Neuro-Inspired Inverse Learning for Planning and Control

Maryna Kapitonova* **Tonio Ball**
NeuroMentum AI; IMBIT, University of Freiburg, Germany
*Correspondence: maryna.kapitonova@neuromentum.ai

Abstract

We present a neuro-inspired framework for planning and control. Building on three principles that enable fast and highly effective goal-directed behavior in the mammalian brain — hierarchies of paired forward/inverse internal models, open-loop multi-step motor commands, and sequential organization of action — our *Inverter* framework combines analytic and trained components, the latter trained end-to-end through *inverse learning*, a paradigm we formalize and delineate from supervised, reinforcement, and imitation learning. Our approach combines RL-style amortized inference with optimal-control-style sequence reasoning, emitting a full action sequence in one forward pass. Single Inverters or hierarchical n=2 Inverter stacks exceed comparable offline-RL and diffusion-planner mean scores on all 3 `maze2d-v1` and 6 `antmaze-v2` D4RL variants by an average of $+25.0\%$ (range $+1.0\%$ to $+76.2\%$), at one-to-two orders of magnitude less inference compute. Across both benchmark families, we find complementary evidence for inverse learning beyond behavior cloning: policies closer to the analytic optimum than the data-generating policy in Maze2D, and target-specific deceleration with reduced overshoot in AntMaze. We identify forward-model hacking under narrow training-data coverage as a failure mode, which we show can be mitigated by using random training data with broader coverage. We also demonstrate an $n = 3$ hierarchy in which a Strategy Inverter learns to schedule lower-level control according to requested reliability, progress, and compute cost. We propose probabilistic, latent, and domain-general extensions of the Inverter framework as a neuro-grounded world-interface, especially for latency- and compute-critical embodied AI.

1 Introduction

What happens if we apply a planning and control framework whose principles are inspired by the functional organization of the mammalian brain to goal-directed, embodied control? This paper develops one such answer. Our starting point is considering the brain as an inversion machine.

The brain as an inversion machine. The dominant Bayesian-brain view has conceptualized the brain as an inference machine [1, 2]. We conceptualize the brain first and foremost as an *inversion* machine: goal-directed behavior poses the inverse problem "given a desired outcome, which actions realize it?" Solving this inverse problem minimizes the discrepancy between desired and actual outcome. The computational core is inversion of a model of how the world responds to action, which admits many implementations: iterative, or amortized (a learned direct inverse mapping in one feedforward pass); probabilistic or deterministic. We presume the brain flexibly uses whichever variant fits the task, timescale, and resources [3]. Here we focus on the amortized, deterministic, single-agent, and fully observable direct mapping variant, following the classical motor control concepts of Jordan & Rumelhart's distal-teacher framework [4] and Kawato's internal-model theory [5, 6].

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

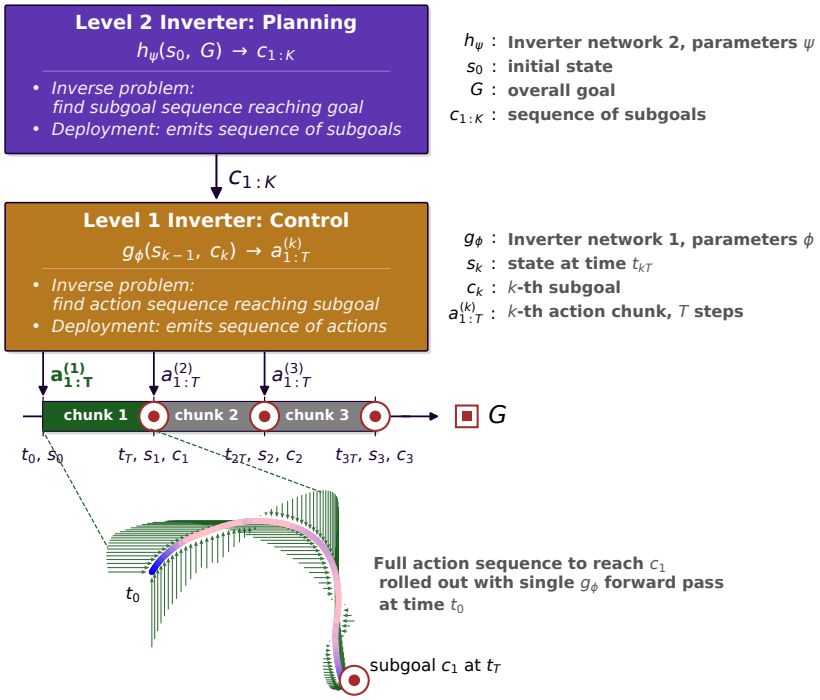


Figure 1: **The Inverter framework, shown for an $n=2$ stack. Top:** The Level 2 Inverter emits subgoal sequences; the Level 1 Inverter emits T-step action chunks conditioned on each chunk's initial state and subgoal. Neural Inverters at both levels are trained by inverse learning through their respective FoMs; each emits its sequence in one feedforward pass, without test-time optimization, autoregressive decoding, or iterative denoising. Sequential action chunks drive the agent toward G. **Bottom:** One chunk's actions (green arrows) produce goal-conditioned motion from s_0 toward c_1, with an emergent rather than predefined trajectory.

The combination of three essential principles organizes our approach:

- *Hierarchies of paired forward models (FoMs) and inverse models (IMs)*, with each FoM providing the instructive error signal for its paired IM[1] [7, 8].
- *Open-loop multi-step motor commands* ballistically executable in pre-planned chunks too fast for sensory correction [9].
- *Sequences of motor chunks*, segmenting behaviors into sequential sub-plans [10, 11].

The Inverter planning/control framework. We organize the three principles above into a planning/control framework (Fig. 1) consisting of a hierarchy of Inverters at $n=1,2,3,\ldots$ abstraction levels, whose neural implementations use inverse learning through a FoM and emit sequences in one feedforward pass; algorithmic components can also fill individual slots. This first principle reflects the hierarchical organization of mammalian motor control, with high-level decision/sequencing circuits issuing subgoals to lower-level motor loops [6, 10, 11]. Single-level Inverters instantiate the basic forward/inverse module; hierarchical variants compose these modules across abstraction levels. Table 1 organizes our implementations by level: Level 1 emits physical controls, Level 2 emits subgoals, and Level 3 schedules calls to the lower hierarchy. We evaluate both two learned levels (AntMan) and a three-level stack with learned control and strategy around a frozen breadth-first-search (BFS) Path Inverter (App. A.16). The framework is agnostic to the network architectures used to implement each level.

[1] Abbreviations used throughout the paper are listed in App. A.1.

Table 1: **Inverter implementations, ordered by level.** The neural Inverters emit sequences in one feedforward pass, providing fully amortized planning across goals, action sequences, and across levels in hierarchical variants.

Level	Inverter	Task / requested outcome	Emitted sequence
1	Motor (neural)	Reach a position (MuJoCo point mass)	2-D forces
1	Locomotion (neural)	Reach a waypoint (MuJoCo Ant)	8-D joint torques
1	Pulse (neural)	Realize a target unitary (quantum gate synthesis)	Microwave control pulses
2	Path (BFS)	Route to a goal through a maze	Waypoints
2	Game (neural)	Collect pellets in AntMan	Waypoints
3	Strategy (neural)	Balance reliability, progress, and compute budget	Steps between controller calls along the planned route

2 Neuro-Inspired Inverse Learning

We formalize **Inverse Learning** as amortized optimization of a task objective through a differentiable known or learned FoM, providing **fully amortized planning across goals, action sequences, and across levels in hierarchical variants**. Relationships to supervised control, imitation, and reinforcement learning, including differentiable model-based RL and deep deterministic policy gradient (DDPG), are detailed in App. A.2.

Acting to achieve a goal admits a direct formulation as action optimization: given state s_0 and a task conditioning c (the framework's per-chunk subgoal c_k at Level 1, the overall goal G at Level 2), find

$$a^*_{1:T} \;=\; \arg\min_{a_{1:T}} \; \mathcal{J}(s_{0:T},\, a_{1:T},\, c) \qquad \text{s.t. } \; s_t = f(s_{t-1}, a_t) \;\; \text{for } t = 1, \ldots, T, \tag{1}$$

a discrete-time Bolza problem of classical optimal control (OC) [12, 13] parameterized by (s_0, c), with $\mathcal{J}$ any differentiable combination of terminal cost, running reward, and action regularizer. Pontryagin's necessary conditions couple state and costate equations with control optimality and boundary conditions. The adjoint recursion corresponds to backpropagation through f, used in gradient-based per-query trajectory optimization. The *inverse planning equation*, Eq. (1), contrasts with the Bellman equation $V^*(s) = \max_a[r(s,a) + \gamma V^*(f(s,a))]$ [14], which recursively relates values across states.

Inverse learning: amortizing the planner. Inverse learning replaces classical per-query trajectory optimization by a *learned solver*: a neural network $g_\phi(s_0, c) \to a_{1:T}$ trained, through a FoM f_θ (typically learned; analytic when the physics is known, e.g., the Lindbladian in Sec. 4.5), to approximate a model-relative solution in one feedforward pass per action chunk:

$$\min_{\phi} \; \mathbb{E}_{(s_0,c)} \Big[\, \mathcal{J}\Big(f_\theta^{(1:T)}\big(s_0,\; g_\phi(s_0,c)\big),\;\; g_\phi(s_0,c),\;\; c \Big) \, \Big]. \tag{2}$$

At deployment, Level 1 is a goal-conditioned controller executing open-loop chunks. "Amortized planning" refers to learning a predictive solution map through inverse task optimization. We call the inverse model networks trained this way **inverse world models (IWMs)**. Jordan & Rumelhart's distal-teacher framework [4] trains inverse mappings through learned forward models of unknown dynamics. Our implementation builds on this principle with jointly emitted multi-step action sequences and hierarchical composition (Secs. 4.1–4.4). In our experiments f_θ is trained first and then frozen during inverse training, but joint (f_θ, g_ϕ) training is equally natural and would tie FoM accuracy to the action distribution the Inverter actually visits.

In our framework, directly using a known, observable target provides stronger task structure than sampled scalar reward alone. Our comparisons do not address the problem of discovering suitable final goals from reward. Our experiments compare complete methods with different information/model interfaces but with access to the identical task-related information. Supplementary Tables 5 and 6 in App. A.2 give a detailed comparison of inverse learning with other learning paradigms.

3 Related work

Amortized optimal control. Inverse learning is an instance of *amortized optimal control*, closely related to differentiable predictive control (DPC) [15] – training a network to emit a multistep OC solution in a single feedforward pass. Related instances differ in either output shape or in whether deployment-time iteration is avoided: trajopt distillation into a policy [16] clones the trajectories of a separate, non-differentiable offline OC solver; value-gradient backprop through learned dynamics [17] differentiates the value through model rollouts but emits one action at a time; differentiable MPC layers [18] embed a constrained QP solver as a differentiable layer and so retain test-time iterative optimization.

Differentiable world-model control. Backpropagating a control objective through a learned dynamics model has a long lineage [19, 4]; subsequent variants – PILCO [20], Dreamer [21], Universal Planning Networks [22], TD-MPC [23] – retain test-time iteration, value bootstrapping, or one-step outputs. Our implementation emphasizes joint sequence output and query cost: a separate inverse network emits the full T-step action sequence in one feedforward pass.

Trajectory-level sequence models and action-chunked imitation. A parallel line treats the action sequence as a first-class object: autoregressive sequence modeling [24], return-conditioned decoding [25], iterative trajectory denoising [26], online tree search over a learned model [27], and test-time-iterative latent planners [28, 29]; broader hierarchical-planning agendas with learned predictive world models [30, 31] share these commitments. The Inverter is the zero-iteration counterpart, folding trajectory optimization into training. Action Chunking Transformers [32] and Diffusion Policy [33] also produce action chunks and are trained on expert demonstrations. MPNet [34] learns next-configuration predictions from expert paths and constructs paths through iterative bidirectional planning.

Hierarchical, goal-conditioned, and explicit-MPC control. Hierarchical RL builds on the options framework [35], typically mixing level-specific mechanisms (e.g., high-level policy gradient over low-level actor-critic); hierarchical latent world models have been used for visual humanoid control but still rely on test-time planning [36]. From classical control, the closest analogue to our framework is explicit MPC [37]; our approach is its learned, nonlinear, goal-conditioned generalization.

4 Planning and control with Inverters: properties, performance, and failure modes

We organize the experiments by increasing structural complexity: a single Level 1 Inverter (`maze2d-umaze-v1`; Sec. 4.1), a Level 1 Inverter coupled with a (simple) algorithmic Path Inverter at Level 2 (larger `maze2d` layouts and all six `antmaze-v2` variants; Secs. 4.2–4.3), two paired learned Inverters (AntMan on `antmaze-large-diverse-v2`; Sec. 4.4), and a three-level Strategy experiment (App. A.16). Quantum gate synthesis tests the Level 1 Inverter in a different physical domain (Sec. 4.5). Scores and trajectories across all nine maze variants from D4RL [38], a benchmark suite providing offline reinforcement-learning datasets, are summarized in Suppl. Tab. 7 and Suppl. Fig. 6.

4.1 Single Motor Inverter (`maze2d-umaze`)

We evaluate `maze2d-umaze-v1` with a frozen Transformer FoM ($T = 16$ chunks) paired with a Transformer IWM that emits a 128-step action sequence from the current state and goal in one forward pass (architectures, dataset, baselines: App. A.3).

Performance and inference-time compute: Table 2 summarizes D4RL and per-episode compute across all three `maze2d` variants. On `umaze-v1` a one-shot $K = 128$ plan delivers $\mathbf{161.6 \pm 2.2}$ in just 3 NN forward passes (one to reach the goal, 2 to stay at the goal until the end of the fixed 300 time step evaluation window) and $\mathbf{11.4}$ ms total per episode – $\mathbf{37}\times$ less than Diffuser, $\mathbf{47}\times$ less than SAC-N, and an estimated nearly three orders of magnitude less than DecisionLLM [40]. The Inverter takes 3.19 ms per forward pass in this measurement campaign; emitting a full 128-step action chunk reduces the fixed 300-step episode to three calls (App. A.4).

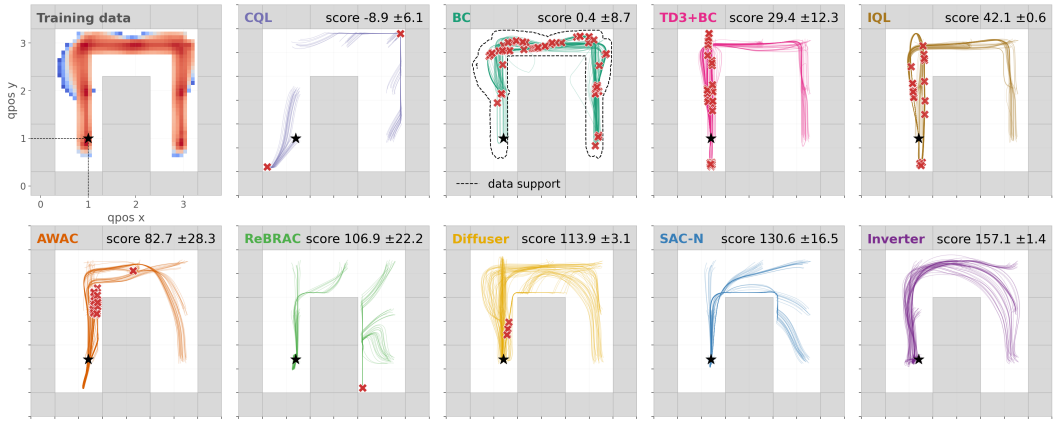


Figure 2: **Trajectory comparison on maze2d-umaze-v1.** Top-left: training data heatmap showing the U-shaped corridor coverage. Remaining panels: 100 evaluation trajectories per method, ordered by D4RL score. ★ = goal; red × = timeout (episode did not reach the goal). The Inverter (ours) produces smooth, direct paths that are not restricted to the angular data support. Offline-RL baselines use CORL implementations [39]; Diffuser uses its separate implementation.

Table 2: **Maze2D: Inverter vs. best offline-RL or diffusion-planner baseline per maze and fastest PyTorch BC-10%. Top**: D4RL score (mean ± SD; Inverter: across four training seeds; Diffuser: across 100 episodes of one checkpoint). **Bottom**: Neural Network (NN) forward passes and total wall time per episode (NN passes plus algorithmic overhead, single A40 GPU, PyTorch, batch 1 except Diffuser: 64 trajectories per plan, cuda-synced). Best-per-column bolded; the Inverter row reports the $K = 128$ one-shot configuration on umaze (Tab. 8, App. A.5) and $K = 16$ on medium/large (Tabs. 9, 10). Full per-method tables (10+ baselines, including JAX+JIT ReBRAC and Diffuser): App. A.5.

	`umaze-v1`	`medium-v1`	`large-v1`
D4RL score			
Best offline-RL/diffusion baseline	SAC-N: 130.6 ± 16.5	Diffuser: 130.1 ± 22.7	AWAC: 209.1 ± 8.2
Inverter (ours)	**161.6 ± 2.2**	**166.8 ± 1.2**	**220.7 ± 0.2**
# NN forward passes per episode / total wall time per episode (ms)			
BC-10% (fastest PyTorch)	300 / 374.9	600 / 749.8	800 / 999.7
Inverter (ours)	**3 / 11.4**	**38 / 72.9**	**50 / 93.7**

Smooth, sequence-coherent trajectories. Figure 2 shows that the Motor Inverter produces smooth, direct paths through the maze, while every baseline yields visibly noisier, more angular, or more constrained motion. App. A.7 reports per-chunk displacement directions and curvature variance: the Inverter achieves the broadest directional spectrum and, by a large margin, the lowest median curvature variance among the compared methods. These geometric observations complement the speed measurements in Fig. 3.

Action saturation and local optimal control. For an idealized one-dimensional damped point mass with bounded input, minimum-time transfer to a resting target uses bang-bang control (App. A.6). This local straight-segment model motivates our action-saturation analysis. Figure 3 shows that the Inverter concentrates actions near the bounds of the action box; its mean absolute action components lie closer to full saturation than those of the training data. The Inverter also attains the highest measured average speed in Fig. 3 and the strongest D4RL headline score among the compared offline-RL and diffusion-planner baselines on `maze2d-umaze-v1` (Table 2). During training, task gradients flow through the frozen FoM across complete action sequences, allowing optimization beyond the demonstrated action statistics.

Noise robustness and replanning. In a separate open-arena point-mass test, we vary execution-prefix length under injected action noise with one frozen Inverter. At the strongest tested noise, replanning every 64 steps achieves 100% success with 1.70 calls on average, versus 51.5% with

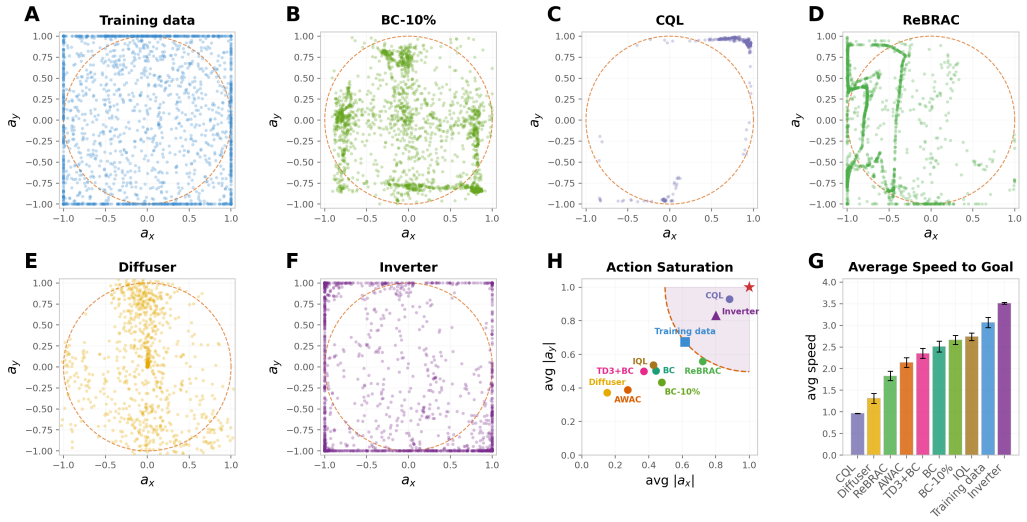


Figure 3: **Action-space structure and navigation performance.** Per-step action scatter plots (a_x, a_y) for the training data and a representative subset of methods (Panels A–F). Note that BC-10% [41] lands *below* the training data on action saturation (Panel H) – the multimodal-action BC failure mode [42, 33]. Panel H: ★ at $(1, 1)$ marks the full-saturation reference. Panel G: average speed to goal, using the same trajectory data as in Figure 2. The Inverter combines greater action saturation than the training data with high measured speed to goal (Panel G). CQL also produces highly saturated actions but fails the task at a negative D4RL score, illustrating that saturation alone does not ensure successful navigation.

one-shot execution. Replanning every step achieves only 60.5% despite 250.5 calls, demonstrating a nonmonotonic robustness–compute trade-off (full protocol and sweeps: App. A.10).

4.2 Motor Inverter + simple algorithmic Path Inverter (`maze2d-medium/large`)

On `maze2d-medium-v1` and `maze2d-large-v1`, the dataset's longest corridor paths exceed the Level 1 Inverter's 128-step horizon, so a single feedforward plan is insufficient. We therefore add a simple **algorithmic Path Inverter (BFS built exclusively on the offline-data density)** at Level 2: it builds a free-space occupancy grid from the training-data density alone, runs a 4-connected BFS, and emits one sub-goal per corridor elbow; the Level 1 Inverter then executes chunk by chunk toward the current sub-goal. No maze geometry, simulator, or FoM call enters the planner. App. A.8 gives the full construction; Fig. 8 shows a representative plan overlaid on the data heatmap.

Headline numbers for medium and large at $K = 16$ (highest D4RL across our K-sweep on medium and large; Suppl. Tab. 12) appear in Table 2; full per-method breakdowns are in App. A.5. The stacked Inverter reaches $\mathbf{100/100}$ success across 4 seed IWMs – roughly $\mathbf{10}$–$\mathbf{30}\times$ faster than step-wise offline-RL baselines and $\mathbf{39}$–$\mathbf{51}\times$ faster than Diffuser (DecisionLLM reference for this task not available). The only compute-competitive method is ReBRAC† (JAX+JIT, ~ 0.2 ms/pass), which still lags the Inverter on D4RL by 63 points on medium and 137 on large. Increasing $K \in \{16, 32, 64, 128, 256\}$ trades D4RL for inference speed (App. A.9); scores drop sharply at $K = 128$. The $K = 256$ row uses a separately trained 256-step IWM. This is qualitatively reminiscent of Fitts' law and submovement / iterative-correction motor psychophysics [43–45], and of the optimal-control principles motivating our framework [8].

4.3 `antmaze-v2`: Locomotion and FoM hacking

We find that the same setup (Level 1 Locomotion Inverter + data-only BFS Path Inverter at Level 2) successfully transfers to all six `antmaze-v2` variants – 29-dim MuJoCo ant locomotion with 8-dim torque actions on the same topologies plus play/diverse goal distributions – and has a higher mean score than the published CORL ReBRAC baseline on every variant, with the gap widening on the large mazes (Tab. 3 and App. A.5; for comparison to published model-based offline RL baselines see

Table 3: **AntMaze headline: Inverter vs. strongest CORL D4RL competitor (ReBRAC) and fastest PyTorch baseline (BC-10%). Top**: D4RL score (%), mean ± std over 4 seeds; Inverter: 100 paired cases per seed, data-only BFS (App. A.13). **Bottom**: ms per environment step (single A40 GPU, PyTorch, batch 1, cuda-synced). Best-per-column bolded. Full 10-baseline breakdown including JAX+JIT ReBRAC timing: Tab. 11, App. A.5.

	u-umaze	u-divrs	m-play	m-divrs	l-play	l-divrs
D4RL score (%)						
ReBRAC	97.8 ± 1.5	83.5 ± 7.0	89.5 ± 3.4	83.5 ± 8.2	52.2 ± 29.0	64.0 ± 5.4
Inverter (ours)	$\mathbf{98.8} \pm 0.4$	$\mathbf{99.5} \pm 0.5$	$\mathbf{94.8} \pm 4.5$	$\mathbf{99.0} \pm 0.0$	$\mathbf{92.0} \pm 4.2$	$\mathbf{93.8} \pm 2.5$
ms / env step						
BC-10% (fastest PyTorch)	1.25	1.25	1.25	1.25	1.25	1.25
Inverter (ours)	**0.14**	**0.14**	**0.13**	**0.13**	**0.13**	**0.13**

Table 4: **Matched-controller ablation with identical Level 2 routing.** All methods use the same algorithmic BFS Path Inverter and 16-step waypoint-update schedule; chunked BC also matches the Inverter's 16-step action execution. The Inverter retains the highest average score in both Maze2D and AntMaze, showing that its advantage is not explained by algorithmic routing alone. D4RL scores on Maze2D-medium/large (2 tasks), all AntMaze-v2 variants (6 tasks), and all 8 tasks, with 100 matched evaluation episodes per seed. Means weight tasks equally. Uncertainty is $\sqrt{\sum_{t=1}^{N} \sigma_t^2}/N$ for N tasks, using each task's reported spread σ_t across four training seeds. Best means are bold. Protocol and task results: App. A.13.

Level 1 controller	Maze2D avg.	AntMaze avg.	Overall avg.
Reactive BC	118.0 ± 4.0	93.0 ± 1.1	99.3 ± 1.3
Chunked BC	178.4 ± 0.4	22.0 ± 1.8	61.1 ± 1.3
BC-10%	145.1 ± 16.5	54.0 ± 3.0	76.8 ± 4.7
IQL	123.9 ± 31.8	84.4 ± 2.1	94.3 ± 8.1
AWAC	172.8 ± 12.0	85.9 ± 2.2	107.6 ± 3.4
ReBRAC	136.2 ± 11.3	52.5 ± 7.3	73.4 ± 6.1
CQL	130.4 ± 63.4	69.2 ± 2.7	84.5 ± 16.0
SAC-N	108.6 ± 3.3	49.4 ± 5.0	64.2 ± 3.9
EDAC	145.5 ± 25.1	26.0 ± 9.2	55.9 ± 9.3
Inverter (ours)	$\mathbf{193.7 \pm 0.6}$	$\mathbf{96.3 \pm 1.1}$	$\mathbf{120.7 \pm 0.9}$

App. A.11). The Locomotion Inverter emits 16-step chunks, with measured inference and planning cost of 0.13–0.14 ms per executed environment step across all six variants. Relative to the BC-10% timing used in Table 3, this corresponds to approximately $9\times$ lower per-step cost (timing provenance: App. A.12).

