LEARNING TO OPTIMIZE ACTIONS WITH FLOWS

Anonymous authors
Paper under double-blind review

ABSTRACT

Model-based planning requires repeatedly solving complex non-convex optimization problems over sequences of actions. These problems are predominantly tackled by sampling-based optimizers, whose hand-crafted update rules cannot exploit structure patterns across planning instances. From a complementary perspective, generative modeling has enabled expressive policies that directly generate action sequences, yet without optimizing them over a horizon through a predictive model. We bridge these two paradigms by introducing **FlowOpt Policy**, which casts planning as a learned transport problem over action sequences: rather than generating actions directly, a flow iteratively transports a population of candidate action sequences toward lower cost, conditioned on their rollout evaluations. The flow is parameterized by an attention mechanism in which every candidate attends over the evaluated population, so that its update depends jointly on where the other candidates lie and how they scored. We further reveal connections to classical methods such as MPPI and CEM, whose updates emerge as particular fixed attention patterns. FlowOpt Policy is trained in a self-supervised manner via flow matching on improvement steps computed by a sampling-based optimizer on the populations the planner encounters, without requiring any demonstration data. Experiments on classical robotics and world model planning tasks show that FlowOpt Policy consistently outperforms sampling-based planners at equal rollout budget, and is more robust than state-conditioned generative policies, generalizing better to unseen objectives because it learns how to optimize rather than what to output.

1 INTRODUCTION

An autonomous agent equipped with a predictive model of its environment has the ability to plan ahead, since it can select actions by optimizing their simulated consequences before executing them (Betts, 2010; M. Moerland et al., 2023). This paradigm is central to sequential decision-making across AI, with applications ranging from robotics (LaValle, 2006) to autonomous navigation (Paden et al., 2016) and interactive agents acting in simulated or virtual environments (Schrittwieser et al., 2020). The underlying predictive model may range from physics simulators (Todorov et al., 2012; Makoviychuk et al., 2021) to learned world models that predict future outcomes in latent space from raw observations (Ha & Schmidhuber, 2018; Hansen et al., 2024; Zhou et al., 2024; Assran et al., 2025). Regardless of the model's form, the induced optimization problems over action sequences are most often highly non-convex, with actions tightly coupled across the horizon through the dynamics. Moreover, when deployed in closed loop as model predictive control (MPC), these problems must be re-solved under strict computational budgets, which makes their efficient solution critical.

While gradient-based solvers excel in smooth, well-behaved landscapes (Jacobson & Mayne, 1970; Li & Todorov, 2004), sampling-based planners such as Cross-Entropy Method (CEM) (De Boer et al., 2005) and Model Predictive Path Integral (MPPI) (Williams et al., 2017), have become the method of choice for planning under non-smooth, contact-rich, and learned models, thanks to their derivative-free and highly parallelizable nature (Howell et al., 2022; Bhardwaj et al., 2022). Nevertheless, these planners rely on fixed, hand-crafted update rules, and their performance hinges on carefully tuned hyperparameters and large sample budgets. More fundamentally, their core optimization mechanism compresses the population into a single nominal sequence through a fixed collapse rule, discarding most of the information carried by the samples. The same rule is moreover applied to every problem instance, so patterns shared across problems from the same task distribution are never exploited. This raises a natural question: can the transport of the population itself be learned, so that candidates move in a problem-adapted way given how they scored?

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.

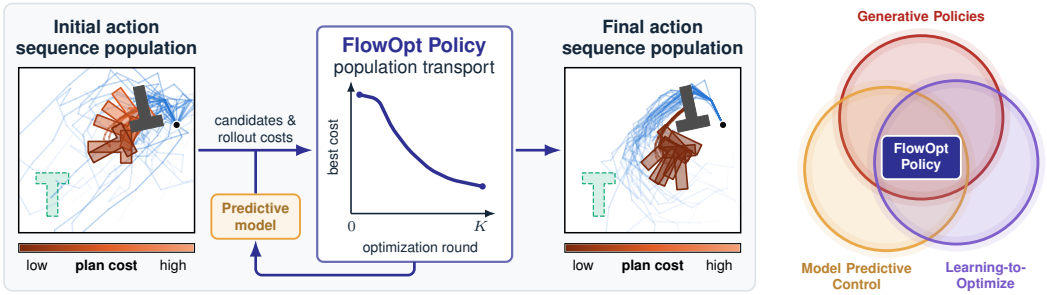

Figure 1: **Conceptual overview.** Our method **FlowOpt Policy** iteratively transports a population of candidate action sequences via a learned flow model toward lower cost, conditioned on their rollout scores, essentially bridging model predictive control, generative modeling and learning-to-optimize.

At the other end of the spectrum, generative policies based on diffusion and flow models have become widely popular, providing a mapping from states to action sequences that captures complex action distributions (Ajay et al., 2022; Janner et al., 2022; Chi et al., 2025; Black et al., 2024). Such policies, however, are trained against a fixed objective, whether by imitating demonstrations or through reinforcement learning, so the task that shaped them is absorbed into the weights and never appears explicitly at deployment. Essentially, the policy returns a sample from its learned conditional distribution without consulting a predictive model, so the returned actions are never optimized against the task at hand. As a result, their performance is bound to the distribution they were trained on, and any change in the cost or constraints at deployment would require retraining.

These two paradigms thus exhibit complementary limitations: sampling-based planners optimize at deployment but do not amortize across instances, while generative policies learn from data but perform no optimization at deployment. This suggests a third option: learn the planner's optimization procedure itself, so that deployment retains an iterative, model-in-the-loop search across action sequences while its update rule is shaped by data. This viewpoint connects with another field, learning to optimize (L2O), where an optimizer is trained on a distribution of problems and then applied to new instances in place of a hand-designed method (Chen et al., 2022; Amos et al., 2023). Sampling-based planners invite this treatment, yet existing learned variants remain limited, primarily learning the parameters of the sampling distribution (Sacks & Boots, 2022; Power & Berenson, 2022).

Contributions. Our key insight is that a sampling-based planner's iteration can be viewed a map transporting a population of candidate action sequences toward lower cost, conditioned on their evaluations under the model, precisely the class of objects that modern generative models are designed to represent. Building on this perspective, we introduce **FlowOpt Policy**, a learned flow that iteratively transports a population of candidate action sequences toward lower cost, conditioned only on their rollout evaluations (Fig. 1). Since the flow is conditioned on costs rather than states, it inherits the flexibility of a planner: at deployment it can optimize objectives it was never trained on. Training is self-supervised and requires no demonstrations, as the flow is trained by flow matching to fit the actions of a sampling-based planner run with a larger rollout budget on the populations the planner encounters. FlowOptPolicy outperforms sampling-based planners at an equal number of rollouts, and, unlike state-conditioned generative policies, generalizes to unseen objectives, both in simulators and in learned world models. Our main contributions are the following:

- We introduce a novel self-supervised framework that casts planning as a learned transport of a population of candidate action sequences. Each planning iteration is represented by a conditional flow that improves the candidates using only the rollout costs of its members.
- We motivate attention over the population as the natural parameterization of this flow. Each candidate attends to the other members through their costs, yielding an update that is invariant to actions and cost scale, allowing to generalize well across instances.
- We reveal a close connection between the proposed method and classical sampling-based planners: the updates of MPPI and CEM correspond to fixed attention patterns, further motivating attention as the right inductive bias for learning to optimize actions.
- We provide an experimental demonstration on physics-based and world model tasks, showing that our approach consistently outperforms well-established sampling-based planners, and exhibits substantially better generalization than state-conditioned generative policies.

2 Related Work

Sampling-based predictive control. Sampling-based planners such as MPPI (Williams et al., 2017), CEM (De Boer et al., 2005), CMA-ES (Hansen & Ostermeier, 2001) and others, optimize action sequences by drawing candidates from a parametric proposal, scoring them through model rollouts, and refitting the proposal around the best-scoring ones. Recent approaches have interpreted these planners through diffusion: model-based diffusion (Pan et al., 2024) estimates the score of a noise-smoothed cost from model rollouts, making each reverse step an MPPI-style weighted average at a shrinking noise level, while DIAL-MPC (Xue et al., 2025) anneals the sampling variance across iterations and along the horizon. OT-MPC (Pacelli et al., 2026) computes an entropy-regularized optimal transport coupling between the candidates and their low-cost proposals, so that each candidate is refined toward nearby promising samples, though the coupling itself remains a fixed Sinkhorn rule. Amortization has primarily entered these planners only at the level of the sampling distribution, with Sacks & Boots (2022) learning the update of its Gaussian parameters by imitating an MPPI expert with a larger sample budget, while Sacks & Boots (2023) and Power & Berenson (2022) learn richer proposals from which the classical cost-weighted update then draws.

Generative policies. Diffusion and flow models have become a standard policy class for control, generating action sequences conditioned on the current observation (Chi et al., 2025) or on returns and constraints (Ajay et al., 2022). A second line generates full trajectories and steers the sampling process at test time, through value-gradient guidance (Janner et al., 2022), constraint enforcement inside the reverse process via barrier functions (Xiao et al., 2025) or constrained Langevin steps (Zhang et al., 2026), or reward-guided generation of synthetic data for self-improvement (Liang et al., 2023). Closest to us are methods that pair a generator with a predictive model: D-MPC (Zhou et al., 2025) learns a diffusion action proposal alongside a diffusion dynamics model and plans by sampling, scoring and ranking, while Kurtz & Burdick (2025) train a state-conditioned flow policy on the outputs of a sampling-based planner and uses its samples to warm-start it.

Learning-to-optimize. Learning-to-optimize replaces hand-designed optimization algorithms with learned ones trained on a specific distribution of problem instances (Chen et al., 2022; Amos et al., 2023). More recently, generative models have been repurposed as optimizers, by treating diffusion or flow models as samplers concentrated on low-cost or feasible regions (Krishnamoorthy et al., 2023; Kong et al., 2024; Uehara et al., 2024; Saravanos et al., 2026). Within trajectory optimization and MPC, learning-to-optimize approaches have so far been used to warm-start a downstream solver, either through learned initializations (Sambharya et al., 2023), replacing the optimization with with learned components (Sacks et al., 2024), or through diffusion and flow models that propose candidate trajectories subsequently refined by a local solver (Li et al., 2024; Idoko et al., 2026).

3 Background

Notation. Three time-like indices appear in this work: subscript t denotes the planning timestep, superscript k the optimization iteration, and $\tau \in [0, 1]$ the flow time.

3.1 Problem Setting

Model predictive control (MPC) is a framework for sequential decision-making in which a finite-horizon sequence of actions is optimized using a predictive model. At its core, the underlying optimization problem that is being solved recursively in a receding horizon fashion has the form:

$$\min_{\bar{\boldsymbol{a}}} \; J(\boldsymbol{s}_0, \bar{\boldsymbol{a}}) := \sum_{t=0}^{H-1} \ell(\boldsymbol{s}_t, \boldsymbol{a}_t) + \phi(\boldsymbol{s}_H) \tag{1a}$$

$$\text{s.t.} \quad \boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \boldsymbol{a}_t), \quad \boldsymbol{s}_0 = \hat{\boldsymbol{s}}_0, \quad t = 0, \dots, H-1, \tag{1b}$$

where $\boldsymbol{s}_t \in \mathbb{R}^{n_s}$ and $\boldsymbol{a}_t \in \mathbb{R}^{n_a}$ denote the state and action at time step t, respectively, $\hat{\boldsymbol{s}}_0$ is a given initial state, $f(\cdot)$ describes the system dynamics and $J(\cdot)$ defines the cost to minimize (or reward to maximize) consisting of a running term $\ell(\cdot)$ and a terminal one $\phi(\cdot)$. The state and action sequences over a planning horizon H are defined as $\bar{\boldsymbol{s}} = \{\boldsymbol{s}_0, \dots, \boldsymbol{s}_H\}$ and $\bar{\boldsymbol{a}} = \{\boldsymbol{a}_0, \dots, \boldsymbol{a}_{H-1}\}$, respectively.

