scale. Let $c = (z_t, g)$ denote the current-state and goal condition. An action-only generative policy models $q_\phi^A(A \mid c)$, a world model represents action-conditioned dynamics $p_\theta(Z \mid A, c)$, and a joint WAM models $q_\phi(A, Z \mid c)$. What differs is the modeled object. A joint WAM generates both an action trajectory and an intended future trajectory, with action marginal $q_{\phi,A}(A \mid c) = \int q_\phi(A, Z \mid c)\, dZ$.

**Model class and scale are separate choices.** A joint WAM does not require a particular architecture, generative parameterization, dataset scale, or pretraining regime. Autoregressive, diffusion, and flow-based models can all represent distributions over world-action trajectories. A joint WAM can likewise be task-specific or pretrained on heterogeneous data. Recent WAMs emphasize large-scale pretraining and generalization (Zhu et al., 2025; Cen et al., 2025; Ye et al., 2026). Here, we focus on the control problem that comes with generating a joint trajectory.

**Joint generation exposes an additional control reference.** An action-only generator gives downstream control only an action trajectory. A joint WAM also provides a future trajectory $\tilde{Z}$. For this extra output to matter for control, it should affect the resulting actions rather than act only as an auxiliary prediction. WATO uses $\tilde{A}$ and $\tilde{Z}$ as references in the trajectory projection $\Pi_{\theta,g}$, so both can shape the optimized action trajectory. Proposition 1 in Sec. 3.4 makes this explicit: the state reference acts through the world model rollout Jacobian.

**Scaling does not affect the control objective.** Learning a broad trajectory distribution and choosing good behavior for a downstream task are different problems. A large joint WAM may represent diverse behaviors and futures, but imitation alone does not ensure that a generated trajectory minimizes the task objective or matches the dynamics used for control. Pretraining determines what trajectory distribution the model represents. WATO shapes how that distribution is used for control.

**The formulation is independent of the generator parameterization.** Our implementation uses flow matching, but WATO is defined at the level of generated world-action trajectories. The operator $\Pi_{\theta,g}(\tilde{A}, \tilde{Z})$ depends on action and state references and differentiable action-conditioned dynamics, not on the flow architecture itself. The same idea may therefore extend to other differentiable joint generative models, including larger pretrained WAMs. We leave this direction to future work.

# B Why WATO Should Be Your Favorite Controller

Our results suggest WATO as a *default* controller for visual offline goal-conditioned control with local goals or waypoints. This raises a broader question: why does WATO also outperform value-based GCRL methods designed for long-horizon goal reaching? We look at local versus long-horizon control, representation, candidate selection, and the way each method learns a controller.

**Short horizons do not make policy learning trivial.** OGBench uses goal relabeling during training and predefined start-goal tasks for evaluation (Park et al., 2025a). Our evaluation instead samples goals 25 steps ahead in test trajectories and allows 50 execution steps. Each pair has an observed local connection, so little cross-trajectory stitching is needed. But the controller still needs accurate contact, pose control, and recovery from execution errors. Being able to propagate goal-reaching values over long horizons does not automatically give a strong local controller.

**The gap is not explained by the encoder alone.** The main comparisons use the same pretrained LeWM encoder to control for representation quality. End-to-end training with the official IMPALA-based setup raises GCIQL's average success from $50.9\%$ to $53.9\%$, with larger gains on `scene-play` and `puzzle-3x3`. So encoder training can help GCIQL, but it does not close the gap.

**The advantage persists without candidate selection.** WATO with $K = 1$ achieves $88.7\%$ average success and outperforms both GCIQL variants in all 7 environments. Increasing $K$ to 8 adds another 3.0 percentage points. With $K = 1$, world model ranking cannot change the executed proposal, so best-of-$K$ selection alone cannot explain the advantage. Table 4 summarizes both controls.

Table 4: **Encoder and candidate-count controls.** Success rates (%, mean ± std over 4 seeds). End-to-end GCIQL uses the official IMPALA-based setup.

| | GCIQL | | WATO: pretrained LeWM | |
|---|---|---|---|---|
| Environment | Pretrained LeWM | End-to-end training | $K = 1$ | $K = 8$ |
| `TwoRooms` | 80 ±5 | 81 ±8 | 90 ±3 | 93 ±0 |
| `Reacher` | 71 ±4 | 67 ±6 | 82 ±5 | 85 ±2 |
| `PushT` | 39 ±0 | 38 ±5 | 96 ±3 | 100 ±0 |
| `cube-single` | 52 ±1 | 56 ±7 | 97 ±2 | 99 ±1 |
| `cube-double` | 37 ±5 | 40 ±10 | 86 ±2 | 90 ±3 |
| `scene-play` | 40 ±4 | 51 ±5 | 90 ±1 | 94 ±1 |
| `puzzle-3x3` | 37 ±6 | 44 ±4 | 80 ±3 | 81 ±2 |
| Average | 50.9 | **53.9** | **88.7** | 91.7 |

**Trajectory supervision directly matches the local control problem.** Each training window gives a current state, an attained future goal, and the actions and states in between. Goal-conditioned imitation uses the same kind of hindsight supervision (Ghosh et al., 2021), and GCRL also uses relabeled goals and dataset actions. So the key difference is not whether the methods see goal-reaching trajectories, but how they use them. GCIQL learns goal-conditioned values and extracts a policy using those values and a behavioral constraint (Park et al., 2025a). WATO instead learns the joint trajectory and sends task gradients through its optimized solution. In this sense, it trades bootstrapped value learning for model-based trajectory supervision while keeping the imitation signal.

Simply matching the goal distribution throughout trajectories is not enough. GCBC and the flow policy achieve 66.4% and 69.3% average success, compared with 79.9% for Joint WAF and 91.7% for WATO. Neither hindsight imitation nor action-only generation reaches the performance of WATO.

**One trajectory sample can still represent coordinated behavior.** Reducing $K$ removes best-of-$K$ selection, but not the structure within a sampled trajectory. A joint WAF can generate an action sequence whose approach, contact, and subsequent motion follow one behavioral mode. This may help compared with extracting individual actions from a unimodal policy, becasue the continuous-action OGBench baselines use Gaussian policies evaluated at their means (Park et al., 2025a). Still, our comparisons do not cleanly separate temporal coordination, policy expressiveness, and the training objective. We therefore view trajectory generation as a useful inductive bias, not as something a Markov policy could never represent.

**Lower GCRL performance does not point to value error alone.** The bottleneck could lie in value estimation, policy extraction, or generalization to states encountered during execution. Prior work shows that policy extraction and test-time generalization can limit offline RL even when value learning works well (Park et al., 2024). Our controls do not isolate these factors one by one, but they show that neither encoder choice nor best-of-$K$ selection explains the gap.

WATO takes a different route: it gets continuous task supervision through a learned rollout and trajectory optimization. This still depends on world model accuracy and on transferring optimized behavior into the unprojected generator, as analyzed in Appendix C.4.

**WATO as the default local controller.** Taken together, our results point to a simple default: if the goal is to turn visual offline trajectories into a goal-conditioned controller, start with a joint WAM and optimize it for control. WATO outperforms the evaluated GCRL methods even when GCIQL trains its visual encoder end-to-end, and $K = 1$ WATO remains far ahead without best-of-$K$ selection. The gap is therefore not explained by representation quality or test-time candidate selection alone. WATO also keeps the resulting controller simple at deployment, requiring only trajectory generation and a small number of world model evaluations.

This does not make WAMs and GCRL mutually exclusive. Generative policies can be combined with value learning (Park et al., 2025b), and values may be useful when the problem requires trajectory stitching or reasoning far beyond locally supported trajectories. But that is a different job from learning a strong local controller. For the controller itself, our results give a clear answer: directly learn the trajectory, optimize it for the task, and amortize that optimization into the generator.

For visual offline goal-conditioned control, ***WATO should be your favorite controller***.

# C Theory and Derivations

This section develops the reference sensitivities and local geometry used in Sec. 3.4. We also study the local stability of the trajectory projection and connect its residuals to optimization amortization. All results are *local*, since the world model rollout and trajectory optimization problem are generally nonlinear and nonconvex.