Matched-controller ablation. To test the contribution of learned inverse control, we replace the Level 1 controller while holding the algorithmic Path Inverter, waypoint logic, goal conditioning, starts/goals, and evaluation protocol fixed. All methods use the same underlying offline data set. Across eight tasks, the Inverter has the highest overall mean (120.7 versus 107.6 for AWAC, $+13.1$ points, $+12.2\%$; Table 4) and leads on 6/8 tasks. It outperforms reactive and chunked BC on all eight tasks. Chunked BC matches its 16-step execution cadence, whereas reactive BC and the offline-RL controllers require one network call per step. These results support a contribution from learned inverse control beyond shared routing and chunked output alone (full protocol and task results: App. A.13).

One important new observation from AntMaze suggested that a plain task-reward Locomotion Inverter on AntMaze may be *forward-model hackable* – the ant tips, joints jam, body yaw drifts in simulation, while the FoM predicts successful action – a failure we suspected to be related to the fact that the offline training data contains relatively few failure cases and the FoM is therefore uncalibrated off-support. We compensated with two additive, per-time-step auxiliary losses – a BC action-fidelity anchor and a body-yaw regularizer – which compose linearly with the task gradient (ablations in App. A.12, Suppl. Tab. 16). These additions work in practice but are conceptually unsatisfactory: anchoring toward dataset actions partially compromises inverse learning's defining property of being FoM-gradient-driven rather than data-anchored. These observations motivated detailed analysis of

the ant's kinematic data (below), and designing the new AntMan task (Sec. 4.4), which lets us (i) control the offline training data ourselves and therefore study FoM hacking systematically, and (ii) demonstrate a full $n=2$ Hierarchical Inverter – two trained Inverters, one for planning and one for control.

Behavior beyond cloning. A detailed analysis of the ant's kinematic data shows that the action-fidelity anchor permits learning useful behavior beyond imitation. In matched U-Maze first-block queries, the Inverter reduces target overshoot by 56.1% and post-crossing forward displacement by 48.5% relative to chunked BC (Suppl. Fig. 9; protocol and metrics: App. A.14). None of 503 observable first true-goal arrivals in the training data meets the controlled-stop criterion. These observations support that inverse learning resulted in helpful target-specific deceleration beyond the examples in the training data.

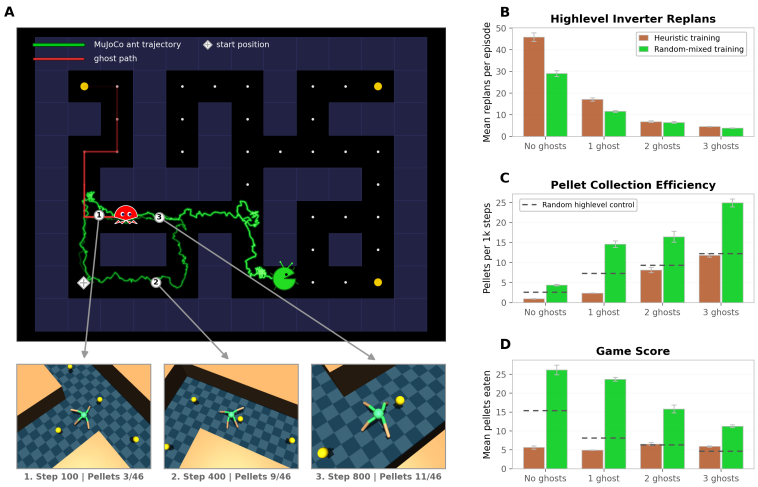


Figure 4: **Simultaneous planning and control with a coupled Game and Locomotion Inverter on the AntMan task.** Panel A shows a representative rollout in the MuJoCo maze with an abstract rendering of the game situation and 3 snapshots showing the underlying motion of the MuJoCo Ant through the maze. Successful planning and control was achieved when training on random training data, but not on training data only reflecting a heuristic game controller (Panels B–D). Metrics are averaged over 100 evaluation games (25 each for no, 1, 2, and 3 ghosts). Only random training yields above chance pellet-collection efficiency and pellet counts at every difficulty, while replanning is comparable or lower.

4.4 Hierarchical Inverse Learning for planning and control

To study a fully learned $n = 2$ Hierarchical Inverter, we designed the new AntMan task, which requires pure offline learning of both low-level motor control of the MuJoCo Ant and a Pac-Man-style game embodied through the same Ant placed in the `antmaze-v2 large` setup.

We solve this challenge with a *Level 1 Locomotion Inverter* which controls the Ant in `antmaze-large-diverse-v2`, while a paired *Level 2 Game Inverter* controls the Locomotion Inverter by outputting 32-step waypoint directions from a 58-dim game state (ant pose, ghost states, pellet map, mode indicators). A causal-transformer FoM is trained on simulated AntMan games; the Game Inverter is then trained purely through forward-model reward to collect as many pellets as possible. To isolate the training-distribution effect, two matched FoM/IWM pairs differ only in the offline training data generating behavior policy: (i) heuristic (chases pellets, avoids ghosts) and (ii) random (uniform random neighbor-cell moves).

End-to-end hierarchical Inverse Learning for planning and control. The two levels are trained separately through their respective forward models and composed for execution. The AntMan task demonstrates that a full $n = 2$ Hierarchical Inverter, whose Game Inverter is trained using FoM-predicted game reward, successfully plays AntMan games well above chance – proof of principle for increasing hierarchical depth and for extending the framework from continuous physics to symbolic/discrete planning.

Forward-model hacking and how to mitigate it. A second, striking observation reverses the offline-learning intuition that more competent demonstrations make better data (Fig. 4). With heuristic training data the FoM is severely hacked – predicted 37.98 pellets vs. 4.48 realized ($8.48\times$) – and the Game Inverter drops to chance at 2–3 ghosts. With random training data the FoM is calibrated and the Inverter stays above chance across all difficulty levels. The failure mechanism (App. A.15, Fig. 10) is that under expert-skewed data the Inverter generates off-support plans for which the FoM-predicted reward *decorrelates* from the actual game outcome. These observations highlight the importance of dynamics coverage beyond demonstrator competence. Our findings thus support that inverters are particularly suited to settings where learning must proceed from random rather than expert data; the converse – combining inverters with imitation learning beyond plain BC – is an interesting open future direction.

Three-level Inverter. Here, Level 1 generates motor action sequences, and Level 2 supplies waypoints for Level 1. With the lower levels frozen, the task of the Level 3 Strategy Inverter is to learn through a frozen macro-FoM to balance requested reliability, progress, and compute budget. On held-out disturbed Maze2D-large cases, its reliability-focused profile averages 30.74 controller calls versus 100 when calling every eight steps: a 69.26% reduction (paired 95% interval $[65.01\%, 73.37\%]$). It achieves 100% observed success and meets the prespecified success criteria (App. A.16). Thus, the third level successfully learns to allocate controller calls along the route while maintaining reliable navigation.

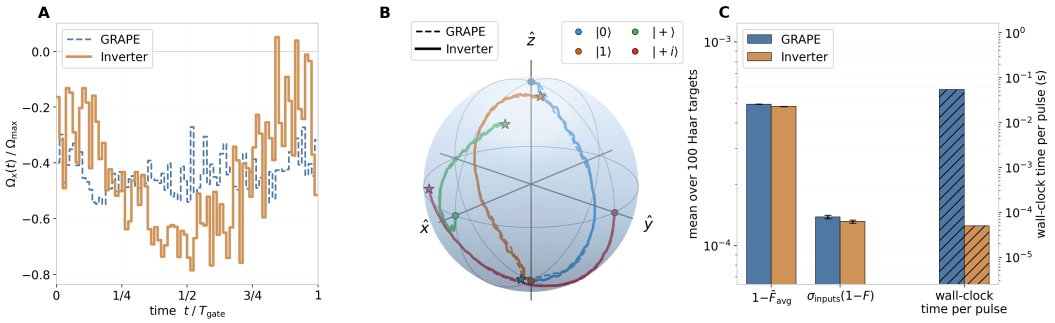


Figure 5: **Single-shot gate synthesis on a noisy three-level transmon.** **(A)** Pulse $\Omega_x(t)$ for target 6: GRAPE dashed blue, Inverter solid orange. **(B)** Four input-state Bloch trajectories; stars mark ideal outputs. **(C)** Same 100 fresh Haar targets: exact infidelity and six-state fidelity spread (means±SEM), and warm blocked CPU latency. The Pulse Inverter combines high-fidelity gate synthesis with rapid pulse generation in a single feedforward pass.

4.5 Quantum gate synthesis

Quantum gates require shaped electromagnetic pulses. GRAPE [46] can precompute pulses for a fixed library, but storing a separately optimized pulse for every continuously varying gate requested by an algorithm or measurement feedback is impractical. A Pulse Inverter learns this mapping and generates pulses on demand. Related work explores random-layer optimization [47] and neural compilation [48].

We train on a noisy three-level transmon to emit 80 bounded (Ω_x, Ω_y) slices, minimizing exact unconditional Haar-average infidelity through its Lindblad channel, without GRAPE labels. On 4096 fresh Haar targets, 4051 meet $1-\bar{F}_{\text{avg}} \leq 5\times10^{-4}$. On 250 matched targets, fidelity is very close to heavy-budget optimized GRAPE: mean infidelity is 4.823×10^{-4} versus 4.807×10^{-4}. Figure 5 uses a separate 100-target threshold benchmark: warm blocked CPU latency is $50.1\,\mu$s versus 54.4 ms ($1086\times$); interleaving gives $777\times$, with $98/100$ versus $96/100$ successes. Two-qubit results reach $\bar{F}_{\text{avg}}$=0.9832 with learned expert selection (App. A.18). Together, these results extend inverse learning to rapid, high-fidelity pulse generation for previously unseen quantum operations.

5 Discussion, limitations, and outlook

Learning beyond behavior cloning. Maze2D and AntMaze provide complementary evidence for learning beyond demonstrations. In Maze2D, the Inverter combines action saturation beyond the training distribution with high speed to goal, consistent with the local bang-bang optimum (Fig. 3). In AntMaze U-Maze, a BC-anchored Inverter reduces end-block overshoot by 56.1% relative to chunked BC, supporting temporal recombination of demonstrated actions into target-specific deceleration (App. A.14). Thus, our findings support the conclusion that task optimization through inverse learning can improve both action selection and its temporal organization beyond the examples in the training data.

Task-specificity vs. generalization. Across the 9 `maze2d`/`antmaze-v2` benchmarks alone, the strongest offline-RL scores require switching among three structurally distinct algorithms (AWAC, SAC-N, ReBRAC), each selected per benchmark. Our framework uses a single universal architecture (universal causal-transformer FoM and Inverter PyTorch classes plus a simple Path Inverter) across all 9 maze variants, but with the auxiliary-loss setup as the main per-task adaptation. E.g., the quantum case required an MLP-based rather than transformer-based Pulse Inverter. A full list is given in Suppl. Tab. 22. Making these useful task-specific adaptations learnable remains a goal shared with meta-RL and conditional sequence modeling. Limitations of amortization relative to deployment-time iteration and complementary combinations with RL are discussed in App. A.2.

Toward a fully learnable Inverter framework. For this aim, we delineate four complementary strategies (Suppl. Tab. 21), each operating on a different lever: (1) **Data** – cover the failure modes the Inverter is incentivized to exploit (proof of principle: AntMan random-mixed); (2) **Model** – cross-task / cross-embodiment pretraining of the FoM-and-Inverter core with per-task adapters, in the foundation-model pattern; (3) **Objective** – meta-learn the Bolza objective $\mathcal{J}$ across task families; (4) **Deployment** – replace remaining algorithmic slots with learned Inverters, including the frozen BFS in our three-level stack, and test deeper hierarchies. The framework may offer a suitable substrate for this aim: its Inverter core is differentiable end-to-end and its slots (FoM, IWM, objective $\mathcal{J}$, hierarchy depth) are modular and defined, helping specialized concrete instances to be learnable within a shared organizational principle.

Shared organizational principles with specialized concrete instances. This pattern is analogous to how mammalian motor systems achieve high performance on radically different niches under one architecture: primates carry a direct corticomotor neuronal pathway for independent finger control [49, 50], echolocating bats overlay a ms-scale audio-motor loop on the same paired forward/inverse architecture [51], rodents add a dedicated brainstem Central Pattern Generator (CPG) for $\sim$5–12 Hz whisking [52], and elephants control a hydrostatic trunk via a massively expanded facial motor nucleus [53] – all within the fundamental principles of mammalian motor organization. Our inverse-learning framework, too, may enable such specialized concrete instances based on generic organizational principles.

Extension. Our Inverters are deterministic, and our experiments use fully-observable, single-agent settings; extending to stochastic / partially-observable / multi-agent settings requires Inverters at each level to accommodate the added uncertainty. A natural first direction is to develop **probabilistic Inverters** spanning a range from amortized stochastic optimal control at one end, through control-as-inference and variational forms, to active-inference Inverters. A second direction is to develop **latent Inverters**, e.g. with bidirectional latent world models [28, 36] as a more abstract planning substrate. A third direction is **broader and deeper hierarchies** (prospective applications: App. A.2.4), beyond our three-level stack: a meta-Inverter could treat the choice of subgoal specification for lower levels as an inverse problem given a task description – a natural slot for VLLM substrates at the top layer, given their language-conditioned task-decomposition and broad world-knowledge capabilities.

A differentiable world-interface for latency- and compute-critical embodied AI. Inspired by the mammalian brain's paired forward/inverse internal models, open-loop multi-step motor commands, and sequential organization of action, the Inverter framework enables fast and effective planning and control through a feedforward FoM-and-IWM core that emits entire action chunks in single passes. Neuro-inspired inverse learning thus offers a differentiable world-interface for latency- and compute-critical embodied AI.

Acknowledgments and Disclosure of Funding

Funding. This work used Machine-Brain Interfacing (MBI) large-scale equipment funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 428605208.

Competing interests. Maryna Kapitonova and Tonio Ball are founders and shareholders of NeuroMentum AI, an R&E company focusing on NeuroAI.

References

[1] David C. Knill and Alexandre Pouget. The Bayesian brain: The role of uncertainty in neural coding and computation. *Trends in Neurosciences*, 27(12):712–719, 2004. doi: 10.1016/j.tins.2004.10.007.

[2] Karl Friston. The free-energy principle: A unified brain theory? *Nature Reviews Neuroscience*, 11(2): 127–138, 2010. doi: 10.1038/nrn2787.

[3] Samuel J. Gershman, Eric J. Horvitz, and Joshua B. Tenenbaum. Computational rationality: A converging paradigm for intelligence in brains, minds, and machines. *Science*, 349(6245):273–278, 2015. doi: 10.1126/science.aac6076.

[4] Michael I. Jordan and David E. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16(3):307–354, 1992. doi: 10.1207/s15516709cog1603_1.

[5] Mitsuo Kawato. Internal models for motor control and trajectory planning. *Current Opinion in Neurobiology*, 9(6):718–727, 1999. doi: 10.1016/S0959-4388(99)00028-8.

[6] Daniel M. Wolpert and Mitsuo Kawato. Multiple paired forward and inverse models for motor control. *Neural Networks*, 11(7–8):1317–1329, 1998. doi: 10.1016/S0893-6080(98)00066-5.

[7] Masahiko Haruno, Daniel M Wolpert, and Mitsuo Kawato. Hierarchical mosaic for movement generation. In *International congress series*, volume 1250, pages 575–590. Elsevier, 2003.

[8] Emanuel Todorov and Michael I. Jordan. Optimal feedback control as a theory of motor coordination. *Nature Neuroscience*, 5(11):1226–1235, 2002. doi: 10.1038/nn963.

[9] Michel Desmurget and Scott Grafton. Forward modeling allows feedback control for fast reaching movements. *Trends in Cognitive Sciences*, 4(11):423–431, 2000.

[10] Ann M. Graybiel. The basal ganglia and chunking of action repertoires. *Neurobiology of Learning and Memory*, 70(1–2):119–136, 1998. doi: 10.1006/nlme.1998.3843.

[11] Jörn Diedrichsen and Katja Kornysheva. Motor skill learning between selection and execution. *Trends in Cognitive Sciences*, 19(4):227–233, 2015.

[12] Lev S. Pontryagin, Vladimir G. Boltyansky, Revaz V. Gamkrelidze, and Evgenii F. Mishchenko. *The Mathematical Theory of Optimal Processes*. Interscience Publishers, New York, 1962.

[13] Oskar Bolza. *Vorlesungen über Variationsrechnung*. B. G. Teubner, Leipzig and Berlin, 1909.

[14] Richard E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[15] Ján Drgoňa, Karol Kiš, Aaron Tuor, Draguna Vrabie, and Martin Klaučo. Differentiable predictive control: Deep learning alternative to explicit model predictive control for unknown nonlinear systems. *Journal of Process Control*, 116:80–92, 2022.

[16] Igor Mordatch and Emo Todorov. Combining the benefits of function approximation and trajectory optimization. In *Robotics: Science and Systems*, 2014. doi: 10.15607/RSS.2014.X.052.

[17] Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, 2015.

[18] Brandon Amos, Ivan Dario Jimenez Rodriguez, Jacob Sacks, Byron Boots, and J. Zico Kolter. Differentiable MPC for end-to-end planning and control. In *Advances in Neural Information Processing Systems*, 2018.

[19] Jürgen Schmidhuber. Making the world differentiable: On using self-supervised fully recurrent neural networks for dynamic reinforcement learning and planning in non-stationary environments. Technical Report FKI-126-90, Institut für Informatik, Technische Universität München, 1990.

[20] Marc Peter Deisenroth and Carl Edward Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning*, pages 465–472, 2011.

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

[22] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4732–4741, 2018.

[23] Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations*, 2024.

[24] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, volume 34, pages 1273–1286, 2021.

[25] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[26] Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

[27] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020. doi: 10.1038/s41586-020-03051-4.

[28] Vlad Sobal, Wancong Zhang, Kyunghyun Cho, Randall Balestriero, Tim G. J. Rudner, and Yann LeCun. Learning from reward-free offline data: A case for planning with latent dynamics models, 2025.

[29] Amir Bar, Gaoyue Zhou, Danny Tran, Trevor Darrell, and Yann LeCun. Navigation world models, 2024. URL `https://arxiv.org/abs/2412.03572`.

[30] Yann LeCun. A path towards autonomous machine intelligence, version 0.9.2. OpenReview position paper, 2022. 2022-06-27.

[31] Adrien Bardes, Quentin Garrido, Jean Ponce, Xinlei Chen, Michael Rabbat, Yann LeCun, Mido Assran, and Nicolas Ballas. Revisiting feature prediction for learning visual representations from video. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL `https://openreview.net/forum?id=QaCCuDfBk2`. Featured Certification.

[32] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Robotics: Science and Systems*, 2023. doi: 10.15607/RSS.2023.XIX.016.

[33] 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.

[34] Ahmed H. Qureshi, Anthony Simeonov, Mayur J. Bency, and Michael C. Yip. Motion planning networks. In *IEEE International Conference on Robotics and Automation*, pages 2118–2124, 2019. doi: 10.1109/ICRA.2019.8793889.

[35] Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, 1999. doi: 10.1016/S0004-3702(99)00052-1.

[36] Nicklas Hansen, Jyothir S V, Vlad Sobal, Yann LeCun, Xiaolong Wang, and Hao Su. Hierarchical world models as visual whole-body humanoid controllers. In *International Conference on Learning Representations*, 2025.

[37] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002. doi: 10.1016/S0005-1098(01)00174-1.

[38] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4RL: Datasets for deep data-driven reinforcement learning, 2020. URL https://arxiv.org/abs/2004.07219.

[39] Denis Tarasov, Alexander Nikulin, Dmitry Akimov, Vladislav Kurenkov, and Sergey Kolesnikov. CORL: Research-oriented deep offline reinforcement learning library. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[40] Xiaowei Lv, Zhilin Zhang, Yijun Li, Yusen Huo, Siyuan Ju, Xuyan Li, Chunxiang Hong, Tianyu Wang, Yongcai Wang, Peng Sun, Chuan Yu, Jian Xu, and Bo Zheng. DecisionLLM: Large language models for long sequence decision exploration, 2026. URL https://arxiv.org/abs/2601.10148.

[41] Aviral Kumar, Joey Hong, Anikait Singh, and Sergey Levine. When should we prefer offline reinforcement learning over behavioral cloning? In *International Conference on Learning Representations*, 2022.

[42] Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *5th Annual Conference on Robot Learning*, 2021. URL https://openreview.net/forum?id=rif3a5NAxU6.

[43] Paul M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381–391, 1954.

[44] E. R. F. W. Crossman and P. J. Goodeve. Feedback control of hand-movement and Fitts' law. *The Quarterly Journal of Experimental Psychology Section A*, 35(2):251–278, 1983. doi: 10.1080/14640748308402133.

[45] David E. Meyer, Richard A. Abrams, Sylvan Kornblum, Charles E. Wright, and J. E. Keith Smith. Optimality in human motor performance: Ideal control of rapid aimed movements. *Psychological Review*, 95(3):340–370, 1988. doi: 10.1037/0033-295X.95.3.340.

[46] Navin Khaneja, Timo Reiss, Cindie Kehlet, Thomas Schulte-Herbrüggen, and Steffen J Glaser. Optimal control of coupled spin dynamics: design of nmr pulse sequences by gradient ascent algorithms. *Journal of magnetic resonance*, 172(2):296–305, 2005.

[47] Marco Dall'Ara, Martin Koppenhöfer, Florentin Reiter, Thomas Wellens, Simone Montangero, and Walter Hahn. Random layers for quantum optimal control with exponential expressivity. *arXiv preprint arXiv:2603.08948*, 2026. URL https://arxiv.org/abs/2603.08948v2.

[48] Arash Fath Lipaei, Ebrahim Khaleghian, Gani Göral, Zidong Lin, Selin Aslan, and Özgür E Müstecaplıoğlu. Fidelity-informed neural pulse compilation of a continuous family of quantum gates with uncertainty-margin analysis. *arXiv preprint arXiv:2604.11314*, 2026. URL https://arxiv.org/abs/2604.11314v3.

[49] Roger N. Lemon. Descending pathways in motor control. *Annual Review of Neuroscience*, 31:195–218, 2008. doi: 10.1146/annurev.neuro.31.060407.125547.

[50] Jean-Alban Rathelot and Peter L. Strick. Subdivisions of primary motor cortex based on cortico-motoneuronal cells. *Proceedings of the National Academy of Sciences*, 106(3):918–923, 2009. doi: 10.1073/pnas.0808362106.

[51] Hans-Ulrich Schnitzler and Annette Denzinger. Auditory fovea and Doppler shift compensation: Adaptations for flutter detection in echolocating bats using CF-FM signals. *Journal of Comparative Physiology A*, 197(5):541–559, 2011. doi: 10.1007/s00359-010-0569-6.

[52] David Kleinfeld and Martin Deschênes. Neuronal basis for object location in the vibrissa scanning sensorimotor system. *Neuron*, 72(3):455–468, 2011. doi: 10.1016/j.neuron.2011.10.009.

[53] Jeheskel Shoshani, William J. Kupsky, and Gary H. Marchant. Elephant brain: Part I: Gross morphology, functions, comparative anatomy, and evolution. *Brain Research Bulletin*, 70(2):124–157, 2006. doi: 10.1016/j.brainresbull.2006.03.016.

[54] Arnav Kumar Jain, Vibhakar Mohta, Subin Kim, Atiksh Bhardwaj, Juntao Ren, Yunhai Feng, Sanjiban Choudhury, and Gokul Swamy. A smooth sea never made a skilled sailor: Robust imitation via learning to search. *Advances in Neural Information Processing Systems*, 38:62676–62707, 2026.

[55] Tyler Han, Bat Nemekhbold, Siyang Shen, Rohan Baijal, Richard Ebock, Harine Ravichandiran, Sanghun Jung, Kevin Huang, and Byron Boots. Online world modeling enables real-world inverse reinforcement learning from observation, 2026. URL https://arxiv.org/abs/2602.24121.

[56] Andrew Wagenmaker, Yunchu Zhang, Mitsuhiko Nakamoto, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering your diffusion policy with latent space reinforcement learning. In Joseph Lim, Shuran Song, and Hae-Won Park, editors, *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pages 258–282. PMLR, 27–30 Sep 2025. URL https://proceedings.mlr.press/v305/wagenmaker25a.html.

[57] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 34, pages 20132–20145, 2021.

[58] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 1179–1191, 2020.

[59] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit Q-learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=68n2s9ZJWF8.

[60] Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. AWAC: Accelerating online reinforcement learning with offline datasets, 2020.

[61] Denis Tarasov, Vladislav Kurenkov, Alexander Nikulin, and Sergey Kolesnikov. Revisiting the minimalist approach to offline reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[62] Gaon An, Seungyong Moon, Jang-Hyun Kim, and Hyun Oh Song. Uncertainty-based offline reinforcement learning with diversified Q-ensemble. In *Advances in Neural Information Processing Systems*, 2021.

[63] L. M. Sonneborn and F. S. Van Vleck. The bang-bang principle for linear control systems. *Journal of the Society for Industrial and Applied Mathematics, Series A: Control*, 2(2):151–159, 1964. doi: 10.1137/0302013.

[64] Daniel Liberzon. *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton University Press, 2012. doi: 10.1515/9781400842643.

[65] Kwanyoung Park and Youngwoon Lee. Model-based offline reinforcement learning with lower expectile q-learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=OATPSB5JK1.

[66] Zhengyao Jiang, Tianjun Zhang, Michael Janner, Yueying Li, Tim Rocktäschel, Edward Grefenstette, and Yuandong Tian. Efficient planning in a compact latent action space. In *International Conference on Learning Representations (ICLR)*, 2023.

[67] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, et al. Jax: composable transformations of python+ numpy programs. 2018.

[68] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.

[69] J Robert Johansson, Paul D Nation, and Franco Nori. Qutip: An open-source python framework for the dynamics of open quantum systems. *Computer physics communications*, 183(8):1760–1772, 2012.

[70] Vladislav Kurenkov and Sergey Kolesnikov. Showing your offline reinforcement learning work: Online evaluation budget matters. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 11729–11752, 2022.