3.2 SAMPLING-BASED OPTIMIZATION

Sampling-based methods such as MPPI (Williams et al., 2017), CEM (De Boer et al., 2005) and others, address problem (1) by iteratively sampling from a parametric distribution, evaluating candidate action sequences and updating the distribution parameters. In particular, at optimization iteration k, a population of P candidate action sequences is drawn around the current nominal sequence,

$$\bar{\boldsymbol{a}}_j^k = \bar{\boldsymbol{a}}_{\text{nom}}^k + \boldsymbol{\epsilon}_j^k, \qquad \boldsymbol{\epsilon}_j^k \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}), \qquad j = 1, \dots, P, \tag{2}$$

and each candidate is scored by $J_j^k := J(\hat{\boldsymbol{s}}_0, \bar{\boldsymbol{a}}_j^k)$, which entails rolling it out through the dynamics (1b). The nominal sequence is then updated as a cost-weighted average of the sampled perturbations:

$$\bar{\boldsymbol{a}}_{\text{nom}}^{k+1} = \bar{\boldsymbol{a}}_{\text{nom}}^k + \sum_{j=1}^{P} w_j^k \boldsymbol{\epsilon}_j^k, \qquad w_j^k = \frac{g\big(J_j^k\big)}{\sum_{i=1}^{P} g\big(J_i^k\big)}. \tag{3}$$

Different sampling-based algorithms correspond to different choices of the weighting function $g(\cdot)$. In MPPI, the weights are defined through exponential cost shaping, $g_{\text{MPPI}}(J_j) = \exp\left(-J_j/\lambda\right)$, where $\lambda > 0$ is the temperature parameter. In CEM, only the N_{elite} lowest-cost (elite) samples $\mathcal{E}$ contribute, $g_{\text{CEM}}(J_j) = \mathbf{1}\{j \in \mathcal{E}\}$, so that $w_j^k = 1/N_{\text{elite}}$ for $j \in \mathcal{E}$ and zero otherwise.

4 LEARNING TO OPTIMIZE ACTIONS VIA FLOWS

4.1 PLANNING AS TRANSPORT OF ACTION POPULATIONS

Classical sampling-based methods typically operate by evolving a population of samples, i.e., candidate action sequences, through assigning them with a score, compressing them into a nominal sequence, and re-expanding through sampling around that summary. The distinguishing feature among such methods including MPPI, CEM and others, is the most often merely the collapse rule, which in addition, is a hand-designed function of the rollout costs, while most other operations are fixed. Viewed abstractly, one iteration is therefore a map that takes a population together with its evaluations and returns a new population. Motivated by this perspective, we instead frame planning as a *transport* problem over the space of *action sequences*. Our goal is to learn from data how the candidates should move and redistribute given how they scored. To formalize this viewpoint, let a planner maintain a population of P candidate action sequences:

$$\mathcal{A}^k = \{\bar{\boldsymbol{a}}_j^k\}_{j=1}^P, \qquad \bar{\boldsymbol{a}}_j^k = \big\{\boldsymbol{a}_{j,0}^k, \dots, \boldsymbol{a}_{j,H-1}^k\big\} \in \mathbb{R}^{H \times n_a}. \tag{4}$$

at optimization iteration k. Given the current state $\hat{\boldsymbol{s}}_0$, each candidate $\bar{\boldsymbol{a}}_j^k$ is rolled out under the dynamics (1b), to obtain its cost $J_j^k := J(\hat{\boldsymbol{s}}_0, \bar{\boldsymbol{a}}_j^k)$. All information available to the planner at iteration k (i.e., the population and its costs) is collected in the *iteration context* $\mathcal{C}^k$ specified below. One planning iteration then corresponds to a transport of the evaluated population,

$$\mathcal{A}^{k+1} = \Phi_\theta\big(\mathcal{A}^k; \mathcal{C}^k\big), \tag{5}$$

where θ are learnable parameters. Note that MPPI, CEM, and others, can be viewed as hard-coded instances of (5), collapsing the population onto a nominal sequence through the convex combination (3) and re-expanding by Gaussian sampling around it. In this case though, the scores enter only through the prescribed weighting $g(\cdot)$, and all structure within the population beyond this weighted mean is discarded. Our goal is instead to learn a transport Φ_θ that better uses the costs information, in which both the recombination and the re-diversification of the population are determined by data.

This is also a crucial difference compared to how widely used generative control policies work. Such policies typically learn a state-conditioned policy for generating actions, i.e., $\bar{\boldsymbol{a}} \sim \pi_\theta(\cdot|\hat{\boldsymbol{s}}_0)$, without optimizing over a horizon leveraging a model at deployment. As a result, the objective they were trained on is absorbed into θ, and their performance hinges on the training distribution capturing every state and objective met at deployment, with no mechanism to correct a generated sequence once its cost is known. Our learned model-based planner instead returns

$$\mathcal{A}^K = \Phi_\theta\big(\cdot\,; \mathcal{C}^{K-1}\big) \circ \cdots \circ \Phi_\theta\big(\cdot\,; \mathcal{C}^0\big)\big(\mathcal{A}^0\big), \qquad \bar{\boldsymbol{a}}^\star = \operatorname*{arg\,min}_{\bar{\boldsymbol{a}} \in \mathcal{A}^K} J(\hat{\boldsymbol{s}}_0, \bar{\boldsymbol{a}}), \tag{6}$$

so that the objective enters at deployment through the contexts $\mathcal{C}^k$ rather than through the weights, and what is learned is how to use the costs to improve the population.

4.2 Optimizing Action Populations via Conditional Flows

Learning this transport is challenging since rollout costs assess complete action sequences but do not specify how each candidate should move. We therefore seek an update that moves the population jointly without first reducing it to a single nominal sequence, and a flow is a natural parameterization for it. Unlike the usual diffusion-policy construction, which generates actions by denoising from a noise prior, our optimizer already has a meaningful starting point, namely the previous population. We use it as the initial condition of a cost-conditioned velocity field and learn to transport it toward improved populations, which yields a direct source-to-target training problem through flow matching, while the resulting candidates can be evaluated again before the next optimization iteration.

Let $\boldsymbol{x} \in \mathbb{R}^{P \times Hn_a}$ denote the stacked population, whose j-th row is $\boldsymbol{x}_j = \mathrm{vec}(\bar{\boldsymbol{a}}_j^k)$. We parameterize Φ_θ as the time-one map of a velocity field on the population space,

$$\dot{\boldsymbol{x}}(\tau) = v_\theta\big(\boldsymbol{x}(\tau), \tau; \mathcal{C}^k\big), \qquad \boldsymbol{x}(0) = \mathcal{A}^k, \qquad \mathcal{A}^{k+1} = \boldsymbol{x}(1), \tag{7}$$

over flow time $\tau \in [0, 1]$. The context $\mathcal{C}^k$ is held fixed throughout the integration.

During deployment, starting from an initial population $\mathcal{A}^0$, the planner alternates evaluation and transport for K iterations, each transport integrating (7) with N_τ explicit Euler steps,

$$\boldsymbol{x}^{(m+1)} = \boldsymbol{x}^{(m)} + \frac{1}{N_\tau}\, v_\theta\big(\boldsymbol{x}^{(m)}, \tfrac{m}{N_\tau}; \mathcal{C}^k\big), \qquad m = 0, \ldots, N_\tau - 1, \tag{8}$$

with $\boldsymbol{x}^{(0)} = \boldsymbol{x}^k$ and $\boldsymbol{x}^{k+1} = \boldsymbol{x}^{(N_\tau)}$. After K iterations the planner returns the best member $\bar{\boldsymbol{a}}^\star = \bar{\boldsymbol{a}}_{j^\star}^K$, $j^\star = \arg\min_j J_j^K$, at a total cost of $(K+1)P$ rollouts. In receding-horizon control, the first action of $\bar{\boldsymbol{a}}^\star$ is applied and the next control step is warm-started by shifting $\mathcal{A}^K$ one step forward.

Several key properties follow from the velocity field defined in (7). First, the context is frozen during the flow, so one iteration costs P rollouts regardless of N_τ, exactly as one iteration of MPPI or CEM. Second, in contrast to MPPI and CEM that summarize the population by a Gaussian and resample from it; the velocity field moves the evaluated members directly. Finally, beyond the population itself, v_θ depends exclusively on the rollout costs exclusively through $\mathcal{C}^k$ making it effectively a method learns how to optimize rather than how to output. What is learned is therefore how to move a population given how it scored, which is what (3) hard-codes through $g(\cdot)$.

4.3 Architecture of the Velocity Field

Having cast planning as a transport problem via a flow-based model, the natural next question is *what neural network architecture* is the most suitable. We identify three desiderata. First, the learned optimizer should be invariant to the scale and location of the action sequences and costs, since the velocity field in (7) operates on populations ranging from a random cloud at the start of an optimization to a tight cluster near a solution. Second, each candidate should observe the other candidates together with their costs and derive its update from the most promising combination of them. Third, the architecture should be permutation equivariant among the members.

Transformation invariance. Although the scores assigned to the population is the signal that directs the transport, raw costs can be a poor conditioning signal for the velocity field, as their scale varies. We therefore describe each member relative to its population through a normalized rank $\rho_j^k = \mathrm{rank}(J_j^k)/(P-1)$ and a standardized cost $\tilde{J}_j^k = (J_j^k - \bar{J}^k)/\sigma^k$, where $\bar{J}^k$ and σ^k denote the empirical mean and standard deviation. The iteration context is then $\mathcal{C}^k = \{(\rho_j^k, \tilde{J}_j^k)\}_{j=1}^P$, which is invariant to positive affine rescalings $J \mapsto aJ + b$, $a > 0$, allowing a single v_θ to be applied across instances with different cost scales. Similarly, the actions are also handled relative to the population. With $\bar{\boldsymbol{x}}^k$ and s^k the mean and root-mean-square deviation of the population, the network reads members in standardized coordinates and returns its output in the same units, $v_\theta\big(\boldsymbol{x}, \tau; \boldsymbol{x}^k, \mathcal{C}^k\big) = s^k\, g_\theta\big(\hat{\boldsymbol{x}}, \tau; \hat{\boldsymbol{x}}^k, \mathcal{C}^k, \bar{\boldsymbol{x}}^k/s^k\big)$, with $\hat{\boldsymbol{x}}_j = (\boldsymbol{x}_j - \bar{\boldsymbol{x}}^k)/s^k$, where the scaled mean $\bar{\boldsymbol{x}}^k/s^k$ retains the absolute location. Displacements are thus measured in units of the population's own spread, whether that spread is a random cloud or a tight cluster near a solution.