## C.1 Reference Sensitivity

We start from the reduced WATO objective

$$\Psi(A; \tilde{A}, \tilde{r}) = F(A) + \frac{\lambda_A}{H}\|A - \tilde{A}\|_2^2 + \frac{\lambda_Z}{H}\|r(A) - \tilde{r}\|_2^2, \tag{11}$$

where $F(A) = J(\mathcal{R}_\theta(z_t, A), g)$, $r(A) = \mathrm{vec}\,\mathcal{N}_Z(\mathcal{R}_\theta(z_t, A))$, and $\tilde{r} = \mathrm{vec}\,\mathcal{N}_Z(\tilde{Z})$. Let $S(A) = \partial r(A)/\partial A$ and write the first-order condition as

$$G(A; \tilde{A}, \tilde{r}) := \nabla_A \Psi(A; \tilde{A}, \tilde{r}) = 0. \tag{12}$$

For an interior local solution $A^\star$, this becomes

$$\nabla F(A^\star) + \frac{2\lambda_A}{H}(A^\star - \tilde{A}) + \frac{2\lambda_Z}{H} S_\star^\top \big(r(A^\star) - \tilde{r}\big) = 0, \tag{13}$$

where $S_\star = S(A^\star)$. Let $\mathcal{H}_\star = \nabla^2_{AA}\Psi(A^\star; \tilde{A}, \tilde{r})$. If $\mathcal{H}_\star$ is nonsingular, the implicit function theorem gives

$$\frac{\partial A^\star}{\partial y} = -\mathcal{H}_\star^{-1}\frac{\partial G}{\partial y}, \qquad y \in \{\tilde{A}, \tilde{r}\}. \tag{14}$$

The action reference enters Equation (13) directly, with $\partial G/\partial \tilde{A} = -(2\lambda_A/H)I$. This gives

$$\frac{\partial A^\star}{\partial \tilde{A}} = \frac{2\lambda_A}{H}\mathcal{H}_\star^{-1}. \tag{15}$$

The state reference enters through the world model rollout. Since $\partial G/\partial \tilde{r} = -(2\lambda_Z/H)S_\star^\top$, we get

$$\frac{\partial A^\star}{\partial \tilde{r}} = \frac{2\lambda_Z}{H}\mathcal{H}_\star^{-1} S_\star^\top. \tag{16}$$

Together, Equation (15)-Equation (16) give Proposition 1. The action reference changes the optimized action trajectory through the local optimization Hessian, while the state reference additionally passes through $S_\star^\top$, which maps changes in the predicted future back to action space. For box-constrained solutions, the same derivation applies locally on the free action coordinates when the active set is fixed.

## C.2 Local Optimization Geometry

The Hessian of Equation (11) also shows how the two references shape the local optimization geometry. Define the state-reference residual $e(A) = r(A) - \tilde{r}$. Holding $\tilde{r}$ fixed, its squared norm has gradient

$$\nabla_A \|e(A)\|_2^2 = 2S(A)^\top e(A). \tag{17}$$

Writing $r_i(A)$ and $e_i(A)$ for the corresponding coordinates and differentiating again gives

$$\nabla_A^2 \|e(A)\|_2^2 = 2S(A)^\top S(A) + 2\sum_i e_i(A)\nabla_A^2 r_i(A). \tag{18}$$

The exact Hessian at $A^\star$ is

$$\mathcal{H}_\star = \nabla^2 F(A^\star) + \frac{2\lambda_A}{H}I + \frac{2\lambda_Z}{H}\left[S_\star^\top S_\star + \sum_i e_{\star,i}\nabla_A^2 r_i(A^\star)\right], \tag{19}$$

where $e_\star = r(A^\star) - \tilde{r}$.

Approximating the state-reference Hessian by its Gauss-Newton term gives

$$\mathcal{H}_\star \approx \nabla^2 F(A^\star) + \frac{2\lambda_A}{H} I + \frac{2\lambda_Z}{H} S_\star^\top S_\star. \tag{20}$$

The task Hessian $\nabla^2 F(A^\star)$ is kept exact. If $\|\nabla_A^2 r_i(A)\| \le M_r$ for every coordinate in a neighborhood of $A^\star$, the omitted term satisfies

$$\left\| \frac{2\lambda_Z}{H} \sum_i e_{\star,i} \nabla_A^2 r_i(A^\star) \right\| \le \frac{2\lambda_Z M_r}{H} \|e_\star\|_1. \tag{21}$$

So the approximation error vanishes with the state-reference residual.

The action reference contributes isotropic curvature $(2\lambda_A/H)I$. The state-reference Gauss-Newton term is positive semidefinite since, for any action perturbation $u$,

$$u^\top S_\star^\top S_\star u = \|S_\star u\|_2^2 \ge 0. \tag{22}$$

It weights action directions by how strongly they change the standardized predicted trajectory. The task Hessian and residual-weighted term may still be indefinite, so this does not make the full objective convex.

**Generated versus rollout-derived references.** One interesting distinction is that this geometry does not require a WAM-generated future. For a fixed proposal $\tilde{A}$, replacing $\tilde{r}$ with $r(\tilde{A})$ gives the state-reference penalty $\lambda_Z H^{-1} \|r(A) - r(\tilde{A})\|_2^2$. At the same action trajectory $A$, its Gauss-Newton contribution is still $(2\lambda_Z/H) S(A)^\top S(A)$. What changes is the reference itself, and hence the gradient, residual-weighted Hessian term, and optimized solution. Removing $d_Z$ therefore tests the state-reference penalty as a whole, but does not separate generated-future information from dynamics-aware regularization. This motivates the generated versus rollout-derived reference comparison in Appendix G.1.

### C.3 Projection Stability

The same sensitivity expressions also tell us how stable the projection is locally. Assume that $\Psi$ is locally $\mu$-strongly convex in $A$ around $A^\star$, so that

$$\mathcal{H}_\star \succeq \mu I, \qquad \mu > 0. \tag{23}$$

Then $\|\mathcal{H}_\star^{-1}\| \le 1/\mu$. Using Equation (15),

$$\left\| \frac{\partial A^\star}{\partial \tilde{A}} \right\| \le \frac{2\lambda_A}{H\mu}. \tag{24}$$

Similarly,

$$\left\| \frac{\partial A^\star}{\partial \tilde{r}} \right\| \le \frac{2\lambda_Z}{H\mu} \|S_\star\|. \tag{25}$$

For sufficiently small perturbations $(\Delta\tilde{A}, \Delta\tilde{r})$, these bounds give

$$\|\Delta A^\star\| \le \frac{2}{H\mu} \left( \lambda_A \|\Delta\tilde{A}\| + \lambda_Z \|S_\star\| \|\Delta\tilde{r}\| \right) + o(\|(\Delta\tilde{A}, \Delta\tilde{r})\|). \tag{26}$$

The action-reference sensitivity is controlled by the local curvature of the optimization problem. For the state reference, the rollout Jacobian also matters: it has more influence in directions where small action changes produce larger changes in the future.

### C.4 Optimization Amortization

There is one subtlety in WATO: training evaluates the projected task cost $F(A^\star)$, while deployment uses the unprojected proposal $\tilde{A}$. A good projected trajectory does not automatically imply a good

raw proposal. For example, consider $H = 1$, $r(A) = A$, $F(A) = A^2$, and $\lambda_A = \lambda_Z = 1$. The projection is

$$A^\star = \arg\min_A \left[A^2 + (A - \tilde{A})^2 + (A - \tilde{r})^2\right] = \frac{\tilde{A} + \tilde{r}}{3}. \tag{27}$$

Setting $\tilde{A} = t$ and $\tilde{r} = -t$ gives $F(A^\star) = 0$ but $F(\tilde{A}) = t^2$. This example only concerns the projected task term. WATO still keeps trajectory imitation through $\mathcal{L}_{\text{WAF}}$. The useful question is therefore when a small projection residual is enough to transfer the optimized behavior back to the raw proposal.