[71] Matthew Thomas Jackson, Uljad Berdica, Jarek Liesen, Shimon Whiteson, and Jakob Nicolaus Foerster. A clean slate for offline reinforcement learning, 2025. URL https://arxiv.org/abs/2504.11453.

[72] Tom Le Paine, Cosmin Paduraru, Andrea Michi, Caglar Gulcehre, Konrad Zolna, Alexander Novikov, Ziyu Wang, and Nando de Freitas. Hyperparameter selection for offline reinforcement learning, 2020.

A Technical appendices and supplementary material

Code availability. Code is publicly available at `https://github.com/neuromentum/inverter`.

A.1 Abbreviations

API application programming interface.

AWAC Advantage-Weighted Actor-Critic.

AWG arbitrary waveform generator (quantum hardware).

AWR Advantage-Weighted Regression.

BC behavior cloning; *BC-10%*: BC restricted to the top-10% return episodes.

BFGS Broyden–Fletcher–Goldfarb–Shanno (quasi-Newton optimizer); *L-BFGS-B*: limited-memory BFGS with box constraints.

BFS breadth-first search.

CBOP Conservative Bayesian Offline Policy optimization.

CNN convolutional neural network.

CNOT controlled-NOT (two-qubit quantum gate).

COMBO Conservative Model-Based Offline policy optimization.

CORL Clean Offline Reinforcement Learning (benchmark).

CPG central pattern generator.

CPU/GPU central / graphics processing unit.

CQL Conservative Q-Learning.

D4RL Datasets for Deep Data-Driven Reinforcement Learning.

DT Decision Transformer.

EDAC Ensemble-Diversified Actor-Critic.

FoM Forward Model.

GRAPE Gradient Ascent Pulse Engineering.

IQL Implicit Q-Learning.

IWM inverse world model.

JAX/JIT Python autodiff library / just-in-time (compilation).

KAK KAK decomposition (canonical form for two-qubit unitaries).

KL Kullback–Leibler (divergence).

LLM large language model.

MAML Model-Agnostic Meta-Learning.

MAPLE Model-based Adaptive Policy LEarning.

MBRL model-based reinforcement learning.

MLP multi-layer perceptron.

MOBILE MOdel-Bellman Inconsistency penalized offLinE Policy Optimization.

MOPO Model-based Offline Policy Optimization.

MOReL Model-based Offline Reinforcement Learning.

MPC model predictive control; *MPPI*: model predictive path integral.

NN neural network.

OC optimal control.

OOD out of distribution.

PD proportional–derivative (controller).

PG policy gradient.

PILCO Probabilistic Inference for Learning Control.

QAOA Quantum Approximate Optimization Algorithm.

QEC quantum error correction.

QP quadratic programming.

RAMBO Robust Adversarial Model-Based Offline RL.

ReBRAC Re-evaluated Behavior-Regularized Actor Critic.

RL reinforcement learning.

SAC Soft Actor-Critic; *SAC-N*: SAC with N-critic ensemble.

SEM standard error of the mean.

SR success rate.

$\mathbf{U}(d)$ Unitary group.

$\mathbf{SU}(d)$ Special unitary group.

TAP Trajectory Autoencoding Planner.

TD temporal difference; *TD-MPC*: temporal-difference MPC; *TD3+BC*: TD3 with behavior cloning regularizer.

TT Trajectory Transformer (*TT+Q*: Q-guided beam-search variant).

VQE Variational Quantum Eigensolver.

A.2 Relationship to supervised control, imitation, and reinforcement learning

Supplementary Table 5 separates task information, optimization mechanisms, learned mappings, and model requirements. Its entries describe representative configurations, rather than necessary or sufficient definitions of disjoint learning families. Behavior cloning (BC) represents imitation learning in this comparison; other imitation methods can use different objectives and mechanisms. For the Inverter, the objective $\mathcal{J}$ and task condition c specify the task, while the FoM Jacobian supplies the optimization pathway. Differentiating through frozen f_θ yields model-relative gradients $\partial\mathcal{J}/\partial a_{t,i}$ for all jointly emitted actions. Differentiable model-based reinforcement learning can share this pathway; DDPG instead differentiates a Bellman-trained critic through the current action.

Supplementary Table 5: **Representative learning configurations and their information/model requirements. (a) Direct target supervision, BC, and the Inverter.** Task objectives and optimization mechanisms are separate axes; the families can overlap. BC represents imitation here. The Inverter column is repeated in panel (b) for comparison with RL.

Axis	**Direct target supervision**	**Imitation (BC)**	**Inverter (ours)**
Learning-family grouping	Supervised learning	Supervised learning (BC case)	Supervised learning (distal-target framing in the fixed-goal case)
Task information / objective	Target $y^\star$ and prediction loss	Expert actions/trajectories and imitation loss	Differentiable finite-horizon objective $\mathcal{J}$; explicit target c in the fixed-goal case
Optimization / update mechanism	Gradient of target loss	Gradient of imitation loss	Gradient of the finite-horizon objective through the frozen FoM to jointly emitted actions
Explicit environment-level target used directly in objective	Task-dependent	No in canonical BC; demonstrations supply action targets	**Strong assumption:** required in the fixed-goal comparisons
Dynamics-model requirement	Not generally required	Not generally required	Differentiable known or learned FoM required during Inverter training
Learned mapping / amortization	Input to prediction	Observation/history (optionally goal) to action(s) or trajectory	Fully amortized: Across goals, action sequences, and across levels in hierarchical variants
Information / model requirement	Labeled targets and adequate input coverage	Suitable demonstrations and relevant coverage	Differentiable objective and valid FoM over optimized rollouts; fixed-goal cases also require a known target

Supplementary Table 5: **Representative learning configurations (continued). (b) Model-free RL, model-based RL, and the Inverter.** Entries are representative rather than universal definitions. Inverter entries describe training through a frozen FoM, with explicit targets in the fixed-goal comparisons. Goal-conditioned, sequence, and hierarchical RL variants also exist.

Axis	**Model-free RL**	**Model-based RL**	**Inverter (ours)**
Learning-family grouping	Reinforcement learning	Reinforcement learning	Supervised learning (distal-target framing in the fixed-goal case)
Task information / objective	Reward r_t and expected-return objective	Reward/cost objective	Differentiable finite-horizon objective $\mathcal{J}$; explicit target c in the fixed-goal case
Optimization / update mechanism	Sampled returns, learned values/advantages, policy gradients, or critic gradients	Planning/search, value or policy updates, sampling, or gradients through differentiable model rollouts	Gradient of the finite-horizon objective through the frozen FoM to jointly emitted actions
Explicit environment-level target used directly in objective	Not required; goal-conditioned variants may use one	Not required; goal-conditioned variants may use one	**Strong assumption:** required in the fixed-goal comparisons
Dynamics-model requirement	No transition model in the canonical decision-learning update	Known or learned model used; differentiability optional	Differentiable known or learned FoM required during Inverter training
Learned mapping / amortization	State/history (optionally goal) to policy/value; sequence/hierarchy variants exist	Learned model and/or policy; planning may remain per query	Fully amortized: Across goals, action sequences, and across levels in hierarchical variants
Information / model requirement	Informative reward and adequate interaction/logged-data coverage	Usable model and task objective; validity over planned rollouts	Differentiable objective and valid FoM over optimized rollouts; fixed-goal cases also require a known target

Interpreting the benchmark comparisons. In the fixed-goal benchmarks, the target is a constant parameter defining the reward, rather than an episode-specific solution label. Direct access to a known, observable target nevertheless provides stronger task structure than sampled scalar reward alone. Our benchmark results compare complete methods with different information/model interfaces. Neither the model-mediated gradient pathway nor any individual amortization axis is exclusive to the Inverter.

Operational reductions and overlaps. Supplementary Table 6 distinguishes an exact objective reduction, a temporal special case, overlapping optimization mechanisms, and changes to the algorithm. Training the same sequence architecture only on expert action labels gives exact BC. Retaining a distal objective alongside an action-fidelity anchor, as in AntMaze, instead gives a hybrid. A DDPG-style update requires replacing the FoM-mediated objective with a Bellman-trained critic.

Supplementary Table 6: **Operational reductions and overlaps of inverse learning.** The conditions distinguish exact behavior cloning from a hybrid objective, single-step distal control, shared model-based optimization, per-query trajectory optimization, and replacement by an actor–critic update.

Modification or limiting condition	Resulting method or relation	Interpretation
Remove the distal/FoM objective; train the controller only on expert action/sequence labels	Behavior cloning	Exact BC objective for the same sequence architecture
Retain the distal objective but let an action-matching loss dominate	BC-like hybrid	Limiting case; the weak AntMaze anchor remains a hybrid rather than exact BC
Set $T = 1$ and train from explicit distal outcome error through a frozen FoM	Single-step distal supervised control	Temporal special case following Jordan and Rumelhart [4]
Use a reward/return objective $\mathcal{J}$ and optimize the controller through learned dynamics	Differentiable model-based policy/trajectory optimization	Conceptual and algorithmic overlap rather than a disjoint family
Replace Inverter evaluation with per-query optimization of $a_{1:T}$ through the same FoM	Differentiable trajectory optimization; receding-horizon control if re-solved online	Non-amortized algorithmic relation
Replace the FoM-mediated objective with a Bellman-trained critic and current-action actor update	DDPG-style actor–critic	Different update mechanism, not a parameter-limit reduction

A.2.1 Relationship to differentiable predictive control

Differentiable predictive control (DPC) [15] is a close predecessor to the inverse-learning construction. It trains a sequence-producing neural controller through a previously learned, fixed state-space model, using tracking, smoothing, and constraint penalties without optimal-action labels. Deployment evaluates the controller with receding-horizon feedback, avoiding a numerical optimal-control solve per query. Learned dynamics, sequence output, direct model gradients, and feedforward controller evaluation are therefore shared ingredients. Our work investigates composition across levels of action abstraction, extended action-chunk execution, direct multi-step model architectures, and their behavior in navigation, hierarchical game control, and quantum gate synthesis. The demonstrated hierarchies compose separately trained modules.

A.2.2 Limitations of amortization and deployment-time iteration

Amortization trades training effort and approximation error for lower computation within each control query. Deployment-time search or refinement can adapt a candidate to the current observation and objective, at additional computational cost; its effectiveness still depends on model and objective quality. Predicted horizon, executed chunk length, computation within a query, and online learning are distinct choices. Our disturbance experiment (App. A.10) demonstrates the value of feedback between chunks, not a comparison against iterative optimizers.

Iterative visuomotor action generation. Diffusion Policy [33] generates action sequences through iterative denoising and executes them with receding-horizon feedback. Its representation supports multimodal actions conditioned on visual observations. Denoising samples a learned action distribution; it need not optimize the same explicit task objective as the Inverter. These representational and observational differences matter alongside inference cost.

Search and adaptation. SAILOR [54] combines learned world and reward models with deployment-time search and distills search outcomes into a diffusion policy. MPAIL2 [55] learns from observations and online interaction using world-model planning and inverse reinforcement learning. DSRL [56] uses reinforcement learning to adapt a frozen diffusion policy through its latent noise. Together, these methods address recovery under distribution shift and online adaptation beyond our fixed-goal offline setting, motivating combinations of amortized control, search, and learning.

A.2.3 Combinations with reinforcement learning and optimal control

Inverse learning also admits three complementary combinations with reinforcement learning and optimal control. These are directions for future evaluation.

Learned value or cost-to-go. A learned value/cost-to-go can enter $\mathcal{J}$ to account for consequences beyond the modeled chunk. This component could be trained by temporal-difference learning, adding value approximation and its associated error to the current finite-horizon model-mediated objective.

RL-selected high-level goals. RL can select high-level goals or subgoals while an Inverter realizes the corresponding low-level action chunks. Reward-driven selection could address which outcomes are desirable, while the lower-level controller would still receive an explicit task condition. Learning suitable and reachable subgoals remains part of this proposed combination.

Warm starts and policy improvement. An Inverter sequence can initialize model predictive control (MPC) or model predictive path integral (MPPI) optimization, or supply a behavior prior or initial policy for subsequent RL improvement. An optimizer can then refine the proposed sequence for the current query. The MPC/MPPI hybrid retains the amortized initializer but adds deployment-time optimization; its complete control query is no longer a single feedforward pass. Measuring the resulting quality–latency trade-off would require a separate evaluation.

A.2.4 Known-target applications beyond navigation

Adaptive optics and metamaterial inverse design offer prospective hierarchies with specified task objectives and intermediate targets beyond spatial waypoints. These applications are not evaluated in this paper.

Adaptive optics. A slower Level 2 Inverter could map a scientific-image objective, such as suppressing stellar light, and the current image to a requested wavefront. A faster Level 1 Inverter could then map the wavefront target and current optical state to a sequence of mirror adjustments. The scientific objective is specified, while selecting useful intermediate wavefront targets remains an upper-level task.

Metamaterial inverse design. A Level 2 Design Inverter could map a desired device-level shape or mechanical response to a spatial field of effective material properties. Level 1 Inverters could map local property targets to manufacturable unit-cell designs. This extends the hierarchy to inverse design: the outputs are design parameters rather than temporal action chunks. Validating the forward models and feasibility of the intermediate targets remains future work.

A.3 Experimental details for `maze2d-umaze-v1`

The `maze2d-umaze-v1` benchmark from D4RL [38] is a continuous-control navigation task in which a point mass must reach a goal position in a U-shaped corridor. The offline dataset consists of undirected trajectories generated by a PD controller navigating between random waypoints; these follow the angular U-shaped geometry of the maze walls, visible in the heatmap in Figure 2.

Our final FoM is a chunked Transformer FoM with chunk length $T = 16$, $d_{\text{model}} = 128$, 4 attention heads, 4 layers, and 847k parameters, trained for 300 epochs with Gaussian state noise $\sigma = 0.01$. We then freeze this model and train the final Transformer IWM for 2000 epochs through it, using segment length 16, $d_{\text{model}} = 192$, 6 heads, 4 layers. The IWM maps $(s_0, s_g) \mapsto a_{1:128}$ in one forward pass. **Training goal sampling**: each minibatch sample is a (s_0, s_g) pair with $s_0 = s_t$ drawn uniformly from the offline buffer and $s_g = s_{t+H}$ taken at a fixed horizon offset $H = 128$ ahead in the same trajectory; the sampler is episode-aware (pairs that would cross an episode boundary are rejected). This matches the deployment query distribution: at test time the Path Inverter emits sub-goals at `wp_spacing` = 6 m on `maze2d-medium/large` and the goal directly on `umaze`, both well within one IWM-horizon of typical dataset displacement.

Maze2D IWM training objective. Let $\hat{a}_{1:H} = g_\phi(s_0, s_g)$ be the IWM output, $\hat{s}_{1:H} = f_\theta^{(1:H)}(s_0, \hat{a}_{1:H})$ the segmented FoM rollout (the rollout is chained across $\lceil H/T \rceil$ chunks of length

$T = 16$ so the gradient flows end-to-end through the frozen FoM), and $\hat{p}_t = \hat{s}_t^{(xy)}$ the predicted positions. We train g_ϕ to minimize

$$\mathcal{L}_{\text{IWM-2d}} \;=\; \lambda_{\text{term}} \left\|\hat{s}_H - s_g\right\|^2 \;+\; \lambda_{\text{dense}} \left[-\tfrac{1}{H}\sum_{t=1}^{H} \exp\!\big(-\|\hat{p}_t - p_g\|\big)\right] \;+\; \lambda_{\text{bnd}}\, \tfrac{1}{H}\sum_{t=1}^{H} \big(1 - O(\hat{p}_t)\big), \quad (3)$$

with $\lambda_{\text{term}} = 0$, $\lambda_{\text{dense}} = \lambda_{\text{bnd}} = 5$ in the published runs (the terminal term is absorbed into the dense reward via the exponential profile and was set to 0 on `umaze`). $O : [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \to [0, 1]$ is a precomputed support map of the offline data, queried at the FM-rolled-out positions by bilinear interpolation (`F.grid_sample`, differentiable in $\hat{p}_t$). Construction: discretize the data's (x, y) extent into a 64×64 grid, count visits per cell, apply the chosen *occupancy mode* (`binary`: $\mathbb{1}[\text{count} > 0]$ in published runs, or `frequency`: $\log(1 + \text{count})$), Gaussian-smooth with $\sigma_{\text{cells}} = 1.5$ for sub-cell gradients, and rescale to $[0, 1]$. Because the offline corridors are far from the walls, $1 - O$ is essentially zero on the data manifold and rises sharply outside it; the IWM is therefore penalized only when the FM rolls its plan into wall regions absent from the training distribution.

Loss-balance toggle. The Table 24 entry "boundary mode: {binary, z-score}" is unrelated to the occupancy mode above; "z-score" refers to a separate, orthogonal balance toggle in which each raw loss term ($\mathcal{L}_{\text{term}}, \mathcal{L}_{\text{dense}}, \mathcal{L}_{\text{bnd}}, \mathcal{L}_{\text{fid}}$) is divided by its running EMA standard deviation (momentum 0.99, 100-batch warm-up) before its λ weight is applied – so λ operates on unit-variance signals. The published `maze2d` runs leave this toggle off (raw losses) and use `binary` occupancy.

At evaluation time we consider two execution modes for the same final IWM checkpoint. In *one-shot* mode, the controller executes the generated 128-step action sequence open-loop and only requests a new plan once that horizon has been consumed. In *replanning* mode, it executes the first $K = 64$ actions, observes the new state, and replans if the goal has not yet been reached.

Our comparison uses CORL implementations [39] for BC, TD3+BC [57], CQL [58], IQL [59], AWAC [60], ReBRAC [61], and SAC-N [62]. Diffuser [26] is trained and evaluated with its separate implementation. Local measurements, published CORL scores, and maze-specific evaluation settings are distinguished below.

A.4 Timing measurement protocol

This appendix describes the timing measurements behind Table 8 and distinguishes per-pass latency from the inference budget of a full episode.

Hardware and software. All timings are measured on a single GPU (cuda:0) with PyTorch in `eval` mode under `torch.no_grad`. ReBRAC is the sole exception: its inference is measured from JAX+JIT code and is therefore marked with † in Table 8. MuJoCo stepping uses the default single-thread CPU implementation.

Per-pass wall time. For Maze2D Inverters, forward calls are timed with `time.perf_counter()` and `torch.cuda.synchronize()` on both sides, following 10 discarded warm-up calls. Diffuser uses three warm-up planning calls and CUDA synchronization around the complete policy call. Each Diffuser call generates 64 trajectories; evaluation follows the first, with one planning call per episode. Its reported ms/NN-pass is the complete planning-call time divided by the number of denoising steps (64 on U-Maze; 256 on medium/large). Run-level JSON records supply the scores and timing summaries assembled by `maze2d/timing/make_table2.py`.

Per-episode accounting. We decompose an episode's wall time into four additive components: (i) NN forward passes, (ii) *other* algorithmic overhead – Python control loop, tensor prep, `.cpu().numpy()` transfers, replan dispatch (the Inverter) or the PD tracker (Diffuser), (iii) `env.step`, and (iv) a residual of a few milliseconds for episode-level bookkeeping. Table 8's *sum* column reports $(i) + (ii)$, i.e. the algorithm's inference-compute budget. For Diffuser, component (i) is the complete policy-call time, including its planning overhead. Environment stepping and framework glue are excluded on purpose: we want a quantity that is invariant to the simulator and portable to real-robot deployment, where `env.step` is replaced by physical dynamics.

Fixed evaluation protocol. All methods are evaluated on exactly 100 episodes of `maze2d-umaze-v1` with a fixed benchmark goal, reproducible initial states that vary across episodes, and a hard cap of 300 environment steps per episode. Inverter numbers average over 4 independently-trained seed IWMs (and their matching FMs).

Per-pass latency and chunk amortization. The measurements in Table 8 give 2.67–3.36 ms per Inverter forward across the listed chunk lengths, including 3.19 ms at $K = 128$. For that configuration, three calls contribute 9.57 ms per 300-step episode; adding 1.87 ms of dispatch overhead gives the reported 11.4 ms inference budget. This budget is computed from the recorded planning-cycle time after subtracting the optional visualization FoM rollout; it excludes simulator stepping. The benefit comes from amortizing each call over an action chunk: a step-wise actor requires 300 calls for the same episode.

A.5 Detailed performance tables (`maze2d-medium/large` and `antmaze`)

This appendix collects the per-method performance tables for `maze2d-umaze-v1` (Table 8) referenced in Sec. 4.1, `maze2d-medium-v1` (Table 9) and `maze2d-large-v1` (Table 10) referenced in Sec. 4.2, and for the six `antmaze-v2` variants (Table 11) referenced in Sec. 4.3. Table 7 below summarizes the per-task improvement of the Inverter over the strongest reported offline-RL or diffusion-planner baseline on each variant, both in absolute D4RL-score points and as a percentage of the baseline, with the row-wise mean across all 9 tasks.

Supplementary Table 7: **Per-task D4RL improvement of the Inverter over the strongest reported offline-RL or diffusion-planner baseline, summarized across all 9 `maze2d/antmaze-v2` variants.** Baselines per variant: SAC-N / Diffuser / AWAC are the per-maze winners on `maze2d` (Table 2); ReBRAC is used as the comparator on every `antmaze-v2` variant (Table 3). Δ pts = Inverter $-$ baseline; Δ % = Δ / baseline. AntMaze Inverter scores use the data-only evaluation in App. A.13; differences use unrounded Inverter means. Final row: mean across all 9 tasks (D4RL points and %, equally weighted).

Task	Baseline	Baseline	Inverter	Δ pts	Δ %
`maze2d-umaze-v1`	SAC-N	130.6	**161.6**	+31.0	+23.7
`maze2d-medium-v1`	Diffuser	130.1	**166.8**	+36.7	+28.2
`maze2d-large-v1`	AWAC	209.1	**220.7**	+11.6	+5.5
`antmaze-umaze-v2`	ReBRAC	97.8	**98.8**	+1.0	+1.0
`antmaze-umaze-diverse-v2`	ReBRAC	83.5	**99.5**	+16.0	+19.2
`antmaze-medium-play-v2`	ReBRAC	89.5	**94.8**	+5.3	+5.9
`antmaze-medium-diverse-v2`	ReBRAC	83.5	**99.0**	+15.5	+18.6
`antmaze-large-play-v2`	ReBRAC	52.2	**92.0**	+39.8	+76.2
`antmaze-large-diverse-v2`	ReBRAC	64.0	**93.8**	+29.8	+46.5
Mean (n=9)				**+20.7**	**+25.0**

Protocol for `maze2d-medium/large`. Same D4RL-official protocol as Table 8: 100 episodes at the fixed target, per-episode body-position seed, 600-step cap on medium, 800-step cap on large. CORL numbers are taken from the Tarasov et al. [39] "Last Scores" benchmark (we did not train our own medium/large CORL checkpoints); per-pass times come from our local umaze benchmark (same actor architectures) and are scaled by the target maze's step count for the *sum* column. Diffuser scores and timings are measured locally after 2M-step training runs on the respective datasets, using the final saved checkpoints (stored step 1,999,000). Each checkpoint is evaluated on 100 episodes with seeds $777 + i$ ($i = 0, \ldots, 99$); score spreads are standard deviations across these episodes. The medium/large horizons are 256/384, with 256 denoising steps and 64 sampled trajectories per planning call. Evaluation follows the first trajectory. The *other* column (per-episode non-NN overhead) is $\sim$27 ms on medium and $\sim$36 ms on large because the algorithmic Path Inverter runs a 4-connected BFS on a $\sim$100-cell grid once per replan event and occasionally performs a full re-plan when the stuck-check triggers (App. A.8); even so, this is an order of magnitude below the MLP cost of every step-wise baseline.

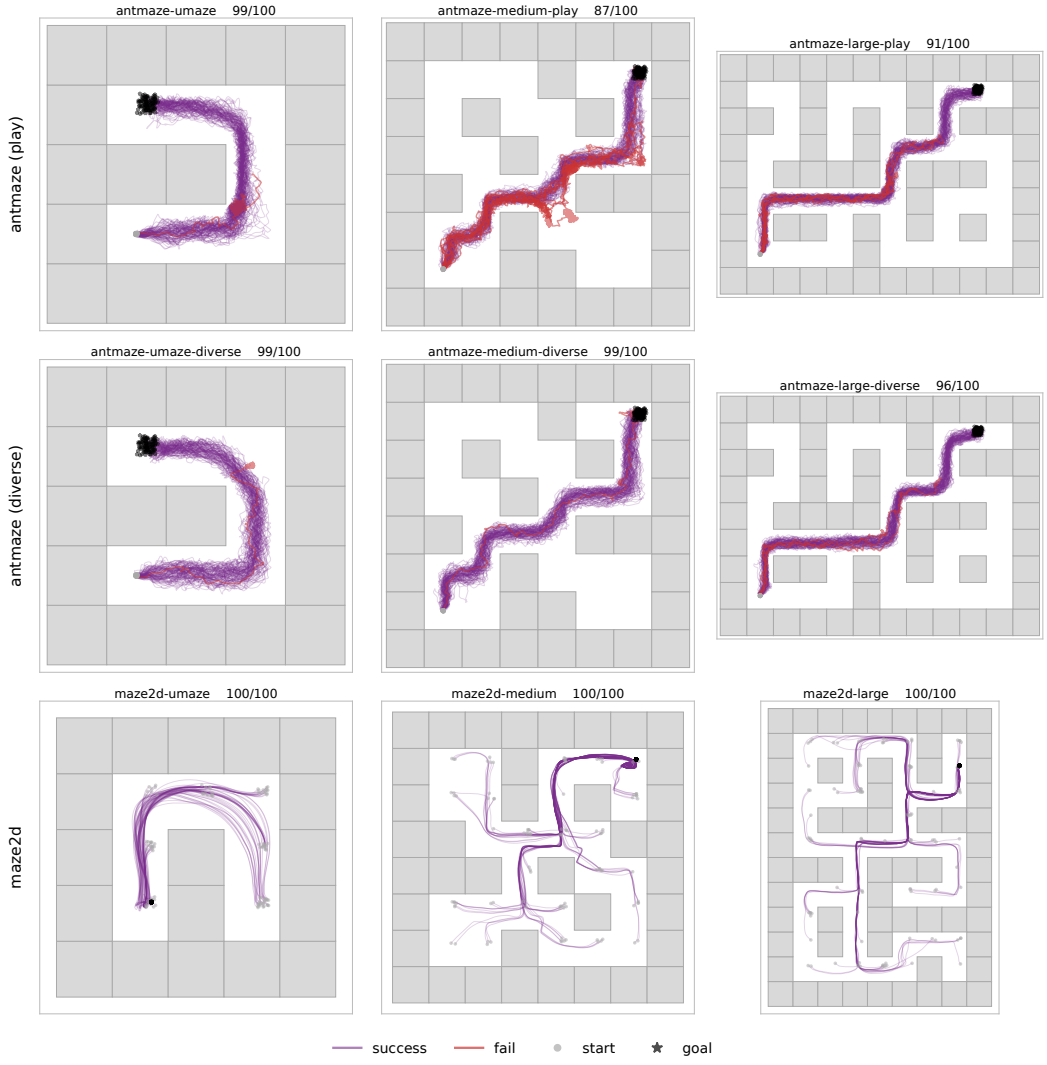


Figure 6: **Per-episode trajectory overlays for the Inverter on every maze variant we evaluate.** 3×3 grid: rows = (`antmaze` play / `antmaze` diverse / `maze2d`); columns = small / medium / large maze. Each AntMaze panel shows all 100 episodes, including failures, from seed 42 of the data-only evaluation used for the headline scores. Titles report this seed's success count, rather than the four-seed mean. Maze2D panels retain their original evaluation runs. Walls are shown for visual context and are not inputs to AntMaze routing. Inverter color matches the per-method palette used in Fig. 3. AntMaze goals lie in the benchmark's corner goal region; Maze2D panels show varying starts and the fixed benchmark target for each maze.