Attention over the population. The evaluated population acts as a scored memory from which the transported members read. A shared embedding pairs a standardized action sequence with its

scores, giving $\boldsymbol{h}_i^0 = \eta_\theta(\hat{\boldsymbol{x}}_i^k, \rho_i^k, \tilde{J}_i^k)$ for each evaluated member and $\boldsymbol{h}_j = \eta_\theta(\hat{\boldsymbol{x}}_j(\tau), \rho_j^k, \tilde{J}_j^k)$ for each transported member at its current position. A cross-attention layer lets each transported member attend to all evaluated members, and a per-member decoder ψ_θ returns its displacement,

$$g_\theta(\cdot)_j = \psi_\theta\Big(\boldsymbol{h}_j + \mathrm{Attn}_\theta\left(\boldsymbol{h}_j;\, \{\boldsymbol{h}_i^0\}_{i=1}^P\right),\ \tau,\ \bar{\boldsymbol{x}}^k/s^k\Big). \tag{9}$$

Pairing $\hat{\boldsymbol{x}}_i^k$ with $(\rho_i^k, \tilde{J}_i^k)$ inside $\boldsymbol{h}_i^0$ associates each score to the position at which it was measured, and the residual term keeps a member's own position and score available to the decoder alongside the attended mixture. Every member's velocity is thus a function of its own state and evaluation and of the entire evaluated population, the information a classical planner reduces to a weighted combination. Permutation equivariance follows since Attn_θ is invariant to the order of the set $\{\boldsymbol{h}_i^0\}$, all per-member maps share weights, and $(\bar{\boldsymbol{x}}^k, s^k)$ are symmetric functions of the members.

4.4 Training

Training the velocity field (7) effectively faces several difficulties. First of all, we assume no demonstration data is available, so the only feedback on a candidate is the cost of rolling it out through the model. Second, the inputs of the field are the populations met in closed loop, whose distribution is induced by the controller itself. As a result, populations drawn around a fixed predefined initial-state distribution would not match those encountered once the learned optimizer drives the system. To address both, we let the learned optimizer run as a receding-horizon controller, label every population it produces with a stronger sampling-based teacher, and refit the field on the labeled pairs.

The initial population of each control step is the final population of the previous step shifted by one action, and the first step of an episode draws $\boldsymbol{x}_j^0 \sim \mathcal{N}(\boldsymbol{0}, \sigma_g^2 \boldsymbol{I})$. The same procedure is used in training and at deployment, so training populations are exactly those the closed loop produces. Each population $\boldsymbol{x}^k$ met by the field at state $\hat{\boldsymbol{s}}_n$ is labelled by a teacher seeded by it: a sampling-based optimizer initialized at the mean and per-coordinate standard deviation of $\boldsymbol{x}^k$ runs K_T iterations of P_T rollouts. The target $\boldsymbol{x}^{k,+}$ consists of the P cheapest sequences among $\boldsymbol{x}^k$ and everything the teacher evaluated, so every target has a known cost, the sorted target costs dominate the sorted source costs elementwise, and a member better than anything the teacher found is its own target. Targets are matched to members by the permutation π minimizing $\sum_j \|\boldsymbol{x}_j^k - \boldsymbol{x}_{\pi(j)}^{k,+}\|^2$, the optimal transport coupling between the two empirical measures under squared Euclidean cost, and the tuple $(\boldsymbol{x}^k, \mathcal{C}^k, \boldsymbol{x}_\pi^{k,+})$ is stored for every k. After every N_E episodes the field is refitted on all stored tuples with the flow-matching loss on straight paths $\boldsymbol{x}_\tau = (1-\tau)\boldsymbol{x} + \tau \boldsymbol{x}^+$,

$$\mathcal{L}(\theta) = \mathbb{E}_{(\boldsymbol{x}, \mathcal{C}, \boldsymbol{x}^+)\sim\mathcal{D},\ \tau\sim U[0,1)}\ \frac{1}{P}\sum_{j=1}^{P}\Big\|v_\theta\big(\boldsymbol{x}_\tau, \tau;\, \mathcal{C}\big)_j - \big(\boldsymbol{x}_j^+ - \boldsymbol{x}_j\big)\Big\|^2, \tag{10}$$

which makes no call to the simulator since $\mathcal{C}$ is rebuilt from stored rollouts. The field thus learns to mimic the performance K_T teacher iterations at the cost of a single population evaluation, while the executed actions remain the field's own and the teacher only labels.

4.5 Connections to Classical Sampling-Based Optimization

Our attention-based parameterization is closely related with the core mechanism underlying classical sampling-based optimizers such as MPPI and CEM. These methods assign greater influence to promising action sequences through their rollout costs and aggregate across the resulting weighted population, which can be viewed as a simplified operation performed by our member-axis attention. Recall that attention computes $\mathrm{Attn}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \mathrm{softmax}(\boldsymbol{Q}\boldsymbol{K}^\top/\sqrt{d})\boldsymbol{V}$, where the rows of $\boldsymbol{Q}$ are the queries of the members being updated, the rows of $\boldsymbol{K}$ are the keys describing the members from which information may be drawn, and the rows of $\boldsymbol{V}$ are the corresponding values to be aggregated. Considering this operation along the population at a fixed timestep t, the output for member j is

$$\boldsymbol{y}_j = \sum_{i=1}^{P} \alpha_{ji}\boldsymbol{v}_i, \qquad \alpha_{ji} = \frac{\exp(r_{ji})}{\sum_{i'=1}^{P}\exp(r_{ji'})}, \tag{11}$$

where α_{ji} is the attention weight determining the influence of member i on the update of member j, $\boldsymbol{v}_i$ is the value contributed by member i, and r_{ji} is the corresponding attention logit. The updates of MPPI and CEM arise from particular choices of these values and logits, as summarized below.

Proposition 1 (MPPI and CEM as special attention patterns)**.** *Let the values in (11) be the candidate actions sequences, i.e., $\boldsymbol{v}_i = \bar{\boldsymbol{a}}_i$. For any normalized weights $\{w_i\}_{i=1}^P$, an attention block followed by one Euler step can implement*

$$\bar{\boldsymbol{a}}_j^+ = \bar{\boldsymbol{a}}_j + \left(\sum_{i=1}^P w_i \bar{\boldsymbol{a}}_i - \frac{1}{P} \sum_{i=1}^P \bar{\boldsymbol{a}}_i \right) \tag{12}$$

*for every population member j. Consequently, the mean of the updated population equals the weighted mean $\sum_{i=1}^P w_i \bar{\boldsymbol{a}}_i$. Setting the attention logits as $r_{ji} = -J_i/\lambda$, yields the **MPPI** weights*

$$\alpha_{ji} = \frac{\exp(-J_i/\lambda)}{\sum_{i'=1}^P \exp(-J_{i'}/\lambda)} = w_i^{\mathrm{MPPI}}. \tag{13}$$

*Setting $r_{ji} = 0$ if $i \in \mathcal{E}$ and $r_{ji} = -\beta$ otherwise, yields the **CEM** weights*

$$\alpha_{ji} \xrightarrow[\beta \to \infty]{} \frac{\mathbf{1}\{i \in \mathcal{E}\}}{|\mathcal{E}|} = w_i^{\mathrm{CEM}}. \tag{14}$$

The above exposition reveals that MPPI and CEM coincide with restricted attention patterns of our architecture, since their logits are fixed functions of rollout cost and identical for every querying member. Our learned attention layer relaxes all three restrictions. Its logits can depend jointly on the member being updated and the members supplying information, using their actions and rollout costs. Consequently, different members may draw information from different promising regions of the population, rather than all being driven toward a single average. The proposed parameterization therefore retains the effective cost-weighted aggregation underlying classical sampling-based optimization while extending it to richer, context-dependent population updates.

5 Experiments

Our experiments span classical physics-based systems and learned latent world models. The results exposition highlights that **FlowOpt Policy** significantly outperforms hand-crafted sampling-based planners at an equal rollout budget, and that that optimizing at deployment yields better performance and generalization under distribution shift than widely used direct state-conditioned generation.

5.1 Tasks and Baselines

Tasks and metrics. We consider four physics-based tasks, CarNav (a custom navigation task among obstacles with moving targets), HalfCheetah and Walker from MuJoCo (Todorov et al., 2012), and Push-T (Chi et al., 2025), as well as the HalfCheetah-WM and Humanoid-WM tasks planned inside the learned latent world model of TD-MPC2 (Hansen et al., 2024). On the physics tasks, all planners roll out the ground-truth simulator, while on the world-model ones, they roll out the same frozen latent dynamics from the current encoded observation and evaluate the same latent cost. For CarNav we report the closed-loop cost accumulated over the episode and for Walker the mean per-step cost, each normalized per task by the mean cost of the best method. For HalfCheetah, whose per-step cost is minus the forward speed plus penalties, we report the corresponding return normalized by that of the best method. For the world-model tasks, we report the episode return over 500 steps, normalized in the same way. For Push-T, we report the success rate, i.e., the fraction of episodes in which the final block pose lies close to the goal, and use the mean stage cost for the performance profiles.

Baselines. We compare **FlowOpt Policy** against MPPI (Williams et al., 2017), CEM De Boer et al. (2005), CMA-ES (Hansen & Ostermeier, 2001), and DIAL-MPC (Xue et al., 2025). under the same number of rollouts, details listed in Appendix D. To illustrate the value of using a generative model for optimizing under a predictive model, we further compare against a policy $\bar{\boldsymbol{a}} \sim \pi_\theta(\cdot \mid \hat{\boldsymbol{s}}_0)$ that generates action sequences conditioned on the current state (and potentially the target state and other problem information), which we refer to as Flow Policy. This baseline is representative of generative policies (Chi et al., 2025; Black et al., 2024), which solely use single conditional generation and never evaluate the model at deployment for planning. For fairness purposes, Flow Policy uses the same neural network architecture and is trained identically to **FlowOpt Policy** .

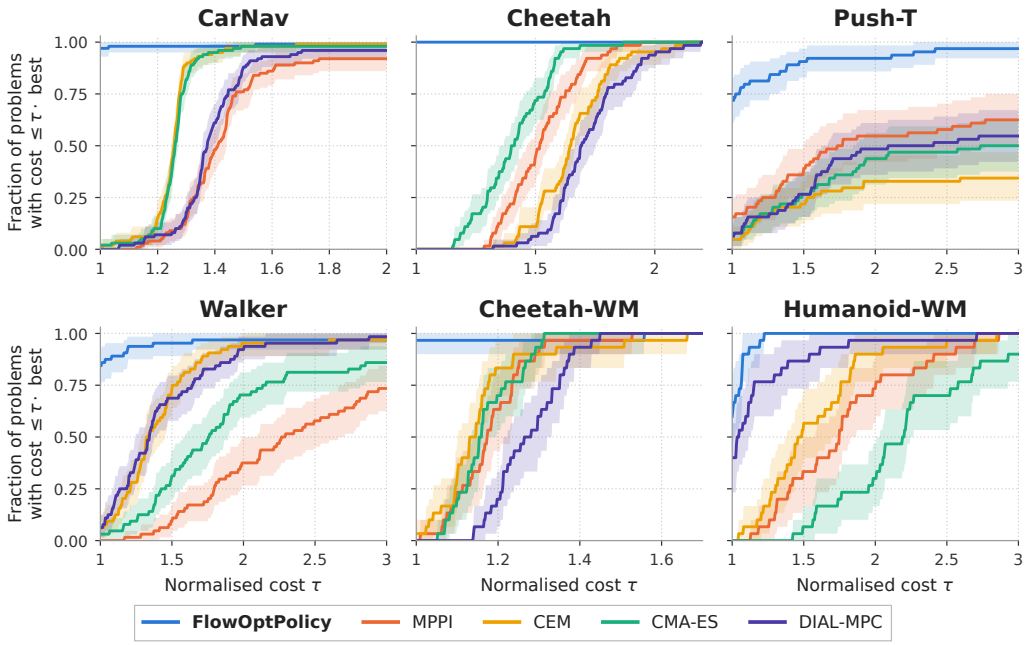

Figure 2: **Performance profiles on physics-based and world-model tasks.** Each curve shows the fraction of problems on which a method's closed-loop cost is within a factor τ of the best performer (closer to the top-left is better). `FlowOpt Policy` is at or near the best cost on nearly all problems.