**Residual-based deployment bounds.** For brevity, write $d_A = d_A(A^\star, \tilde{A})$ and $d_Z = d_Z(Z^\star, \tilde{Z})$, where $Z^\star = \mathcal{R}_\theta(z_t, A^\star)$. By definition,

$$\|A^\star - \tilde{A}\|_2 = \sqrt{H d_A}, \qquad \|r(A^\star) - \tilde{r}\|_2 = \sqrt{H d_Z}. \tag{28}$$

Suppose $A^\star$ is an exact interior local solution. Assume that $\nabla F$ is $L_F$-Lipschitz and $r$ is $L_r$-Lipschitz on a neighborhood containing the segment between $A^\star$ and $\tilde{A}$. Below, matrix norms are spectral norms.

**Residual-based deployment transfer.**

The task-cost gap satisfies

$$\left|F(\tilde{A}) - F(A^\star)\right| \le \left(2\lambda_A + \frac{HL_F}{2}\right) d_A + 2\lambda_Z \|S_\star\| \sqrt{d_A d_Z}. \tag{29}$$

The inconsistency between the generated future and the rollout of the generated actions satisfies

$$\sqrt{d_Z\left(\mathcal{R}_\theta(z_t, \tilde{A}), \tilde{Z}\right)} \le L_r \sqrt{d_A} + \sqrt{d_Z}. \tag{30}$$

*Proof.* The first-order condition in Equation (13) gives

$$\|\nabla F(A^\star)\|_2 \le \frac{2}{\sqrt{H}} \left(\lambda_A \sqrt{d_A} + \lambda_Z \|S_\star\| \sqrt{d_Z}\right). \tag{31}$$

Lipschitz continuity of $\nabla F$ gives

$$\left|F(\tilde{A}) - F(A^\star)\right| \le \|\nabla F(A^\star)\|_2 \sqrt{H d_A} + \frac{HL_F}{2} d_A. \tag{32}$$

Substituting Equation (31) proves Equation (29). For the consistency bound, the triangle inequality gives

$$\|r(\tilde{A}) - \tilde{r}\|_2 \le L_r \|\tilde{A} - A^\star\|_2 + \|r(A^\star) - \tilde{r}\|_2. \tag{33}$$

Dividing by $\sqrt{H}$ gives Equation (30).

These bounds connect the reference distances in Figure 4 to two deployment quantities: the cost of removing optimization and the world model inconsistency of the generated trajectory. The second statement is with respect to the pretrained world model, not the true environment.

**Stationarity and fixed points.** Under the same assumptions,

$$\|\nabla F(\tilde{A})\|_2 \le \frac{2}{\sqrt{H}} \left[\left(\lambda_A + \frac{HL_F}{2}\right) \sqrt{d_A} + \lambda_Z \|S_\star\| \sqrt{d_Z}\right]. \tag{34}$$

This follows from $\|\nabla F(\tilde{A})\|_2 \le \|\nabla F(A^\star)\|_2 + L_F \|\tilde{A} - A^\star\|_2$ and Equation (31). So small residuals also bound the task gradient of the raw proposal.

If $A^\star = \tilde{A}$ and $\tilde{r} = r(\tilde{A})$, both distances vanish and

$$\nabla_A J(\mathcal{R}_\theta(z_t, \tilde{A}), g) = 0. \tag{35}$$

An interior dynamics-consistent fixed point is task-stationary and not necessarily globally optimal.

**Active action bounds.** When action bounds are active, the full task gradient need not vanish. Let $N_{\mathcal{U}^H}(A)$ denote the normal cone of the action box. Constrained optimality gives $0 \in \nabla_A \Psi(A^\star; \tilde{A}, \tilde{r}) + N_{\mathcal{U}^H}(A^\star)$, which implies

$$\operatorname{dist}\left(0,\, \nabla F(A^\star) + N_{\mathcal{U}^H}(A^\star)\right) \le \frac{2}{\sqrt{H}} \left(\lambda_A \sqrt{d_A} + \lambda_Z \|S_\star\| \sqrt{d_Z}\right). \tag{36}$$

At a dynamics-consistent fixed point, this becomes constrained task stationarity.

For feasible $\tilde{A}$ and $A^\star$ in a neighborhood where $F$ is $C_F$-Lipschitz, we can still bound the task-cost gap without assuming interior stationarity:

$$\left|F(\tilde{A}) - F(A^\star)\right| \le C_F \sqrt{H d_A}. \tag{37}$$

The dynamics-consistency bound in Equation (30) also continues to hold at active bounds, since its proof does not use stationarity.

**Inexact optimization.** The same picture extends to approximate solves. For an approximate interior solution, let $\epsilon_{\text{opt}} = \|\nabla_A \Psi(A^\star; \tilde{A}, \tilde{r})\|_2$, with all distances and Jacobians evaluated at the returned trajectory. The right-hand sides of Equations (31) and (34) then gain $\epsilon_{\text{opt}}$, while Equation (29) gains $\epsilon_{\text{opt}} \sqrt{H d_A}$. For a bound-active solution, Equation (36) similarly gains the corresponding KKT residual $\operatorname{dist}(0, \nabla_A \Psi(A^\star; \tilde{A}, \tilde{r}) + N_{\mathcal{U}^H}(A^\star))$. The consistency and Lipschitz cost bounds do not require a converged projection.

**Candidate selection under model error.** Finally, consider the deployed best-of-$K$ rule. For $K$ generated candidates and their projected trajectories, let $k^\star = \arg\min_k F(\tilde{A}^{(k)})$. Let $F_{\text{env}}(A)$ denote the terminal task cost obtained by executing $A$ in the environment, using the same encoder and goal. Suppose $|F_{\text{env}}(A) - F(A)| \le \epsilon_{\text{model}}$ for every generated and projected action trajectory in this candidate set. Let $\eta_k \ge 0$ satisfy $F(\tilde{A}^{(k)}) \le F(A^{\star(k)}) + \eta_k$, as provided by the preceding task-cost bounds. Then

$$\begin{aligned} F_{\text{env}}(\tilde{A}^{(k^\star)}) &\le \min_k \left[F_{\text{env}}(A^{\star(k)}) + \eta_k\right] + 2\epsilon_{\text{model}} \\ &\le \min_k F_{\text{env}}(A^{\star(k)}) + \max_k \eta_k + 2\epsilon_{\text{model}}. \end{aligned} \tag{38}$$

This follows from $F(\tilde{A}^{(k^\star)}) \le F(\tilde{A}^{(k)}) \le F(A^{\star(k)}) + \eta_k$ and the model-error bound on the selected and projected trajectories. The bound separates three effects: projected candidate quality, the cost of removing optimization, and world model error.

These results describe what small projection residuals buy us: WATO does not explicitly minimize them. They are finite-horizon statements and do not imply convergence of WATO training, closed-loop stability, or environment-defined success.

# D Trajectory Optimization and Differentiation

WATO optimizes only the action trajectory and obtains future states from the world model rollout. The world model parameters remain fixed, while derivatives w.r.t. the action trajectory are retained.

## D.1 Reduced-Space Optimization

We implement Equation (5) in standardized action coordinates. The action trajectory $A$ is the only optimization variable, with $Z(A) = \mathcal{R}_\theta(z_t, A)$. We solve

$$A^\star = \arg\min_{A \in \mathcal{U}^H_{\text{std}}} J(Z(A), g) + \lambda_A d_A(A, \tilde{A}) + \lambda_Z d_Z(Z(A), \tilde{Z}), \qquad Z^\star = Z(A^\star), \tag{39}$$

where $\mathcal{U}_{\text{std}}$ denotes the action bounds in standardized coordinates. The optimized future $Z^\star$ is always produced by the world model.

We use

$$J(Z(A), g) = \|Z_H(A) - g\|_2^2, \tag{40}$$

and

$$d_A(A, \tilde{A}) = \frac{1}{H}\sum_{h=0}^{H-1} \|A_h - \tilde{A}_h\|_2^2, \qquad d_Z(Z, \tilde{Z}) = \frac{1}{H}\sum_{h=1}^{H} \|\mathcal{N}_Z(z_h) - \mathcal{N}_Z(\tilde{z}_h)\|_2^2. \tag{41}$$

Primitive actions are standardized using offline-data statistics before forming action blocks. Environment action bounds are transformed with the same statistics and imposed as box constraints. The action reference $\tilde{A}$ uses the same coordinates.