Supplementary Table 8: **Per-episode inference compute on `maze2d-umaze-v1`** (100 episodes, 300 steps/ep). Each cell reports the number of NN forward passes per episode, mean wall time per pass (cuda-synced, GPU, PyTorch; batch 1 except Diffuser: 64 trajectories per plan), *other* — per-episode non-NN overhead inside the algorithm (Inverter: per-chunk replan dispatch only (no sequencer on this maze); Diffuser: the PD tracker running at every env step; step-wise RL baselines: 0), the *sum* = #NN-passes × ms/NN-pass + other, and × slower relative to our fastest configuration on this maze. † JAX+JIT; all others use PyTorch. Diffuser score spreads are across 100 episodes of one checkpoint. Inverter rows are mean ± std over 4 seed IWMs; the forward model is skipped at deployment (only the IWM is charged). ‡ DecisionLLM sum is an order-of-magnitude estimate from the paper.

Method	score ↑	std	#NN-passes	ms/NN-pass	other	**sum**	× slower
BC [39]	0.36	8.69	300	1.70 ms	—	511.3 ms	44.7×
CQL [58]	−8.90	6.11	300	1.84 ms	—	551.2 ms	48.2×
BC-10% [39]	12.18	4.29	300	1.25 ms	—	374.9 ms	32.8×
AWAC [60]	82.67	28.30	300	1.28 ms	—	382.5 ms	33.4×
IQL [59]	42.11	0.58	300	1.69 ms	—	508.4 ms	44.4×
TD3+BC [57]	29.41	12.31	300	1.67 ms	—	502.1 ms	43.9×
ReBRAC [61]	106.87	22.16	300	0.21 ms†	—	63.0 ms†	5.5×
SAC-N [62]	130.59	16.52	300	1.80 ms	—	539.3 ms	47.1×
EDAC [62]	95.26	6.39	300	1.74 ms	—	522.1 ms	45.6×
DT [25]	18.08	25.42	300	3.57 ms	—	1.07 s	93.7×
Diffuser [26]	116.32	34.70	64	6.64 ms	3.0 ms	427.8 ms	37.4×
DecisionLLM [40]	145.20	35.32	300	33.30 ms	—	10.00 s‡	873.9×
Inverter $K=16$	**164.25**	**0.54**	19	3.36 ms	9.3 ms	**73.2** ms	**6.4**×
Inverter $K=32$	**164.77**	**0.34**	10	2.67 ms	4.3 ms	**31.0** ms	**2.7**×
Inverter $K=64$	**165.19**	**0.80**	5	3.14 ms	2.9 ms	**18.7** ms	**1.6**×
Inverter $K=128$	**161.64**	**2.17**	3	3.19 ms	1.9 ms	**11.4** ms	**1.0**×

A.6 Why minimum-time control remains bang-bang under viscous damping

To interpret the smoother IWM trajectories in Figure 2, it is useful to separate the geometry of the *state trajectory* from the structure of the *control signal*. Consider the idealized one-dimensional point-mass model

$$\dot{x} = v, \qquad \dot{v} = -\beta v + u, \qquad |u| \le u_{\max}, \qquad \beta > 0, \quad u_{\max} > 0, \tag{4}$$

where β is a viscous damping coefficient. This captures bounded actuation and linear damping while abstracting away maze walls, goal radii, and replanning. We consider the minimum-time transfer from an initial state (x_0, v_0) to the terminal state $(0, 0)$.

This is a standard time-optimal control problem for a linear system with bounded inputs, so the bang-bang principle applies [63, 64]. A short derivation follows from Pontryagin's maximum principle. With running cost 1, the Hamiltonian is

$$H = \lambda_1 v + \lambda_2(-\beta v + u) + \lambda_0. \tag{5}$$

We use the maximizing convention, with $\lambda_0 \le 0$ and multipliers not all zero. Since the terminal time is free, $H = 0$ along an optimal trajectory. Because H is affine in u, the maximizing control must lie at the boundary of the admissible set whenever $\lambda_2(t) \neq 0$:

$$u^*(t) = u_{\max}\ \mathrm{sgn}(\lambda_2(t)). \tag{6}$$

The costates satisfy

$$\dot{\lambda}_1 = 0, \qquad \dot{\lambda}_2 = -\lambda_1 + \beta\lambda_2, \tag{7}$$

so λ_1 is constant and $\lambda_2(t)$ has the form

$$\lambda_2(t) = c_1 e^{\beta t} + c_2. \tag{8}$$

Supplementary Table 9: **Per-episode inference compute on `maze2d-medium-v1`** (100 episodes, 600 steps/ep). Each cell reports the number of NN forward passes per episode, mean wall time per pass (cuda-synced, GPU, PyTorch; batch 1 except Diffuser: 64 trajectories per plan), *other* — per-episode non-NN overhead inside the algorithm (Inverter: replan dispatch + the data-derived BFS sequencer; Diffuser: the PD tracker running at every env step; step-wise RL baselines: 0), the *sum* = #NN-passes × ms/NN-pass + other, and × slower relative to our fastest configuration on this maze. † JAX+JIT; all others use PyTorch. Diffuser score spreads are across 100 episodes of one checkpoint. CORL baselines have no trained `medium` checkpoint in our local benchmark; per-pass time is measured on `umaze` (same actor architectures) and the *sum* column scales it by 600 environment steps. Diffuser scores and timings are measured locally after 2M-step training runs, using the final saved checkpoints (stored step 1,999,000). Inverter row is mean ± std over 4 seed IWMs (eval seed 42); *other* here is the per-episode cost of the data-derived cardinal-BFS sequencer (Appendix A.8).

Method	score ↑	std	#NN-passes	ms/NN-pass	other	**sum**	× slower
BC	0.79	3.25	600	1.70 ms	—	1.02 s	14.0×
BC-10%	14.25	2.33	600	1.25 ms	—	749.8 ms	10.3×
DT	31.71	26.33	600	3.57 ms	—	2.14 s	29.4×
IQL	34.85	2.72	600	1.69 ms	—	1.02 s	14.0×
AWAC	52.88	55.12	600	1.28 ms	—	765.0 ms	10.5×
EDAC	57.04	3.45	600	1.74 ms	—	1.04 s	14.3×
TD3+BC	59.45	36.25	600	1.67 ms	—	1.00 s	13.8×
CQL	86.11	9.68	600	1.84 ms	—	1.10 s	15.1×
SAC-N	88.61	18.72	600	1.80 ms	—	1.08 s	14.8×
ReBRAC	105.11	31.67	600	0.21 ms†	—	126.0 ms†	1.7×
Diffuser	130.07	22.70	256	14.54 ms	14.5 ms	3.74 s	51.3×
Inverter $K=16$	**166.82**	**1.20**	38	1.62 ms	11.2 ms	**72.9** ms	**1.0**×

The switching function cannot vanish identically: this would force $\lambda_1 = 0$, and $H = 0$ would then force $\lambda_0 = 0$, contradicting nontriviality of the multipliers. An exponential plus a constant can cross zero at most once, so the optimal control is saturated almost everywhere and has at most one switching time. Thus, viscous damping changes the switching condition, but not the bang-bang nature of the minimum-time input.

The switching surface can be written in closed form. Suppose the state is already moving toward the origin with $v > 0$, and apply maximal braking $u = -u_{\max}$. The velocity reaches zero after

$$t_{\text{brake}}(v) = \frac{1}{\beta} \log\left(1 + \frac{\beta v}{u_{\max}}\right), \tag{9}$$

and the distance traveled before stopping is

$$d_{\text{brake}}(v) = \frac{v}{\beta} - \frac{u_{\max}}{\beta^2} \log\left(1 + \frac{\beta v}{u_{\max}}\right). \tag{10}$$

Therefore the switching curve for transfer to $(0, 0)$ is

$$x = -\operatorname{sgn}(v) \left[\frac{|v|}{\beta} - \frac{u_{\max}}{\beta^2} \log\left(1 + \frac{\beta |v|}{u_{\max}}\right)\right]. \tag{11}$$

For a start at rest away from the goal, the optimal policy applies maximal input toward the goal until it hits this curve, then brakes maximally to arrive at the target with zero velocity. In the limit $\beta \to 0$, Eq. (11) reduces to the familiar undamped switching parabola $x = -\frac{1}{2u_{\max}} |v| v$.

This derivation should be interpreted as a *local straight-segment model*, not as an exact theorem for the full `maze2d` benchmark. The benchmark only requires entry into a finite goal region, not arrival with zero terminal velocity, and the full maze introduces walls and path constraints. In that setting the exact optimal law can differ, and the braking phase may disappear if first arrival to the goal set is all that matters. The main point is narrower: once the dynamics are approximated by a damped point mass with bounded control, smooth spatial trajectories remain compatible with bang-bang minimum-time inputs.

Supplementary Table 10: **Per-episode inference compute on `maze2d-large-v1`** (100 episodes, 800 steps/ep). Each cell reports the number of NN forward passes per episode, mean wall time per pass (cuda-synced, GPU, PyTorch; batch 1 except Diffuser: 64 trajectories per plan), *other* — per-episode non-NN overhead inside the algorithm (Inverter: replan dispatch + the data-derived BFS sequencer; Diffuser: the PD tracker running at every env step; step-wise RL baselines: 0), the *sum* = #NN-passes × ms/NN-pass + other, and × slower relative to our fastest configuration on this maze. † JAX+JIT; all others use PyTorch. Diffuser score spreads are across 100 episodes of one checkpoint. CORL baselines have no trained `large` checkpoint in our local benchmark; per-pass time is measured on `umaze` (same actor architectures) and the *sum* column scales it by 800 environment steps. Diffuser scores and timings are measured locally after 2M-step training runs, using the final saved checkpoints (stored step 1,999,000). Inverter row is mean ± std over 4 seed IWMs (eval seed 42); *other* here is the per-episode cost of the data-derived cardinal-BFS sequencer (Appendix A.8).

Method	score ↑	std	#NN-passes	ms/NN-pass	other	**sum**	× slower
BC	2.26	4.39	800	1.70 ms	—	1.36 s	14.6×
BC-10%	11.32	5.10	800	1.25 ms	—	999.7 ms	10.7×
CQL	23.75	36.70	800	1.84 ms	—	1.47 s	15.7×
DT	35.66	28.20	800	3.57 ms	—	2.86 s	30.5×
IQL	61.72	3.50	800	1.69 ms	—	1.36 s	14.5×
ReBRAC	78.33	61.77	800	0.21 ms†	—	168.0 ms†	1.8×
EDAC	95.60	22.92	800	1.74 ms	—	1.39 s	14.9×
TD3+BC	97.10	25.41	800	1.67 ms	—	1.34 s	14.3×
Diffuser	123.07	62.22	256	14.32 ms	17.0 ms	3.68 s	39.3×
SAC-N	204.76	1.19	800	1.80 ms	—	1.44 s	15.4×
AWAC	209.13	8.19	800	1.28 ms	—	1.02 s	10.9×
Inverter $K=16$	**220.66**	**0.20**	50	1.59 ms	14.3 ms	**93.7** ms	**1.0**×

A.7 Directional movement spectrum and trajectory curvature

Figure 7 provides a geometric decomposition of trajectory quality by examining both the *direction spectrum* of agent motion and quantitative curvature measures. The training data distribution is visibly axis-aligned: the point mass predominantly moves along x or y, a consequence of the U-shaped corridor layout. BC faithfully reproduces this distribution. Somewhat surprisingly, several strong offline RL baselines (TD3+BC, IQL, AWAC, ReBRAC) are *more* axis-constrained than the training data–these methods narrow the action distribution around the dominant headings seen during training, likely as a result of pessimistic value regularization pulling the policy toward high-density data regions. CQL, SAC-N, and Diffuser introduce more diagonal motion (exploiting directions not covered by training demonstrations), but still exhibit pronounced axis-aligned peaks.

The Inverter's broad distribution of displacement directions is consistent with the curved, shortcut-taking paths in Figure 2. Its low median peak curvature and curvature variance describe the spatial geometry of those paths, while speed is measured separately in Figure 3G. During training, gradients flow through the frozen FoM across the entire action sequence, permitting coordinated changes to the predicted trajectory. The observed combination of smooth geometry, high speed, and successful navigation is consistent with effective control over complete action chunks.

A.8 Simple algorithmic Path Inverter (Level 2): data-driven BFS over offline-data density

For the larger `maze2d` and `antmaze` layouts, a single Inverter chunk is no longer enough to reach the goal from an arbitrary start: corridors contain multiple 90° turns and the longest shortest path through free space exceeds the IWM's training horizon. We therefore couple the unchanged Inverter, which emits action chunks, with a trivial algorithmic Path Inverter (BFS on offline data) at Level 2, which emits intermediate sub-goals $c_1, c_2, \ldots$ along a feasible corridor, and let the Inverter drive the agent chunk-by-chunk toward the current sub-goal. The key constraint we impose on the Path Inverter is that it must be buildable from the *offline training data alone*

Supplementary Table 11: **AntMaze: D4RL score and per-step inference compute. Top**: D4RL success-rate score (%) over 100 episodes on each `antmaze-v2` variant (CORL baselines [39]: mean ± std over 4 training seeds; Inverter: mean ± population std over seeds 42–45, each on the same 100 frozen cases with data-only BFS (App. A.13); best-per-column in bold). **Bottom**: × slower per env step against the fastest method on each maze, measured on a single A40 GPU (PyTorch, batch 1, cuda-synced). For step-wise actors ms/step = ms/pass (the NN runs once per env step) and is env-independent; ms/pass values are reported in Table 8. For the Inverter, ms/step = (#NN-pass × ms/pass + BFS-sequencer overhead) / mean steps-to-goal, averaged across the 4 IWM seeds. † ReBRAC: JAX+JIT timing. Best-per-column (**1.0×**) bolded.

	D4RL score (%) on antmaze-v2					
Method	u-umaze	u-divrs	m-play	m-divrs	l-play	l-divrs
BC	55.2 ± 4.2	47.2 ± 4.1	0.0	0.8 ± 0.8	0.0	0.0
BC-10%	65.8 ± 5.3	44.0 ± 1.0	2.0 ± 0.7	5.8 ± 9.4	0.0	0.8 ± 0.8
TD3+BC	70.8 ± 39.2	44.8 ± 11.6	0.2 ± 0.4	0.2 ± 0.4	0.0	0.0
AWAC	57.8 ± 10.3	58.0 ± 7.7	0.0	0.0	0.0	0.0
CQL	92.8 ± 1.9	37.2 ± 3.7	65.8 ± 11.6	67.2 ± 3.6	20.8 ± 7.3	20.5 ± 13.2
IQL	77.0 ± 5.5	54.2 ± 5.5	65.8 ± 11.7	73.8 ± 5.5	42.0 ± 4.5	30.2 ± 3.6
ReBRAC	97.8 ± 1.5	83.5 ± 7.0	89.5 ± 3.4	83.5 ± 8.2	52.2 ± 29.0	64.0 ± 5.4
SAC-N	0.0	0.0	0.0	0.0	0.0	0.0
EDAC	0.0	0.0	0.0	0.0	0.0	0.0
DT	57.0 ± 9.8	51.8 ± 0.4	0.0	0.0	0.0	0.0
Inverter (ours)	**98.8** ± 0.4	**99.5** ± 0.5	**94.8** ± 4.5	**99.0** ± 0.0	**92.0** ± 4.2	**93.8** ± 2.5

	× slower per env step on antmaze-v2					
Method	u-umaze	u-divrs	m-play	m-divrs	l-play	l-divrs
BC	12.0×	12.1×	12.8×	13.0×	12.7×	12.7×
BC-10%	8.8×	8.9×	9.4×	9.5×	9.3×	9.3×
TD3+BC	11.8×	11.8×	12.6×	12.7×	12.5×	12.5×
AWAC	9.0×	9.1×	9.6×	9.8×	9.6×	9.6×
CQL	13.0×	13.0×	13.8×	14.0×	13.7×	13.7×
IQL	11.9×	12.0×	12.7×	12.9×	12.6×	12.6×
ReBRAC	1.5×†	1.5×†	1.6×†	1.6×†	1.6×†	1.6×†
SAC-N	12.7×	12.8×	13.5×	13.7×	13.4×	13.4×
EDAC	12.3×	12.3×	13.1×	13.3×	13.0×	13.0×
DT	25.1×	25.3×	26.8×	27.3×	26.6×	26.6×
Inverter (ours)	**1.0×**	**1.0×**	**1.0×**	**1.0×**	**1.0×**	**1.0×**

– no access to the maze geometry, the simulator, or the occupancy grid the data was generated from. A single implementation drives both `maze2d` and `antmaze`; we use it through the same Python class (`common.cardinal_bfs_planner.CardinalBFSPlanner`), only swapping a handful of task-dependent hyperparameters. Figure 8 shows the result on `maze2d-medium-v1`, `maze2d-large-v1`, `antmaze-medium-play-v2`, and `antmaze-large-play-v2`: the log-density of the training data reveals the corridors implicitly, and the Path Inverter's output chain turns at exactly those corners.

Construction (`maze2d` setup). We describe the planner once on `maze2d`, then mark the few task-specific changes for `antmaze` below. The Path Inverter is built in five stages:

(i) *Data-derived occupancy grid.* We discretize the offline dataset's qpos trajectories into a 2-D grid and call a cell *free* if it received at least τ training points. Two choices make this work in practice: **Density-aligned origin**: the sub-cell phase is not chosen with `floor(min)` but by locating the dominant peak of the 1-D marginals modulo the cell size, so corridor centerlines sit exactly at cell centers; without this alignment, a cell can straddle two adjacent corridor peaks and two walls, making the "free" test arbitrary, and the signal-to-noise ratio of the occupancy map jumps by an order of magnitude when it is enabled. **Noise-floor threshold** $\tau = 2000$ visits per 0.5 m cell: true corridor cells in the `maze2d-medium/large` datasets receive 10^4–10^5 visits, while cells that are *wall-adjacent* but see the dataset's random exploration brushing against them receive only 10^1–10^3

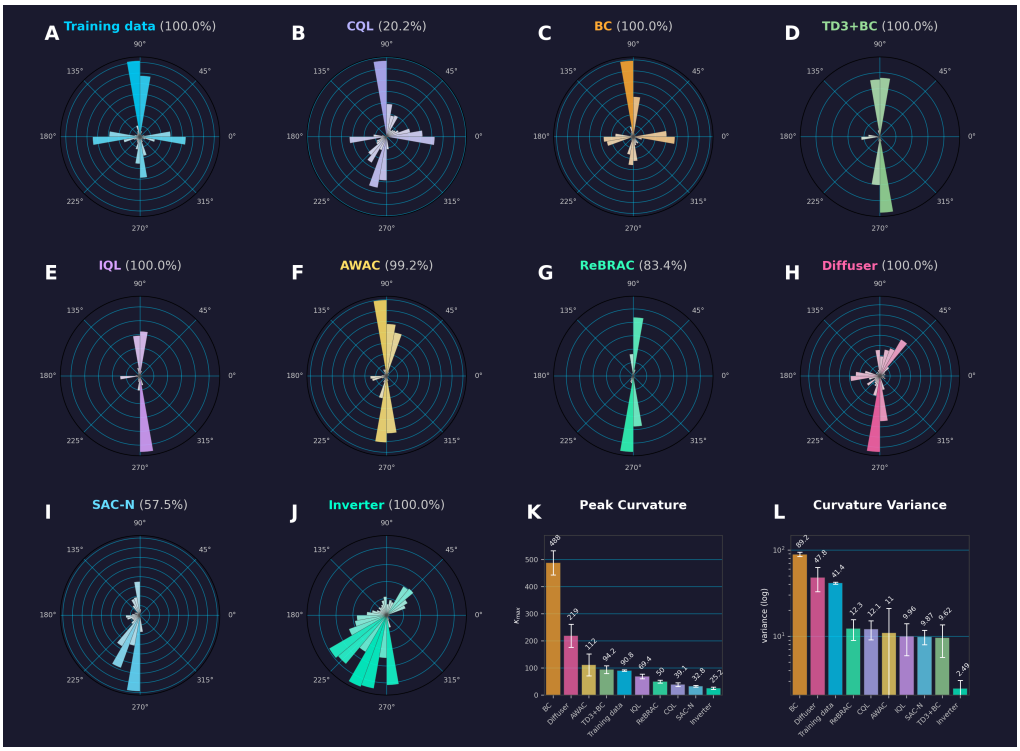


Figure 7: **Directional movement spectrum and trajectory curvature across methods.** Panels A–J show polar histograms of per-chunk displacement directions (chunk length $T = 16$, ≥ 300 rollout steps per method); each wedge's magnitude reflects the fraction of agent steps moving in that direction. Panel K shows median peak curvature $\kappa_{\max}$, and Panel L shows median curvature variance (log scale), both with SEM whiskers over 300 steps per method. *Training data:* movement is mostly axis-aligned (along x or y), reflecting the U-shaped corridor geometry. *BC* closely copies this axis-aligned pattern. *TD3+BC, IQL, AWAC, ReBRAC:* exhibit an *even more restricted* directional spectrum, concentrated on fewer headings. *CQL, SAC-N, Diffuser:* explore some diagonal motion not present in the training data, but still show prominent axis-aligned peaks. *Inverter (ours):* produces the broadest directional distribution across all methods, with substantial off-axis movement, consistent with the smooth curved trajectories visible in Figure 2. The IWM combines broad directional coverage with the lowest median peak curvature (Panel K) and, by a large margin, the lowest median curvature variance (Panel L) among the compared methods. These geometric properties accompany its high measured speed in Fig. 3G.

visits, so $\tau = 2000$ cleanly separates the two populations and is insensitive to small changes. Cell "centers" are the density-weighted mean qpos of the points in that cell, so corners land on the corridor centerline learned from the data rather than on a geometric grid point.

(ii) *4-connected BFS shortest path with centerline tie-break.* Among all equally-short shortest paths from the start cell to the goal cell, we pick the one whose interior cells have the highest cumulative *distance-to-wall* (cardinal taxicab distance, computed once at construction). This is implemented as a single backward DP pass over the BFS layers and remains pure BFS in the sense that the path length is always the minimum cell-step count. Setting the tie-break weight to zero recovers plain BFS.

(iii) *Turn-based polyline.* Rather than emit every visited cell center as a waypoint, we keep only the *corner cells* of the BFS path – the elbows where the cell-index direction changes – plus the start and the goal. This collapses long straight cell-runs to their endpoints and is robust to the small jitter of data-mean centers because direction changes are detected on the cell *indices*, not on the cell-center coordinates.

(iv) *Per-segment perpendicular snap.* Between consecutive turn cells the BFS traveled along exactly one axis (only one of (row, col) changed). Density-weighted centers still jitter on the perpendicular axis, producing tiny diagonal segments. We snap both endpoints of each segment to a shared

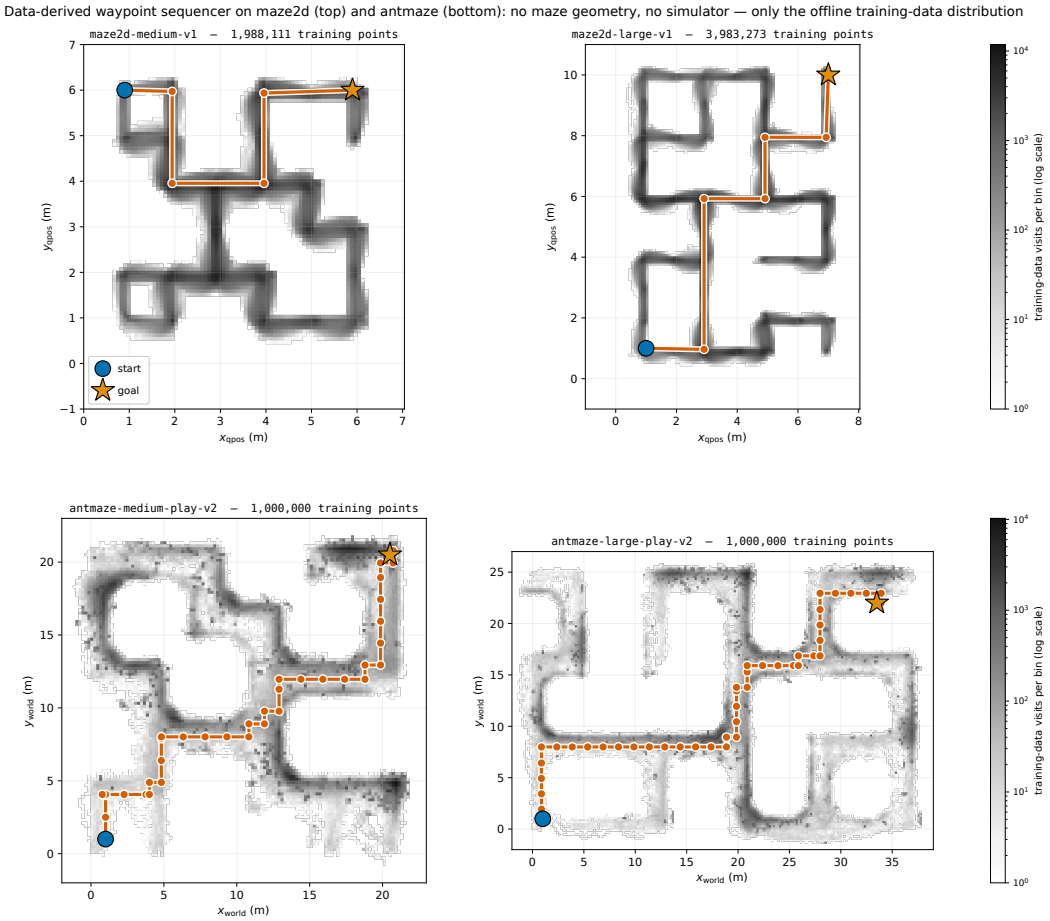


Figure 8: **The simple algorithmic Path Inverter uses only the offline training-data distribution, no maze geometry.** Background: log-density of the training-data states (shared colorbars on the right; top row in qpos coordinates, bottom row in world coordinates). No maze walls or simulator information is drawn or provided to the Path Inverter – the corridors are visible purely because the offline data concentrates there. A cardinal-BFS waypoint chain produced by our data-derived Path Inverter is overlaid in orange (circle markers = intermediate sub-goals); start and goal are shown in blue and orange. **Top row**: `maze2d-medium-v1` (1,988,111 training points) and `maze2d-large-v1` (3,983,273 points), with the Maze2D planner settings (`resolution` $= 0.5$ m, $\tau = 2000$ visits, `min_leg` $= 1$ m). **Bottom row**: `antmaze-medium-play-v2` and `antmaze-large-play-v2` (1,000,000 points each), with the AntMaze planner settings (`resolution` $= 1.0$ m, $\tau = 100$, `min_leg` $= 0$); same code path otherwise. In all four panels the chain turns through exactly those corners where the data concentration itself turns, demonstrating that deployment-time routing can be driven by training-data support alone.

perpendicular coordinate (their average) so the segment becomes exactly axis-aligned; adjacent segments are perpendicular by construction, so their snaps don't conflict at the shared corner cell.

