Amortizing Trajectory Diffusion with Keyed Drift Fields

Anonymous Authors

***Abstract*— Diffusion-based trajectory planners can synthesize rich, multimodal action sequences for offline reinforcement learning, but their iterative denoising incurs substantial inference-time cost, making closed-loop planning slow under tight compute budgets. We study the problem of achieving diffusion-like trajectory planning behavior with one-step inference, while retaining the ability to sample diverse candidate plans and condition on the current state in a receding-horizon control loop. Our key observation is that conditional trajectory generation fails under naïve distribution-matching objectives when the similarity measure used to align generated trajectories with the dataset is dominated by unconstrained future dimensions. In practice, this causes attraction toward average trajectories, collapses action diversity, and yields near-static behavior. Our key insight is that conditional generative planning requires a conditioning-aware notion of neighborhood: trajectory updates should be computed using distances in a compact key space that reflects the condition, while still applying updates in the full trajectory space. Building on this, we introduce Keyed Drifting Policies (KDP), a one-step trajectory generator trained with a drift-field objective that attracts generated trajectories toward condition-matched dataset windows and repels them from nearby generated samples, using a stop-gradient drifted target to amortize iterative refinement into training. At inference, the resulting policy produces a full trajectory window in a single forward pass. Across standard RL benchmarks and real-time hardware deployments, KDP achieves strong performance with one-step inference and substantially lower planning latency than diffusion sampling. Project website, code and videos: https://keyed-drifting.github.io/**

I. INTRODUCTION

Diffusion-based trajectory models have emerged as a compelling foundation for robot decision-making and offline reinforcement learning (RL) because they can model complex, multimodal distributions over future behavior directly from data [1]–[3]. In a typical deployment, a diffusion model is used inside a receding-horizon control loop: at each control step, the system conditions on the current observation and optionally a goal or desired return, samples a short trajectory segment, executes the first action, and replans [2]–[4]. Despite their flexibility, diffusion planners are often impractical at high control rates because they require many sequential denoising steps, leading to substantial latency and large numbers of neural network evaluations [4]–[6]. This paper asks a concrete question: how can we obtain diffusion-like trajectory planning behavior with one-step inference while preserving conditional controllability and sample diversity in a receding-horizon control loop?

Prior work reduces diffusion-planning latency either by cutting denoising steps [6]–[10] or by iteratively sampling and selecting/guiding candidates with various signals [11], [12]. Both approaches remain tied to an iterative sampling loop, requiring added tuning as compute budgets tighten.

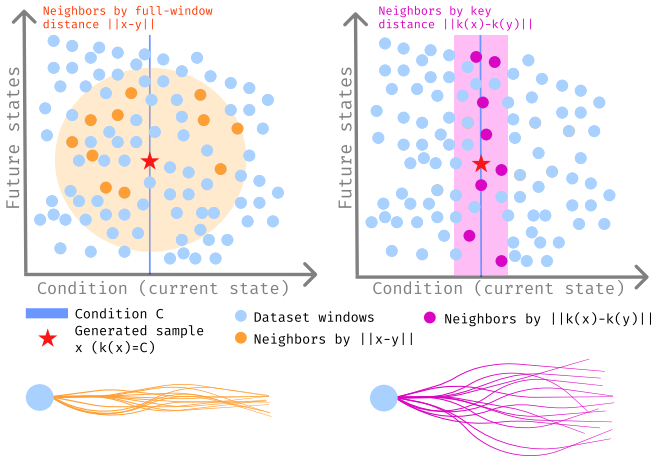

Fig. 1: Given condition c, full-window distance selects neighbors dominated by unconstrained future dimensions (orange), encouraging mode-averaged trajectories (left bottom). Key distance selects condition-matched neighbors (magenta), preserving diverse conditional trajectories (right bottom).

We take a different route: amortize refinement into training to obtain a one-step conditional trajectory generator usable for planning. Naïvely matching the data distribution in full window space is brittle—in high-dimensional state-action sequences, distances are dominated by unconstrained future components, pulling samples toward averages, collapsing action diversity, and yielding inert closed-loop behavior.

Our key observation is that conditional trajectory generation requires a conditioning-aware notion of neighborhood: for a given condition, the learning signal should preferentially compare against data trajectories that match that condition, rather than against arbitrary trajectories that are close under a high-dimensional, poorly aligned metric. We operationalize this observation with Keyed Drifting Policies (KDP), a one-step trajectory generator trained with a keyed drift field objective [13] that pulls samples toward condition-matched dataset windows and repels them from nearby generated samples via a stop-gradient drifted target, amortizing refinement into the model parameters. KDP is deployed in the same receding-horizon planning interface as diffusion planners, and we contribute: (i) a practical keyed drift-field formulation for conditional trajectory generation that avoids collapse under hard conditioning constraints; (ii) a one-step planning recipe that preserves the diffusion-planner interface while eliminating the T-step sequential denoising loop; and (iii) an end-to-end evaluation spanning standard simulation benchmarks and real-time closed-loop hardware deployments under strict latency budgets.