The state normalization is $\mathcal{N}_Z(z) = (z - \mu_Z)/\sigma_Z$, where $\mu_Z$ and $\sigma_Z$ are computed only from frozen-LeWM latents in the offline training split. This normalization is used only in $d_Z$; the task cost is evaluated in the native pretrained latent space. We use $\lambda_A = 10$ and $\lambda_Z = 0.1$ in all experiments.

### D.2 Numerical Solver

We initialize the primal solve from the clipped generated action reference $\tilde{A}$. We run L-BFGS-B for at most 200 iterations under the standardized action bounds, followed by up to 25 projected-gradient polishing iterations. Each polishing iteration uses at most 12 backtracking trials.

Because the world model rollout is nonlinear, the optimization problem is generally nonconvex. We require local stationarity using

$$\left\| \nabla_{\mathrm{proj}} \Psi(A^\star; \tilde{A}, \tilde{r}) \right\|_\infty \le 0.02, \tag{42}$$

where the projected gradient is taken with respect to the standardized action bounds.

During training, a task sample whose solution fails Equation (42) is rejected and retried with another reserved window and proposal. We allow at most four attempts per update. If none satisfies the stationarity criterion, the task update is aborted.

The trajectory optimizer is used only during WATO training. Deployment uses generated proposals and world model evaluation without L-BFGS-B refinement.

### D.3 Implicit Differentiation

We differentiate the local solution through its optimality conditions rather than unrolling the primal solver. Let $y = (\tilde{A}, \tilde{r})$, where $\tilde{r} = \mathrm{vec}\,\mathcal{N}_Z(\tilde{Z})$, and let $A^\star(y)$ denote a local solution. For an interior regular solution, $\nabla_A \Psi(A^\star; y) = 0$. Define $\mathsf{H} = \nabla^2_{AA}\Psi(A^\star; y)$ and $\mathsf{B} = \partial_y \nabla_A \Psi(A^\star; y)$. Differentiating the stationarity condition gives

$$\mathsf{H}\frac{\partial A^\star}{\partial y} + \mathsf{B} = 0. \tag{43}$$

We only need the vector-Jacobian product with the task loss. Let

$$g_{\mathrm{task}} = \nabla_A J(\mathcal{R}_\theta(z_t, A), g)|_{A=A^\star}\,. \tag{44}$$

Rather than forming $\mathsf{H}^{-1}$, we solve

$$\mathsf{H}^\top v = g_{\mathrm{task}}, \qquad \nabla_y J(Z^\star, g) = -\mathsf{B}^\top v. \tag{45}$$

The resulting reference gradients are propagated through the joint generator to update $\phi$.

Let

$$J_R = \left.\frac{\partial \mathcal{R}_\theta(z_t, A)}{\partial A}\right|_{A=A^\star}, \tag{46}$$

and let $D_Z$ denote the block-diagonal linear map induced by $\mathcal{N}_Z$. The standardized rollout Jacobian is $S_\star = D_Z J_R$. The joint WAF directly outputs the normalized state reference $\tilde{r}$, with task gradient

$$\nabla_{\tilde{A}} J(Z^\star, g) = \frac{2\lambda_A}{H} v, \qquad \nabla_{\tilde{r}} J(Z^\star, g) = \frac{2\lambda_Z}{H} D_Z J_R v. \tag{47}$$

In native latent coordinates,

$$\nabla_{\tilde{Z}} J(Z^\star, g) = D_Z^\top \nabla_{\tilde{r}} J(Z^\star, g) = \frac{2\lambda_Z}{H} D_Z^\top D_Z J_R v. \tag{48}$$

Thus, the generated future receives a task gradient through its effect on the optimized actions even though it is not an independent optimization variable.

For a box-constrained solution with a locally fixed active set, we differentiate only through the free action coordinates. Let $E_{\text{free}}$ embed the free coordinates into the full action vector. The theoretical adjoint system is

$$\left(E_{\text{free}}^\top \mathsf{H} E_{\text{free}}\right)^\top v_{\text{free}} = E_{\text{free}}^\top g_{\text{task}}, \qquad v = E_{\text{free}} v_{\text{free}}. \tag{49}$$

This local derivative does not cover points where the active set changes.

In implementation, we solve the damped system

$$\left(E_{\text{free}}^\top \mathsf{H} E_{\text{free}} + 10^{-3} I\right)^\top v_{\text{free}} = E_{\text{free}}^\top g_{\text{task}}. \tag{50}$$

We compute Hessianvector products through the frozen world model and use conjugate gradients for at most 16 iterations with tolerance $10^{-4}$. If conjugate gradients fails, we fall back to a dense solve and, when needed, a pseudoinverse. Damping and finite solver tolerance make the backward pass an approximation to the local sensitivity.

The world model parameters $\theta$ are never updated by this gradient. Derivatives through $\mathcal{R}_\theta$ with respect to the action trajectory are retained.

# E World Model and WAF Implementation

We specify the pretrained world model, joint world-action generator, and WATO training procedure. The world model and joint WAF are trained before WATO. WATO updates only the generator parameters $\phi$ and the world model parameters $\theta$ remain fixed.

## E.1 World Model

We use latent world models following LeWM (Maes et al., 2026). Each model consists of a visual encoder $E_\theta$ and an action-conditioned predictor $F_\theta$ trained jointly end-to-end. The encoder maps an observation to a latent state, while the predictor takes an action block $A_h$ and a finite latent context $\mathcal{C}_h$ and predicts the next latent state. Autoregressive prediction gives the rollout $\mathcal{R}_\theta$ defined in Equation (1).

The released LeWM architecture uses $224\times224$ visual inputs, a ViT-Tiny encoder with patch size 14, and a six-layer action-conditioned Transformer predictor. For released checkpoints, five primitive actions are grouped into each action block. The predictor uses a finite latent context whose length depends on the environment.

LeWM is trained with next-latent prediction and SIGReg. The prediction loss is

$$\mathcal{L}_{\text{pred}} = \mathbb{E}_{\mathcal{D}} \left[\text{MSE}\left(\hat{z}_{h+1}, z_{h+1}\right)\right], \tag{51}$$

and the full objective is $\mathcal{L}_{\text{WM}} = \mathcal{L}_{\text{pred}} + \lambda_{\text{sig}} \mathcal{L}_{\text{SIG}}$. Following LeWM, we use 1024 random projections for SIGReg and $\lambda_{\text{sig}} = 0.1$.

We use released LeWM checkpoints when available. For environments without one, we train the same LeWM architecture and objective from scratch. We retain the default temporal abstraction of five primitive actions per latent transition and a maximum context of three latent states. For models trained in this work, the training entry reports the optimizer step of the selected validation checkpoint. Table 5 summarizes the configurations.

For world models trained in this work, we use AdamW with a learning rate of $5 \times 10^{-5}$, weight decay of $10^{-3}$, and batch size 128. We train for up to 12 epochs (approximately 92k optimizer updates) with a 1% linear warmup followed by cosine decay to zero, and clip the global gradient norm to 1.0. We select the checkpoint with the lowest teacher-forced validation loss, defined as prediction MSE plus 0.09 times the SIGReg loss.

All world models are trained before WATO. During WATO, gradients through $E_\theta$, $F_\theta$, and $\mathcal{R}_\theta$ with respect to their inputs are retained, but $\theta$ is never updated.