(v) *L-corner insertion and final-approach axis lock.* Two bookend segments still need attention. Any residual diagonal step between consecutive keypoints (above a small tolerance) is split into two perpendicular legs by inserting an explicit L-corner; we always pick the corner that traverses the *longer* axis first, because that is the axis the BFS just traveled along, so the corner stays inside free cells. Finally, if the segment from the last keypoint to the goal is still diagonal, we project the goal onto the preceding axis-aligned segment so the very last step is cardinal. An optional sub-cell wobble filter (`min_leg`, default 1 m on `maze2d`) drops any waypoint whose incoming *and* outgoing legs are both shorter than that threshold, removing staircase artifacts from data-mean snapping.

Per-chunk control loop (`maze2d`). The Path Inverter emits the polyline once at episode start; deployment is a tight loop that consumes the polyline chunk by chunk while continuously re-checking the agent's actual position. At every chunk boundary (every $K = 16$ `env.step` calls) the controller (i) *advances* the waypoint pointer by incrementing `wp_idx` for as long as the agent's current state lies within `wp_advance_dist` $= 0.5$ m of `waypoints[wp_idx]` (one cell of slack), then (ii) sets the IWM target to `waypoints[wp_idx]` (the *current* sub-goal, never the final goal directly), (iii) runs one IWM_low forward pass to obtain a 128-step action plan and executes only the first $K = 16$ actions in the env, and (iv) loops. Crucially the loop does not break on goal reach: with the `-no_pd_hold` flag set, when the agent enters `goal_threshold` $= 0.45$ m the `wp_idx` is already clamped at the last waypoint ($\approx$ the goal cell) and the same loop keeps re-planning, which keeps the agent parked at the goal until `step` $=$ `MAX_STEPS`. No PD tracker is ever invoked – terminal regulation is just continued IWM-driven replanning. The Maze2D headline run uses `wp_skip_dist` $= 0$ (no look-ahead skip) and `target_dist_min/max` $= 0$ (target is the literal waypoint xy, not a randomized offset along the heading); both of these knobs become useful on AntMaze (below) and stay off here.

Stuck-check fallback. Training-data occupancy guides waypoint selection but does not guarantee reachability from each encountered state. In practice the Inverter occasionally overshoots a waypoint or commits to a chunk that hits a wall before the next replan boundary – e.g. when the current state sits within a cell boundary and the plan is parameterized with respect to the wrong neighboring sub-goal. We therefore add a conservative stuck-check fallback: if the agent fails to advance to the next waypoint for ν consecutive chunks, we re-run the BFS from the *current* state and replace the remainder of the polyline with the fresh plan. The replan is uncapped and essentially free (a few hundred microseconds of Python). On the reported `maze2d-medium/large` runs ($\nu = 3$, `max_replans` $= 0$ meaning uncapped) the stuck-check fires on every episode at least once and on average 5–13 times; disabling it drops `large` SR from $100/100$ to $93/100$ on the same seeds. A longer chunk length suppresses the fallback ($K = 128$ fires on $25/100$ episodes on `medium` and $40/100$ on `large`), but at the cost of a substantial drop in D4RL score.

What changed for `antmaze`. We invoke the same Path Inverter with a different argument set; the BFS / polyline / snap / L-corner / final-axis-lock pipeline is byte-identical, so changing tasks is a constructor argument, not a code path. The argument differences address AntMaze's smaller, $\sim 4\times$ wider data (coarser grid `resolution` $= 1.0$ m, $\tau = 100$; `min_leg` $= 0$, no sub-cell jitter at this resolution; `data_center_weight` $= 1.0$ for a centerline tie-break, since wide corridors leave many equally-short BFS paths and the high-dim ant tracks the midline more reliably than the edges) and the ant's slower per-step displacement and wider body (chunk-level operating point $K = 16$, `wp_advance_dist` $= 1.5$ m, `stuck_threshold` $= 10$, ≤ 5 replans/episode; `wp_skip_dist` $= 0.8$ m for look-ahead waypoint skip when the wide ant body has cleared a corner before BFS bookkeeping says so; `target_dist_min/max` $= 1.0/2.0$ m for a carrot-on-a-stick along the heading, keeping the IWM's locomotion reach in its calibrated window even when the agent is right on top of a waypoint). Terminal regulation: `goal_reached_dist` $= 0.5$ m breaks the outer loop rather than re-planning, since the ant naturally settles when the IWM stops issuing new chunks.

What this Path Inverter is *not*. The Path Inverter is not a learned module: it is a plain BFS on a data-derived grid, with no parameters to train and no gradient flowing into it. It is not a model-predictive controller either: no FoM is queried at deployment and no inner optimization loop runs per step. Its only job is to cut the long corridor into chunks that fit inside the Inverter's training horizon, using only the offline data's geometry.

A.9 Replan-horizon (K) sweep on `maze2d-medium-v1` and `maze2d-large-v1`

The headline rows of Tables 9 and 10 report a single operating point ($K = 16$). Table 12 shows how D4RL and wall-clock cost trade off as K varies across $\{16, 32, 64, 128, 256\}$. $K \in \{16, 32, 64, 128\}$ all use the same paper-headline IWM (training horizon $h = 128$); the $K = 256$ row uses a separately-trained $h = 256$ IWM so it genuinely measures a 256-step open-loop commitment rather than collapsing to $K = 128$. Three features stand out: (i) success remains 100% for $K \leq 64$, while D4RL decreases as K increases; both drop sharply at $K = 128$ and further at $K = 256$. (ii) Comparing $K = 128$ with $K = 256$ changes both the trained model and open-loop commitment, so it does not

Supplementary Table 12: **Inverter K-sweep on `maze2d-medium-v1` and `maze2d-large-v1`** (100 episodes per seed, d4rl-official protocol; D4RL and SR reported as mean ± std over 4 IWM seeds). Each row is a single replan horizon K: the Inverter emits a fresh plan every K env steps toward the current data-derived BFS sub-goal. $K \in \{16, 32, 64, 128\}$ uses the same paper-headline IWM (training horizon $h = 128$); the $K = 256$ row uses a separately-trained $h = 256$ IWM so the row genuinely measures a 256-step open-loop commitment instead of collapsing to $K = 128$. *#pass* is the number of IWM forward passes per episode; *ms/pass* is the cuda-synced mean wall time per pass (batch 1); *other* is per-episode overhead outside the IWM forward (the data-BFS sequencer + replan dispatch + CPU transfers); *sum* = #pass × ms/pass + other. Success remains 100% through $K = 64$, while D4RL decreases; both drop at $K = 128$ and further at $K = 256$. The $K = 256$ row also changes the trained model, so it does not isolate the effect of commitment length. Runtime drops monotonically as K grows because fewer IWM calls pay the per-chunk overhead, but the $K \geq 64$ runs are all within one order of magnitude of each other in sum wall time — i.e., accuracy is the binding constraint, not speed.

	`maze2d-medium-v1` (600 steps/ep)						`maze2d-large-v1` (800 steps/ep)					
K	D4RL ↑	SR	#pass	ms/pass	other	**sum**	D4RL ↑	SR	#pass	ms/pass	other	**sum**
16	**166.8** ± 1.2	**100** ± 0	38	1.62 ms	11.2 ms	72.9 ms	**220.7** ± 0.2	**100** ± 0	50	1.59 ms	14.3 ms	93.7 ms
32	157.9 ± 0.8	**100** ± 0	19	1.57 ms	5.5 ms	35.3 ms	202.3 ± 2.8	**100** ± 0	25	1.88 ms	8.7 ms	55.6 ms
64	141.1 ± 1.0	**100** ± 0	10	2.21 ms	3.8 ms	25.9 ms	191.8 ± 3.1	**100** ± 0	13	2.33 ms	5.4 ms	35.7 ms
128	76.0 ± 1.8	84 ± 3	5	2.79 ms	2.6 ms	16.6 ms	95.6 ± 5.2	86 ± 2	7	2.34 ms	2.9 ms	19.3 ms
256	41.7 ± 4.0	32 ± 10	3	2.42 ms	1.5 ms	**8.7** ms	27.5 ± 7.1	27 ± 7	4	2.27 ms	1.7 ms	**10.7** ms

isolate either effect or test execution beyond a model's training horizon. (iii) Inference wall time falls monotonically with K, but the reduction from $K = 16$ to $K = 64$ is less than fourfold. We use $K = 16$ for the headline rows because it gives the highest D4RL among the tested settings.

A.10 Action-noise robustness and replanning frequency

Question and frozen test setting. A long action sequence need not be executed entirely before observing the state again. We vary how many cached actions are executed before querying the same frozen Inverter, isolating deployment-time commitment from the learned prediction horizon. This experiment uses a separate bounded Open9x9 MuJoCo point-mass arena, with state $s = (x, y, v_x, v_y)$ and actions in $[-1, 1]^2$; it is distinct from the D4RL maze benchmarks and the three-level Strategy experiment. One deterministic checkpoint emits $H = 128$ actions per query, without retraining or fine-tuning. The 200 held-out start/goal cases have zero initial velocity and start–goal distance at least 5.5 arena units. They are disjoint from exploratory cases and the 64-case suite used for event-threshold selection.

Paired action disturbances and evaluation. The arena's native slippery-patch noise is disabled. For four consecutive steps, starting at zero-based step 4 or 8, we execute

$$a_t^{\text{exec}} = \text{clip}(a_t + \sigma\epsilon_t, -1, 1), \qquad \epsilon_t \sim \mathcal{N}(0, I_2), \qquad \sigma \in \{0, 0.2, 0.4, 0.6, 0.8\}.$$

Outside this interval the commanded action is executed without the added perturbation. Frozen initial states, goals, onset times, and Gaussian streams are paired across policies and severities; clipping makes the realized perturbation depend on the commanded action. Success means any visit within 0.45 units of the goal. The environment terminates at distance below 0.3, or at the 300-step cap, so success reporting and termination use different radii. Seven periodic policies and six event/hybrid policies, at five severities and 200 cases each, give 13,000 episodes. The completed artifact audit verifies all expected condition–case combinations and exposure to all four disturbance steps for positive severities.

Periodic and one-shot execution. Periodic control re-queries the Inverter from the observed state every $K \in \{128, 64, 32, 16, 8, 1\}$ executed actions. The one-shot condition, denoted $K = \infty$, makes one Inverter query, executes its 128 actions, and then applies a clipped proportional–derivative hold around the observed endpoint without further Inverter calls. If x_H is that endpoint, the hold action is $\text{clip}(x_H - x_t - v_t, -1, 1)$. Thus "one-shot" counts neural queries; it does not mean zero actuation or the absence of all feedback after the learned sequence ends. This convention is specific to this robustness protocol.

Event detection and threshold selection. The event controller compares the observed state with the cached FoM rollout prediction for the same timestep:

$$r_t = \|(s_t - \hat{s}_t) \oslash q\|_2, \qquad q = (3.2896, 3.2985, 6.2064, 6.1923),$$

where $\oslash$ is elementwise division and each scale is the corresponding training-observation interquartile range divided by 1.349. This is a cached rollout residual, not a fresh prediction conditioned on the latest observation. A residual above threshold δ triggers a new plan for the next action. Pure-event control also re-queries when the 128-action cache is exhausted; hybrid control additionally enforces a maximum plan age $K_{\max}$. The detector sees the current state but does not receive the disturbance time or severity.

Thresholds are fixed before test outcomes. Seven candidates are obtained at residual quantiles $.50, .70, .80, .90, .95, .975, .99$. On the 64-case validation mixture across all five severities, we select the candidate whose mean call count is nearest the periodic target budget, breaking ties toward the larger threshold. The $K = 64, 32, 16$ target budgets are $1.600, 5.175, 2.013$ calls; selected event budgets are $2.353, 5.159, 2.353$, with thresholds $2.0513, 0.8981, 2.0513$, respectively. The two low-call targets therefore share a threshold and are imperfect budget matches. All six declared event/hybrid conditions, including the duplicate pure-event policies, remain in the test and in Table 14.

Periodic robustness–compute results. Table 13 reports success and mean Inverter calls for the complete sweep. At $\sigma = 0.8$, $K = 64$ achieves 100% success with 1.70 calls, compared with 51.5% and one call for one-shot execution. Wilson 95% intervals over the 200 cases are $[98.1\%, 100\%]$ and $[44.6\%, 58.3\%]$, respectively. Every-step replanning achieves only 60.5% with 250.53 calls. At that severity, its median, 75th-percentile, and 95th-percentile call counts are all 300. Commitments $K = 64, 16, 8$ achieve 100% success at every tested severity, whereas $K = 32$ achieves 57.5–59.5%.

The nonmonotonicity is already present without injected noise: one-shot success is 61.0%, $K = 32$ is 59.5%, and $K = 1$ is 56.5%, versus 100% for $K = 64, 16, 8$. The entire noisy-condition gap therefore cannot be attributed to repairing disturbance-induced errors. Replanning changes the controller's closed-loop behavior as well as its opportunity to correct errors. Applying the predeclared rule of taking the largest tested commitment within two success-percentage points of the observed maximum selects $K = 64$ at every severity; the largest commitment meeting 95% observed success is $K = 128$.

Supplementary Table 13: **Periodic replanning under action perturbations.** Each cell is success percentage / mean Inverter calls per episode over 200 paired held-out cases; 7 commitments and 5 disturbance levels give 7,000 episodes. K is the number of executed actions between plans; $K = \infty$ uses one plan only. Each perturbation lasts four steps; $\sigma = 0$ is the undisturbed reference. Success is nonmonotonic in replanning frequency, including without perturbations.

Commitment	$\sigma = 0$	$\sigma = 0.2$	$\sigma = 0.4$	$\sigma = 0.6$	$\sigma = 0.8$
$K = \infty$	61.0 / 1.00	54.0 / 1.00	49.5 / 1.00	49.5 / 1.00	51.5 / 1.00
$K = 128$	95.5 / 1.57	96.0 / 1.66	96.5 / 1.71	97.5 / 1.71	96.5 / 1.70
$K = 64$	100.0 / 1.54	100.0 / 1.62	100.0 / 1.70	100.0 / 1.71	100.0 / 1.70
$K = 32$	59.5 / 4.96	57.5 / 5.31	57.5 / 5.59	57.5 / 5.66	58.0 / 5.63
$K = 16$	100.0 / 2.12	100.0 / 2.08	100.0 / 2.10	100.0 / 2.13	100.0 / 2.16
$K = 8$	100.0 / 2.83	100.0 / 2.85	100.0 / 2.88	100.0 / 2.91	100.0 / 2.95
$K = 1$	56.5 / 268.88	59.5 / 266.06	61.0 / 263.25	60.5 / 260.41	60.5 / 250.53

Event-triggered and hybrid results. The $\delta = 0.8981$ event policy improves success over its $K = 32$ comparator by 8.5–12.5 percentage points across severities, at similar realized call counts. At $\sigma = 0.8$, it reaches 70.5% versus 58.0%, using 5.62 versus 5.63 calls. The paired outcomes contain 25 event-only successes and no periodic-only successes (two-sided exact McNemar $p = 5.96 \times 10^{-8}$, unadjusted). The corresponding hybrid with $K_{\max} = 64$ reaches 72.0% at 5.63 calls, with 28 hybrid-only and no periodic-only successes. At this severity, event-call count percentiles (5th/median/95th) are $1/3/14$ for $\delta = 0.8981$ and $1/2/6$ for $\delta = 2.0513$.

The event policies do not dominate the best periodic policies. At $\sigma = 0.8$, even the $\delta = 2.0513$, $K_{\max} = 64$ hybrid has 96.5% success at 2.64 calls, versus 100% at 1.70 for periodic $K = 64$. The

$K_{\max} = 128$ guard reproduces the corresponding pure-event outcomes in these cases. Consequently, adaptive event triggering helps relative to a particular cadence and budget, but is not uniformly preferable to fixed replanning.

Supplementary Table 14: **Event-triggered and hybrid replanning.** Cells report success percentage / mean Inverter calls over the same 200 cases per condition (6,000 episodes). E_j and H_j denote event and hybrid policies whose thresholds were selected to approximate the validation call budget of periodic $K = j$, without test outcomes. Thresholds are $\delta = 2.0513$ for $j = 64, 16$ and 0.8981 for $j = 32$. Hybrid maximum commitments are $K_{\max} = 128, 64, 64$ for H_{64}, H_{32}, H_{16}, respectively. E_{64} and E_{16} selected the same threshold; both evaluated rows are retained.

Policy	$\sigma = 0$	$\sigma = 0.2$	$\sigma = 0.4$	$\sigma = 0.6$	$\sigma = 0.8$
E_{64}	90.0 / 2.42	92.0 / 2.48	92.5 / 2.51	93.0 / 2.48	90.5 / 2.60
E_{32}	71.0 / 5.05	67.5 / 5.61	66.0 / 5.83	66.0 / 5.91	70.5 / 5.62
E_{16}	90.0 / 2.42	92.0 / 2.48	92.5 / 2.51	93.0 / 2.48	90.5 / 2.60
H_{64}	90.0 / 2.42	92.0 / 2.48	92.5 / 2.51	93.0 / 2.48	90.5 / 2.60
H_{32}	72.0 / 5.07	69.0 / 5.63	66.5 / 5.84	66.5 / 5.96	72.0 / 5.63
H_{16}	93.5 / 2.41	95.5 / 2.46	96.5 / 2.51	98.0 / 2.50	96.5 / 2.64

Recovery and continuous-error diagnostics. We additionally compare each disturbed goal-distance trajectory with its paired $\sigma = 0$ trajectory under the same policy, padding a shorter reference with its final value. After the disturbance ends, recovery is the first occurrence of three consecutive states whose excess goal distance is at most 0.10 units, or the first recorded success, whichever occurs first; success already attained by disturbance end counts as immediate recovery. At $\sigma = 0.8$, recovery occurs in 86.5% of one-shot cases, 97.0% for $K = 32$, and 100% for $K = 128, 64, 16, 8, 1$. Among recovered cases, median recovery times are 19, 11, 8, and 6 steps for $K = 64, 16, 8, 1$. Fast recovery relative to the nominal trajectory is therefore insufficient to establish high task success: the nominal $K = 1$ controller itself often fails.

A descriptive compute-aware objective can separate nominal loss, disturbance effects, and realized calls:

$$\mathcal{J}(K; \sigma, \rho) = B(K) + D(K, \sigma) + \rho\, \overline{N}(K, \sigma).$$

Here B is the mean squared goal distance over the nominal rollout, D is the mean squared deviation of the disturbed goal-distance trajectory from its paired nominal trajectory after disturbance onset, and $\overline{N}$ is mean Inverter calls. Trajectories are padded with their final values to the fixed 300-step evaluation horizon. This formulation retains cadence-dependent nominal errors and actual call counts; neither need vary monotonically with K. As a coarse diagnostic, fitting $D \approx a\sigma^2 K^p$ to the 24 finite-K, positive-severity conditions gives $a = 0.00755$, $p = 1.383$ (case-bootstrap 95% interval $[1.279, 1.515]$), and log-space $R^2 = 0.747$. Excluding conditions with more than 10% clipped action steps gives $p = 1.287$ and $R^2 = 0.806$.

Inference cost and limitations. Calls count Inverter invocations, not complete end-to-end runtime. The instrumented evaluator builds a FoM prediction cache for every plan, including periodic plans, although only residual-triggered control requires that cache. A separate uncontended A40 timing audit (batch one, one CPU thread, 20 warm-up calls, 1,000 synchronized planner calls) measures 1.478 ms mean action generation, 13.342 ms FoM cache construction, and 15.010 ms total per plan. Thus event detection has a material forward-model cost. The study covers one frozen checkpoint, fully observable state, a bounded open arena, and four-step action disturbances.

A.11 Comparison to model-based offline RL baselines

The performance tables above (Tables 8–11) compare against the CORL [39] offline-RL benchmark and Diffuser [26]. Reviewers asked whether we should also include published *model-based* offline RL (MBRL) numbers – since our framework itself learns through a FoM, the framing question is fair. This subsection collects what is publicly reported for the canonical MBRL methods on these specific D4RL tasks, with a candid disclosure of two coverage gaps that prevent a direct head-to-head. Citations in this subsection are listed in App. A.22 (separate reference list), so the main paper's bibliography is unchanged.

Coverage gap on `maze2d`. None of the canonical MBRL papers (MOPO [S1], MOReL [S2], COMBO [S3], RAMBO [S4], MOBILE [S5], CBOP [S6], MAPLE [S7], ARMOR [S8], TT [S9], TAP [S10]) report results on the single-task `maze2d-umaze/medium/large-v1` benchmarks in their published evaluation tables; their evaluations focus on D4RL MuJoCo locomotion and (less often) Adroit / NeoRL. The only widely-cited "model-based planner" with published `maze2d-v1` numbers is Diffuser [S11], which we include through local evaluations in Tables 8, 9, and 10. We did not re-run MOPO/MOReL/COMBO/RAMBO/MOBILE/CBOP/TT/TAP on `maze2d-v1` ourselves under our compute envelope (App. A.21); we flag this as a missing baseline rather than fabricate numbers.

Coverage gap on `antmaze`: v0 vs. v2. Every published MBRL number we could locate on `antmaze` is for the `antmaze-v0` datasets, not the `antmaze-v2` datasets that we and the CORL benchmark use. RAMBO's authors explicitly state "we used the AntMaze-v0 datasets" (App. B.7 of [S4]); TAP states "v2 version of the datasets for locomotion control and v0 for the other tasks" (Sec. 6 of [S10]). The v0 vs. v2 distinction matters – v2 corrected reward-shaping and termination handling that affected v0 long-horizon evaluations – so a same-row comparison would not be apples-to-apples. We therefore report the v0 numbers in Table 15 with an explicit footnote, alongside our v2 Inverter row reproduced from Table 11, and leave it to the reader to apply the appropriate caveat.

Supplementary Table 15: **Model-based offline RL baselines on `antmaze`** (D4RL Normalized Score, %). Top block: published MBRL numbers, all on `antmaze-v0` (rows above the rule), as reported in the cited papers. Bottom block: our Inverter, evaluated on `antmaze-v2` (reproduced from Table 11). * AntMaze-v0 vs. v2 mismatch: the v2 datasets we use have corrected reward and termination handling; published MBRL papers as of NeurIPS 2025 evaluate on v0 only, so cross-row comparison is approximate. † COMBO row is the antmaze-v0 baseline reported in RAMBO [S4] (Table 1); the original COMBO paper [S3] does not report AntMaze. § TT result is the TT+Q variant (Q-guided beam search) from [S9] Table 2. ¶ TAP result is the TAP+G variant (goal-conditioned) from [S10] Table 6. ‖ LEQ values are the original LEQ paper's own results [65] from Table 11, averaged over 5 seeds (true termination function). `n/a` entries indicate antmaze-v0 was not reported in the source paper; we list `n/a` for MOPO [S1], MOReL [S2], MOBILE [S5], and CBOP [S6] rather than impute values, as none of these papers report AntMaze in their published evaluation tables. "0.0" entries are reported as such in the source – consistent with the well-known difficulty MBRL methods face on long-horizon sparse-reward tasks.

	D4RL score (%) – antmaze-v0* for MBRL rows					
Method	u-umaze	u-divrs	m-play	m-divrs	l-play	l-divrs
MOPO [S1]	n/a	n/a	n/a	n/a	n/a	n/a
MOReL [S2]	n/a	n/a	n/a	n/a	n/a	n/a
COMBO†	80.3	57.3	0.0	0.0	0.0	0.0
RAMBO [S4]	25.0 ± 12.0	0.0 ± 0.0	16.4 ± 17.9	23.2 ± 14.2	0.0 ± 0.0	2.4 ± 3.3
MOBILE [S5]	n/a	n/a	n/a	n/a	n/a	n/a
CBOP [S6]	n/a	n/a	n/a	n/a	n/a	n/a
TT+Q [S9]§	100.0 ± 0.0	n/a	93.3 ± 6.4	100.0 ± 0.0	66.7 ± 12.2	60.0 ± 12.7
TAP+G [S10]¶	n/a	n/a	78.0 ± 4.1	85.0 ± 3.6	74.0 ± 4.4	82.0 ± 5.0
LEQ [65]‖	94.4 ± 6.3	71.0 ± 12.3	58.8 ± 33.0	46.2 ± 23.2	58.6 ± 9.1	60.2 ± 18.3
Our Inverter (antmaze-v2):						
Inverter (data-BFS, ours)	98.8 ± 0.4	**99.5** ± 0.5	**94.8** ± 4.5	99.0 ± 0.0	**92.0** ± 4.2	**93.8** ± 2.5

Reading the table. Two patterns are visible across the AntMaze suite. First, four canonical MBRL methods (MOPO, MOReL, MOBILE, CBOP) do not report AntMaze in their original published evaluation tables, so the public MBRL coverage on this benchmark is sparse. Among the reported MBRL results in Table 15, TT+Q has the highest means on `u-umaze` and the two `medium` variants (100.0, 93.3/100.0), while LEQ has the highest reported mean on `u-divrs` (71.0). TAP+G is the strongest listed MBRL comparator on both `large` variants (74.0/82.0 on l-play/l-divrs). The Inverter's reported `antmaze-v2` means are 92.0/93.8 on these large variants, giving differences of 18.0/11.8 points relative to TAP+G's `antmaze-v0` means. We emphasize the v0/v2 caveat once more: a clean comparison would require us to re-run RAMBO, TT+Q, and TAP+G on antmaze-v2; the public codebases (`marc-rigter/rambo`, `jannerm/trajectory-transformer`, `ZhengyaoJiang/latentplan`) make this tractable but lay outside our 8-GPU / seven-week compute envelope (App. A.21) for the original submission.