II. Related Work

Our work sits at the intersection of diffusion-based planning, offline RL, and distribution matching.

Offline RL: Offline RL learns policies from fixed datasets [14] and combats distribution shift via behavior constraints, regularization, and conservative value learning [15]–[17]. Practical variants blend cloning-style objectives with critic learning or advantage weighting [18], [19], while model-based offline RL uses learned dynamics with pessimism to mitigate compounding error [20]–[22]. Sequence-model approaches reframe decision making as conditional sequence prediction [23]–[25]. These families are effective but usually yield reactive policies or require costly decoding or optimization, and they do not directly address reducing inference-time iterations for sampling-based planning.

Diffusion-based planners: Diffusion-based planners model future trajectories or actions and deploy them by sampling candidate plans or actions [1], [2], [26]. They support flexible conditioning via inpainting, return conditioning, or guidance [1], [2] and have been extended through hierarchical, latent, and energy-guided variants [27], [28]. Robotics variants apply diffusion to trajectory generation and control under constraints [3], [4]. Despite strong modeling capacity, these methods typically rely on iterative denoising, which increases inference latency and can introduce closed-loop brittleness for real-time planning.

One-step generators: Faster samplers reduce denoising steps via deterministic trajectories, improved parameterizations, or specialized solvers, and guidance mechanisms steer samples toward conditions without explicit classifiers [6], [7]. Distillation and alternative objectives target few-step generation and reduces per-step compute [8]–[10]. Score-based foundations formalize diffusion and score learning in continuous time and in latent spaces, and hybrid objectives combine diffusion training with GAN-style efficiency [29]–[31]. These approaches often require additional stages or tuning; our focus is amortizing refinement into training for one-step conditional planning.

Distribution matching: Kernel and particle-based perspectives emphasize that distribution matching depends on the neighborhood metric: mean-shift-style updates and density-gradient estimators motivate attraction dynamics, while kernel mean embeddings formalize distributional discrepancies [32]–[34]. Recent methods combine attraction with kernel-based repulsion to maintain diversity, including policy-gradient adaptations [35], [36]. Energy-based and score-based learning provide complementary views: score matching links learning to density-gradient estimation, while energy-based learning, and noise-contrastive estimation use negative samples to fit unnormalized models [37], [38]. Adversarial objectives and contrastive learning operationalize attraction/ repulsion through learned discrepancies and positive/negative sampling [39], [40]. Autoregressive sequence models provide another form of distribution matching by fitting trajectory distributions directly through conditional sequence prediction [23], [24].

III. Background

A. Trajectory diffusion models for planning

We consider diffusion models that generate short trajectory windows for receding-horizon control. Let a window be represented as $x_0 \in \mathbb{R}^{H\times D}(D = d_s + d_a)$, where each timestep concatenates state and action, i.e., $x_0[t] = (s_t, a_t)$. At decision time, the model is conditioned on a context c derived from the current observation (and optionally a goal). In this paper, the conditioning signal we primarily care about is the current state, written as $c = s_0$.

Diffusion introduces a sequence of latent variables $\{x_t\}_{t=1}^T$ via a forward noising process [5]

$$q(x_t \mid x_{t-1}) = \mathcal{N}(\sqrt{\alpha_t}\, x_{t-1},\ (1-\alpha_t)I)\,, \tag{1}$$

with $\alpha_t = 1 - \beta_t$ and a user-chosen noise schedule $\{\beta_t\}_{t=1}^T$. Defining $\bar{\alpha}_t = \prod_{i=1}^t \alpha_i$, the marginal admits the closed form

$$q(x_t \mid x_0) = \mathcal{N}\big(\sqrt{\bar{\alpha}_t}\, x_0,\ (1-\bar{\alpha}_t)I\big)\,, \tag{2}$$

also written as $x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \varepsilon$ with $\varepsilon \sim \mathcal{N}(0, I)$.