Table 5: **World model configurations.** $B$ is the number of primitive actions per latent transition and $l_c$ is the maximum latent context length.

| Environment | Source | $B$ | $l_c$ | Selected checkpoint |
|---|---|---|---|---|
| `TwoRooms` | Released LeWM | 5 | 1 | Released |
| `Reacher` | Released LeWM | 5 | 3 | Released |
| `PushT` | Released LeWM | 5 | 3 | Released |
| `cube-single` | Released LeWM | 5 | 3 | Released |
| `cube-double` | Trained in this work | 5 | 3 | 92,052 updates |
| `scene-play` | Trained in this work | 5 | 3 | 76,252 updates |
| `puzzle-3x3` | Trained in this work | 5 | 3 | 68,724 updates |

### E.2 Joint World Action Flow

The joint WAF models $X = (A, Z)$, where $A = (A_0, \ldots, A_{H-1})$ and $Z = (z_1, \ldots, z_H)$, conditioned on the current and goal latents $c = (z_t, g)$. A single conditional flow generates the action and future-state trajectories jointly rather than autoregressively over the horizon.

The vector field $v_\phi$ is parameterized by a Transformer with model width 256, 4 Transformer layers, 8 attention heads, feed-forward dimension 1024, no dropout, and a 64-dimensional flow-time embedding. Action and latent variables are projected to the common model width, processed jointly, and mapped back to their respective output spaces. The model has approximately 3.5M parameters.

Primitive actions are standardized before action blocks are formed. Future state targets are obtained by encoding dataset observations with the pretrained encoder. For each minibatch, we sample a horizon-$H$ trajectory window, form $X = (A, Z)$, and train with the joint WAF objective in Equation (3). We use $\lambda_{\text{WAF}} = 1$, batch size 512, and 50,000 pretraining updates.

At generation time, Gaussian noise with the same structure as $X$ is integrated from flow time 0 to 1. We use Euler integration with NFE $= 8$. All action and state variables are updated jointly at each flow step. Unless otherwise stated, deployment draws $K = 8$ independent candidates.

### E.3 WATO Training

WATO fine-tunes the pretrained joint WAF using the objective in Equation (9). The flow matching term preserves trajectory imitation, while the task term differentiates through the trajectory-optimization layer. Only the WAF parameters $\phi$ are updated. The world model remains fixed.

We use AdamW with a learning rate of $5 \times 10^{-5}$, weight decay $10^{-4}$, and a flow matching batch size of 64 for 10,000 WATO updates. The task term uses one projected trajectory per update with $\lambda_{\text{task}} = 5 \times 10^{-3}$. For each update, trajectories for the task term are generated with Euler integration using NFE $= 2$ and passed through the trajectory projection in Equation (4). We draw up to four proposals to obtain a stationarity-qualified trajectory. Only solutions satisfying the criterion in Equation (42) contribute to the task gradient.

We clip the global generator gradient norm to 1. The learning rate uses a linear warmup over the first 100 updates followed by cosine decay to zero. We select the final checkpoint using the minimum fixed-validation objective, $\mathcal{L}_{\text{WAF}} + \lambda_{\text{task}}\mathcal{L}_{\text{task}}$.

### E.4 Training Cost and Solver Statistics

WATO adds training-time computation because each task update solves a trajectory optimization problem and differentiates through its solution. Table 7 reports statistics from the full 10,000-update `PushT` run. All wall-clock times are measured on a single NVIDIA RTX Pro 6000 Blackwell GPU.

WATO fine-tuning takes 5.82 hours, compared with 1.76 hours for joint WAF pretraining. The primal solver uses 90 L-BFGS-B iterations on average, but the stationarity filter rarely rejects a solve: only 5 of 10,005 attempts are rejected, and no training update is aborted. The additional computation is confined to training; deployment uses generation, world model evaluation, and best-of-$K$ selection without iterative trajectory refinement.

Table 6: **Joint WAF training and WATO training configuration.** NFE $= 2$ is used to generate trajectories for the task loss during training, whereas all controllers use NFE $= 8$ at deployment.

| Setting | Joint WAF training | WATO training |
|---|---|---|
| Optimizer | AdamW | AdamW |
| Weight decay | $10^{-4}$ | $10^{-4}$ |
| Learning rate | $3 \times 10^{-4}$ | $5 \times 10^{-5}$ |
| Batch size | 512 | 64 (FM), 1 (task) |
| Updates | 50,000 | 10,000 |
| Warmup | 500 updates | 100 updates |
| Learning-rate schedule | Cosine decay | Cosine decay |
| Task-loss flow integration | – | Euler, NFE $= 2$ |
| Deployment flow integration | Euler, NFE $= 8$ | Euler, NFE $= 8$ |
| $\lambda_{\mathrm{WAF}}$ | 1 | 1 |
| $\lambda_{\mathrm{task}}$ | – | $5 \times 10^{-3}$ |
| Gradient-norm clipping | 1 | 1 |
| Checkpoint criterion | Minimum validation FM loss | Minimum validation weighted objective |

Table 7: **Training cost and solver statistics on `PushT`.** Statistics are from the full 10,000-update WATO run on a single GPU.

**Training cost**

| Metric | Value |
|---|---|
| Joint WAF pretraining | 1.76 h |
| WATO training | 5.82 h |
| Total pipeline | 7.58 h |
| WATO / Joint WAF | 3.31× |
| Pipeline / Joint WAF-only | 4.31× |
| FM-only continuation | 0.65 h |
| WATO / FM-only | 9.0× |
| Direct rollout | 0.9–1.5 h |
| WATO / direct rollout | 3.9–6.5× |

**Solver workload**

| Metric | Value |
|---|---|
| Mean L-BFGS-B iter. | 90 |
| Median L-BFGS-B iter. | 84 |
| Interquartile range | 53–116 |
| Maximum L-BFGS-B iter. | 200 |
| Maximum polish iter. | 25 |
| Total L-BFGS-B iter. | $8.8 \times 10^5$ |

**Solver reliability**

| Metric | Value |
|---|---|
| Optimizer updates | 10,000 |
| Task-solve attempts | 10,005 |
| Rejected samples | 5 |
| First-attempt success | 99.95% |
| Success within $\leq 4$ | 100% |
| Aborted updates | 0 |
| Aborted-update rate | 0% |

# F Experimental Protocol

This section specifies goal construction, baseline training, environment-specific configurations, and compute measurement. Unless stated otherwise, all methods use the same offline data, evaluation goals, execution budget, and pretrained visual representation. Figure 5 shows representative observations from the seven evaluation environments.

## F.1 Goal Construction and Evaluation

We evaluate local visual goal reaching from evaluation trajectories. For an initial observation $o_t$, the goal is the observation $o_{t+L}$ from the same trajectory, with $L = 25$ environment steps. All evaluation episodes are *nontrivial*: the initial state never satisfies the goal criterion. Only pairs with at least $L$ remaining steps are used. This defines the same goal distribution across all methods.

Each episode has a 50-step execution budget. A controller repeatedly observes the current image, conditions on the fixed goal image, executes its selected action segment, and replans until success or the budget is exhausted. All methods use the same environment-specific success criterion.

Success rate is the primary metric. We also report the final latent goal error $\|E_\theta(o_T) - g\|_2$, where $o_T$ is the final observation. All results use 4 seeds and identical start-goal pairs across methods.

## F.2 Baseline Training

All baselines are trained on the same offline dataset and goal distribution as WATO. Evaluation uses identical start-goal pairs and the same execution budget. All methods use the same pretrained LeWM encoder to separate control performance from representation quality. Each baseline otherwise keeps its original objective, goal-relabeling procedure, and training scheme.

**Goal-conditioned learning.** GCBC (Ding et al., 2019) learns a goal-conditioned policy by behavioral cloning. GCIVL (Kostrikov et al., 2022) and GCIQL (Park et al., 2025a) learn goal-conditioned

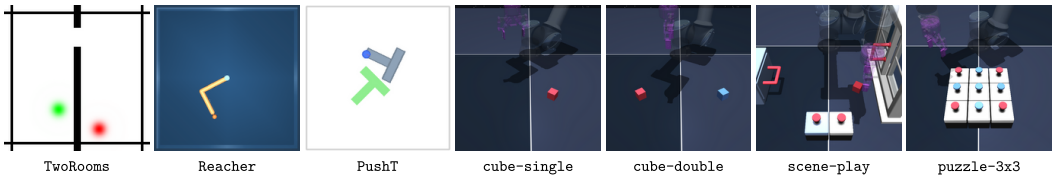


Figure 5: Environments for WATO experiments

value functions and policies from the same offline data. QRL (Wang et al., 2023) learns goal-conditioned control through its quasimetric objective. All four methods are trained from scratch for this goal distribution rather than evaluated with published final-goal checkpoints.