Where TT, TAP, and the Inverter sit on the amortized-vs.-iterative axis. Beyond the headline numbers, TT [24] and TAP [66] differ from the Inverter on two structural axes that organize the related-work paragraph in Sec. 3. (i) *Where the optimization runs.* Both TT and TAP place trajectory optimization at *inference*: TT runs a beam search over per-timestep tokens (state and action dimensions discretized into a vocabulary, with a Q-function added as a search heuristic in the TT+Q variant on AntMaze); TAP runs a beam search over a state-conditioned VQ-VAE's discrete latent action codes (length-L=3 chunks, codebook size K=512). Both therefore retain a sample-time iteration whose cost grows with the search budget. The Inverter folds task optimization into *training* (FM-gradient amortization) and emits the full T-step plan in a single feedforward pass at deployment, with no inner loop. This is the amortized-vs.-iterative axis our related work draws (Sec. 3). (ii) *What the training loss is.* TT and TAP learn trajectory models from offline data and use reward-guided search at deployment; their training objectives do not preclude improvement over the data-generating policy. The Inverter instead trains its sequence-producing network with task-objective gradients through a frozen FoM.

A.12 AntMaze: data, forward and inverse models, and training objective

This appendix gives the technical details behind Sec. 4.3.

Datasets. We use the six `antmaze-v2` variants from D4RL [39, 38]: `umaze`, `umaze-diverse`, `medium-play`, `medium-diverse`, `large-play`, `large-diverse`. Each dataset comprises ~ 1 M ant-locomotion transitions; we serialize them once with `antmaze/extract_data.py` into per-variant `states.npy / actions.npy / next_states.npy / goals.npy` arrays under `data/antmaze/<variant>/`.

State and action encoding. Actions are the standard 8-D joint-torque vector $\mathbf{a} \in [-1, 1]^8$ (8 actuated joints). States are the 29-dim observation supplied by the D4RL env, factored (using Python half-open ranges) as

- $\mathbf{s}_{0:2}$ – root (x, y) position;
- $\mathbf{s}_{2:3}$ – root z (height);
- $\mathbf{s}_{3:7}$ – root quaternion (q_w, q_x, q_y, q_z);
- $\mathbf{s}_{7:15}$ – 8 hip / ankle joint angles;
- $\mathbf{s}_{15:18}$ – root linear velocity $(\dot{x}, \dot{y}, \dot{z})$;
- $\mathbf{s}_{18:21}$ – root angular velocity $(\omega_x, \omega_y, \omega_z)$;
- $\mathbf{s}_{21:29}$ – 8 joint velocities.

Forward model. `AntMazeFull29TransformerFM`: causal transformer that maps $(s_0,\, a_{1:L})$ to $\hat{s}_{1:L}$ with chunk length $L = 16$, $d_{\text{model}} = 384$, 6 heads, 6 layers, ff-mult 4, dropout 0.1, totalling ≈ 11.1 M parameters. Trained on the offline transitions for 300 epochs (batch 512, AdamW with lr $= 3 \times 10^{-4}$, weight decay 10^{-4}, $\sigma_{s_0} = 0.01$ initial-state noise). The training loss is a sum of 7 per-component MSEs (xy, z, quaternion, joint angles, linear velocity, angular velocity, joint velocities), each with unit weight; we use a running-z-score normalizer per dimension (momentum 0.99, 100-epoch warm-up) so the per-component losses share a comparable scale.

Inverse world model. `AntMazeIWM_L16_Healthy29D`: causal transformer mapping (s_0, G_{xy}) to a 16-step action chunk $\hat{a}_{1:16}$, with $d_{\text{model}} = 192$, 6 heads, 4 layers, ff-mult 3, dropout 0.1, totalling ≈ 1.5 M parameters. Trained for 200 epochs (batch 256, AdamW with lr $= 3 \times 10^{-4}$, weight decay 10^{-4}, 10-epoch warm-up) by back-propagation through the frozen FoM.

Single-pass non-autoregressive decoding. The two-token nominal input (s_0, G_{xy}) is first reduced to a single conditioning vector $c \in \mathbb{R}^{d_{\text{model}}}$ by a two-layer MLP (Linear-GELU-LN $\times 2$). An $L = 16$-position token bank is then formed in one shot as $x_t = c + \text{posemb}[t]$ for $t = 0, \ldots, L-1$, where $\text{posemb} \in \mathbb{R}^{L \times d_{\text{model}}}$ is a learned position embedding; the same conditioning is thus broadcast to every output position. These L tokens are processed in parallel by a pre-norm Transformer encoder with a triangular causal self-attention mask (token t attends only to tokens $0..t$), and a per-position linear head followed by $\tanh$ emits one action vector at every position. The whole chunk $\hat{a}_{1:16}$ thus comes out of a single feedforward pass; the "causal" label refers only to the attention pattern – there is no autoregressive sampling, no decoder cross-attention, no iterative refinement. The Maze2D Motor Inverter (App. A.3) shares the same architecture pattern with $L = 128$.

Training goal sampling. Each training minibatch sample is an (s_0, G_{xy}) pair with $s_0 = s_t$ drawn uniformly from the offline buffer and $G_{xy} = (s_{t+L})_{xy}$, the (x, y) of the state at a fixed horizon offset $L = 16$ ahead in the *same* trajectory; the sampler is episode-aware so pairs spanning an episode boundary are rejected. The IWM internally encodes the goal as a displacement $G_{xy} - (s_0)_{xy}$ before its first projection layer, so the network never sees a global (x, y) at input. This L-step-ahead distribution is what aligns training with the deployment query distribution: the Sec. 4.3 Path Inverter emits sub-goals at `wp_spacing` $= 1.5$ m, which is within the typical 16-step ant displacement in the offline data (~ 1.5 m / chunk for `md` ≥ 0.5 m / `z` ≥ 0.3 m filtered states).

Training objective. Let $\hat{a}_{1:L} = g_\phi(s_0, G_{xy})$ be the IWM output, $\hat{s}_{1:L} = f_\theta^{(1:L)}(s_0, \hat{a}_{1:L})$ the FoM rollout, and $a^{\text{data}}_{1:L}$ the corresponding offline action chunk for the same (s_0, G_{xy}) pair. We train g_ϕ to minimize

$$\mathcal{L}_{\text{IWM}} = \lambda_{\text{term}} \left\| \hat{s}_L^{(xy)} - G_{xy} \right\| \; + \; \lambda_{\text{yaw}} \left\| (\hat{s}_{1:L}^{(q_w)}, \hat{s}_{1:L}^{(q_z)}) - (1, 0) \right\|^2 \; + \; \lambda_{\text{fid}} \left\| \hat{a}_{1:L} - a^{\text{data}}_{1:L} \right\|^2, \quad (12)$$

with $\lambda_{\text{term}} = \lambda_{\text{yaw}} = \lambda_{\text{fid}} = 5$ in the published runs. *Note:* the FM's training loss (above) and the IWM's loss in Eq. 12 are distinct losses for distinct networks; the IWM's terminal term is intentionally unsquared, see explanation below. The first term is the FoM-gradient task signal: minimize the predicted final-distance to the goal. The use of the *un-squared* L2 norm here is intentional: $\partial \|x\| / \partial x = x / \|x\|$ has unit magnitude independent of $\|x\|$, so the goal-reaching gradient remains effective whether the sub-goal is 1.5 or 6 m away; a squared $\|x\|^2$ would yield gradient $2x$ that vanishes as the agent approaches the sub-goal and over-weights distant ones, which we found empirically destabilizes training on the longer chunks. The second term is the *ant-happiness* regularizer: it pushes the predicted body orientation across the chunk toward $(q_w, q_z) = (1, 0)$, which corresponds to yaw $= 0$ and the upright orientation overwhelmingly represented in the offline data. The third term is the *BC action-fidelity anchor*: it pulls each predicted action $\hat{a}_t$ toward the recorded data action a_t^{data} at the same *time index* t within the chunk. Since $\hat{a}_{1:L}$ is generated open-loop from s_0, the matching is purely temporal, not state-based — the open-loop predicted states $\hat{s}_t$ will in general drift from s_t^{data}, and the anchor still penalizes deviations of $\hat{a}_t$ from a_t^{data} at every t regardless of that drift. Both regularizers act on bounded per-step quantities, where squared-norm scaling is harmless; the asymmetry between the terminal $\| \cdot \|$ and the regularizer $\| \cdot \|^2$ is therefore deliberate, not a typographical omission. All three terms are per-action, per-time-step, differentiable, and additive – no value function, no sample-time guidance, no KL trust region.

Level 2 substitute (deployment). The data-only Path Inverter follows the Maze2D construction (Appendix A.8: density-aligned origin, noise-floor threshold, turn-based polyline) with one AntMaze-specific addition: among all equally-short shortest paths, we pick the one whose interior cells have the maximum cumulative cardinal distance-to-wall, so the resulting waypoint chain stays on the corridor centerline rather than along the edges. This is implemented as a single backward DP pass over the BFS layers and remains pure BFS in the sense that path length is always the minimum cell-step count. AntMaze-specific hyperparameters: `resolution` $= 1.0$ m (one AntMaze cell), $\tau = 100$ visits, `data_center_weight` $= 1.0$ (centerline tie-break enabled), `min_leg` $= 0$ (sub-cell wobble filter disabled, not needed at this grid resolution), `wp_spacing` $= 1.5$ m, `wp_advance_dist` $= 1.5$ m,

wp_skip_dist $= 0.8$ m, target-distance window $(1.0, 2.0)$ m, stuck_threshold $= 10$ chunks, ≤ 5 full BFS replans per episode.

Per-chunk waypoint-tracking and sub-goal logic (deployment). The BFS produces a polyline $\mathrm{wp}[0:N]$ once at episode start. At every chunk boundary (every $K = 16$ `env.step` calls), the controller executes the following loop, with the current full state $s \in \mathbb{R}^{29}$ and its XY position $p = s^{(xy)}$:

1. **Advance.** While wp_idx $< N - 1$ and $\|p - \mathrm{wp}[\texttt{wp_idx}]\|_2 <$ wp_advance_dist $=$ 1.5 m, increment wp_idx by 1.
2. **Look-ahead skip.** If wp_idx $< N - 1$ and $\|p - \mathrm{wp}[\texttt{wp_idx+1}]\|_2 <$ wp_skip_dist $= 0.8$ m, set wp_idx $\leftarrow \min(\texttt{wp_idx} + 2, N - 1)$ and return to step 1; otherwise proceed to step 3.
3. **Target shaping (carrot-on-a-stick).** Let $\hat{u} = (\mathrm{wp}[\texttt{wp_idx}] - p)/\|\mathrm{wp}[\texttt{wp_idx}] - p\|$. Sample $d \sim \mathcal{U}(\texttt{target_dist_min}, \texttt{target_dist_max}) = \mathcal{U}(1.0, 2.0)$ m and set the IWM target $G_{xy} \leftarrow p + d\,\hat{u}$. This keeps the IWM input distance inside its calibrated locomotion-reach window even when the agent sits on top of a waypoint, where a literal target $\mathrm{wp}[\texttt{wp_idx}]$ would degenerate to zero distance and collapse joint-torque magnitudes.
4. **Plan.** Run one IWM forward pass conditioned on (s, G_{xy}) to obtain a 16-step action chunk $\hat{a}_{1:16}$.
5. **Execute.** Apply $\hat{a}_{1:16}$ to the env (16 `env.step` calls), updating s and $p = s^{(xy)}$.
6. **Stuck check.** If wp_idx has not advanced for stuck_threshold $= 10$ consecutive chunks and the global cap of ≤ 5 replans/episode is not exhausted, re-run BFS from the current p and replace the remaining polyline with the fresh plan.
7. **Terminate.** If $\|p - G_{\text{episode}}\|_2 \leq$ goal_reached_dist $= 0.5$ m, exit the outer loop; otherwise loop back to step 1.

Ordinary advancement in step 1 can pass multiple waypoints at one chunk boundary. Step 2 additionally skips past the next waypoint when its proximity test holds. The carrot-on-a-stick in step 3 is the only deviation from passing the literal current waypoint into the IWM as a target. Both are off (wp_skip_dist $= 0$, target_dist_min/max $= 0$) on `maze2d` where the point-mass dynamics do not require them.

Evaluation protocol. The headline Inverter scores reuse the data-only BFS evaluation of seeds 42–45 in App. A.13: 100 frozen cases drawn from default D4RL resets, shared goals and target-distance streams, and population standard deviations across seeds. Episode caps are 700 steps on `umaze*` and 1000 on `medium*`/`large*`. Timing rows retain the separate May campaign, whose jobs each used one A40 GPU. CORL D4RL scores in Table 11 are taken from Tarasov et al. [39] (mean over 4 training seeds); per-pass time for the PyTorch CORL actors is measured on `maze2d-umaze` (same actor architectures), since per-pass NN cost is architecture-bound, not env-bound. ReBRAC is JAX+JIT (marked †); all other baselines are PyTorch.

Per-method D4RL breakdown. The data-only Inverter+BFS row in Table 11 has higher mean D4RL scores than the published ReBRAC baseline on all six antmaze-v2 variants. The Inverter leads by 5.3 points on `medium-play`, 15.5 on `medium-diverse`, 39.8 on `large-play`, and 29.8 on `large-diverse`. CQL and IQL score below the Inverter on all six variants, with gaps of 6.0–73.3 and 21.8–63.6 points, respectively. On the `medium`/`large` variants, ReBRAC scores 52.2–89.5, CQL 20.5–67.2, and IQL 30.2–73.8; the remaining seven baselines (BC, BC-10%, TD3+BC, AWAC, SAC-N, EDAC, DT) have single-digit scores throughout. On `u-umaze`, ReBRAC scores 97.8 and the Inverter 98.8. These are descriptive comparisons of means; gaps use unrounded Inverter scores.

Per-step compute breakdown. The May four-run aggregates underlying Table 11 give 0.13–0.14 ms per executed environment step across all six variants. For each run, we divide mean per-episode Inverter inference, BFS, and other non-physics overhead by mean executed steps, including failed episodes, then average the four run-level ratios. Rounded to three decimals, the six costs are 0.142, 0.141, 0.133, 0.132, 0.134, and 0.134 ms in table order. The table carries baseline timings from the Maze2D evaluation archive: BC-10% at 1.25 ms/pass and JAX+JIT ReBRAC at 0.21 ms/pass.

With those inputs, the Inverter has the lowest tabulated cost on all six variants, with 8.8–$9.5\times$ lower cost than BC-10% and 1.5–$1.6\times$ lower cost than ReBRAC.

How other paradigms accommodate data-support priors. Auxiliary objectives can introduce interacting trade-offs in both actor–critic methods and inverse learning. Here, BC and body-yaw priors enter as additive training losses alongside the FoM-mediated task objective.

Loss-component ablation. Table 16 quotes the contribution of λ_{yaw} on top of λ_{fid} on `large-diverse-v2`. The two ablated checkpoints share architecture, optimizer, and dataset with the published runs and differ only in the value of λ_{yaw} (0 vs 5); both were evaluated under the canonical Path Inverter config above.

Supplementary Table 16: **Loss-component ablation on `antmaze-large-diverse-v2`.** 100 episodes per seed, canonical waypoint config (`wp_spacing` $= 1.5$, target-distance window 1.0–2.0 m); D4RL reported as mean $\pm$ std over 4 IWM seeds. Removing the body-yaw regularizer drops D4RL by 11 points; the BC anchor alone is not sufficient. We do not separate $\lambda_{\text{fid}} = 0$ configurations on AntMaze because the offline-data generation process is fixed by the benchmark; we instead study the underlying FoM-hacking mechanism in AntMan (Sec. 4.4), where the data distribution is under our control.

Loss configuration on `large-diverse-v2`	D4RL
BC anchor only ($\lambda_{\text{fid}} = 5$, $\lambda_{\text{yaw}} = 0$)	83.0 ± 3.5
BC anchor + body-yaw ($\lambda_{\text{fid}} = 5$, $\lambda_{\text{yaw}} = 5$)	$\mathbf{94.0} \pm \mathbf{5.0}$

A.13 Matched-controller ablation under shared waypoint routing

Question and controlled comparison. The long-horizon navigation results combine algorithmic routing with a learned low-level controller. To distinguish their contributions, we hold the data-derived BFS Path Inverter, waypoint-update rules, goal interface, starts/goals, and evaluation protocol fixed while replacing the Level 1 controller. We compare reactive BC, 16-action chunked BC, IQL, AWAC, ReBRAC, CQL, SAC-N, EDAC, and the Inverter, with BC-10% as a reduced-data diagnostic. Table 4 summarizes the task-family averages; Supplementary Table 17 supplies all eight task results. Shared routing means shared rules, not identical realized waypoint sequences once controllers follow different trajectories. This comparison tests the complete learned controller under shared routing.

Maze2D protocol. We evaluate `maze2d-medium-v1` and `maze2d-large-v1`, with four controller seeds (42, 123, 456, 789) and the same 100 evaluation tasks per seed. Episode i resets with seed $42 + i$ and uses the official fixed target. The state is (x, y, v_x, v_y); the local target is the active waypoint with zero target velocity. The planner updates every 16 steps. Reactive BC and offline-RL actors use the current state at each step, whereas chunked BC and the Inverter execute 16 actions between queries. Chunked BC predicts exactly 16 actions; the Maze2D Inverter predicts a longer sequence and executes its first 16. A full episode has 600 steps on medium and 800 on large, including steps after first entering the 0.45-radius success region. Thus reactive methods use 600/800 controller calls and chunked methods use 38/50.

AntMaze protocol. We evaluate all six `antmaze-v2` variants with controller seeds 42–45. Each seed uses the same 100 frozen simulator states and global goals, with paired random streams for target-distance selection. A local target is chosen every 16 steps using the shared projected-waypoint rules and remains fixed during that block. Controllers receive the current 29-dimensional state and the local goal; the BC/offline-RL input concatenates the state with two target-relative XY coordinates. Reactive methods query every step, while chunked BC and the Inverter query every 16 steps. Episodes terminate on global success (environment success reward or distance below 0.5 m), with caps of 700 steps for the U-Maze variants and 1,000 for medium/large. Scores are success percentages. The Inverter checkpoints are re-evaluated with the same data-derived BFS as every baseline. The AntMaze headline tables use these same primary Inverter scores. Their comparator rows retain published CORL results; the matched-controller tables instead report controllers evaluated with the shared data-derived router.

Data, training, and checkpoint selection. All methods use the same underlying offline data set. Example construction and training splits need not coincide. Maze2D BC replays the BFS rules on logged trajectories to label active waypoints. Reactive BC uses individual transitions; chunked BC uses complete within-episode 16-action blocks, excluding partial tails. The BC split is episode-pure, with 5% validation episodes and split seed 20260505. AntMaze BC and offline RL use eligible 16-step segments: no episode crossing, endpoint displacement at least 0.25 m, and height at least 0.3 m at all 17 states. The recorded endpoint supplies the local goal; reactive examples retain that goal at each offset.

Both BC baselines use two 256-unit ReLU hidden layers with a bounded $\tanh$ action head, action MSE, Adam at 3×10^{-4}, batch size 256, and one million updates. The chunked head emits 16×2 actions in Maze2D or 16×8 in AntMaze; the reactive head emits one action vector. Normalization uses training data only, and checkpoint selection uses the lowest offline-validation action MSE. Offline-RL baselines follow goal-conditioned CORL implementations [39], preserving the same absolute local goal on both sides of the Bellman update, with sparse local-goal rewards (radius 0.5). Maze2D terminals follow prepared episode ends; AntMaze terminals follow the end of the 16-step segment. These baselines use fixed final checkpoints, without online-return selection. Training budgets are method-specific: SAC-N and EDAC use three million updates, versus one million for the other newly trained baselines. AntMaze CQL retains its $10r - 5$ reward preprocessing.

Task results and aggregation. For task means μ_t and seed spreads σ_t, the reported macro-average and propagated spread are

$$\bar{\mu} = \frac{1}{N} \sum_{t=1}^{N} \mu_t, \qquad \sigma_{\text{macro}} = \frac{\sqrt{\sum_{t=1}^{N} \sigma_t^2}}{N},$$

with $N = 2$, 6, or 8 for Maze2D, AntMaze, or the combined comparison. Tasks receive equal weight.

The Inverter has the highest eight-task mean (120.7, versus 107.6 for AWAC) and the highest task mean on six of eight tasks. The exceptions are AntMaze medium-play (SAC-N 99.25 versus Inverter 94.75) and large-play (ReBRAC 93.00 versus Inverter 92.00). The Inverter exceeds both reactive and chunked BC on every task. Chunking alone helps BC in Maze2D but performs poorly in AntMaze; matching the Inverter's call cadence does not remove the advantage of learned inverse control.

Supplementary Table 17: **Matched-controller task results with shared routing. (a)** Maze2D-medium/large-v1 and AntMaze-umaze/umaze-diverse-v2. Each cell is mean ± seed spread over four controller seeds, each evaluated on 100 paired cases. Maze2D uses normalized full-episode return; AntMaze uses success percentage. Spread conventions are specified in the text; they are not confidence intervals. Highest means are bold.

Controller	Maze2D medium	Maze2D large	AntMaze umaze	AntMaze umaze-diverse
Reactive BC	41.21 ± 7.18	194.81 ± 3.31	94.50 ± 2.69	93.00 ± 3.16
Chunked BC	153.51 ± 0.20	203.26 ± 0.70	75.50 ± 4.09	56.50 ± 9.91
BC-10%[†]	91.37 ± 32.35	198.86 ± 6.27	81.50 ± 7.30	83.00 ± 4.30
IQL	67.18 ± 50.52	180.62 ± 38.69	74.00 ± 4.95	87.50 ± 5.89
AWAC	143.93 ± 23.63	201.66 ± 4.21	84.75 ± 4.76	91.00 ± 2.83
ReBRAC	150.87 ± 21.50	121.46 ± 6.87	79.50 ± 15.34	2.00 ± 3.46
CQL	105.31 ± 71.08	155.47 ± 105.00	78.00 ± 8.76	87.00 ± 3.74
SAC-N	77.24 ± 2.23	140.01 ± 6.32	6.00 ± 9.25	0.00 ± 0.00
EDAC	120.18 ± 41.45	170.89 ± 28.29	50.25 ± 24.66	30.75 ± 11.26
Inverter (ours)	$\mathbf{166.82 \pm 1.20}$	$\mathbf{220.66 \pm 0.20}$	$\mathbf{98.75 \pm 0.43}$	$\mathbf{99.50 \pm 0.50}$

Supplementary Table 17: **Matched-controller task results (continued).** AntMaze medium and large variants. Means and task-specific seed spreads use the same conventions as panel (a).

Controller	AntMaze medium-play	AntMaze medium-diverse	AntMaze large-play	AntMaze large-diverse
Reactive BC	93.75 ± 3.27	94.75 ± 0.43	90.75 ± 2.68	91.50 ± 3.20
Chunked BC	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
BC-10%†	74.50 ± 4.15	16.75 ± 13.50	3.25 ± 2.86	65.00 ± 6.04
IQL	88.50 ± 4.03	88.00 ± 3.67	83.50 ± 7.83	84.75 ± 1.92
AWAC	85.75 ± 5.80	88.25 ± 8.17	84.25 ± 2.38	81.25 ± 5.76
ReBRAC	36.00 ± 36.17	13.75 ± 16.13	$\mathbf{93.00 \pm 3.54}$	91.00 ± 9.00
CQL	70.25 ± 4.35	72.25 ± 4.99	59.00 ± 8.37	48.50 ± 7.72
SAC-N	$\mathbf{99.25 \pm 1.30}$	98.75 ± 1.64	34.00 ± 17.85	58.25 ± 22.41
EDAC	22.00 ± 37.53	51.50 ± 29.77	0.00 ± 0.00	1.50 ± 1.66
Inverter (ours)	94.75 ± 4.55	$\mathbf{99.00 \pm 0.00}$	92.00 ± 4.18	$\mathbf{93.75 \pm 2.49}$

A.14 AntMaze target approach and deceleration

Question and matched-query protocol. The action-fidelity anchor in Eq. (12) limits forward-model exploitation, but need not reduce the controller to cloning the demonstrated motion. We compare a 16-action chunked BC controller with the anchored Inverter on the first block of 100 frozen `antmaze-umaze-v2` evaluation cases for each of seeds 43–45 (300 paired queries). Within each pair, both receive the identical 29-dimensional initial state and projected BFS waypoint, and execute 16 actions before another controller query. The Inverter is trained through the frozen FoM with terminal, yaw, and action-fidelity losses; chunked BC is trained to reproduce the demonstrated action sequence. Using the first block avoids comparing queries whose initial states have already diverged under different controllers. Seed 42 supplied the exploratory analysis; seeds 43–45 were held out from that exploration, but the target-reach metrics were expanded post hoc. We therefore report these mechanistic comparisons descriptively.

Target-relative metrics. Let $x_0, \ldots, x_{16} \in \mathbb{R}^2$ be the executed XY positions and g the requested projected waypoint. Define the initial target distance $d = \|g - x_0\|_2$, direction $u = (g - x_0)/d$, and along-target displacement $z_t = u^\top (x_t - x_0)$. The remaining signed distance in Fig. 9A is $d - z_t$; negative values lie beyond the plane through g normal to u. We measure

$$
\begin{aligned}
O &= \max(0, z_{16} - d), \\
\tau &= \min\{t \in \{0, \ldots, 16\} : z_t \geq d\}, \\
D &= z_{16} - z_\tau \quad \text{when crossing occurs.}
\end{aligned}
$$

Here O is *end-block overshoot*, not maximum excursion during the block. The conditional post-crossing quantity D is signed forward displacement, not path length; its mean is computed separately over each controller's crossing cases. Those conditional cohorts need not contain the same paired queries. XY "speed" denotes $v_t = \|x_t - x_{t-1}\|_2$ in metres per environment step, without conversion to metres per second. Late speed averages v_t over steps 13–16; late action magnitude averages the absolute torque command over the same four steps and all eight actuators.