A standard parameterization trains a denoiser ε_θ to predict the injected noise under conditioning c:

$$\min_\theta\ \mathbb{E}_{x_0\sim\mathcal{D},\, t\sim\{1,\dots,T\},\, \varepsilon\sim\mathcal{N}(0,I)} \left[\|\varepsilon - \varepsilon_\theta(x_t, t, c)\|_2^2\right]. \tag{3}$$

At inference, trajectories are sampled by reversing the process from $x_T \sim \mathcal{N}(0, I)$ using a learned reverse transition

$$p_\theta(x_{t-1} \mid x_t, c) = \mathcal{N}\big(\mu_\theta(x_t, t, c),\ \sigma_t^2 I\big)\,, \tag{4}$$

where the mean typically takes the DDPM [5] form

$$\mu_\theta(x_t, t, c) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\, \varepsilon_\theta(x_t, t, c)\right), \tag{5}$$

and σ_t^2 is set from the schedule.

B. Receding-horizon control and conditioning

In receding-horizon deployment, at each environment step we observe the current state s and set $c = s$. A sampled window $\hat{x}_0$ yields the control action by extracting its first action component: $a = \hat{x}_0[0,\, d_s{:}d_s{+}d_a]$. Implementations [1] further improve reliability by sampling K candidate windows $\{\hat{x}_0^{(k)}\}_{k=1}^K$ and selecting via a learned scorer J_ϕ.

C. Computational challenges with diffusion planning

Diffusion planning is dominated by sequential denoising: generating one candidate trajectory requires T denoiser evaluations, and best-of-K selection scales this to $\approx KT$ evaluations per control step, plus scoring. Decreasing T reduces latency but changes the sampling distribution and harms conditional fidelity; increasing K improves selection but increases cost linearly. A natural alternative is a one-step conditional generator $g_\psi(z, c)$ that outputs a full horizon-H window in a single forward pass. However, naïvely training such a generator by matching distributions in full window space is unstable: distances and losses are dominated by unconstrained future components, encouraging collapse toward average trajectories and producing inert controllers despite numerically stable training.

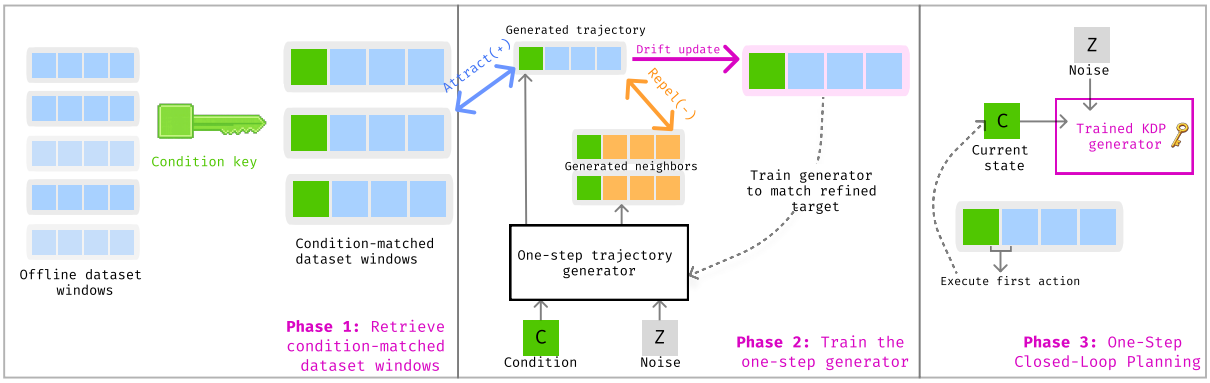

Fig. 2: Overview of KDP. A condition key retrieves condition-matched dataset windows, which train a one-step generator by attracting trajectories toward matched data and repelling them from nearby generated samples, enabling one-step closed-loop planning at inference.

IV. Keyed Drifting Policies

We introduce Keyed Drifting Policies (KDP), which train a conditional generator by evolving its training-time pushforward distribution with a drifting field, following the stop-gradient drifted target recipe of drifting models [13]. The modification for conditional planning is that we make neighborhood structure *conditioning-aware*: when we decide which dataset windows should influence a generated sample, we measure similarity in a *key space* aligned with the condition, while applying updates in the full trajectory space.

A. Trajectory windows and hard conditioning constraints

Recall that a trajectory window is represented as x and $x[t] = (s_t, a_t)$. We use the block operators: $\mathrm{state}(x)[t] = x[t, 0{:}d_s]$, $\mathrm{act}(x)[t] = x[t, d_s{:}D]$.