5.2 Comparison with Sampling-Based Planners

We compare `FlowOpt Policy` with the classical sampling-based planners on the six tasks at matched rollout budgets. Figure 2 reports performance profiles (Dolan & Moré, 2002), which complement the aggregate metrics of Table 1 in Appendix D, by assessing how consistently each method performs across individual problem instances. For each method, the curve shows the fraction of evaluation episodes on which its closed-loop cost is within a factor τ of the best planner on that episode. `FlowOpt Policy` has the highest profile at almost all τ on all six tasks. On HalfCheetah it is the best planner on every episode, whereas the baselines need $\tau \approx 2$ to cover all episodes. On CarNav, Walker, Cheetah-WM and Humanoid-WM, it is best on at least 85% of episodes and saturates by $\tau \approx 1.5$, while the baselines require τ between 1.4 and 2.5, and `MPPI` and `CMA-ES` do not saturate within $\tau = 3$ on Walker. The challenging Push-T separates the methods most sharply, as `FlowOpt Policy` covers 96% of episodes by $\tau = 1.5$, whereas every baseline plateaus between 35% and 65% even at $\tau = 3$. Since all planners roll out the same model under the same budget, these gaps are attributable to the ability of the underlying optimization in finding effective action sequences.

5.3 Generalization

Next, we compare against `Flow Policy`, which shares the architecture and training with `FlowOpt Policy` but directly outputs state-conditioned action sequences. We evaluate both on the training distribution and on shifted settings, CarNav with four and eight obstacles instead of two, and Push-T with a wider initial block range and altered friction. As shown in Figure 3, the two methods are indistinguishable on CarNav in distribution, but as obstacles are added, `FlowOpt Policy` attains roughly 17% lower closed-loop cost, with the cumulative-cost curves showing that the gap opens after each goal switch (dashed lines), when the warm start is invalidated and the planner must find a path through an obstacle configuration the generator never saw. On Push-T, `FlowOpt Policy` is better already in distribution and the gap persists under both shifts, appearing within the first steps of the episode, consistent with the task being multimodal from the start rather than becoming so after a change in the objective. Optimizing against the model at deployment is thus beneficial even in distribution when the task has several competing solution modes, and under distribution shift the advantage is structural: the shift alters the rollout costs, which is the only interface `FlowOpt Policy` was trained on, whereas for `Flow Policy` it alters the input, which hurts its generalization.

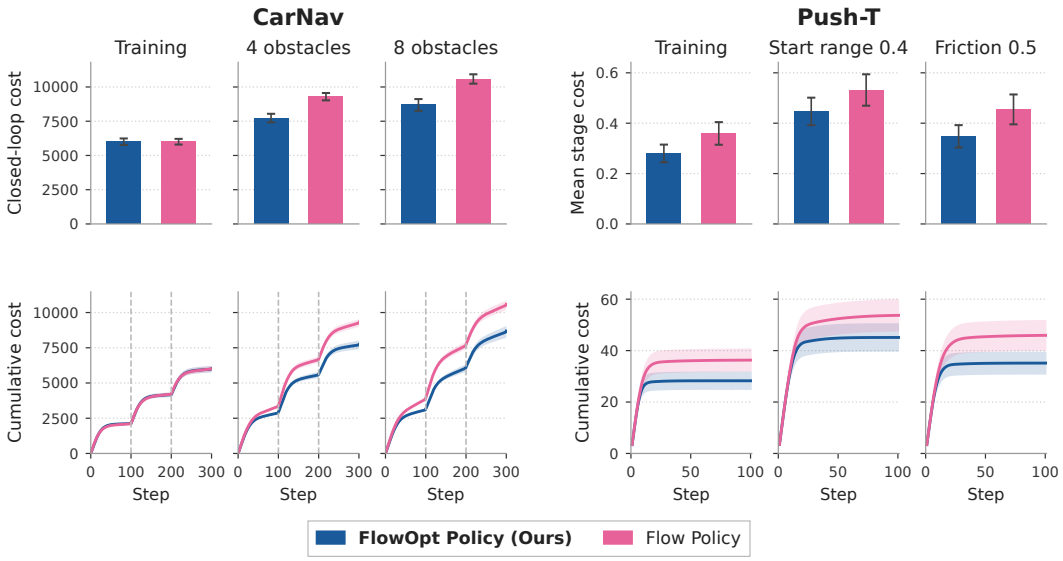

Figure 3: **Generalization capabilities.** `FlowOpt Policy` against `Flow Policy`, a state-conditioned generative policy with identical architecture and training data on CarNav with more obstacles than seen in training and on Push-T with a wider initial block range and altered friction. `FlowOpt Policy` attains lower cost on every shifted setting, thanks to optimizing using the model at deployment.

5.4 Ablations

We further evaluate our method as the number of optimization iterations per control step increases beyond the number used in training. As shown in Figure 4, `FlowOpt Policy` at three iterations attains a clear advantage against `MPPI` and `CEM` at nine, on both CarNav and Push-T. Additional iterations keep improving `FlowOpt Policy` on CarNav, whereas on Push-T its cost remains flat, indicating that the learned transport has largely converged within the trained budget; the baselines improve steadily but do not close the gap. Applying the learned optimizer for more iterations than it was trained for thus never degrades its performance, and the advantage over hand-crafted updates persists across the full range of budgets.

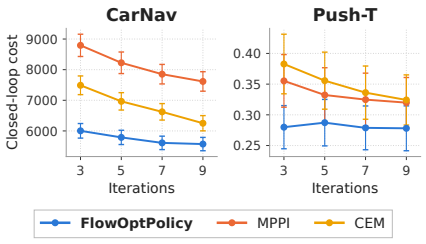

Figure 4: The effect of increasing iterations in the CarNav and Push-T tasks.

Limitations. `FlowOpt Policy` is essentially a learning-to-optimize method so although it is shown to generalize better than direct generative policies, its learned transport remains primarily tailored to the training distribution. The current training scheme is potentially myopic, since the teacher supervises a single population update over a finite horizon rather than the closed-loop return; therefore combining the optimizer with a learned terminal value function is a natural next direction. Finally, as the method drives populations toward low-cost regions more aggressively, it may exploit inaccuracies of a learned world model in regions where the model was never validated.

6 Conclusion

We introduced FlowOpt Policy, which casts planning as a learned transport problem over action sequences and learns, via a flow-based model parameterized by attention over the population, how candidates should move given how they scored. We further show that classical sampling-based planners are recovered as special cases as fixed attention patterns. Across physics-based and world-model tasks, FlowOpt Policy consistently outperforms sampling-based planners, and generalizes better than state-conditioned generative policies to unseen objectives. We believe that this work paves the way toward planners that learn their own optimization procedure, combining the adaptability of model-based search with the expressiveness of modern generative models.

AI USE STATEMENT

In this work, we used generative AI tools for code implementation methodologies and polishing the writing. We have not used generative AI tools for formulating the proposed methodology and framework. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work develops a learned optimizer for planning in simulated control and robotics environments, evaluated entirely on synthetic benchmarks and learned world models. The research does not involve human subjects, personal data, or sensitive information and raises no immediate concerns related to privacy, security, or fairness. We have not identified any potential conflicts of interest, legal compliance issues, or potential harmful applications from this research.

REPRODUCIBILITY STATEMENT

To ensure reproducibility, Appendices B and D provide extensive details of the architecture and the experiments, respectively. Complete proofs of theoretical statements are given in Appendix C. The full implementation, including code and the trained models, will be released upon publication.

REFERENCES

Anurag Ajay, Yilun Du, Abhi Gupta, Joshua Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making? *arXiv preprint arXiv:2211.15657*, 2022.

Michael Samuel Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=li7qeBbCR1t.

Brandon Amos et al. Tutorial on amortized optimization. *Foundations and Trends® in Machine Learning*, 16(5):592–732, 2023.

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

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

Mohak Bhardwaj, Balakumar Sundaralingam, Arsalan Mousavian, Nathan D Ratliff, Dieter Fox, Fabio Ramos, and Byron Boots. Storm: An integrated framework for fast joint-space model-predictive control for reactive manipulation. In *Conference on Robot Learning*, pp. 750–759. PMLR, 2022.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to optimize: A primer and a benchmark. *Journal of Machine Learning Research*, 23(189):1–59, 2022.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025. doi: 10.1177/02783649241273668. URL https://doi.org/10.1177/02783649241273668.

Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

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

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.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

Taylor Howell, Nimrod Gileadi, Saran Tunyasuvunakool, Kevin Zakka, Tom Erez, and Yuval Tassa. Predictive sampling: Real-time behaviour synthesis with mujoco. *arXiv preprint arXiv:2212.00541*, 2022.

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.

David H Jacobson and David Q Mayne. Differential dynamic programming. *(No Title)*, 1970.

Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9902–9915. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/janner22a.html.

Lingkai Kong, Yuanqi Du, Wenhao Mu, Kirill Neklyudov, Valentin De Bortoli, Dongxia Wu, Haorui Wang, Aaron Ferber, Yi-An Ma, Carla P Gomes, et al. Diffusion models as constrained samplers for optimization with unknown constraints. *arXiv preprint arXiv:2402.18012*, 2024.

Siddarth Krishnamoorthy, Satvik Mehul Mashkaria, and Aditya Grover. Diffusion models for black-box optimization. In *International Conference on Machine Learning*, pp. 17842–17857. PMLR, 2023.

Vince Kurtz and Joel W Burdick. Generative predictive control: Flow matching policies for dynamic and difficult-to-demonstrate tasks. *arXiv preprint arXiv:2502.13406*, 2025.

Steven Michael LaValle. *Planning algorithms*, volume 1. Cambridge university press Cambridge, UK, 2006.

Anjian Li, Zihan Ding, Adji Bousso Dieng, and Ryne Beeson. Diffusolve: Diffusion-based solver for non-convex trajectory optimization. *arXiv preprint arXiv:2403.05571*, 2024.

Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *First International Conference on Informatics in Control, Automation and Robotics*, volume 2, pp. 222–229. SciTePress, 2004.

Zhixuan Liang, Yao Mu, Mingyu Ding, Fei Ni, Masayoshi Tomizuka, and Ping Luo. Adaptdiffuser: Diffusion models as adaptive self-evolving planners. *arXiv preprint arXiv:2302.01877*, 2023.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=PqvMRDCJT9t.

Xingchao Liu, Chengyue Gong, and qiang liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=XVjTT1nw5z.

Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. Model-based reinforcement learning: A survey. *Foundations and Trends in Machine Learning*, 16(1):1–118, 2023.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance GPU based physics simulation for robot learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=fgFBtYgJQX_.

Vincent Pacelli, Akash Ratheesh, and Evangelos A Theodorou. Sampling-based control via entropy-regularized optimal transport. *arXiv preprint arXiv:2605.02147*, 2026.

Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, 1(1):33–55, 2016.

Chaoyi Pan, Zeji Yi, Guanya Shi, and Guannan Qu. Model-based diffusion for trajectory optimization. *Advances in Neural Information Processing Systems*, 37:57914–57943, 2024.

Thomas Power and Dmitry Berenson. Variational inference mpc using normalizing flows and out-of-distribution projection. *arXiv preprint arXiv:2205.04667*, 2022.

Jacob Sacks and Byron Boots. Learning to optimize in model predictive control. In *2022 International Conference on Robotics and Automation (ICRA)*, pp. 10549–10556. IEEE, 2022.

Jacob Sacks and Byron Boots. Learning sampling distributions for model predictive control. In *Conference on Robot Learning*, pp. 1733–1742. PMLR, 2023.

Jacob Sacks, Rwik Rana, Kevin Huang, Alex Spitzer, Guanya Shi, and Byron Boots. Deep model predictive optimization. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 16945–16953. IEEE, 2024.

Rajiv Sambharya, Georgina Hall, Brandon Amos, and Bartolomeo Stellato. End-to-end learning to warm-start for real-time quadratic optimization. In *Learning for dynamics and control conference*, pp. 220–234. PMLR, 2023.