**Generative control.** The Flow Policy generates only the action trajectory $A$ and serves as the action-only generative baseline. Joint WAF uses the joint generator $q_\phi(A, Z \mid z_t, g)$ trained only with the flow matching objective in Equation (3). It isolates the effect of WATO training from joint trajectory imitation. For GPC (Kurtz & Burdick, 2026), we adapt the method to the same offline setting and use the same pretrained LeWM dynamics model as WATO. All generative methods use the same goals and execution budget.

**Model-based control.** CEM (Maes et al., 2026) optimizes action trajectories online using the same pretrained world model and terminal task cost $J$ as WATO. It uses a population of 300 trajectories, retains 30 elites (an elite fraction of 0.1), and performs 30 optimization rounds for PushT and Reacher and 10 rounds for the OGBench manipulation environments. The initial standard deviation is 1, no warm start is used, and the final Gaussian mean is returned as the action plan. Gradient-based MPC uses 8 independently initialized action trajectories, 40 gradient steps, and a learning rate of 0.05. Neither method updates the world model during evaluation.

We use the released LeWM CEM hyperparameters without tuning. Hyperparameters for learned baselines are selected on validation tasks and then fixed for test evaluation.

### F.3 Hyperparameters

We use the same WATO configuration across all environments. The planning horizon is $H = 5$ latent transitions with $B = 5$ primitive actions per block. At deployment, WATO generates $K = 8$ candidates using Euler integration with NFE $= 8$. We set $\lambda_A = 10$, $\lambda_Z = 0.1$, and $\lambda_{\text{task}} = 5 \times 10^{-3}$. The trajectory optimizer uses at most 200 L-BFGS-B iterations during WATO training and is not used at deployment. Ablations change only the component specified by each experiment.

### F.4 Success Criteria

We evaluate success using environment-specific semantic state criteria rather than the latent-space terminal cost used for planning. The latent cost ranks candidate action trajectories, while success is computed from simulator state. Table 8 summarizes the criterion for each environment.

An episode is successful if its criterion is satisfied at least once during execution. WATO and the other WAF-based controllers terminate immediately upon success. The released Reacher CEM parity evaluator is an exception: it continues until the full episode budget is exhausted but keeps the first observed success for evaluation.

## G Full Results

This section reports the per-environment results underlying the aggregate comparisons in the main paper. We provide the full WATO ablations, the absolute success rates used to measure optimization amortization, and final goal errors.

Table 8: **Environment-specific semantic success criteria.** These criteria use privileged simulator state and are distinct from the pretrained world model's latent-space planning cost.

| Environment | Success criterion |
|---|---|
| `TwoRooms` | The Euclidean distance between the agent position and target position is less than 16 coordinate units. |
| `Reacher` | The maximum absolute joint-position error over the two target joints is less than $0.05$ radians. |
| `PushT` | The positional error over the first four task-state coordinates is less than 20, and the wrapped block-orientation error is less than $\pi/9$ radians. |
| `cube-single` | The Euclidean distance between the cube position and target position is at most $0.04$ meters. |
| `cube-double` | Both cubes are within $0.04$ meters of their respective targets, i.e., $\max_{i\in\{1,2\}} \left\|p_i - p_i^g\right\|_2 \leq 0.04$. |
| `scene-play` | The cube-position error is at most $0.04$ meters, the absolute drawer and window position errors are each at most $0.04$, and every discrete button state matches its target state. All conditions must hold simultaneously. |
| `puzzle-3x3` | All nine binary button states match their target states, i.e., $\sum_{i=1}^{9} \mathbf{1}\left[s_i \neq s_i^g\right] = 0$. |

### G.1 Full WATO Ablations

Table 9 reports the complete training-objective and reference-design ablations. All variants use the same joint WAF architecture, offline data, pretrained world model, and evaluation protocol.

The training-objective ablation isolates how trajectory optimization enters learning. Joint WAF uses flow matching alone. Direct rollout adds task supervision through the world model, direct rollout with consistency additionally aligns generated states with the rollout, and distillation uses optimized trajectories as detached targets. WATO achieves the highest average success rate, improving from $79.9\%$ for Joint WAF to $91.7\%$.

The reference-design ablation isolates the two components of the world-action trajectory. Using only the action reference achieves $85.7\%$, while using only the state reference achieves $85.0\%$. Their combination reaches $91.7\%$, showing that neither reference alone recovers full WATO performance.

Removing $d_Z$ eliminates both its dynamics-aware regularization and the generated future reference. To separate these effects, the rollout-derived variant replaces $\tilde{Z}$ with $\mathcal{R}_\theta(z_t, \tilde{A})$ while retaining both reference penalties. The backward pass includes the dependence of this reference on $\tilde{A}$. This retains dynamics-aware regularization without using the generated future as a control reference, as discussed in Appendix C.2.

The average success rate of $87.9\%$ place this variant $2.2$ pp above the action-reference-only variant and $3.8$ points below WATO. This indicate that dynamics-aware regularization explains part, but not all, of the state-reference benefit. The remaining gap support using generated futures to guide optimization beyond the consequences already predicted for the proposed actions. Conversely, performance comparable to WATO suggest that rollout-derived references capture most of this benefit.

Table 9: **Full WATO ablation results.** Results are success rates (%, mean ± std over 4 seeds).

| | Training objective | | | | Reference design | | | |
|---|---|---|---|---|---|---|---|---|
| Environment | Joint WAF | Direct | Direct + Consistency | Distillation | Action only | State only | Rollout-derived reference | **WATO** |
| `TwoRooms` | 83 ±1 | 85 ±1 | 87 ±2 | 90 ±0 | 86 ±2 | 88 ±3 | 88 ±3 | **93** ±0 |
| `Reacher` | 67 ±3 | 77 ±5 | 79 ±4 | 81 ±4 | 79 ±3 | 76 ±4 | 81 ±2 | **85** ±2 |
| `PushT` | 84 ±4 | 94 ±3 | 95 ±3 | 93 ±2 | 95 ±3 | 91 ±4 | 96 ±4 | **100** ±0 |
| `cube-single` | 88 ±1 | 96 ±2 | 97 ±2 | 96 ±1 | 96 ±2 | 94 ±2 | 97 ±1 | **99** ±1 |
| `cube-double` | 89 ±2 | 87 ±3 | 88 ±3 | 86 ±2 | 84 ±3 | 85 ±3 | 87 ±5 | **90** ±3 |
| `scene-play` | 78 ±3 | 90 ±4 | 91 ±3 | 91 ±3 | 87 ±4 | 89 ±4 | 90 ±3 | **94** ±1 |
| `puzzle-3x3` | 70 ±3 | 71 ±4 | 75 ±2 | 79 ±3 | 73 ±4 | 72 ±4 | 76 ±2 | **81** ±2 |
| Average | 79.9 | 85.7 | 87.4 | 88.0 | 85.7 | 85.0 | 87.9 | **91.7** |

### G.2 Training Horizon and Test Goal Distance

The main experiments use a training horizon $L_{\text{train}} = 25$ and test goals $D = 25$ primitive steps ahead. We additionally vary $L_{\text{train}} \in \{10, 25, 50, 100\}$ and $D \in \{10, 25, 50, 100\}$ on `PushT`, `cube-single`, and `scene-play`. For each goal distance $D$, the execution budget is $2D$ steps. All other settings are unchanged.

These horizons correspond to physical durations. In `PushT`, one environment step is $0.1\,\mathrm{s}$, so $L_{\mathrm{train}} = \{10, 25, 50, 100\}$ corresponds to $\{1, 2.5, 5, 10\}\,\mathrm{s}$. In `cube-single` and `scene-play`, one step is $0.05\,\mathrm{s}$, giving $\{0.5, 1.25, 2.5, 5\}\,\mathrm{s}$. The execution budget is twice the test-goal duration.