In receding-horizon control, the condition is the current observation (state conditioning), $c = s_0 \in \mathbb{R}^{d_s}$. Because the plan must start at the true current state, we enforce this as a hard constraint via a clamping operator $\mathcal{C}_c(\cdot)$:

$$\mathcal{C}_c(x)[0, 0{:}d_s] = c, \qquad \mathcal{C}_c(x)[t, :] = x[t, :] \ \text{ otherwise.} \quad (6)$$

Clamping is necessary because small prediction errors in the conditioned dimensions accumulate across replanning steps, which can destabilize the closed-loop behavior.

a) Goal-conditioned inpainting: For goal-conditioned tasks we augment the condition with a goal vector $g \in \mathbb{R}^{d_g}$ and enforce goal satisfaction by inpainting selected state coordinates at a designated timestep t_g. Let $\mathcal{I} \subseteq \{1, \dots, d_s\}$ denote the goal-relevant state indices. We define an extended clamping operator $\mathcal{C}_{(c,g)}$ that overwrites both the initial state and the goal coordinates:

$$\mathcal{C}_{(c,g)}(x)[0, d_a{:}D] = c, \qquad \mathcal{C}_{(c,g)}(x)[t_g, d_a + \mathcal{I}] = g, \quad (7)$$

leaving all other entries unchanged. During training we apply the same mask to the drift field so drift updates cannot modify clamped coordinates.

B. Conditional pushforward at training time

Let g_ψ be a conditional generator that maps noise $z \sim \mathcal{N}(0, I)$ and condition c to a trajectory window:

$$\mathbf{x} = \mathcal{C}_c(g_\psi(z, c)) \in \mathbb{R}^{H \times D}. \quad (8)$$

For a fixed condition c, the model induces a conditional pushforward distribution $q_\psi(\cdot \mid c)$. During stochastic optimization, the parameter sequence $\{\psi_i\}$ induces a sequence of conditional pushforwards $\{q_{\psi_i}(\cdot \mid c)\}$. Drifting models [13] view training as an evolution of the pushforward distribution governed by a drifting field; we adopt the same viewpoint, but the conditional setting forces a question that does not arise in unconditional generation: *how should we define neighborhood structure so that the learning signal reflects the condition rather than being dominated by unconstrained future dimensions?*

Empirically, if similarity is measured over full windows early in training, the unconstrained future dominates distance computations and leads to unstable or collapsed behavior (e.g., action variance collapse and near-static rollouts). KDP addresses this by defining neighborhoods in a key space aligned with the condition.

C. Key space and conditioning-aware neighborhoods

KDP introduces a *key map* $\kappa(\cdot)$ used *only* for neighborhood formation. In the state-conditioned case we use the initial state block: $\kappa(x) = \mathrm{state}(x)[0] \in \mathbb{R}^{d_s}$.

Key-space distances are Euclidean: $d_\kappa(x, y) = \|\kappa(x) - \kappa(y)\|_2$. Because we always clamp $\mathrm{state}(\mathbf{x})[0] = c$, the key of a generated window is exactly the condition. As a result, neighborhood selection depends on c rather than on the current (potentially noisy) future portion of $\mathbf{x}$. This is the mechanism by which KDP makes conditional learning signals robust in high-dimensional trajectory spaces.

D. A keyed drifting field in full trajectory space

Having fixed *how* we form neighborhoods (Sec. IV-C), we still need a *training-time refinement rule*: given a clamped generated window $\mathbf{x}_i$, what update direction should move it toward the conditional data distribution while avoiding mode collapse? Following drifting models [13], we construct a drifting field $V(\mathbf{x})$ with two roles: (i) *attraction* toward dataset windows that match the condition, and (ii) *repulsion* away from nearby generated windows to maintain diversity. KDP uses the key distance d_κ to decide which windows are near, but applies the update in the full window space $\mathbb{R}^{H \times D}$.

From mean-shift to a keyed drift. A convenient way to think about drift is as a mean-shift-like update. Given a normalized kernel $\tilde{k}(\cdot,\cdot)$ and sources of samples: $\mathbf{y}^+ \sim p$ from the dataset and $\mathbf{y}^- \sim q$ from the current model—a standard attraction–repulsion field can be written as

$$V_{p,q}(\mathbf{x}) = \mathbb{E}_{\mathbf{y}^+\sim p}\big[\,\tilde{k}_\kappa(\mathbf{x},\mathbf{y}^+)\,\mathbf{y}^+\,\big] \;-\; \mathbb{E}_{\mathbf{y}^-\sim q}\big[\,\tilde{k}_\kappa(\mathbf{x},\mathbf{y}^-)\,\mathbf{y}^-\,\big], \quad (9)$$

where $\tilde{k}_\kappa$ emphasizes that similarity is computed in key space. This form is equivalent to the mean of differences presentation in drifting models [13] (the $\mathbf{x}$ terms cancel between attraction and repulsion), and it makes explicit what KDP needs: a way to compute $\tilde{k}_\kappa$ that reflects the condition.