Augustinos D Saravanos, Oswin So, H M Sabbir Ahmad, and Chuchu Fan. Learning-to-optimize via deep unfolded flows. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=ZOtOq7hxJP.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033. IEEE, 2012.

Alexander Tong, Kilian FATRAS, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Guy Wolf, and Yoshua Bengio. Improving and generalizing flow-based generative models with minibatch optimal transport. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=CD9Snc73AW. Expert Certification.

Masatoshi Uehara, Yulai Zhao, Ehsan Hajiramezanali, Gabriele Scalia, Gokcen Eraslan, Avantika Lal, Sergey Levine, and Tommaso Biancalani. Bridging model-based optimization and generative modeling via conservative fine-tuning of diffusion models. *Advances in Neural Information Processing Systems*, 37:127511–127535, 2024.

Grady Williams, Andrew Aldrich, and Evangelos A. Theodorou. Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, 40 (2):344–357, 2017. doi: 10.2514/1.G001921.

Wei Xiao, Johnson Wang, Chuang Gan, Ramin Hasani, Mathias Lechner, and Daniela Rus. Safediffuser: Safe planning with diffusion probabilistic models. In *International Conference on Learning Representations*, volume 2025, pp. 100391–100412, 2025.

Haoru Xue, Chaoyi Pan, Zeji Yi, Guannan Qu, and Guanya Shi. Full-order sampling-based mpc for torque-level locomotion control via diffusion-style annealing. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4974–4981. IEEE, 2025.

Jichen Zhang, Liqun Zhao, Antonis Papachristodoulou, and Jack Umenberger. Constrained diffusers for safe planning and control. *Advances in Neural Information Processing Systems*, 38:34965–34998, 2026.

Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. Dino-wm: World models on pre-trained visual features enable zero-shot planning. *arXiv preprint arXiv:2411.04983*, 2024.

Guangyao Zhou, Sivaramakrishnan Swaminathan, Rajkumar Vasudeva Raju, J Swaroop Guntupalli, Wolfgang Lehrach, Joseph Ortiz, Antoine Dedieu, Miguel Lazaro-Gredilla, and Kevin Patrick Murphy. Diffusion model predictive control. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=pvtgffHtJm.

A Background on Flow Matching

Flow-based generative models (Chen et al., 2018; Lipman et al., 2023) transport a base distribution p_0 to a target p_1 on $\mathbb{R}^D$. The transport is defined by a learnable velocity field $v_\theta : \mathbb{R}^D \times [0,1] \to \mathbb{R}^D$ which induces the ordinary differential equation (ODE):

$$\dot{\boldsymbol{x}}(\tau) = v_\theta\big(\boldsymbol{x}(\tau), \tau\big), \qquad \boldsymbol{x}(0) \sim p_0, \tag{15}$$

whose time-one map pushes p_0 forward to p_1. Flow matching (Lipman et al., 2023; Liu et al., 2023; Albergo & Vanden-Eijnden, 2023) trains v_θ through direct regression, without integrating (15) during training. Sampling a coupling $(\boldsymbol{x}_0, \boldsymbol{x}_1) \sim \pi$ with marginals p_0 and p_1 (e.g., the independent one $\pi = p_0 \otimes p_1$) and interpolating linearly, $\boldsymbol{x}_\tau = (1-\tau)\,\boldsymbol{x}_0 + \tau\,\boldsymbol{x}_1$, the conditional flow matching loss

$$\mathcal{L}_{\mathrm{CFM}}(\theta) = \mathbb{E}_{\tau\sim\mathrm{Unif}[0,1],\,(\boldsymbol{x}_0,\boldsymbol{x}_1)\sim\pi}\Big[\big\|v_\theta(\boldsymbol{x}_\tau,\tau) - (\boldsymbol{x}_1 - \boldsymbol{x}_0)\big\|^2\Big] \tag{16}$$

is minimized by $v^\star(\boldsymbol{x},\tau) = \mathbb{E}\big[\boldsymbol{x}_1 - \boldsymbol{x}_0 \mid \boldsymbol{x}_\tau = \boldsymbol{x}\big]$, the velocity field that generates the marginals p_τ of $\boldsymbol{x}_\tau$ and hence transports p_0 to p_1 for any such π (Tong et al., 2024). Inference then amounts to integrating (15), typically with a few Euler steps. Conditional generation is obtained by providing context c to the velocity field, $v_\theta(\boldsymbol{x},\tau,c)$, trained on triplets $(\boldsymbol{x}_0,\boldsymbol{x}_1,c)$ drawn jointly.

B Architecture Details

B.1 Rollout and context construction

Given the measured state $\hat{\boldsymbol{s}}_0 \in \mathbb{R}^{n_s}$ and the incoming population $\mathcal{A}^k = \{\bar{\boldsymbol{a}}_j^k\}_{j=1}^P$, all members are rolled out in parallel under the dynamics and scored under the cost,

$$\boldsymbol{s}_{j,0} = \hat{\boldsymbol{s}}_0, \qquad \boldsymbol{s}_{j,t+1} = f(\boldsymbol{s}_{j,t},\, \boldsymbol{a}_{j,t}^k), \qquad J_j^k = \sum_{t=0}^{H-1} \ell(\boldsymbol{s}_{j,t},\, \boldsymbol{a}_{j,t}^k) + \phi(\boldsymbol{s}_{j,H}), \tag{17}$$

for $t = 0, \dots, H-1$. The context of iteration k consists of the normalized rank and standardized cost of every member,

$$\mathcal{C}^k = \big\{(\rho_j^k,\, \tilde{J}_j^k)\big\}_{j=1}^P \in [0,1]^P \times \mathbb{R}^P, \tag{18}$$

as defined in Appendix B.2. It is computed once per outer iteration, at the cost of P parallel rollouts of H steps each, and held fixed for the flow integration of that iteration.

B.2 Normalization

All cost quantities enter the network only through statistics that are invariant to positive affine rescaling of the cost. The member-level total-cost channel and rank channel are

$$\tilde{J}_j = \frac{J_j - \mu^J}{\sigma^J + \varepsilon}, \qquad \rho_j = \frac{\mathrm{rank}(J_j) - 1}{P - 1} \in [0,1], \tag{19}$$

with μ^J, σ^J computed across members and $\mathrm{rank}(J_j) \in \{1, \dots, P\}$ the rank of J_j within the population (ascending).

The actions are normalized in the same spirit, relative to the evaluated population $\boldsymbol{x}^k$ at $\tau = 0$. Let

$$\bar{\boldsymbol{x}}^k = \frac{1}{P}\sum_{j=1}^P \boldsymbol{x}_j^k, \qquad s^k = \sqrt{\frac{1}{PHn_a}\sum_{j=1}^P \big\|\boldsymbol{x}_j^k - \bar{\boldsymbol{x}}^k\big\|^2} + \varepsilon, \tag{20}$$

denote the population mean and the root-mean-square deviation of its entries, both computed once per outer iteration and held fixed during the flow. Every member, evaluated or transported, is presented to the network in standardized coordinates $\hat{\boldsymbol{x}}_j = (\boldsymbol{x}_j - \bar{\boldsymbol{x}}^k)/s^k$, and the network output g_θ is rescaled to actions units, $v_\theta = s^k g_\theta$. Since centering discards the absolute location of the population, the scaled mean $\bar{\boldsymbol{x}}^k/s^k$ is appended to the global conditioning vector. As a result, $v_\theta(\alpha\boldsymbol{x}, \tau; \alpha\boldsymbol{x}^k, \mathcal{C}^k) = \alpha\, v_\theta(\boldsymbol{x}, \tau; \boldsymbol{x}^k, \mathcal{C}^k)$ for $\alpha > 0$, so displacements are measured in units of the population's own spread.

B.3 Attention Mechanism

The network has a single cross-attention layer, which is the only place members communicate. Let $c_j = (\rho_j, \tilde{J}_j) \in \mathbb{R}^2$ denote the context of member j and let $W_x : \mathbb{R}^{Hn_a} \to \mathbb{R}^d$ and $W_c : \mathbb{R}^2 \to \mathbb{R}^d$ be linear embeddings, with $d = 64$. The transported member i at flow position $\hat{\boldsymbol{x}}_i(\tau)$ forms a query token, and every evaluated member j of the population forms a key/value token,

$$q_i = \mathrm{LN}_1\big(W_q\,(W_x\hat{\boldsymbol{x}}_i(\tau) + W_c c_i)\big), \qquad k_j = \mathrm{LN}_1\big([\,W_x\hat{\boldsymbol{x}}^k_j\,;\; W_c c_j\,]\big), \qquad q_i, k_j \in \mathbb{R}^{2d}, \quad (21)$$

where $[\,\cdot\,;\,\cdot\,]$ is concatenation and $W_q : \mathbb{R}^d \to \mathbb{R}^{2d}$. Under the aligned coupling, transported member i carries the context c_i of its own evaluated copy. Multi-head attention with $n_h = 4$ heads of width $d_h = 2d/n_h$ then computes

$$\alpha^h_{ij} = \mathrm{softmax}_j\left(\frac{\langle W^h_Q q_i,\, W^h_K k_j\rangle}{\sqrt{d_h}}\right), \qquad o_i = W_O\Big[\textstyle\sum_{j=1}^P \alpha^h_{ij}\, W^h_V k_j\Big]_{h=1}^{n_h}, \qquad a_i = \mathrm{LN}_2(q_i{+}o_i). \quad (22)$$

The output o_i is a convex combination of the population tokens with weights depending on the positions and costs of both i and j; the residual retains member i's own token, so the head below can express a displacement *from* $\hat{\boldsymbol{x}}_i$. Since no positional encoding is used and the statistics of Appendix B.2 are symmetric in j, the layer is permutation-equivariant over members.

B.4 Velocity head

The attended token is concatenated with the flow-time embedding and the scaled population mean,

$$h_i = \big[\,a_i\,;\; W_\tau\tau + b_\tau\,;\; W_\mu\,\bar{\boldsymbol{x}}^k/s^k\,\big] \in \mathbb{R}^{4d}, \quad (23)$$

and passed through a member-wise MLP,

$$g_\theta(\hat{\boldsymbol{x}}_i, \tau) = \mathrm{MLP}(h_i) \in \mathbb{R}^{Hn_a}, \quad (24)$$

$$\mathrm{MLP} = \mathrm{Lin}(4d, 256) \to \mathrm{LN} \to \mathrm{ELU} \to [\mathrm{Lin}(256, 256) \to \mathrm{ELU}] \times 2 \to \mathrm{Lin}(256, Hn_a). \quad (25)$$

The velocity in action units is $v_\theta = s^k g_\theta$, as in Appendix B.2. The head acts on each member independently; all cross-member information enters through a_i. In summary, the field computes

$$v_\theta(\boldsymbol{x}_i, \tau) = s^k\, g_\theta\Big(\hat{\boldsymbol{x}}_i,\; c_i,\; \textstyle\sum_j \alpha_{ij}\, V(\hat{\boldsymbol{x}}^k_j, c_j),\; \tau,\; \bar{\boldsymbol{x}}^k/s^k\Big), \quad (26)$$

i.e. a member's own position together with a cost-weighted summary of the population, the same information a CEM or MPPI update consumes, with the weighting and the step size learned.

C Connections to Classical Sampling-Based Optimization

C.1 Proof of Proposition 1

We provide an explicit construction showing how the attention block over the population recovers the population-mean updates of MPPI and CEM. Throughout, each candidate $\bar{\mathbf{a}}_i \in \mathbb{R}^{H\times n_a}$ is identified with its vectorization $\mathbf{x}_i = \mathrm{vec}(\bar{\mathbf{a}}_i) \in \mathbb{R}^{Hn_a}$, so attention acts on entire action sequences; we keep the notation $\bar{\mathbf{a}}_i$ for readability.