Table 10 shows that WATO is not limited to short local control. The best training horizon generally grows with the test goal distance, although exact matching is not always optimal. For example, $L_{\mathrm{train}} = 50$ achieves $82\%$ success on `PushT` at $D = 50$, corresponding to a $5\,\mathrm{s}$ trajectory. An average `PushT` episode spans roughly $125$ steps, or $12.5\,\mathrm{s}$, so a $50$-step WATO horizon already covers about $40\%$ of a full episode. A complete task therefore spans only about two to three such horizons, leaving less work for a separate high-level planner than a purely local-controller view would suggest.

Performance still decreases as the temporal scale grows. At $D = 100$, the best success rates are $48\%$, $58\%$, and $52\%$ on `PushT`, `cube-single`, and `scene-play`, respectively. Longer tasks with more objects, stages, or distant goals may therefore benefit from hierarchy or high-level planning. Such a planner, however, can operate over multi-second WATO controllers rather than primitive local actions. These results further support the perspective in Appendix B: WATO is a strong default controller over a broader temporal range than the main $25$-step setting alone suggests.

Table 10: **Training horizon and test goal distance.** WATO success rates (%) for different training horizons $L_{\mathrm{train}}$ and test goal distances $D$, measured in primitive environment steps. Bold indicates the best training horizon for each environment and test goal distance.

| Environment | $L_{\mathrm{train}} \backslash D$ | 10 | 25 | 50 | 100 |
|---|---|---|---|---|---|
| `PushT` | 10 | **100** | 80 | 52 | 16 |
| | 25 | 74 | **100** | 70 | 30 |
| | 50 | 30 | 66 | **82** | **48** |
| | 100 | 8 | 24 | 50 | 40 |
| `cube-single` | 10 | **100** | 68 | 32 | 10 |
| | 25 | 70 | **99** | 40 | 24 |
| | 50 | 24 | 58 | **92** | 44 |
| | 100 | 5 | 18 | 42 | **58** |
| `scene-play` | 10 | **97** | 64 | 28 | 6 |
| | 25 | 68 | **99** | 50 | 14 |
| | 50 | 20 | 55 | **76** | 32 |
| | 100 | 4 | 14 | 35 | **52** |

### G.3 Full Optimization-Amortization Results

The main paper reports the additional success obtained by applying trajectory optimization before and after WATO training. Table 11 reports the corresponding absolute success rates.

Before WATO, trajectory optimization raises average success from $79.9\%$ to $90.3\%$. After WATO, the same optimizer raises success only from $91.7\%$ to $93.0\%$. Thus, most of the performance previously obtained through online optimization is already present in the WATO generator.

Table 11: **Absolute performance before and after trajectory optimization.** The table expands the pp gains reported in Table 2. Results are success rates (%) from 100 tests.

| | Before WATO | | After WATO | |
|---|---|---|---|---|
| Environment | Joint WAF | + Traj. Opt. | WATO | + Traj. Opt. |
| `TwoRooms` | 83 | 91 | 93 | 94 |
| `Reacher` | 67 | 82 | 85 | 88 |
| `PushT` | 84 | 98 | 100 | 100 |
| `cube-single` | 88 | 97 | 99 | 100 |
| `cube-double` | 89 | 92 | 90 | 91 |
| `scene-play` | 78 | 92 | 94 | 95 |
| `puzzle-3x3` | 70 | 80 | 81 | 83 |
| Average | 79.9 | 90.3 | 91.7 | 93.0 |

The absolute results also separate optimization quality from amortization. Online trajectory optimization is already strong when applied to the imitation-trained generator, reaching $90.3\%$ average success. WATO reaches $91.7\%$ without this online optimization, while an additional optimization step provides only a small remaining gain. This supports the interpretation that WATO transfers the useful effect of trajectory optimization into generation rather than relying on a weak optimizer.

### G.4 Final Goal Error

Success is a thresholded measure and does not distinguish failures that terminate near the goal from those that remain far away. We therefore also report the final latent goal error $\|E_\theta(o_T) - g\|_2$, where $o_T$ is the final observation after the execution budget. Lower values indicate trajectories that terminate closer to the target.

Table 12 reports the metric for the same methods and start-goal pairs as Table 1. We report results separately for each environment because latent scales need not be comparable across independently trained world models.

Table 12: **Final latent goal error.** Lower is better. Results use the same start-goal pairs and execution budget as the success-rate evaluation.

| Environment | GCBC | GCIVL | GCIQL | QRL | Flow Policy |
|---|---|---|---|---|---|
| TwoRooms | 12.7 ±4.3 | 13.6 ±4.7 | 12.9 ±4.9 | 11.9 ±4.2 | 13.1 ±4.4 |
| Reacher | 124.3 ±24.8 | 114.5 ±35.2 | 125.0 ±15.1 | 149.5 ±24.8 | 162.8 ±35.7 |
| PushT | 26.3 ±4.8 | 33.0 ±6.2 | 35.4 ±6.3 | 28.6 ±5.2 | 23.6 ±4.5 |
| cube-single | 14.0 ±4.2 | 23.0 ±6.4 | 20.5 ±5.6 | 14.7 ±4.9 | 12.7 ±4.5 |
| cube-double | 17.1 ±4.9 | 34.1 ±8.7 | 26.2 ±7.4 | 18.0 ±5.1 | 17.4 ±5.6 |
| scene-play | 3.5 ±1.6 | 4.5 ±2.2 | 5.0 ±1.1 | 3.6 ±1.0 | 3.7 ±1.1 |
| puzzle-3x3 | 20.8 ±5.8 | 21.5 ±7.1 | 26.2 ±7.6 | 23.0 ±6.0 | 19.8 ±5.8 |

| Environment | Joint WAF | GPC | CEM | Grad. MPC | **WATO** |
|---|---|---|---|---|---|
| TwoRooms | 12.3 ±4.0 | 11.5 ±4.1 | 65.5 ±14.3 | 10.6 ±4.1 | 10.4 ±3.5 |
| Reacher | 116.0 ±5.1 | 110.8 ±4.0 | 285.3 ±17.7 | 112.7 ±4.7 | 92.9 ±4.2 |
| PushT | 22.1 ±4.6 | 29.5 ±3.5 | 70.0 ±14.9 | 18.8 ±4.0 | 9.3 ±3.3 |
| cube-single | 11.3 ±3.9 | 13.4 ±4.6 | 88.1 ±19.4 | 13.8 ±4.5 | 9.5 ±3.5 |
| cube-double | 11.1 ±4.0 | 16.8 ±6.1 | 93.9 ±21.4 | 15.0 ±4.5 | 11.0 ±4.2 |
| scene-play | 4.4 ±1.6 | 4.2 ±2.8 | 25.2 ±8.7 | 5.3 ±2.9 | 2.3 ±1.7 |
| puzzle-3x3 | 15.2 ±5.0 | 25.4 ±7.2 | 120.5 ±24.6 | 17.7 ±5.2 | 12.7 ±4.3 |

Final goal error complements success rate but does not replace it. Success remains the primary metric because the evaluation criterion is defined in the environment, while latent distance measures how closely the final observation matches the goal under the shared representation.

## H Deployment Efficiency Details

This section expands the deployment-efficiency comparison in Sec. 5.8. We report per-environment latency, define the world model computation count, and study the effect of WATO sampling budget on success and deployment cost.

### H.1 Per-Environment Latency

We measure the wall-clock cost of one planning step. WATO includes joint trajectory generation, world model evaluation of all generated action candidates, candidate selection, and action extraction. CEM includes all online optimization iterations and world model rollouts.

Table 13 reports absolute deployment latency in seconds. The corresponding relative speedups are reported in Table 3. Absolute latency varies across environments, so we report per-environment values rather than averaging raw times.

Table 13: **Per-environment deployment latency.** Wall-clock time in seconds per planning step.

| Environment | CEM (s) | **WATO (s)** |
|---|---|---|
| `TwoRooms` | 1.909 | 0.101 |
| `Reacher` | 1.999 | 0.147 |
| `PushT` | 2.400 | 0.096 |
| `cube-single` | 1.201 | 0.207 |
| `cube-double` | 1.399 | 0.222 |
| `scene-play` | 1.293 | 0.106 |
| `puzzle-3x3` | 1.904 | 0.207 |

### H.2 World Model Evaluations

Wall-clock latency depends on hardware and batching, so we also report world model computation directly. We count one world model evaluation as one latent transition $F_\theta(\mathcal{C}_h, A_h)$ for one candidate trajectory. A full horizon-$H$ rollout therefore uses $H$ evaluations.