Mini-batch construction. In stochastic training, we approximate the expectations in Eq. (9) with minibatch averages. Let $\{\mathbf{y}^+{}_j\}_{j=1}^B$ be a minibatch of dataset windows. We set the condition for the i-th sample to be the dataset key, i.e.,

$$c_i = \kappa(\mathbf{y}^+{}_i);\;\; \mathbf{x}_i = \mathcal{C}_{c_i}(g_\psi(z_i, c_i))\,;\;\; z_i \sim \mathcal{N}(0, I), \quad (10)$$

so that the empirical distribution of conditions seen by the generator matches the data. We use the current minibatch of generated samples as negatives, $\mathbf{y}^-{}_j := \mathbf{x}_j$, so repulsion is always defined *relative to the model's current conditional output distribution*.

Keyed kernel weights. A drifting field is only as useful as its notion of nearby. If we computed similarity in full window space early in training, distances would be dominated by unconstrained future components, and the resulting weights would not reflect the condition. KDP therefore computes kernel weights using key distances:

$$D^+_{ij} = d_\kappa(\mathbf{x}_i, \mathbf{y}^+{}_j), \qquad D^-_{ij} = d_\kappa(\mathbf{x}_i, \mathbf{y}^-{}_j). \quad (11)$$

We adopt the same exponential kernel family as [13] implemented via a softmax normalization:

$$W^+{}_{ij} = \frac{\exp(-D^+_{ij}/\tau)}{\sum_{k=1}^B \exp(-D^+_{ik}/\tau)}, \qquad W^-{}_{ij} = \frac{\exp(-D^-_{ij}/\tau)}{\sum_{k=1}^B \exp(-D^-_{ik}/\tau)}. \quad (12)$$

Crucially, because $\mathbf{x}_i$ is clamped, $\kappa(\mathbf{x}_i)$ is determined by c_i; thus D^+_{ij} compares conditions rather than unconstrained futures, and W^+ concentrates on condition-matched windows.

Self-negative masking. If we include $j = i$ in the repulsion softmax, then $D^-_{ii} = 0$ makes $W^-{}_{ii}$ dominate, yielding $\mu^-{}_i \approx \mathbf{x}_i$. This collapses the repulsion term into a no-op and reduces the drift to a pure attraction update, which empirically encourages mode averaging. We therefore exclude self-negatives by masking the diagonal before normalization (equivalently $D^-_{ii} \leftarrow +\infty$ in Eq. (12)), matching the self-negative masking fix used in our implementation.

Attraction and repulsion means in full window space. We form weighted averages of full windows:

$$\mu^+{}_i = \sum_{j=1}^B W^+{}_{ij}\,\mathbf{y}^+{}_j, \qquad \mu^-{}_i = \sum_{j=1}^B W^-{}_{ij}\,\mathbf{y}^-{}_j. \quad (13)$$

Here $\mu^+{}_i$ is a condition-local prototype of the dataset in full trajectory space, $\mu^-{}_i$ is the corresponding prototype of the model's current samples under the same key-locality notion. The drift direction is their difference: $V_i^{(\tau)} = \mu^+{}_i - \mu^-{}_i$. This is precisely the batch approximation of Eq. (9).

Multi-temperature averaging. The temperature τ controls how local the neighborhood is: smaller τ emphasizes nearest neighbors in key space, while larger τ averages over a broader set of condition-similar windows. To reduce sensitivity to a single bandwidth choice, we average drifts across a small set of temperatures $\{\tau_m\}_{m=1}^M$: $V_i = \frac{1}{M}\sum_{m=1}^M V_i^{(\tau_m)}$.

Constraint-respecting drift (masking) and stable magnitudes (normalization). Because KDP enforces hard constraints via $\mathcal{C}$, refinement must not alter clamped entries. We therefore apply a binary mask $M \in \{0,1\}^{H\times D}$ that zeros clamped coordinates: $V_i \leftarrow M \odot V_i$.

Finally, we normalize nonzero drift vectors per sample: $V_i \leftarrow V_i / \big(\sqrt{\frac{1}{HD}\|V_i\|_2^2 + \epsilon}\big)$. This keeps equilibrium at $V_i = 0$ unchanged and prevents drift from varying wildly.

E. Amortizing refinement with a stop-gradient drifted target

The keyed drift field V_i defines a refinement step that would improve $\mathbf{x}_i$ if applied iteratively. Instead, we amortize refinement into g_ψ using a stop-gradient drifted target, as in [13]. Let $\mathrm{sg}(\cdot)$ denote stop-gradient. We define the target:

$$\widetilde{x}_i = \mathcal{C}_{c_i}(\mathrm{sg}(\mathbf{x}_i + V_i))\,. \quad (14)$$

We then regress the generator output toward this target in full trajectory space.