Aggregate target-approach results. Table 18 gives the pooled first-block results underlying Fig. 9A–B. The Inverter reduces mean end-block overshoot by 56.1% and conditional post-crossing forward displacement by 48.5%. Crossings occur in $91/300$ Inverter queries versus $126/300$ BC queries. The overshoot reduction has the same direction in every seed: BC/Inverter means are $0.1268/0.0736$ m (43), $0.1082/0.0480$ m (44), and $0.0606/0.0082$ m (45); conditional post-crossing displacement is also lower in each seed. Figure curves show query means and interquartile bands, not confidence intervals; open bar-chart markers show seed means.

Supplementary Table 18: **Target approach on matched U-Maze first-block queries.** Pooled means over seeds 43–45. Unconditional quantities use all 300 queries per controller; crossing-time and post-crossing means condition separately on the 126 BC and 91 Inverter crossings. "Late" means steps 13–16. These are descriptive means, without a significance claim.

Metric	Chunked BC	Inverter
End-block overshoot O (m)	0.0985	0.0433
Target-plane crossing fraction	42.0%	30.3%
First crossing step τ (conditional)	14.19	14.88
Post-crossing forward displacement D (m; conditional)	0.1699	0.0874
Late XY displacement (m/step)	0.1061	0.1031
Late mean absolute action	0.3506	0.4225

Across all 300 queries, late XY speed differs only modestly (-2.9%), while the Inverter's late action magnitude is 20.5% larger. A separate, outcome-selected subset comprises the 67 pairs where BC crosses the target plane and the Inverter does not. Within this subset, late speed decreases from 0.1343 to 0.0861 m/step (-35.9%), while mean absolute action increases from 0.3746 to 0.4126 ($+10.2\%$). These conditional statistics must not be read as full-cohort effects. Together the observations support target-dependent organization of motion.

Training-data endpoint and true-goal analysis. Chunked BC receives $[s_t, s^{xy}_{t+16} - s^{xy}_t]$ and predicts the recorded actions $a_{t:t+15}$. Its training target is therefore a retrospective endpoint on a

trajectory, not an instruction that the demonstrator should stop there. We examine all $519{,}567$ chunks in the frozen U-Maze training split. The median ratio of last-four-step to first-four-step XY speed is 0.985, consistent with continued locomotion through typical chunk endpoints.

For the separate goal-aligned cohort in Fig. 9C, we find the first entry into the true environment-goal radius ($0.5\,\mathrm{m}$) in each training episode. We retain an arrival only if its preceding 16-step segment belongs to the frozen training split: 570 arrivals qualify, of which 503 have eight observable continuation steps within the same episode. These are true environment goals, distinct from the projected waypoints used in the controller comparison. Writing the arrival time as t_g, a controlled stop requires all three conditions:

$$\frac{1}{4}\sum_{j=0}^{3} v_{t_g-j} \le 0.05\ \mathrm{m/step}, \qquad \|x_{t_g+8} - x_{t_g}\|_2 \le 0.2\ \mathrm{m}, \qquad \sum_{j=1}^{8} v_{t_g+j} \le 0.4\ \mathrm{m}.$$

None of the 503 observable arrivals meets this joint criterion. Mean speed rises from $0.136\,\mathrm{m/step}$ over the first four steps of the preceding chunk to $0.152\,\mathrm{m/step}$ over its final four; the median path length over the next eight steps is $0.887\,\mathrm{m}$. The zero-stop count is conditional on this cohort and these operational thresholds. Isolated low-motion fragments elsewhere in the training data are not ruled out.

Illustrative close-waypoint approach. Supplementary Figure 9D–F adds a selected example to the aggregate evidence. We select the example post hoc from all 300 paired queries: retain cases where the Inverter's mean XY displacement over steps 13–16 is below its mean over steps 5–8, and its final-step displacement is no greater than the late-window mean. Among the 61 eligible queries, choose the smallest endpoint Euclidean distance to the requested waypoint, breaking ties by seed and case. This selects seed 45, case 76, ending $6.5\,\mathrm{cm}$ from the waypoint versus $35.8\,\mathrm{cm}$ for matched BC. The Inverter's middle-to-late mean displacement decreases from 0.163 to $0.054\,\mathrm{m/step}$, compared with 0.170 to $0.130\,\mathrm{m/step}$ for BC. The Inverter remains in motion (final step: $0.045\,\mathrm{m}$); the example illustrates a close, slowing approach rather than a completed stop or representative cohort performance. BC crosses the waypoint plane at step 14; the Inverter does not. The aggregate results in panels A–C provide the population-level evidence independently of this selected illustration. Both trajectories share a rigid translation/rotation, equal axis scales, and the same displacement color scale; values are unsmoothed.

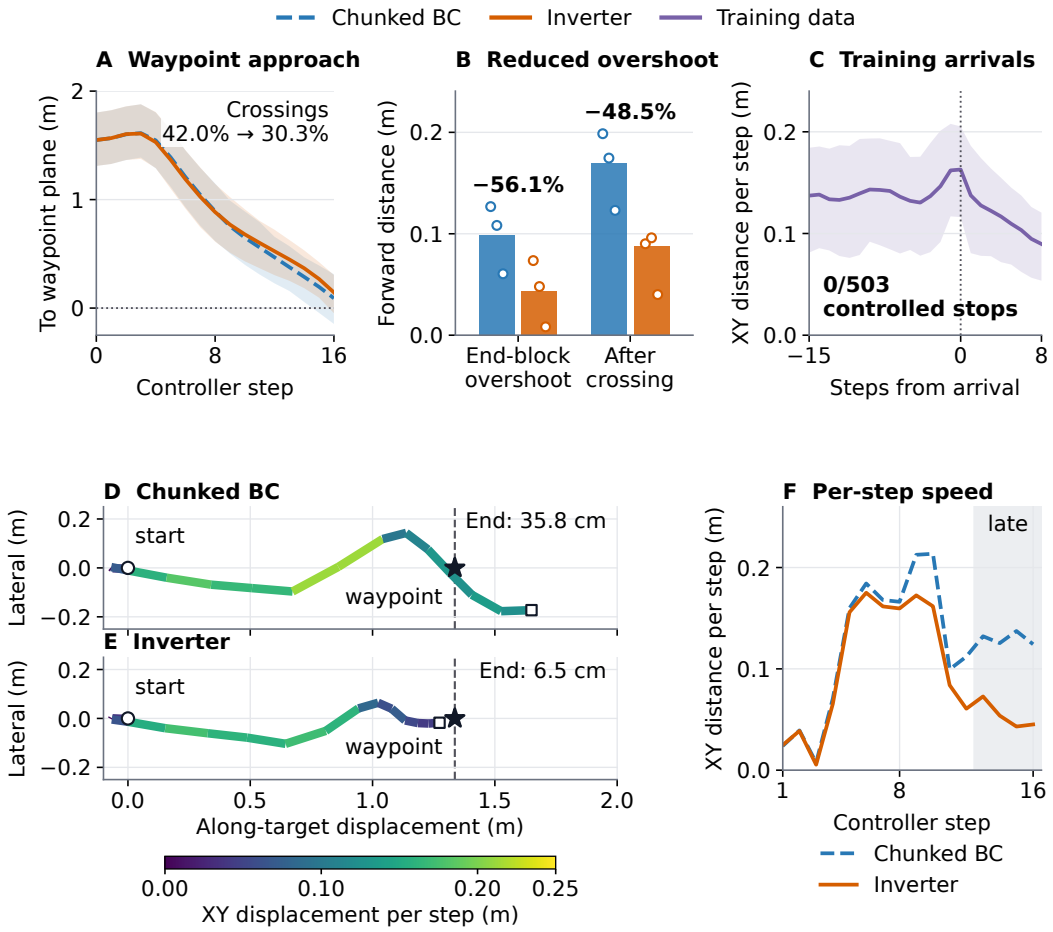


Figure 9: **Target-specific deceleration despite the BC anchor. (A,B)** Aggregate results from 300 paired first-16-step U-Maze queries, seeds 43–45: signed distance to the requested waypoint plane (A), end-block overshoot and forward displacement after first crossing (B). Bars show pooled means; open markers show seed means. Post-crossing displacement is conditional on crossing (BC: 126 cases; Inverter: 91). **(C)** Training traces aligned to first true-goal arrival, with eight subsequent steps available. None of 503 arrivals meets the joint stop criterion: mean speed over the last four approach steps ≤ 0.05 m/step, next-eight-step net displacement ≤ 0.2 m, and path length ≤ 0.4 m. Curves show means with interquartile bands. **(D,E)** Selected matched approach, seed 45, case 76, from identical initial states and waypoints; colors encode XY displacement per step on a shared scale. Both paths share a rigid rotation, equal aspect ratio, and axis limits. Stars mark the waypoint, dashed lines its plane, and squares step 16; endpoint distances are Euclidean. **(F)** Corresponding unsmoothed displacements; shading marks steps 13–16. The example illustrates slowing while still moving; selection is described above.

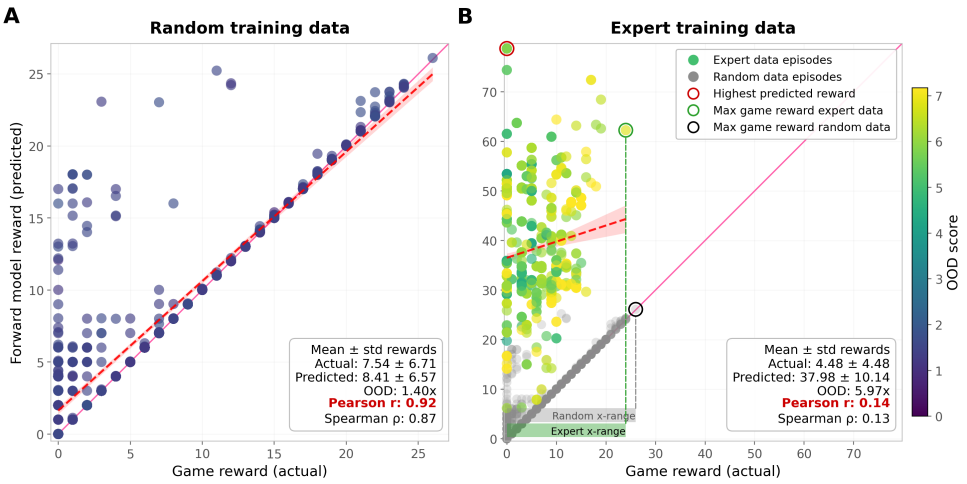


Figure 10: **Random training data yields calibrated forward-model rewards; narrow expert data induces forward-model hacking.** Each panel compares forward-model-predicted reward to realized game reward over 1000 sampled start states for the final high-level IWM checkpoints; point color encodes OOD score (1 = typical train nearest-neighbor distance), the pink diagonal marks perfect calibration, and dashed red lines show linear fits with 95% confidence bands. Panel A (random training data) stays near the training support, has low OOD values, and shows strong correlation between forward-model reward and actual game reward. Panel B (expert training data, without additional stabilizers such as a BC-style loss) shows the opposite failure mode: the Inverter generates highly OOD plans and the FoM assigns them large rewards that are no longer predictive of actual outcomes. The green and gray x-axis projections in Panel B show the realized game-reward ranges for expert-data and random-data plans, with dashed connectors and circles marking the maximum game-reward sample from each group; these ranges are very similar. Thus the failure is not that expert-data training removes high-game-reward plans from the sample, but that forward-model reward becomes decoupled from actual reward. In particular, the highest forward-model-reward sample in Panel B (red circle) achieves zero game reward.

A.15 AntMan forward-model calibration scatter and extended discussion

This appendix shows the predicted-vs-realized reward scatter referenced in Sec. 4.4 and unpacks the implication for inverse-learning data design.

Extended interpretation. The AntMan + Maze2D pair gives a first empirical check of a broader hypothesis running through the paper: inverse learning through a frozen FoM can benefit from training data covering failure modes. In Maze2D, successful trajectories alone do not reveal the wall geometry sharply enough, so we inject a support-derived boundary loss; AntMan asks whether the same principle appears without such preset geometric structure, and the answer is yes – when offline data broaden from expert-skewed to random-mixed interaction, the FoM becomes markedly less exploitable and the inverse policy improves accordingly. The point is not that random data is inherently superior to competent data; it is that inverse planners benefit from data that teach *what fails* as much as *what succeeds*. This highlights a limitation of datasets concentrated on successful behavior. In real systems, unrestricted random control may be unsafe or expensive – the design objective is therefore not random crashes but *safely diverse coverage*: shielded exploration, recovery policies, conservative perturbations, or uncertainty-aware FoM regularization.

A.16 Level-3 Strategy Inverter

Construction and training. We instantiate a three-level stack on `maze2d-large-v1`: the frozen learned Level 1 Motor Inverter produces action chunks, the frozen algorithmic Level 2 Path Inverter supplies a BFS route, and a learned Level 3 Strategy Inverter schedules lower-level calls. Given route/disturbance/compute context and a requested operating point $c_3 = (\alpha, \rho_q, \lambda_{\text{call}})$—reliability, progress, and call price—Level 3 jointly emits all route-leg commitments in one pass.

Each commitment selects $K \in \{8, 16, 32, 64, 128\}$ environment steps between lower-controller calls. Random schedules executed through the frozen lower stack train a macro-FoM to predict success, progress, trajectory loss, final error, disturbance effects, and calls. We freeze this macro-FoM and train the Strategy Inverter through distal outcome error, without schedule labels. A matched model trained on oracle schedule labels serves as an imitation control.

Frozen held-out protocol and metrics. Methods, schedules, seeds, metrics, intervals, and decision thresholds were fixed before test outcomes. Sixteen methods, three requested profiles, four controller checkpoints, and 256 held-out contexts yield 49,152 paired nominal/disturbed cases (98,304 rollouts; all 768 tasks completed). Starts and disturbance streams are paired across methods. Route-relative action noise has standard deviation 0.35 for 96 steps (192 in the long-interval condition), with contexts covering straight segments, before/after turns, and long disturbance intervals. Reported outcomes use the disturbed halves. Each rollout lasts $H = 800$ steps; success means entering the 0.45-unit goal radius at any step. For position x_t, goal G, final error $e_H = \|x_H - G\|_2$, and initial distance $d_0 = \max(\|x_0 - G\|_2, 10^{-8})$, we report

$$q = \mathrm{clip}(1 - e_H/d_0, 0, 1)\left(1 - \frac{\tau}{2H}\right), \qquad L = \frac{1}{H+1}\sum_{t=0}^{H} \|x_t - G\|_2^2,$$

where τ is the first success step, or H if unsuccessful. Thus q combines normalized final progress and arrival time; L measures squared goal distance throughout the rollout. Distances use maze coordinates. Calls count lower-controller forward passes.

Supplementary Table 19: **Requested and realized Level-3 operating points.** Means over 4×256 held-out disturbed cases per profile. Higher success and q are better; lower L, calls, and final error are better. Fixed $K = 8$ is the primary stability reference and does not depend on the requested profile.

Profile	$(\alpha, \rho_q, \lambda_{\mathrm{call}})$	Success	q	L	Calls	Final error
Critical	$(.99, .85, .01)$	1.000	.851	5.538	30.74	.057
Balanced	$(.95, .70, .03)$	1.000	.830	5.643	18.55	.099
Efficient	$(.90, .55, .10)$	1.000	.805	5.991	12.69	.116
Fixed $K = 8$	—	1.000	.839	5.560	100.00	.112

Primary result. All three operating points satisfy their requested reliability and progress targets (Table 19); success is saturated, while progress, loss, error, and calls distinguish the profiles. The critical success Wilson 95% lower bound is .9963, above the requested .99. For the critical profile, a hierarchical paired bootstrap uses 10,000 draws, resampling checkpoints and then cases within checkpoints. All four prospectively fixed criteria pass (Table 20). The primary call reduction is 69.26% relative to fixed $K = 8$; comparison with the validation-selected best fixed strategy ($K = 16$, 50 calls) also favors Level 3.

Supplementary Table 20: **Critical-profile primary comparisons.** Paired 95% bootstrap intervals and the prospectively fixed acceptance bounds. Relative loss and call reduction are means of per-case fractional changes.

Comparison	Estimate	Paired 95% interval	Required bound
Success difference vs. $K = 8$	.000	$[.000, .000]$	Lower $\geq -.02$
Relative loss change vs. $K = 8$	$-.0005$	$[-.0206, .0232]$	Upper $\leq .13$
Call reduction vs. $K = 8$	69.26%	$[65.01\%, 73.37\%]$	Lower $\geq 20\%$
Calls minus best fixed	-19.26	$[-23.37, -15.01]$	Upper < 0

Controls and scope. Call-matched random schedules increase L by 2.13–3.40 across profiles and reduce success, indicating that call placement matters. Removing the requested operating point collapses all profiles to the same $q = .832$, $L = 5.632$, and 18.52 calls. Removing or shuffling context raises loss by .25–.42 or .13–.25; using one commitment throughout an episode raises loss by $.28/.45/.88$ and uses more calls. The oracle-label imitation control has .47–1.09 higher loss and 3.73–13.94 more calls. These controls are descriptive support for the primary comparison. This

experiment demonstrates one functional three-level stack with learned inverse reuse at Levels 1 and 3 and frozen BFS at Level 2.

A.17 Quantum gate synthesis: setup, GRAPE baseline, ablations

Sampling and objective. Haar $\mathrm{U}(2)$ targets and Haar $\mathrm{SU}(2)$ representatives induce the same global-phase-invariant objective. For the computational-subspace channel S_{cc}, let $R = (U^{\mathsf{T}} \otimes U^{\dagger})S_{cc}$. We optimize the exact unconditional average $\bar{F}_{\mathrm{avg}} = \mathrm{Re}[\mathrm{Tr}\, R + \mathrm{vec}(I)^{\dagger} R\, \mathrm{vec}(I)]/[d(d+1)]$, with $d = 2$. The survival term retains leakage; outputs are never renormalized. Six Pauli-axis states independently reproduce the exact average; their fidelity standard deviation is a uniformity diagnostic, not Haar variance.

System and numerical evaluation. Three-level transmon: $\alpha = -4\Omega_{\max}$, $T_1 = 10^4$, $T_\phi = 8 \times 10^3$, $T_{\mathrm{gate}} = 2\pi$ in $1/\Omega_{\max}$ units. Both quadratures obey $|\Omega_{x,y}| \le \Omega_{\max} = 1$ over 80 equal slices. Lindblad propagation uses per-slice matrix exponentials. JAX [67] complex128 propagation uses a degree-18 Taylor kernel with a uniform control-box truncation bound below 10^{-20} per slice. Final pulses are independently Padé-rescored, with SciPy/six-state spot checks; this certifies numerical agreement, not control optimality.

Optimized GRAPE. SciPy L-BFGS-B [68] optimizes physical amplitudes directly within box bounds, using JIT-compiled JAX loss/gradients, float64 controls, Gaussian initialization scale 0.1, and the best evaluated feasible pulse. The quality reference uses 10 restarts$\times$500 iterations, `ftol`=10^{-12}, `gtol`=10^{-9}, without threshold stopping. Timed GRAPE permits 3×200, both tolerances 10^{-9}, stopping at the specified exact-infidelity threshold; failures remain included. Independent checks with QuTiP [69] (version 5.3.1; Qtrl 0.2.0) use Fréchet propagator derivatives and our custom unconditional-Haar fidelity computer, not Qtrl's default objective. The accelerated JAX implementation supplies the timing baseline.

Inverter and selection. GeLU MLPs use three hidden layers of width 128 or four of width 256, stable `trig6` features, and tanh-bounded outputs. Encoding uses complex128 targets; network/features/Adam use float32, physics complex128. Selected parents trained from scratch for 8000 steps (cosine lr $2 \times 10^{-3} \to 10^{-6}$) receive 64000 further updates with fresh Adam, cosine lr $2 \times 10^{-5} \to 2 \times 10^{-7}$, batch 128 (112 Haar, eight near-identity, eight near-Z-axis rotations). Thirty-two continuations cross architectures, two data seeds, and eight recipes: tail weights $0/0.2$, smoothing weights $0/10^{-6}/10^{-5}$, plus Haar-only/low-lr controls. Data seeds share a parent per architecture; they are not independent initialization replications. Validation (2048 Haar/512 structured targets), every 4000 steps, minimizes 0.8 Haar mean+0.1 worst-5% Haar mean+0.1 structured mean. Selected compact/larger models use smoothing $0/10^{-6}$ and no tail penalty. No GRAPE labels or deployment optimization are used. Continuations total 2.048 million updates and 8.42 measured GPU-hours, excluding setup, compilation, and evaluation.

Matched quality (250 targets). Mean infidelity: compact 4.823234×10^{-4}, larger 4.815919×10^{-4}, heavy GRAPE 4.806862×10^{-4}; successes at 5×10^{-4}: 249, 250, 250. GRAPE wins all 250 pairs against either network. Its mean fidelity advantages are 1.637 and 0.906 parts per million, with paired target-bootstrap 95% intervals $[1.513, 1.772]$ and $[0.816, 1.004]$. These intervals condition on fixed models and simulator.

Matched online latency. On 100 fresh targets, three passes use JAX 0.10.0 on an AMD EPYC 7542, CPU affinity 0–3, 64–67 (four physical cores with SMT), and one BLAS thread. We report the median across targets of each target's three-pass median, including failures. At 5×10^{-4}, blocked GRAPE/compact-Inverter medians are 54.438 ms/50.134 μs (1086$\times$); interleaved medians are 51.706 ms/66.555 μs (777$\times$). Successes are 96/100 and 98/100, with 95 common successes. The same returned pulses have mean infidelity $4.947 \times 10^{-4}/4.827 \times 10^{-4}$; these are distinct from the heavy-budget quality comparison. At 10^{-3}, blocked/interleaved ratios are 703 $\times$ /536$\times$, both methods succeeding on all 100. Passes repeat target-specific seeds. Timed scalar calls include target conversion, encoding, bounded-pulse output, dispatch and synchronization; GRAPE includes all target-dependent optimization. Training, compilation, reusable setup and independent scoring are excluded. Both schedules randomize method order; blocked passes receive ten untimed warm calls.

Ratios describe these warm implementations and budgets. A fixed offline gate library amortizes either method's pulse-generation cost.

Precomputation and related work. New operations can still be composed from precomputed elementary gates; direct pulse compilation can reduce the resulting depth and calibration overhead [48]. Dall'Ara et al. [47] optimize random pulse layers; Lipaei et al. [48] learn pulse compilers through differentiable dynamics, with risk-aware training, matched GRAPE benchmarks, and NMR hardware validation.

A.18 Quantum gate synthesis: encoding ablation, robustness sweeps, and two-qubit extension

Waveform trade-offs. Choices were fixed using validation before scoring 4096 fresh Haar targets. Compact/larger mean infidelities are $4.825000 \times 10^{-4}/4.817386 \times 10^{-4}$, 95th percentiles $4.940080 \times 10^{-4}/4.926393 \times 10^{-4}$, and 5×10^{-4} successes $4051/4096$ (98.90%) and $4072/4096$ (99.41%). On 512 separate near-identity/near-Z-axis rotations, successes rise from the parents' $399/446$ to $464/483$. Training tail penalties add the worst-12.5% Haar-batch mean; smoothing averages summed squared adjacent-control differences, normalized by $\Omega_{\max}^2$. A separately validation-selected 256×4 smoother reduces mean Parseval-consistent f_{95} from the parent's 4.946 to 3.336 cycles per natural time unit, with mean infidelity 4.817748×10^{-4} and $4068/4096$ successes. This is a bandwidth/quality choice, not the headline model. On the matched 250, compact/larger/GRAPE mean leakage is $(6.652, 6.345, 5.851) \times 10^{-6}$, six-state fidelity spread $(1.326, 1.355, 1.406) \times 10^{-4}$, and f_{95} $(4.892, 4.868, 3.318)$.

Frozen-pulse device drift. We evaluate the selected nominal models on the same 64 targets at seven settings, varying one parameter at a time; no GRAPE reoptimization enters this test. Here $T_{\text{gate}}/T_1 = 6.28 \times 10^{-4}$. For the compact model, mean infidelity changes from 4.828×10^{-4} nominally to $6.920/4.130 \times 10^{-4}$ at $T_1 = 5000/15000$, $7.555/3.919 \times 10^{-4}$ at $T_\phi = 4000/12000$, and $1.811 \times 10^{-3}/9.801 \times 10^{-4}$ at $\alpha = -3.2/-4.8$. All selected variants improve these endpoint means over their parents, but tails can worsen: the larger smoother's worst error at $\alpha = -3.2$ increases from 0.00683 to 0.00756. Anharmonicity remains a sensitive axis.

Two-qubit extension. This is a four-dimensional qubit-only ZZ model, without transmon leakage levels: $H_{\text{drift}} = (J/2)\sigma_z \otimes \sigma_z$, $J = 0.3$, $T_{\text{gate}} = 4\pi$ ($JT_{\text{gate}} = 3.77$), four bounded drive channels, 80 slices, and the same T_1, T_ϕ per qubit. Exact open-system Haar fidelity on the 100-target inverter reference is 0.95682 and 0.99774 for optimized GRAPE.

A.19 Summary of task-specific adaptations

This subsection inventories the framework components that were specified per task family in this paper, alongside how each could become more general. Three groups: (A) auxiliary loss terms used in `maze2d`/`antmaze`; (B) Level 2 Inverter slot instances; (C) per-task architecture and parameterization. The rightmost column of Tab. 22 maps each item to one of four broad strategy classes (Tab. 21) that organize the route toward more general solutions – operating on the FoM's training data (1), the FoM and Inverter as models (2), the Bolza objective $\mathcal{J}$ (3), or the deployment-time hierarchical / adaptive control structure (4).

A.20 Hyperparameter budget

Our Inverter stack is assembled from two kinds of components. One kind is explicitly *designed to be shared across tasks*: the chunked Transformer FoM and the Transformer inverse-world model (IWM), whose architecture and optimizer choices are counted below. The other kind is explicitly *task-specific*: the IWM's auxiliary loss terms (e.g. the "ant happiness" term in Sec. 4.4) and the Level 2 instance (simple algorithmic Path Inverter for `maze2d`/`antmaze`, Transformer-based Game Inverter for `AntMan`). A natural question is whether this framework saves us hyperparameter budget relative to actor–critic baselines, whose policy and value-learning objectives involve interacting trade-offs.