WATO generates $K$ action candidates and rolls each candidate through the world model once. Its cost per planning step is therefore

$$N_{\mathrm{WM}}^{\mathrm{WATO}} = KH. \tag{52}$$

The flow matching-only joint WAF uses the same deployment procedure and therefore has the same world model count,

$$N_{\mathrm{WM}}^{\text{Joint WAF}} = KH. \tag{53}$$

Let $N_{\mathrm{CEM}}$ denote the number of CEM iterations and $M_{\mathrm{CEM}}$ the population size. Each iteration evaluates all candidates over the full horizon, giving

$$N_{\mathrm{WM}}^{\mathrm{CEM}} = N_{\mathrm{CEM}} M_{\mathrm{CEM}} H. \tag{54}$$

For gradient-based MPC, let $N_{\mathrm{grad}}$ be the number of optimization steps and $M_{\mathrm{init}}$ the number of parallel initial action trajectories. Each optimization step requires a forward rollout through the horizon, so

$$N_{\mathrm{WM}}^{\text{Grad. MPC}} = N_{\mathrm{grad}} M_{\mathrm{init}} H. \tag{55}$$

Backward differentiation is included in wall-clock latency but not counted as an additional forward world model evaluation.

Because all compared methods use the same horizon $H$, the relative forward-model cost is independent of $H$. For example, the CEM-to-WATO ratio is

$$\frac{N_{\mathrm{WM}}^{\mathrm{CEM}}}{N_{\mathrm{WM}}^{\mathrm{WATO}}} = \frac{N_{\mathrm{CEM}} M_{\mathrm{CEM}}}{K}. \tag{56}$$

This count is independent of batching: evaluating $K$ candidates in one batched forward pass still counts as $KH$ latent transitions.

Table 14: **World model computation per planning step.** Counts are latent transition evaluations.

| Method | Count | Relative to WATO |
|---|---|---|
| Joint WAF | $KH$ | $1\times$ |
| CEM | $N_{\mathrm{CEM}} M_{\mathrm{CEM}} H$ | $\frac{N_{\mathrm{CEM}} M_{\mathrm{CEM}}}{K}$ |
| Grad. MPC | $N_{\mathrm{grad}} M_{\mathrm{init}} H$ | $\frac{N_{\mathrm{grad}} M_{\mathrm{init}}}{K}$ |
| **WATO** | $KH$ | $\mathbf{1}\times$ |

### H.3 Success-Compute Tradeoff

WATO exposes two deployment-time compute parameters. The candidate count $K$ controls how many generated action trajectories are evaluated by the world model, while NFE controls the cost of flow integration. Increasing either can improve candidate quality or coverage at additional compute.

We evaluate $K \in \{1, 4, 8, 16\}$ with NFE fixed at 8, and NFE $\in \{4, 8, 16\}$ with $K$ fixed at 8. All other settings remain unchanged. Figure 6 reports average success across the 7 environments against deployment latency, with the number of world model evaluations shown for each setting.

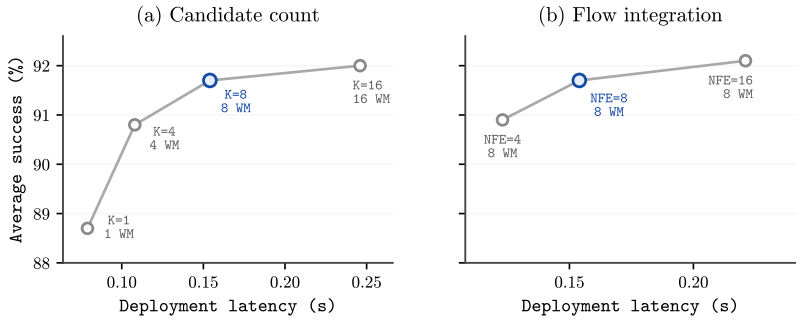


Figure 6: **WATO success-compute tradeoff.** *Left:* varying the number of generated candidates $K$ with NFE $= 8$. *Right:* varying flow integration steps with $K = 8$. The blue marker denotes the default setting used in the main experiments.

Increasing $K$ gives a clear gain from one to eight candidates, while the improvement from $K = 8$ to $K = 16$ is small despite the additional world model evaluations and latency. Increasing NFE shows a similar pattern: NFE $= 8$ captures most of the gain, with little improvement from doubling the integration steps to 16. We therefore use $K = 8$ and NFE $= 8$ as the default, which lies near the knee of both tradeoffs.

# I Call for Research

WATO leaves further questions: *what should we learn to do quickly, and what should we still plan online?* A strong local controller lets us ask where reasoning, value learning, and additional computation are actually most useful.

**A strong controller is only the beginning.** Long-horizon goal reaching has benefited from hierarchical learning (Park et al., 2023; 2025a), but reliably composing local behaviors from offline visual data remains open. Reaching each waypoint is not enough if the resulting state makes the next transition difficult. A high-level plan must choose reachable goals, leave the controller in useful states, and recover when execution deviates from the plan. This is more than simply increasing the planning horizon. We need work that separate local execution, subgoal selection, and their composition.

**World action models beyond scale.** WAM research need not be only about building larger foundation models. As discussed in Appendix A, joint WAMs also form a generative control model class whose objectives, references, and deployment rules deserve study. How should generated futures influence actions? When does joint generation help beyond an equally expressive action-only policy? Which control principles transfer across architectures and scales? Large-scale pretraining and control optimization address different questions. Better generation and better control need not come from the same design choices.

**Put values and broad knowledge where they help most.** Our results also raise a question about where value learning belongs. If local trajectories can already be generated reliably, values may be more useful for selecting subgoals and stitching behaviors than for extracting every motor action. Hierarchical goal-conditioned learning provides a natural starting point (Park et al., 2023). High-level values should also reflect what the deployed low-level controller can actually execute, including its failures, rather than only what transitions appear in the dataset.

Large foundation models may similarly be most useful for semantic reasoning, task decomposition, and goal selection, while a dynamics-aware controller handles local execution. SayCan illustrates how broad knowledge can be grounded through executable skills and their values (Ahn et al., 2022). At the same time, pretrained WAMs can directly generate low-level control (Ye et al., 2026). The question is therefore not only how much to scale, but *where to put that scale*.

**Act fast, deliberate when needed.** *Thinking fast and slow* (Kahneman et al., 2011) suggests allocating computation according to the decision. A familiar subgoal may need only one forward pass, while difficult contact may justify extra sampling or optimization. A useful extension of WATO would decide when direct generation is enough and when to refine a trajectory, increase sampling, or reconsider the goal. Progress, uncertainty, or disagreement between generated and predicted futures could provide such signals, but they should be validated against execution outcomes. The goal is to spend extra computation when it actually improves control.

**Will more compute solve planning, or exploit model error?** Faster hardware, GPU parallelism, and better solvers may make online trajectory optimization attractive. Systems such as cuRobo already show large gains from parallel optimization (Sundaralingam et al., 2023). CEM is also not simply brute-force search: variants such as iCEM use temporal correlations and memory to improve sampling efficiency (Pinneri et al., 2020). Future planners may further use learned proposals or references, erasing the distinction between online optimization and learned generation.

The harder question is whether computation or model error becomes the bottleneck. More search can reduce predicted cost without making the world model more accurate, and may even find trajectories that exploit prediction errors (Yu et al., 2020). WATO's imitation loss and generated references guide optimization, but they do not guarantee model accuracy either. A generator can also learn model exploitation. Optimization error and model error therefore matter for both online planning and amortized control, as reflected in Appendix C.4.

For local visual goals, *WATO should be your favorite controller*. The next challenges are:
**where to go** • **when to think harder** • **how to leverage local abilities in a long horizon**.