Weighted trajectory regression We use a weighted squared error that upweights action dimensions:

$$\min_\psi \;\mathbb{E}\left[\|\mathbf{x}_i - \widetilde{x}_i\|_W^2\right], \quad (15)$$

with $\|u\|_W^2 = \sum_{t=0}^{H-1}\Big(\lambda_s\,\|u[t, 0{:}d_s]\|_2^2 + \lambda_a\,\|u[t, d_s{:}D]\|_2^2\Big)$.

In our implementation, this weighting is a practical guard against action-collapse: it biases the amortized refinement toward producing control-relevant diversity in the action block while still training the full window. Algorithm 1 summarizes one SGD step.

Input: Dataset minibatch $\{\mathbf{y}^+{}_i\}_{i=1}^B$; key map κ; generator g_ψ; temperatures $\{\tau_m\}_{m=1}^M$; clamp operator $\mathcal{C}$; clamp mask M; weights λ_a, λ_s.

$c_i \leftarrow \kappa(\mathbf{y}^+{}_i)$, sample $z_i \sim \mathcal{N}(0, I)$.
$\mathbf{x}_i \leftarrow \mathcal{C}_{c_i}(g_\psi(z_i, c_i))$.
$\mathbf{y}^-{}_j \leftarrow \mathbf{x}_j$ and mask self-negatives ($j = i$) in repulsion weights.
$V_i \leftarrow 0$ for all i.
for $m = 1$ to M **do**
  $W^+{}_{ij} \leftarrow \mathrm{softmax}\big(-d_\kappa(\mathbf{x}_i, \mathbf{y}^+{}_j)/\tau_m\big)$.
  $W^-{}_{ij} \leftarrow \mathrm{softmax}\big(-d_\kappa(\mathbf{x}_i, \mathbf{y}^-{}_j)/\tau_m\big)$.
  $\mu^+{}_i \leftarrow \sum_j W^+{}_{ij}\mathbf{y}^+{}_j$, $\mu^-{}_i \leftarrow \sum_j W^-{}_{ij}\mathbf{y}^-{}_j$.
  $V_i \leftarrow V_i + \frac{1}{M}\big(\mu^+{}_i - \mu^-{}_i\big)$.
end for
$V_i \leftarrow M \odot V_i$. // constraint-aware drift
$\widetilde{x}_i \leftarrow \mathcal{C}_{c_i}(\mathrm{sg}(\mathbf{x}_i + V_i))$.
$\Delta^s_{i,t} \leftarrow \mathbf{x}_i[t, 0{:}d_s] - \widetilde{x}_i[t, 0{:}d_s]$
$\Delta^a_{i,t} \leftarrow \mathbf{x}_i[t, d_s{:}D] - \widetilde{x}_i[t, d_s{:}D]$
$\mathcal{L} \leftarrow \sum_{i,t}\big(\lambda_s\|\Delta^s_{i,t}\|_2^2 + \lambda_a\|\Delta^a_{i,t}\|_2^2\big)$
Update ψ by SGD on $\mathcal{L}$.

Algorithm 1: Keyed drifting policy training (one SGD step)

F. Online receding-horizon planning

At each environment step, KDP conditions on the current observation c and draws K samples in parallel:

$$\hat{x}^{(k)} = \mathcal{C}_c\big(g_\psi(z_k, c)\big), \qquad z_k \sim \mathcal{N}(0, I), \qquad k = 1, \dots, K. \quad (16)$$

A scorer J_ϕ (a learned return model) selects a candidate,

$$\hat{x}^* = \arg \max_{k \in \{1,\dots,K\}} J_\phi(\hat{x}^{(k)}, c), \quad (17)$$

the executed action is the first action block of the window:

$$a \leftarrow \mathrm{act}(\hat{x}^*)[0] = \hat{x}^*[0, d_s{:}D]. \quad (18)$$

This compute profile differs from diffusion planning: diffusion requires T sequential policy evaluations per environment step (each acting on a batch of K candidates under best-of-K), while KDP requires a single policy forward pass on K candidates (plus the optional scorer). KDP therefore trades sequential refinement depth T for embarrassingly-parallel candidate count K, which is easier to amortize on modern accelerators.

Input: Current context c; generator g_ψ; number of candidates K; optional scorer J_ϕ.
1 Sample $z_1, \dots, z_K \sim \mathcal{N}(0, I)$.
2 Generate candidates: $\hat{x}^{(k)} \leftarrow \mathcal{C}_c(g_\psi(z_k, c))$ for $k = 1, \dots, K$.
3 **if** J_ϕ is available **then**
4 Select $\hat{x}^* \leftarrow \arg\max_k J_\phi(\hat{x}^{(k)}, c)$.
5 **else**
6 Select $\hat{x}^* \leftarrow \hat{x}^{(1)}$.
7 **end if**
8 Execute $a \leftarrow \hat{x}^*[0, d_s{:}D]$ and repeat at the next control cycle.