Let

$$\boldsymbol{\mu} = \frac{1}{P}\sum_{i=1}^P \bar{\mathbf{a}}_i \qquad \text{and} \qquad \boldsymbol{\mu}^w = \sum_{i=1}^P w_i \bar{\mathbf{a}}_i \quad (27)$$

denote the empirical and cost-weighted population means, respectively. We construct a two-head attention block whose first head returns $\boldsymbol{\mu}^w$ and whose second head returns $\boldsymbol{\mu}$. The output projection then produces their difference, $\boldsymbol{\mu}^w - \boldsymbol{\mu}$, which serves as the velocity applied to every population member.

MPPI as an attention pattern. Every token $\mathbf{h}_i$ associated with member i carries its standardized cost $\tilde{J}_i = (J_i - \mu^J)/\sigma^J$. We may therefore choose a constant query and a cost-dependent key such that

$$\mathbf{q}_j = \sqrt{d_h}\,\mathbf{e}_1, \qquad \mathbf{k}_i = -\frac{\sigma^J}{\lambda}\,\tilde{J}_i\,\mathbf{e}_1, \tag{28}$$

where $\mathbf{e}_1$ is the first canonical basis vector and d_h is the head dimension. The corresponding scaled dot-product logit is

$$r_{ji} = \frac{\mathbf{q}_j^\top \mathbf{k}_i}{\sqrt{d_h}} = -\frac{\sigma^J \tilde{J}_i}{\lambda} = -\frac{J_i}{\lambda} + \frac{\mu^J}{\lambda}, \tag{29}$$

which is independent of the querying member j. Since the constant μ^J/λ is common to all i, it cancels in the softmax, and the resulting attention weights are

$$\alpha_{ji} = \frac{\exp(-J_i/\lambda)}{\sum_{i'=1}^{P} \exp(-J_{i'}/\lambda)} = w_i^{\mathrm{MPPI}} \qquad \text{for every } j. \tag{30}$$

Choosing the value projection such that $\mathbf{W}_V \mathbf{h}_i = \bar{\mathbf{a}}_i$ (possible whenever the action embedding in the token is injective) gives

$$\mathbf{o}_j^w = \sum_{i=1}^{P} \alpha_{ji} \bar{\mathbf{a}}_i = \sum_{i=1}^{P} w_i^{\mathrm{MPPI}} \bar{\mathbf{a}}_i = \boldsymbol{\mu}^{\mathrm{MPPI}}, \tag{31}$$

which is identical for every querying member. Moreover, since the MPPI samples satisfy $\bar{\mathbf{a}}_i = \bar{\mathbf{a}}^{\mathrm{nom}} + \boldsymbol{\epsilon}_i$ and the weights sum to one,

$$\boldsymbol{\mu}^{\mathrm{MPPI}} = \sum_{i=1}^{P} w_i^{\mathrm{MPPI}} \left(\bar{\mathbf{a}}^{\mathrm{nom}} + \boldsymbol{\epsilon}_i\right) = \bar{\mathbf{a}}^{\mathrm{nom}} + \sum_{i=1}^{P} w_i^{\mathrm{MPPI}} \boldsymbol{\epsilon}_i, \tag{32}$$

which is the MPPI update of the nominal sequence.

CEM as an attention pattern. Let $\mathcal{E}$ denote the elite set selected by CEM and let $K = |\mathcal{E}|$. Membership in $\mathcal{E}$ is a threshold on the normalized rank carried by the tokens, $i \in \mathcal{E}$ if and only if $\rho_i \le (K-1)/(P-1)$. Conditioned on this set, choose query-independent logits $r_{ji} = 0$ for $i \in \mathcal{E}$ and $r_{ji} = -\beta$ otherwise. The resulting attention weights are

$$\alpha_i^{(\beta)} = \begin{cases} \dfrac{1}{K + (P-K)e^{-\beta}}, & i \in \mathcal{E}, \\ \dfrac{e^{-\beta}}{K + (P-K)e^{-\beta}}, & i \notin \mathcal{E}. \end{cases} \tag{33}$$

Therefore,

$$\alpha_i^{(\beta)} \xrightarrow[\beta\to\infty]{} \frac{\mathbf{1}\{i \in \mathcal{E}\}}{K} = w_i^{\mathrm{CEM}}. \tag{34}$$

The approximation error at a finite logit scale is

$$\left\| \boldsymbol{\alpha}^{(\beta)} - \mathbf{w}^{\mathrm{CEM}} \right\|_1 = \frac{2(P-K)e^{-\beta}}{K + (P-K)e^{-\beta}} \le \frac{2(P-K)}{K} e^{-\beta}, \tag{35}$$

which vanishes exponentially as $\beta \to \infty$. Using the same value projection as above, the attention output consequently converges to

$$\mathbf{o}_j^w = \sum_{i=1}^{P} \alpha_i^{(\beta)} \bar{\mathbf{a}}_i \longrightarrow \frac{1}{K} \sum_{i \in \mathcal{E}} \bar{\mathbf{a}}_i = \boldsymbol{\mu}^{\mathrm{CEM}}, \tag{36}$$

the elite mean used by CEM.

Realizing the update as a flow step. It remains to convert the weighted average produced by attention into the population translation stated in Proposition 1. For the second attention head, set all query and key projections to zero. All logits are then equal, yielding uniform attention weights $1/P$. Using the same value projection, this head returns

$$\mathbf{o}_j^u = \frac{1}{P}\sum_{i=1}^{P} \bar{\mathbf{a}}_i = \boldsymbol{\mu} \qquad \text{for every } j. \tag{37}$$

Choose the multi-head output projection to subtract the uniform head from the weighted head. The resulting velocity is

$$\mathbf{v}_j = \mathbf{o}_j^w - \mathbf{o}_j^u = \boldsymbol{\mu}^w - \boldsymbol{\mu}, \tag{38}$$

which is shared by all population members. The residual connection and per-member decoder ψ_θ are taken as pass-through (identity on the attended output), which is a particular choice of their parameters.

Two remarks are in order. First, the queries in (28) are constant and the keys and values are built from the evaluated population only, so $\mathbf{v}_j$ depends neither on the flow time τ nor on the current position $\mathbf{x}(\tau)$ of the transported member. The Euler integration is therefore exact and returns $\mathbf{x}(0) + \mathbf{v}$ for any number of steps N_τ, so a unit step suffices. Second, the network operates in the standardized coordinates $\hat{\mathbf{x}}_i = (\mathbf{x}_i - \bar{\mathbf{x}}^k)/s^k$ and rescales its output by s^k. Since both terms in (38) are convex combinations with weights summing to one,

$$s^k\Big(\sum_{i=1}^{P} w_i \hat{\mathbf{x}}_i - \frac{1}{P}\sum_{i=1}^{P} \hat{\mathbf{x}}_i\Big) = \sum_{i=1}^{P} w_i \mathbf{x}_i - \frac{1}{P}\sum_{i=1}^{P} \mathbf{x}_i, \tag{39}$$

so the same construction in standardized coordinates yields exactly (38) in action units.

Initializing the flow state at $\mathbf{z}_j(0) = \bar{\mathbf{a}}_j$ and taking a single Euler step of unit length gives

$$\mathbf{z}_j(1) = \bar{\mathbf{a}}_j + \mathbf{v}_j = \bar{\mathbf{a}}_j + \left(\sum_{i=1}^{P} w_i \bar{\mathbf{a}}_i - \frac{1}{P}\sum_{i=1}^{P} \bar{\mathbf{a}}_i\right), \tag{40}$$

which is exactly the update in Proposition 1. Averaging over the updated population yields

$$\frac{1}{P}\sum_{j=1}^{P} \mathbf{z}_j(1) = \frac{1}{P}\sum_{j=1}^{P} \bar{\mathbf{a}}_j + \boldsymbol{\mu}^w - \boldsymbol{\mu} \tag{41}$$

$$= \boldsymbol{\mu}^w = \sum_{i=1}^{P} w_i \bar{\mathbf{a}}_i. \tag{42}$$

Thus, the empirical population mean after the flow step equals the MPPI weighted mean exactly. Under the CEM attention pattern, it converges to the CEM elite mean as $\beta \to \infty$. This proves Proposition 1. □

D Experiment Details

D.1 Environments and Tasks

D.1.1 CarNav environment

CarNav is a custom planar navigation task with a car-like vehicle, circular obstacles, and a goal that moves during the episode. It is designed so that a planner must handle a nonholonomic constraint (no turning in place, bounded turning radius), obstacle avoidance with a finite body, and repeated re-targeting that invalidates the warm start.

Dynamics. The vehicle is a kinematic bicycle with the rear axle as reference point. The state is $s = (p_x, p_y, \theta, v)$ (position, heading, forward speed) and the action is $a = (\alpha, \delta)$ (longitudinal acceleration, steering angle):

$$\dot{p}_x = v\cos\theta, \qquad \dot{p}_y = v\sin\theta, \qquad \dot{\theta} = \tfrac{v}{L}\tan\delta, \qquad \dot{v} = \alpha, \tag{43}$$

with wheelbase $L = 1\,\mathrm{m}$, integrated by forward Euler at $\Delta t = 0.05\,\mathrm{s}$. The steering angle is clamped to $|\delta| \le \delta_{\max} = 0.6\,\mathrm{rad}$ inside the dynamics, so the turning radius is bounded below by $L/\tan\delta_{\max} \approx 1.5\,\mathrm{m}$ and the heading cannot change at zero speed. The acceleration is unbounded and is regularized by the cost. The vehicle occupies a disk of radius $r_{\mathrm{car}} = 0.5\,\mathrm{m}$ around its reference point.

Cost. For a plan of horizon H with states $s_{0:H}$ and actions $a_{0:H-1}$, and a goal $g = (g_x, g_y, g_\theta)$, the objective is

$$\begin{aligned} J(s_{0:H}, a_{0:H-1}) = w_{\mathrm{run}} \sum_{t=0}^{H-1} \|p_t - g_p\|^2 + \sum_{t=0}^{H-1} \left(w_\alpha\, \alpha_t^2 + w_\delta\, \delta_t^2\right) \\ + w_{\mathrm{term}} \Big(\|p_H - g_p\|^2 + w_\theta\, \mathrm{wrap}(\theta_H - g_\theta)^2 + w_v\, v_H^2 \Big) \\ + w_{\mathrm{obs}} \sum_{t=0}^{H} \sum_{i} \max\big(0,\ \rho_i - \|p_t - c_i\|\big)^2, \end{aligned} \tag{44}$$

where $\mathrm{wrap}(\cdot)$ maps an angle difference to $(-\pi, \pi]$, and $\rho_i = r_i + r_{\mathrm{car}} + m$ is the exclusion radius of obstacle i with centre c_i, physical radius r_i, and safety margin $m = 0.1\,\mathrm{m}$ (a Minkowski inflation of the obstacle by the vehicle body). The obstacle term is zero outside the exclusion disk and grows quadratically inside it. The weights are $w_{\mathrm{run}} = 1$, $w_\alpha = 0.5$, $w_\delta = 2$, $w_{\mathrm{term}} = 5$, $w_\theta = 1$, $w_v = 1$, and $w_{\mathrm{obs}} = 5000$. The terminal term asks the vehicle to arrive at the goal position with the goal heading and zero speed.