Counting methodology. For each method we count the algorithmically active hyperparameters in the method's `maze2d-umaze-v1` YAML or `config.json`: learning rates, architectural sizes,

Supplementary Table 21: **Four strategy classes for the path toward more general adaptations.** Each class operates on a different lever of the inverse-learning framework. Classes are complementary, not alternatives: most concrete strategies combine two or more (e.g., the AntMan strategy combines Class 1 with Class 4).

Class	Framework lever	Mechanisms	Status in this paper
1. Data	FoM training distribution	Broaden coverage to expose the dynamics the Inverter is incentivized to explore; safely-diverse exploration scaffolding; active data acquisition near low-density regions.	Demonstrated by AntMan's random-mixed condition (Sec. 4.4, App. A.15).
2. Model	FoM, Inverter, and their I/O representations	Uncertainty-aware FoM (Bayesian / ensemble / conformal); foundation-style pretraining of the FoM-and-Inverter core with task-specific adapter heads; meta-learned initializations / hypernetworks / in-context adaptation; equivariant encodings; analytic+residual FoM hybrids.	Foundation-model direction outlined in the Discussion; meta-learning, equivariance, and residual hybrids open.
3. Objective	Bolza objective $\mathcal{J}$ (bridge to OC and RL)	Meta-learning $\mathcal{J}$ across task families; RL value function as cost-to-go in $\mathcal{J}$; standard OC regularizers (bandwidth, smoothness, energy) replace task-specific loss terms.	Meta-learning outlined in the Discussion; value/cost-to-go combinations in App. A.2.3 remain prospective.
4. Deployment	Hierarchical Inverter slots; chunk-boundary / replan policy	Learned Level 2 Inverter replaces algorithmic ones; adaptive replan conditioned on FoM uncertainty; learned termination predictor; deeper hierarchies (meta-Inverter selecting the subgoal specification, language and vision-language models as a top-layer substrate).	AntMan: learned Levels 1–2. Strategy: learned Levels 1 and 3 with frozen BFS at Level 2 (App. A.16). Broader/deeper hierarchies and meta-Inverters based on language or vision-language models remain open.

loss weights, temperature/entropy/KL coefficients, target-update rates, normalization flags, and any algorithm-specific quantity. We exclude purely bookkeeping fields (seed, device, checkpoint path, project/group names, evaluation-episode counts, eval-frequency) that do not affect the learned policy or its deployment.

The total knob count of our stack is of the same order as the larger offline RL baselines (CQL, ReBRAC). Two structural observations qualify this raw count. First, the Inverter stack factorizes the knobs into three sub-modules (FoM, IWM, Path Inverter / deployment) that are optimized and evaluated independently and share no joint optimizer state. When we change, e.g., the Path Inverter's occupancy threshold for the `large` layout, we do not have to re-tune any FoM or IWM knob. Actor–critic baselines couple policy and value learning, so their hyperparameters can interact. BC and DT use different training objectives. Second, the 12 FoM knobs and 13 of the 17 IWM knobs are *architectural / optimizer* knobs (transformer depth, width, heads, FF multiplier, dropout, batch size, learning rate, epochs, training/validation set sizes, chunk/segment length) that are set once from standard Transformer practice and *not re-tuned per benchmark*. The knobs that are genuinely task-specific are concentrated in a handful of loss-weight and deployment fields listed next.

What we actually tuned. Table 24 lists the hyperparameters optimized. All other knobs in the config files above were fixed at their values at the start of the project (for transformer architecture / optimizer fields, these are CORL/DT-style defaults; the optional loss weights $\lambda_{\text{terminal}}, \lambda_{\text{fidelity}}, \lambda_{\text{action_boundary}}$ default to 0 — i.e. the corresponding loss term is turned off — in the `maze2d-umaze-v1` configuration tracked here, and are explicitly set to non-zero values in the `antmaze` runs as listed in the inventory of task-specific adaptations, Tab. 22). No automated sweeps (grid / random / Bayesian / PBT) were run: all values were chosen by hand from small manual scans on a single seed, using held-out IWM validation loss during training and the D4RL normalized score on 10 eval episodes at deployment.

Supplementary Table 22: **Inventory of task-specific adaptations in this paper.** Group A: auxiliary loss terms. Group B: Level 2 Inverter slot instances. Group C: per-task architecture and parameterization. The rightmost column maps each item to the strategy class(es) defined in Tab. 21.

Component	What is task-specific	Toward more general	Class
A. Auxiliary loss terms			
`maze2d` boundary loss, $\lambda_{\text{boundary}}=5$ (App. A.3)	Support-derived per-step penalty pushing predicted states onto the data manifold near walls.	Broader FoM training data with wall-grazing trajectories; uncertainty-aware FoM regularization (same recipe as AntMan's random-mixed condition).	1, 2
`antmaze` yaw / "ant-happiness" regularizer, $\lambda_{\text{yaw}}=5$ (Eq. 12)	Per-step pull of predicted body quaternion toward upright.	Broader FoM training data including falls and recoveries (AntMan's random-mixed condition is the direct demonstration).	1
`antmaze` BC action-fidelity anchor, $\lambda_{\text{fid}}=5$ (Eq. 12)	Per-step pull of predicted actions toward recorded data actions.	Broader FoM training data; in AntMan, the random-mixed condition removes the need for an action anchor entirely.	1
`maze2d` dense intermediate-goal loss, $\lambda_{\text{dense}}=5$ (App. A.3)	Per-step state-vs-goal supervision over the chunk (denser than terminal-only).	Value / cost-to-go term in the Bolza objective $\mathcal{J}$ (prospective; App. A.2.3).	3
B. Level 2 Inverter slot			
Simple algorithmic Path Inverter (`maze2d-medium/large` + all six `antmaze-v2` variants, App. A.8)	4-connected BFS on a data-density occupancy grid with polyline extraction, perpendicular snap, L-corner insertion, sub-cell wobble filter.	Learned closed-loop Level 2 Inverter (Sec. 4.4, AntMan); the algorithmic Path Inverter is the cheaper slot filling for static corridors with abundant data.	4
Per-chunk dispatch controller between the two Inverter levels (App. A.8)	Hand-tuned thresholds for waypoint progress, target-distance window, stuck-recovery, and goal-region behavior.	Closed-loop Level 2 Inverter re-queried per chunk; broader low-level training data with near-target conditions, or direction-conditioning instead of position-conditioning.	4, 2, 1
Replan-horizon K choice (K=128 `maze2d-umaze` one-shot; K=16 `maze2d-medium/large` + all six antmaze-v2 variants; K=80 pulse slices for quantum, no deployment replan; App. A.9)	Per-task constant value chosen by sweep.	Adaptive K conditioned on FoM uncertainty estimates.	4, 2
C. Per-task architecture and parameterization			
Per-task Inverter / FoM networks (Apps. A.3, A.12, A.17; Sec. 4.4)	`maze2d`/`antmaze`: causal-transformer FoM+Inverter at Level 1 (sized to state/action dim). AntMan: reuses Level-1 AntMaze pair plus a causal-transformer Level-2 FoM and Transformer Level-2 Inverter (32-step waypoint-direction sequence). Quantum: 3–4-hidden-layer MLP Inverter, analytic Lindblad FoM (no learned FoM).	Cross-task pretraining of the FoM-and-Inverter core with task-specific adapter heads (foundation-model pattern); or a meta-learned core (MAML, hypernetworks, in-context adaptation) for few-shot specialization.	2
Quantum: 80-slice piecewise-constant pulse, $\Omega_{\max}\tanh(\cdot)$ squash (App. A.17)	Per-slice Lindblad exponentials with bounded controls (same slices/bounds as GRAPE).	Continuous-time spline / Fourier controls; smoothing penalties evaluated in App. A.18.	3
Quantum: stable `trig6` encoding, 6-dim (App. A.17)	Algebraic rotation-angle/axis features with canonicalization.	Equivariant encodings (Bloch-vector, quaternion, Lie-algebra) that scale to higher-dim systems.	2

Supplementary Table 23: **Number of algorithmically active hyperparameters per `maze2d-umaze-v1` configuration.** CORL baselines counted from `CORL/configs/offline/METHOD/maze2d/umaze_v1.yaml`. Our stack counted from the training and deployment configs checked into the repository. The bottom rows are sub-modules that are trained and evaluated independently of each other.

Method	# HPs	Source
BC / BC-10	7	`bc/maze2d/umaze_v1.yaml` (differs only in `frac`)
AWAC	8	`awac/maze2d/umaze_v1.yaml`
TD3+BC	12	`td3_bc/maze2d/umaze_v1.yaml`
IQL	13	`iql/maze2d/umaze_v1.yaml`
SAC-N	13	`sac_n/maze2d/umaze_v1.yaml`
EDAC	14	`edac/maze2d/umaze_v1.yaml`
DT	18	`dt/maze2d/umaze_v1.yaml`
ReBRAC	20	`rebrac/maze2d/umaze_v1.yaml`
CQL	27	`cql/maze2d/umaze_v1.yaml`
FoM (ours)	12	`scripts/train_maze2d_fm_universal.py`
IWM (ours)	17	`scripts/train_iwm_maze2d_v1.py`
Deployment (ours)	6	`maze2d/navigate_maze2d.py`
Path Inverter (medium / large)	5	App. A.8
Total stack – `umaze`	**35**	FoM + IWM + deployment
Total stack – `medium, large`	**40**	+ Path Inverter

Supplementary Table 24: **Hyperparameters we manually optimized.** "Range explored" is the set of values we ran at least once (evidence: sweep directories checked into `maze2d/umaze/iwm/opt170/{lambda-sweep,noise-sweep}/` and `maze2d/umaze/navigate/`). This is orders of magnitude smaller than a published CORL tuning campaign, which typically explores 10^3–10^5 configurations per method. The stuck-check window was 4 chunks in earlier U-Maze planner evaluations and 3 in the reported Maze2D medium/large runs.

Component	Knob	Final value	Range explored
FoM	state noise σ_{s_0}	0.01	$\{0.005, 0.01, 0.015, 0.02, 0.025, 0.03\}$
FoM	chunk length T	16	$\{16, 32, 64\}$
IWM	λ_{dense}	5	$\{5, 10, 15\}$
IWM	$\lambda_{\text{boundary}}$	5	$\{5, 10, 20, 30, 40, 45, 50, 60, 100\}$
IWM	boundary mode	`binary`	{binary, z-score}
IWM	segment length / horizon	16 / 128	horizon $\in \{64, 128\}$
IWM	epochs	2000	$\{500, 1000, 2000\}$
Deploy	replan chunk size K	reported across all	$\{16, 32, 64, 128\}$ (Tab. 8)
Deploy	goal threshold	0.45 (D4RL default)	fixed
Seq.	occupancy threshold τ	2000 visits / 0.5 m cell	$\{500, 1000, 2000, 5000\}$
Seq.	cell size	0.5 m	$\{0.25, 0.5, 1.0\}$ m
Seq.	origin alignment	density-aligned	{`floor-min`, density-aligned}
Seq.	stuck-check window	3 chunks (Maze2D medium/large)	Recorded settings: $\{3, 4\}$
Seq.	min-leg filter	1.0 m	$\{0, 1.0\}$ m

The comparison between the two tables is intentionally asymmetric. The CORL scores quoted in the main text are the *output* of published hyperparameter campaigns across learning rates, discount factors, entropy coefficients, critic ensemble sizes, and architecture widths; they are seed-averaged with well-tuned means. The Inverter results audited here are *not* the product of any comparable sweep. We read this as a partial validation of the architectural bias: the FoM/IWM decomposition removes many of the trade-offs that RL sweeps discover empirically (value pessimism vs. policy ambitiousness, actor/critic LR ratio, entropy temperature), so a small manual scan suffices to put the method in a competitive regime. It is also a caveat to keep in mind when reading Table 8: a more heavily tuned baseline run on our hardware could plausibly move within its CORL-published band, which we display for reference.

Online tuning budget (in the sense of Kurenkov and Kolesnikov [70], Jackson et al. [71]). A recurring concern in offline RL is that published scores are implicitly supported by large, undocumented pre-deployment online evaluation campaigns used for hyperparameter selection [72, 70, 71]. We therefore give a direct empirical audit of the online-evaluation episodes recorded by the original navigation audit. Each deployment-time evaluation included in this audit produced a run directory with a `monitor.png` summary; by scanning the `navigate/` trees under `maze2d/` and `antmaze/` the original audit enumerated the recorded runs and classified them by hand into four categories. Because a substantial fraction of runs in the first five project weeks crashed before completion (before the stability change of 2026-04-23), we charge each inspected run an average of 50 online evaluation episodes – half of the nominal 100-goal protocol – rather than the full 100.

Supplementary Table 25: **Navigation interaction audit.** One row per category of deployment-time run found in `maze2d/*/navigate/` and `antmaze/*/navigate/`. Runs are classified by manual pattern rules over the run's directory path; see `scripts/audit_online_budget.py` for the exact rules. Episodes charged per run: 50 on average (crash-adjusted).

Category	Runs	Online eps	What this is
Selection	88	4 400	actual hyperparameter choices: λ, noise, horizon, chunk size, Path Inverter thresholds, boundary mode, etc.
Reproducibility / seeds	104	5 200	`repro_*, seedsweep_*, s{123..1000}` reruns – measure seed variance, not pick values
Architecture ablation	49	2 450	CNN vs. Transformer IWM, smooth vs. z-score boundary loss, cardinal-variant design, smoke tests
Final reporting	29	1 450	runs whose numbers appear in the paper's tables
Audited total	**270**	**13 500**	audited `monitor.png`-bearing navigate runs, crash-adjusted

Per-task-cell *selection-only* budgets are:

Supplementary Table 26: **Per-task-cell hyperparameter-selection budget** (crash-adjusted, 50 online eps per inspected run). Sum matches the 4 400-ep total in Table 25.

Task cell	# selection runs	Selection eps
`maze2d-umaze-v1`	24	1 200
`antmaze-umaze-v2`	19	950
`antmaze-large-diverse-v2`	14	700
`antmaze-medium-v2`	8	400
`maze2d-large-v1`	8	400
`maze2d-medium-v1`	7	350
`antmaze-AntMan`	5	250
`antmaze-medium-diverse-v2`, `umaze-diverse-v2`, `large-play-v2`	1 each	50 each
Mean per cell	**$\sim$9**	**$\sim$440**

Where this stands in the literature. Kurenkov and Kolesnikov [70] audit typical published offline-RL work on D4RL and attribute to each (method, task) cell an implicit online tuning budget of $\sim 10^5$–10^6 episodes; our audit gives 50–1,200 estimated episodes per cell, with a mean of approximately 440. This mean is therefore *two to three orders of magnitude below* that range. Paine et al. [72] propose the 10^2–10^3 eps-per-cell regime as the deployment-realistic target – our mean is within this band, while individual cells range above and below it. Jackson et al. [71] argue for the sub-100-pulls regime as the stringent "clean-slate" target for new methods; our mean exceeds that threshold, while some cells fall below it. Finally, the CORL library [39] conventions use $\sim$300–1000 eps per cell for *reporting* alone.

We note the qualitative point that a substantial fraction of the audited online-evaluation episodes ($\sim$9 100 of 13 500) went into seed-variance measurement, architecture ablations, and final reporting rather than hyperparameter selection, reflecting the original seven-week development phase, which was spent *designing and implementing* the Inverter / IWM architecture from concept rather than

fine-tuning an existing one. This contrasts sharply with typical offline-RL campaigns, where the vast majority of online episodes are spent on extensive per-method tuning of an already-fixed algorithm.

A.21 Compute environment

The experiments listed here (Inverter FoM and IWM training, ILHL fine-tuning and reward modeling, the full 8-baseline $\times$ 4-layout $\times$ 3-seed CORL baseline sweep, the `AntMan`-IWM experiments, and every timing measurement in Table 8) were run on a single internal 8-GPU node between **2026-03-18** and **2026-05-06** (50 days of wall-clock project time, or roughly 7 calendar weeks). During this phase, no external cluster, TPU, or cloud GPU was used.

Hardware and software.

- **GPUs**: $8\times$ NVIDIA A40 (GA102GL, 48 GB GDDR6 per card, driver 535.274), all on one PCIe host.
- **CPU**: $2\times$ AMD EPYC 7513 (32 physical cores / 64 threads each; 64 physical cores / 128 threads total).
- **RAM**: 503 GiB DDR4 shared across sockets.
- **Software**: Ubuntu 24.04, Linux 6.8.0; PyTorch + CUDA for navigation; JAX + JIT for quantum and ReBRAC (marked † in Table 8); MuJoCo 2.3 for `maze2d`/`antmaze`, a MuJoCo-based `AntMan` environment for Sec. 4.4.

Representative per-run wall times. A single Inverter FoM + IWM training pair for `maze2d-umaze-v1` took $\sim$7 h on one A40 (300 FoM epochs + 2000 IWM epochs, both at batch 128–512); `maze2d-medium-v1` and `maze2d-large-v1` pairs scale roughly linearly with sequence length and training-set size. A single CORL baseline training on `maze2d-umaze-v1` took 0.5–3 h per seed depending on the method (BC and TD3+BC fastest, CQL and ReBRAC slowest). A single ILHL fine-tune was $\sim$1 min per epoch on the base IWM and ran for 300 epochs. The `AntMan` IWM training runs in Sec. 4.4 were the longest single jobs of that phase at $\sim$12 h each. Deployment (Table 8) is essentially free: a full 100-episode evaluation of any method completes in 2–5 minutes on one GPU.

Rebuttal experiments. Navigation follow-ups used the A40 node described above. Recorded per-run training times were 0.72–1.32 h for AntMaze reactive/chunked BC, 8.80–9.31 h for Maze2D SAC-N, and 11.72–12.21 h for Maze2D EDAC. With eight parallel workers, the noise-robustness sweeps evaluated 7,000 periodic and 6,000 event/hybrid episodes in approximately 11 and 2 minutes, respectively, using a frozen checkpoint. The three-level Strategy Inverter evaluation used all eight A40 GPUs for 98,304 rollouts. AntMaze deceleration analysis reused saved trajectories and offline data, requiring no additional training or simulator evaluations. The quantum follow-up trained 32 continuations on A40 and RTX A6000 GPUs (48 GB per card), totaling 8.42 measured GPU-hours for training updates, excluding setup, compilation, and evaluation (App. A.17). These figures cover the indicated jobs; preliminary and failed runs are excluded.

A.22 Supplementary references (model-based RL comparison)

The following references are cited only by the model-based-RL comparison in App. A.11 (Table 15). They are kept in a separate `supprefs.bib` file so the main paper's bibliography (`refs.bib`) is not modified by this addition; the rendered list below is auto-generated from `supprefs.bib` by `scripts/gen_supprefs_list.py`.

[S1] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Y. Zou, S. Levine, C. Finn, T. Ma. *MOPO: Model-based offline policy optimization*. Advances in neural information processing systems 2020.

[S2] R. Kidambi, A. Rajeswaran, P. Netrapalli, T. Joachims. *MOREL: Model-based offline reinforcement learning*. Advances in neural information processing systems 2020.

[S3] T. Yu, A. Kumar, R. Rafailov, A. Rajeswaran, S. Levine, C. Finn. *COMBO: Conservative Offline Model-Based Policy Optimization*. Advances in neural information processing systems 2021.

[S4] M. Rigter, B. Lacerda, N. Hawes. *RAMBO-RL: Robust Adversarial Model-Based Offline Reinforcement Learning*. Advances in neural information processing systems 2022.

[S5] Y. Sun, J. Zhang, C. Jia, H. Lin, J. Ye, Y. Yu. *Model-Bellman inconsistency for model-based offline reinforcement learning*. International Conference on Machine Learning 2023.

[S6] Jihwan Jeong, Xiaoyu Wang, Michael Gimelfarb, Hyunwoo Kim, Baher abdulhai, Scott Sanner. *Conservative Bayesian Model-Based Value Expansion for Offline Policy Optimization*. The Eleventh International Conference on Learning Representations 2023.

[S7] X. Chen, Y. Yu, Q. Li, F. Luo, Z. Qin, W. Shang, J. Ye. *Offline Model-based Adaptable Policy Learning*. Advances in neural information processing systems 2021.

[S8] M. Bhardwaj, T. Xie, B. Boots, N. Jiang, C. Cheng. *Adversarial Model for Offline Reinforcement Learning*. Advances in Neural Information Processing Systems 2023.

[S9] M. Janner, Q. Li, S. Levine. *Offline Reinforcement Learning as One Big Sequence Modeling Problem*. Advances in neural information processing systems 2021.

[S10] Zhengyao Jiang, Tianjun Zhang, Michael Janner, Yueying Li, Tim Rocktäschel, Edward Grefenstette, Yuandong Tian. *Efficient Planning in a Compact Latent Action Space*. 3rd Offline RL Workshop: Offline RL as a "Launchpad" 2022.

[S11] M. Janner, Y. Du, J. Tenenbaum, S. Levine. *Planning with Diffusion for Flexible Behavior Synthesis*. Proceedings of the 39th International Conference on Machine Learning 2022.

[S12] Kwanyoung Park, Youngwoon Lee. *Model-based Offline Reinforcement Learning with Lower Expectile Q-Learning*. The Thirteenth International Conference on Learning Representations 2025.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Sec. 1 state the contributions (Hierarchical Inverter framework, inverse learning paradigm, AntMan benchmark, beyond-data-support optimum approach on maze2d-umaze, FM-hacking failure mode and the data-coverage fix, quantum-gate application example), each backed by a dedicated experimental section (Secs. 4.1–4.5) and discussed alongside its limitations in Sec. 5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Sec. 5 explicitly discusses (i) the task-specific slot-fillings still required in this first instantiation of the framework (auxiliary loss terms, per-task Level 2 Inverter choice, per-task architecture / parameterization), with a full inventory in Suppl. Tab. 22 and a four-strategy research program (Suppl. Tab. 21); (ii) the deterministic, fully-observable, single-agent scope of the experiments; (iii) the forward-model-hacking failure mode under narrow expert data and the data-coverage fix demonstrated by AntMan (Sec. 4.4 and App. A.15); and (iv) the AntMaze auxiliary-loss additions (BC anchor + body-yaw regulariser) whose data-anchoring partially compromises inverse learning's defining FM-gradient-driven property; the original submission-phase compute environment and timeline are described in App. A.21.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.

- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not claim formal theoretical results requiring proof; Eqs. 1–2 are problem formulations of the inverse-planning / inverse-learning objectives, with all assumptions (deterministic dynamics, frozen FM, differentiability) stated in the surrounding text.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Architectures, datasets, optimisers, hyperparameters, and evaluation protocols are documented in App. A.3, A.8, A.12, and A.17; the original submission-phase hyperparameter counts and tuning ranges are reported in App. A.20 (Suppl. Tab. 23, 24). The public code release is described in the code-availability statement.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example

(a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code is publicly available at `https://github.com/neuromentum/inverter` (see the code-availability statement). The underlying offline datasets are the public D4RL `maze2d-v1` and `antmaze-v2` suites [38], and the quantum experiments use Haar-sampled $\mathrm{SU}(2)$ targets generated on the fly.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Headline numbers and per-method breakdowns appear in Tab. 2, Tab. 3, and Suppl. Tab. 16; data splits, training horizons, optimisers, replan settings, and hyperparameter ranges explored are documented in App. A.3–A.12 and App. A.17, with timing methodology in App. A.4 and the explicit hyperparameter-tuning record in App. A.20.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Inverter rows on all 3 `maze2d-v1` variants and all 6 `antmaze-v2` variants now report mean $\pm$ std over 4 independently-trained seed IWMs (Tab. 2, Tab. 3, Suppl. Tab. 8–11); the K-sweep (Suppl. Tab. 12) and the AntMaze loss-component ablation (Suppl. Tab. 16) likewise report mean $\pm$ std over 4 IWM seeds. CORL baselines report mean $\pm$ std over 4 training seeds throughout, matching CORL conventions. AntMan (Sec. 4.4) reports per-condition results from one trained Game Inverter per data-condition, evaluated over 100 games per condition; the quantum experiments report per-target paired statistics over 250 Haar-sampled $\mathrm{SU}(2)$ targets from a single trained Pulse Inverter (App. A.17). Standard deviations are computed across IWM seeds; AntMaze headline scores use seeds 42–45 on the same 100 frozen evaluation cases, as in the matched-controller ablation. The original submission-phase compute environment is documented in App. A.21.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: App. A.21 documents the hardware, original submission-phase calendar span, and representative training/evaluation times for the original and added experiments; Suppl. Tab. 25 gives the original navigation audit's estimated evaluation counts. Per-method inference costs and the timing protocol are reported in Tabs. 2, 3, and 8–11, and App. A.4. App. A.10 additionally reports an A40 planner-latency audit.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.

- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work uses only public benchmark datasets (D4RL maze2d/antmaze) and synthetic Haar-sampled $\mathrm{SU}(2)$ targets, involves no human subjects, no personal or private data, and no scraped material; licenses for the existing software and datasets are listed in item 12, and the public code release is described in item 5.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: The paper introduces a generic planning/control framework and evaluates it on standard public research benchmarks (D4RL maze2d/antmaze and Haar-sampled single-qubit gate synthesis) without targeting a specific deployment context; consistent with the checklist guideline that "it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster," we identify no direct path from the present work to negative societal applications and therefore do not include a dedicated impacts section.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The release comprises research code and the AntMan navigation/decision benchmark; it does not include pre-trained generative models, language models, or scraped datasets that would carry an elevated misuse risk.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We use D4RL [38] (Apache 2.0) for `maze2d-v1` and `antmaze-v2`, the CORL library [39] (Apache 2.0) for offline-RL baselines, MuJoCo (Apache 2.0) for the ant simulator, and PyTorch (BSD-3) and JAX (Apache 2.0) as DL frameworks; each is cited at first use, used within its terms, and reproduced unmodified.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The original submission code package includes the core Inverter framework and AntMan benchmark, a top-level `README.md` documenting components and training/evaluation entry-points, and configuration files. This description concerns the original submission package; code is publicly available as described in the code-availability statement. Setup details are provided in App. A.15 and App. A.12.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The work involves no crowdsourcing and no human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The work involves no human subjects, so no IRB review was required.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core methodology (forward models, Inverters, BFS sequencer, training and inference pipelines) does not use LLMs; DecisionLLM appears only as a published baseline in Tab. 2 and Suppl. Tab. 8.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.