Algorithm 2: Receding-horizon planning with KDP

V. Experiments

Our experiments are designed to validate four claims: (i) KDP supports amortized refinement, (ii) KDP offers a favorable compute–performance tradeoff relative to diffusion-based planners, (iii) KDP supports conditional planning, and (iv) the performance gains are attributable to specific components of the method, verified by ablations.

Experimental setup

1) Simulated benchmarks: We evaluate on D4RL benchmark [41]: (i) locomotion, (ii) goal-conditioned navigation, (iii) long-horizon navigation, (iv) dexterous manipulation.

Fig. 3: D4RL simulation environments.

2) Hardware implementation: To showcase real-time performance of KDP, we evaluate on two domains: (i) quadrotor navigation and (ii) tabletop manipulation with a 6-DoF arm.

3) Metrics: We report the following per environment step: (i) *Sequential NFEs (NFE):* number of sequential policy forward calls required to produce candidates. (ii) *Batch-equivalent forwards (BEF):* total effective batch processed per step, accounting for candidate count. (iii) *Planning latency (ms/step) (PL):* end-to-end planning time per environment step. (iv) *End-to-end control latency (ms/step) (E2E):* planning latency plus environment and wrapper overhead.

4) Baselines: We separate baselines by their inference-time behavior: *(i) fast-policy / offline-selection* baselines (BC [14], CQL [15], IQL [16], Decision Transformer [42], Trajectory Transformer [43]), and representative model-based offline RL methods (MOPO [20], MOReL [21], and MBOP [22]); *(ii) Generative trajectory methods* (Diffuser [1], trained on the same trajectory-window representation and evaluated with the same receding-horizon control as KDP). For locomotion we additionally report recent low-NFE generative or trajectory-modeling approaches (DQL [26], CAC [7], and CBC [7]). Where selection is used, we evaluate both (i) unranked sampling and (ii) ranked best-of-K sampling using the same learned scorer used for KDP.

A. Results: Short-horizon continuous control

We begin with D4RL locomotion, a hard benchmark for one-step generative planning. These tasks have dense rewards, relatively short horizons, and strong state-action regularities; as a result, they favor methods that can produce a sharp immediate action without needing explicit trajectory-level uncertainty. We therefore ask: can a one-step generative planner remain competitive while preserving multimodal sampling and collapsing inference to one NFE?

	HalfCheetah		Hopper		Walker2D		Avg↑	NFE↓
	ME	M	ME	M	ME	M		
Offline selection								
BC [14]	55.2	42.6	52.5	52.9	107.5	75.3	64.3	–
CQL [15]	91.6	44.0	105.4	58.5	108.8	72.5	80.1	–
IQL [16]	86.7	47.4	91.5	66.3	109.6	78.3	80.0	–
DT [42]	86.8	42.6	107.6	67.6	108.1	74.0	81.1	–
TT [43]	95.0	46.9	110.0	61.1	101.9	79.0	82.3	–
MOPO [20]	63.3	42.3	23.7	28.0	44.6	17.8	36.6	–
MOReL [21]	53.3	42.1	108.7	**95.4**	95.6	77.8	78.8	–
MBOP [22]	**105.9**	44.6	55.1	48.8	70.2	41.0	60.9	–
Generative trajectory planners								
DQL [26]	96.8	**69.1**	**111.1**	90.5	110.1	87.0	**91.1**	5
CAC [7]	84.3	**69.1**	100.4	80.7	**110.4**	83.1	88.0	2
CBC [7]	32.7	31.0	90.6	71.7	**110.4**	83.1	69.9	2
Diffuser [1]	88.9	42.8	103.3	74.3	106.9	79.6	82.6	20
KDP	92.5	62.1	103.6	90.3	108.5	**87.2**	90.7	**1**

TABLE I: Performance in short horizon control on D4RL Locomotion (ME=Medium-Expert, M=Medium).

The answer is yes (refer Table I). KDP is competitive with the strongest low-NFE generative trajectory methods while substantially improving over Diffuser. Locomotion is not the benchmark that most naturally favors trajectory

sampling: when the correct action is strongly determined by the current state, conservative value-based methods and sequence models remain extremely difficult to beat. The key observation is that KDP does not need a diffusion-style denoising chain to remain in the same performance band as these methods. In other words, moving to one-step inference does not collapse the planner into a weak imitation policy. A second point is that the performance pattern is not uniform across dataset qualities. Medium-only datasets place greater pressure on the model to preserve diverse but dynamically consistent futures under weaker support, while medium-expert settings more strongly reward sharp action selection and better value discrimination. KDP remains robust across both regimes, which suggests that the gain is not coming from overfitting a single dataset type, but from replacing iterative refinement with a condition-aware amortized update that still preserves useful trajectory diversity.