Problem generation. Each problem instance is drawn as follows. The start position is uniform in $[-4, 4]^2\,\mathrm{m}$, the start heading uniform in $[-\pi, \pi]$, and the start speed uniform in $[-0.5, 0.5]\,\mathrm{m/s}$. The first goal position is uniform in $[-6, 6]^2\,\mathrm{m}$, redrawn until it is at least $6.5\,\mathrm{m}$ from the start so that the corridor between them can hold the obstacles; the goal heading is uniform in $[-\pi, \pi]$, independent of the bearing. Exactly two circular obstacles with radii uniform in $[0.3, 0.8]\,\mathrm{m}$ are placed along the start-to-goal segment with a lateral offset uniform in $[-0.5, 0.5]\,\mathrm{m}$, so they sit close to the straight-line path. Obstacle centres are kept at least $2.4\,\mathrm{m}$ (largest radius, margin, body radius, and $1\,\mathrm{m}$ of turning room) from both the start and the goal, and non-overlapping with each other. The obstacles are fully known to every planner through the cost.

Moving goal. An episode lasts 300 closed-loop steps ($15\,\mathrm{s}$). The goal jumps to a new draw every 100 steps ($5\,\mathrm{s}$), giving three goals per episode: the first goal above and two further goals, each uniform in $[-6, 6]^2\,\mathrm{m}$ with a uniform random heading, rejected if its position lies within the exclusion radius plus $1\,\mathrm{m}$ of turning room of any obstacle. The obstacles stay fixed for the whole episode. Because every jump makes the warm-started plan wrong, the planner has to recover from an uninformative initialization three times per episode.

Closed-loop protocol and metric. All planners run in receding horizon with $H = 60$ ($3\,\mathrm{s}$ of look-ahead), re-planning at every step and warm-starting from the previously executed plan shifted by one step. The executed action is the planner's nominal (the population mean for the sampling-based methods). The reported closed-loop cost is the sum over the three goal segments of the cost above evaluated on the executed trajectory of that segment under its own goal, so each segment pays its running, action, and obstacle terms at every step and its terminal terms at the step of the next goal switch (or the episode end). The evaluation set consists of 100 held-out instances drawn with a fixed seed and shared by all methods.

D.1.2 HALFCHEETAH ENVIRONMENT

HalfCheetah is the standard planar MuJoCo running task. We use the Gym half-cheetah model unchanged (the same bodies, joint limits, stiffness, damping, and actuator gears) and simulate it with MuJoCo Warp for GPU-batched rollouts, so every candidate action sequence of a planner is evaluated on the true simulator rather than on a learned model. The task is included because it has no goal and no obstacles: the objective is a running gait, and the difficulty lies in the contact dynamics and the coordination of six actuators.

Dynamics. The state is $s = (q, \dot{q}) \in \mathbb{R}^{18}$ with $q = (p_x, p_z, \phi, q_{1:6})$: the horizontal and vertical position of the torso, its pitch angle, and the six leg joint angles (back thigh, shin, foot; front thigh, shin, foot). The action $a \in [-1, 1]^6$ is the normalized motor command of the six joints, scaled by the model's gear ratios into torques; commands outside $[-1, 1]$ are clipped before simulation. The planner acts at $\Delta t = 0.05\,\mathrm{s}$ with two semi-implicit Euler substeps of $0.025\,\mathrm{s}$ each, and the action is held constant within a step.

Cost. For a plan of horizon H with post-step states $s_{1:H}$ and actions $a_{0:H-1}$, the objective is

$$J(s_{1:H}, a_{0:H-1}) = \frac{1}{H}\sum_{t=1}^{H}\Big(- w_{\mathrm{fwd}}\,\dot{p}_{x,t} + w_{\mathrm{alive}}\max(0,\ z_{\min} - p_{z,t})^2 + w_{\mathrm{pitch}}\,\phi_t^2\Big) + \frac{w_a}{H}\sum_{t=0}^{H-1}\tfrac{1}{6}\|a_t\|^2, \tag{45}$$

with $w_{\mathrm{fwd}} = 1$, $w_{\mathrm{alive}} = 5$, $z_{\min} = -0.3\,\mathrm{m}$ (relative to the torso's rest height), $w_{\mathrm{pitch}} = 3$, and $w_a = 0.1$. The first term rewards forward speed, the second penalizes the torso dropping below a height floor, the third penalizes flipping, and the last penalizes effort. All terms are averaged over the horizon, so the cost has the units of a per-step quantity and is comparable across horizons and episode lengths. It is negative whenever the cheetah runs forward, and we report $R = -J$ as a return (higher is better).

Problem generation. A problem instance is an initial state. Each is drawn by perturbing the model's default standing pose with independent Gaussian noise: standard deviation $0.05\,\mathrm{m}$ on the torso height, $0.2\,\mathrm{rad}$ on the pitch, $0.1\,\mathrm{rad}$ on each of the six joint angles, and 0.5 on every generalized velocity. The horizontal position is set to zero. This gives a distribution of off-balance starts from which the planner has to recover and accelerate into a gait rather than continue an existing one. The held-out evaluation set consists of 64 such starts drawn with a fixed seed and shared by all methods.

Closed-loop protocol and metric. All planners run in receding horizon with $H = 20$ (1 s of look-ahead), re-planning at every step and warm-starting from the previously executed plan shifted by one step. The executed action is the planner's nominal (the population mean for the sampling-based methods). An episode lasts 100 closed-loop steps ($5\,\mathrm{s}$). The reported closed-loop return is $R = -J$ of the executed 100-step trajectory under the cost above, i.e., the average forward speed over the episode minus the average height, pitch, and action penalties.

D.1.3 PUSH-T ENVIRONMENT

Push-T is the planar non-prehensile manipulation task initially introduced in (Chi et al., 2025). A point pusher must push a T-shaped block across a floor to a fixed goal pose. It is included because the contact between the pusher and the block is intermittent and unilateral: progress requires the pusher to move around the block and push from different sides, which makes the cost landscape multi-modal and poorly served by a single-mode Gaussian planner.

Dynamics. The scene is simulated in MuJoCo (via MuJoCo Warp for GPU-batched rollouts) with a $0.01\,\mathrm{s}$ timestep and a sliding friction coefficient of 1.0 on every surface. The T-block is the union of a $2 \times 8\,\mathrm{cm}$ bar and a $6 \times 2\,\mathrm{cm}$ crossbar, resting on the floor with three planar degrees of freedom (two translations and a yaw). The pusher is a sphere of radius $1\,\mathrm{cm}$ on two planar slide joints driven by velocity servos. The state is $s = (b_x, b_y, b_\theta, q_x, q_y, \dot{b}_x, \dot{b}_y, \dot{b}_\theta, \dot{q}_x, \dot{q}_y) \in \mathbb{R}^{10}$, the block pose and pusher position with their rates. The action $a = (v_x, v_y)$ is the commanded pusher velocity, clipped to $[-v_{\max}, v_{\max}]^2$ with $v_{\max} = 0.5\,\mathrm{m/s}$. The planner's action knots are spaced $\Delta t = 0.1\,\mathrm{s}$ apart (10 simulator substeps), with the command held constant within a knot, so a single knot moves the pusher by at most $5\,\mathrm{cm}$.

Cost. The goal is the block pose $g = (0, 0, 0)$ at the origin with zero yaw. With the per-state cost

$$c(s) = w_{\mathrm{pos}}\|b - g_p\|^2 + w_{\mathrm{ang}}\,\mathrm{wrap}(b_\theta - g_\theta)^2 + w_{\mathrm{close}}\|b - q\|^2, \tag{46}$$

where $b = (b_x, b_y)$ is the block position, q the pusher's world position, and $\mathrm{wrap}(\cdot)$ maps an angle difference to $(-\pi, \pi]$, the planning objective over a horizon of H knots is

$$J(s_{0:H}, a_{0:H-1}) = \Delta t \sum_{t=0}^{H-1} c(s_t) + c(s_H), \tag{47}$$

with $w_{\mathrm{pos}} = 1$, $w_{\mathrm{ang}} = 1$, and $w_{\mathrm{close}} = 0.01$. The first two terms drive the block to the goal pose, and the small third term keeps the pusher near the block so that it can act on it. There is no action penalty.

Problem generation. A problem instance is an initial state, drawn from the reset distribution of Generative Predictive Control: the block's position offset from the goal is uniform in $[-0.2, 0.2]^2\,\mathrm{m}$ and its yaw uniform in $[-\pi, \pi]$; the pusher's offset from its home position, $10\,\mathrm{cm}$ in front of the goal, is uniform in $[-0.2, 0.2]^2\,\mathrm{m}$; all velocities are zero. The pusher may start in contact with, or inside, the block. The evaluation set consists of 3 seeds of 64 starts each, 192 instances in total, shared by all methods.

Closed-loop protocol and metric. All planners run in receding horizon with $H = 20$ knots ($2\,\mathrm{s}$ of look-ahead), re-planning at every knot and warm-starting from the previous plan shifted by one knot. An episode lasts 100 closed-loop steps ($10\,\mathrm{s}$). Because the population mean of a sampling-based planner averages over incompatible pushing strategies and typically fixes the block's angle without moving it, every sampler executes the cheapest sample of its final population rather than its mean (predictive sampling with the sampler's proposal). An episode is a success if, at its final step, the block position is within $2\,\mathrm{cm}$ of the goal and its yaw within $0.1\,\mathrm{rad}$ ($5.7°$); we also report the time to success, the first step from which this tolerance holds until the end of the episode.

D.1.4 WALKER ENVIRONMENT

Walker is the standard MuJoCo planar biped system. The robot has to get up from a random, often fallen, configuration and walk at a target speed while keeping its torso at a target height and upright. It is included because the cost describes no gait: a walking gait has to emerge from the planner as the cheapest way to hold the speed and the height at once with bounded torques, and the short planning horizon of the original setup makes this a hard receding-horizon problem.

Dynamics. The scene is simulated in MuJoCo (via MuJoCo Warp for GPU-batched rollouts) with a $0.01\,\mathrm{s}$ timestep. The robot is a torso on a planar root (vertical and horizontal slides and a pitch hinge) with two legs of hip, knee, and ankle hinges, capsule geometry, and a sliding friction coefficient of 0.5 against the floor. The joint ranges are $[-20°, 100°]$ for the hips, $[-150°, 0°]$ for the knees, and $[-45°, 45°]$ for the ankles. The state is $s = (q, \dot{q}) \in \mathbb{R}^{18}$ with $q = (z, p_x, \phi, q_{1:6})$: the torso's vertical and horizontal displacement, its pitch, and the six leg joint angles. The action $a \in [-1, 1]^6$ is the normalized torque command of the six joints, scaled by gear ratios of 100, 50, and 20 at the hip, knee, and ankle. The planner's action knots are spaced $\Delta t = 0.15\,\mathrm{s}$ apart (15 simulator substeps) with the command held constant within a knot.

Cost. With the torso height $h = 1.3\,\mathrm{m} + z$ and the horizontal velocity v_{com} of the whole-body centre of mass, the per-state cost is

$$\phi(s) = w_h\,(h - h^*)^2 + w_{\mathrm{up}}\,(\cos\phi - 1)^2 + w_v\,(v_{\mathrm{com}} - v^*)^2, \tag{48}$$

and the planning objective over a horizon of H knots is

$$J(s_{0:H}, a_{0:H-1}) = \Delta t \sum_{t=0}^{H-1} \big(\phi(s_t) + w_a \|a_t\|^2\big) + \phi(s_H), \tag{49}$$

with the targets $h^* = 1.2\,\mathrm{m}$ and $v^* = 1.5\,\mathrm{m/s}$ and the weights $w_h = 10$, $w_{\mathrm{up}} = 3$, $w_v = 1$, and $w_a = 0.1$, as in `hydrax`. The upright term is the vertical component of the torso's axis minus one, which is zero when the torso is vertical and largest when it is horizontal. The centre-of-mass velocity is computed from the state in closed form and matches MuJoCo's subtree velocity sensor.

Problem generation. A problem instance is an initial state, drawn from the reset distribution of Generative Predictive Control: the root slides are zero (torso at its home height), the pitch is uniform in $[-1.5, 1.5]\,\mathrm{rad}$, each leg joint is uniform within its range, and every generalized velocity is uniform in $[-0.1, 0.1]$. Since a pitch of $1.5\,\mathrm{rad}$ is nearly horizontal and the legs are placed at random, many starts are fallen or falling, and the planner has to stand the robot up before it can walk. The held-out evaluation set consists of 64 such starts drawn with a fixed seed and shared by all methods.

Closed-loop protocol and metric. All planners run in receding horizon with $H = 4$ knots ($0.6\,\mathrm{s}$ of look-ahead) and re-plan at $50\,\mathrm{Hz}$, i.e., every two simulator steps, well inside a knot. As in `hydrax`, the knots are not shifted between re-plans: the planner is warm-started from its previous plan and re-optimizes it from the new state. The executed action is the planner's nominal (the population mean for the sampling-based methods). An episode lasts 250 closed-loop steps ($5\,\mathrm{s}$). The reported closed-loop cost is the mean running cost $\phi(s_t) + w_a\|a_t\|^2$ over the executed steps of the episode (cost per step, the metric of Generative Predictive Control), so it is independent of the episode length and lower is better.

D.1.5 CHEETAH-WM ENVIRONMENT (PLANNING IN A TD-MPC2 WORLD MODEL)

Cheetah-WM is the `cheetah-run` task of the DeepMind Control Suite, solved by planning inside the learned latent world model of a pretrained TD-MPC2 agent (Hansen et al., 2024) instead of the physics simulator. Every planner optimizes exactly the objective that TD-MPC2's own MPPI planner optimizes, so the comparison isolates the optimizer from the model. The task is included to show that the flow optimizer transfers to learned latent dynamics with a value-based objective, where the cost is a neural network rather than a hand-written function of a physical state.

World model. We use the single-task TD-MPC2 checkpoint for `cheetah-run` (the 5M-parameter model, latent dimension 512, five Q-heads), frozen, with the upstream sources unchanged. The encoder maps the 17-dimensional proprioceptive observation o_t to a latent $z_t \in \mathbb{R}^{512}$, and the latent dynamics $z_{t+1} = f(z_t, a_t)$ are deterministic. The action $a_t \in [-1, 1]^6$ is the environment's torque command; actions are clamped to the box inside the rollout and the cost, so every solver sees the bounded objective the model was trained on. The reward head $R(z, a)$ and the Q-ensemble $Q_k(z, a)$ are TD-MPC2's two-hot decoded heads, and $\pi(z)$ is its policy prior.

Cost. For a latent z_0 and an action sequence $a_{0:H-1}$, the planning objective is minus TD-MPC2's value estimate of the sequence,

$$J(a_{0:H-1};\, z_0) = -\Big[\sum_{t=0}^{H-1} \gamma^t R(z_t, a_t) + \gamma^H\, \bar{Q}\big(z_H, \pi_{\mathrm{mean}}(z_H)\big)\Big], \qquad z_{t+1} = f(z_t, a_t), \tag{50}$$

with the horizon $H = 3$ and discount γ of the checkpoint, and π_{mean} the mean of the policy prior at the terminal latent. $\bar{Q}$ is the mean over the five Q-heads. TD-MPC2's planner instead averages a random two-head subsample at every evaluation; we use the deterministic mean so that the objective is the same for every candidate and a best-so-far tracking planner cannot select evaluation noise. Termination heads are ignored, as the task is non-episodic.

Problem generation. A problem instance is an initial latent. For the closed-loop evaluation in the real environment, the instances are the initial observations of `dm_control` episodes: episode k of planner seed s is reset with seed $1000\,s + k$, so every method sees the same start states. We run 3 planner seeds of 10 episodes each, 30 paired instances in total. For model selection we also use an imagined closed loop, 64 latents from held-out TD-MPC2 episodes rolled out for 500 steps inside the frozen model; those numbers are used only to choose planner settings and are not reported.

Closed-loop protocol and metric. At every environment step the observation is encoded, the planner optimizes J over $H = 3$ actions from the current latent, and the first action is applied to the real environment. The next step is warm-started from the plan shifted by one action with a zero action appended. Because TD-MPC2 itself executes an elite sample rather than the population mean, the sampling-based planners execute the cheapest sample of their final population; DIAL-MPC executes its nominal, as its cheapest sample destabilizes the episode. An episode lasts 500

Task (metric)	MPPI	CEM	CMA-ES	DIAL-MPC	**FlowOpt Policy (Ours)**
CarNav ($J_{\text{norm}}\downarrow$)	1.46 ± 0.61	1.25 ± 0.51	1.26 ± 0.51	1.40 ± 0.58	$\mathbf{1.00 \pm 0.40}$
Cheetah ($R_{\text{norm}}\uparrow$)	0.66 ± 0.05	0.61 ± 0.05	0.71 ± 0.05	0.58 ± 0.04	$\mathbf{1.00 \pm 0.09}$
Walker ($R_{\text{norm}}\uparrow$)	0.86 ± 0.41	0.88 ± 0.49	0.87 ± 0.40	0.79 ± 0.42	$\mathbf{1.00 \pm 0.52}$
Push-T (SR %$\uparrow$)	60.4 ± 4.8	38.0 ± 2.4	54.7 ± 5.6	53.1 ± 4.1	$\mathbf{99.0 \pm 0.9}$
Cheetah-WM ($R_{\text{norm}}\uparrow$)	0.86 ± 0.06	0.88 ± 0.07	0.87 ± 0.04	0.79 ± 0.04	$\mathbf{1.00 \pm 0.09}$
Humanoid-WM ($R_{\text{norm}}\uparrow$)	0.39 ± 0.10	0.42 ± 0.11	0.35 ± 0.08	0.53 ± 0.16	$\mathbf{1.00 \pm 0.08}$

Table 1: Average performance metrics on six main tasks presented in Section 5.2.

decision steps. The reported metric is the undiscounted episode return of the real environment (the sum of the true task rewards, in $[0, 1000]$), which is the quantity TD-MPC2 is evaluated on, normalized by the mean return of the best method.

D.1.6 HUMANOID-WM ENVIRONMENT (PLANNING IN A TD-MPC2 WORLD MODEL)

Humanoid-WM is the `humanoid-walk` task of the DeepMind Control Suite, solved by planning inside the learned latent world model of a pretrained TD-MPC2 agent (Hansen et al., 2024), with the same construction as Cheetah-WM (Appendix D.1.5). It is the hardest of our tasks: the humanoid has 21 actuators and must keep its balance while walking, so a short-horizon planner that reads the value function poorly falls within a few steps and the episode return collapses. The gap between cheap and expensive sampling-based planners is correspondingly the largest of all tasks, which makes it the clearest test of whether the flow optimizer recovers the expensive planner's behaviour at a small budget.

World model. We use the single-task TD-MPC2 checkpoint for `humanoid-walk` (the 5M-parameter model, latent dimension 512, five Q-heads), frozen, with the upstream sources unchanged. The encoder maps the 67-dimensional proprioceptive observation to a latent $z_t \in \mathbb{R}^{512}$, and the latent dynamics $z_{t+1} = f(z_t, a_t)$ are deterministic. The action $a_t \in [-1, 1]^{21}$ is the environment's torque command, clamped to the box inside the rollout and the cost. The reward head, the Q-ensemble, and the policy prior are those of the checkpoint.

Cost. The planning objective is minus TD-MPC2's value estimate of the action sequence, exactly as in Appendix D.1.5: the discounted sum of the predicted rewards over $H = 3$ steps plus the discounted mean over the five Q-heads at the terminal latent, evaluated at the mean of the policy prior. The Q-aggregation is deterministic so that every candidate sees the same objective.

Problem generation. A problem instance is an initial latent. For the closed-loop evaluation in the real environment, the instances are the initial observations of `dm_control` episodes: episode k of planner seed s is reset with seed $1000\,s + k$, so every method sees the same start states. We run 3 planner seeds of 10 episodes each, 30 paired instances in total. For model selection we also use an imagined closed loop, 64 latents from 5 held-out TD-MPC2 episodes rolled out for 500 steps inside the frozen model; those numbers are used only to choose planner settings and are not reported.

Closed-loop protocol and metric. At every environment step the observation is encoded, the planner optimizes the objective over $H = 3$ actions from the current latent, and the first action is applied to the real environment. The next step is warm-started from the plan shifted by one action with a zero action appended. The sampling-based planners execute the cheapest sample of their final population, as TD-MPC2 executes an elite; DIAL-MPC executes its nominal. An episode lasts 500 decision steps. The reported metric is the undiscounted episode return of the real environment (the sum of the true task rewards, in $[0, 1000]$), normalized by the mean return of the best method.

D.2 PLANNING DETAILS

Receding-horizon loop. Every method plans a sequence of H control knots at each closed-loop step, executes the first knot, and is warm-started at the next step from the executed plan advanced

Table 2: CarNav: closed-loop cost (mean ± SE over 100 paired problems) on the training and shifted settings, and number of problems on which **FlowOpt Policy** is better.

Setting	FlowOpt Policy (Ours)	Flow Policy	Ours better
Training	6004 ± 238	6004 ± 208	52/100
Wheelbase 2	6693 ± 287	6669 ± 231	72/100
4 obstacles	7730 ± 312	9292 ± 268	85/100
8 obstacles	8689 ± 429	10585 ± 342	80/100
16 obstacles	9066 ± 430	10934 ± 356	81/100

Table 3: Push-T: mean stage cost (mean ± SE over 64 paired problems), number of problems on which **FlowOpt Policy** is better, and success rate on the training and shifted settings.

	Cost			Success (%)	
Setting	FlowOpt Policy (Ours)	Flow Policy	Ours better	Ours	Flow Policy
Training	0.280 ± 0.035	0.359 ± 0.045	63/64	100.0	59.4
Friction 0.5	0.348 ± 0.044	0.455 ± 0.059	59/64	92.2	51.6
Friction 2	0.242 ± 0.031	0.287 ± 0.036	56/64	100.0	43.8
Start range 0.4	0.447 ± 0.055	0.532 ± 0.062	59/64	96.9	39.1
$w_{\Delta u} = 3 \times 10^{-3}$	0.302 ± 0.038	0.375 ± 0.046	56/64	9.4	7.8

by one step. Across all tasks, we use $K = 3$ optimization iterations. The hyperparameters of the baselines are tuned through grid search on a separate validation dataset. In FlowOpt Policy, the velocity field is integrated using 5 Euler steps. In all tasks, we used population size $P = 30$, except for the Humanoid-WM one where we used $P = 150$.

D.3 Training Details

The velocity field takes the flattened plan $\boldsymbol{x}$, the flow time τ and the cost features $\mathcal{C}$. Each is embedded in 64 dimensions, the embeddings are concatenated and passed to an MLP with three hidden layers of 256 units, ELU activations and a LayerNorm after the first layer. The plan is standardised by the RMS spread of the batch of source samples before the network and the output is rescaled by it. We train using Adam, learning rate 0.001 and batch size 100.

D.4 Additional Results

Table 1 provides the average metrics across all tasks for every method accompanying Section 5.2. Tables 2 and 3 report the full generalization results of Section 5.3, comparing **FlowOpt Policy** with Flow Policy on the training distribution and on shifted settings that alter the environment (number of obstacles on CarNav), the dynamics (wheelbase on CarNav, friction on Push-T), the initial-state distribution (start range on Push-T), or the objective (an added control-rate penalty $w_{\Delta u}$ on Push-T).