Ablation: *Compute–performance tradeoff.* The systems-level distinction becomes much clearer when we look at inference cost (refer Table II. Diffuser scales with the number of sequential denoising steps, so increasing candidate count compounds an already expensive inner loop. KDP changes that scaling law: candidate count grows primarily through parallel batch size, while the sequential depth of the policy remains fixed at one.

Method	NFE↓	BEF↓	PL	E2E
BC [14]	**1**	**1**	**1.01**	**1.22**
D-UR(K=1, T=20)	20	20	149.32	149.51
D-R(K=16, T=20)	20	320	172.16	172.34
D-RK=64, T=20)	20	1280	279.98	280.03
KDP-UR(K=1)	**1**	**1**	1.98	2.11
KDP-R(K=16)	**1**	32	3.97	4.10
KDP-R(K=64)	**1**	128	4.76	4.96

TABLE II: Analysis of planning cost. Hardware: 1xNVIDIA A100. (D=Diffuser, UR=Unranked, R=Ranked)

Ranked planning is only useful when extra candidates are cheap enough to sample inside the control budget. For Diffuser, larger K becomes a latency bottleneck because every additional candidate inherits the full denoising stack. For KDP, increasing K remains a practical knob: ranking acts as an inexpensive robustness improvement rather than a fundamentally different inference regime.

B. Results: Long-horizon, goal-conditioned planning

We next evaluate on Maze2D and AntMaze, where the core challenge is propagating conditioning information across a long horizon under delayed or sparse reward. These tasks stress the failure mode that motivated KDP.

1) Maze2D: horizon and candidate scaling: KDP improves on Diffuser while operating at essentially real-time planning cost (refer Table III), which suggests that the gain is not just from being cheaper, but from using a more appropriate training signal for conditional trajectory generation. In Maze2D, once the neighborhood structure is aligned with the conditioned state/goal geometry, one-step generation appears sufficient to recover strong planning behavior. This result suggest that in a genuinely trajectory-level conditional problem, amortizing refinement into training is not merely a systems trick, it can also be the better inductive bias.

Method	Maze2D				AntMaze		
	UM	Med.	Lar.	PL	UM	Med.	PL
CQL [15]	5.7	5.0	12.5	–	**84.0**	53.7	–
IQL [16]	47.4	34.9	58.6	–	62.2	**70.0**	–
Diffuser [1]	113.9	121.5	123.0	3.848	76.0	31.9	6.153
KDP	**122.3**	**130.4**	**133.0**	**0.031**	81.3	67.6	**0.052**

TABLE III: Long horizon manipulation analysis. PL is in s/step. (UM=U-Maze, Med.=Medium, Lar.=Large)

A second takeaway is that candidate scaling is only useful when the sampler stays inside the latency budget. Maze2D makes this visible because longer horizons and goal constraints make single-sample failure more likely, so best-of-K selection should matter. KDP benefits from this regime precisely because it can afford larger candidate sets without paying a sequential-denoising penalty.

2) AntMaze: long-horizon sparse-reward planning: AntMaze is substantially harder than Maze2D because success depends not only on reaching the right goal region, but on discovering and maintaining globally correct route structure over long horizons under sparse feedback. The results reflect this difficulty. KDP retains its latency advantage and remains competitive with diffusion-based planning, but the gap over strong offline RL baselines narrows relative to Maze2D. AntMaze is a setting where more sampling steps alone are not enough: route selection, better goal representations, stronger ranking, or hierarchical structure matter more than simply denoising longer. KDP preserves fast, condition-aware trajectory generation even in this hard sparse-reward regime, and thus provides a stronger foundation for future improvements in selection or hierarchy.

Ablation: *Training/objective.* Table IV isolates the training-time components of KDP. We ablate the neighborhood definition by replacing the default condition-keyed distance with full-window distances, test the role of collapse prevention by including self-negatives or removing repulsion entirely, and study the stability contribution of drift normalization and temperature averaging.

Ablation	Locom.↑	Maze2D↑	Act. div.↑
Full method (KDP)	**90.7**	128.2	0.93
No keying (full-window distances)	2.91	9.6	0.14
Include self-negatives	78.08	91.3	0.93
Attraction-only (no repulsion)	1.05	3.7	2.2e6
No drift normalization	86.93	**131.5**	0.93
Single τ	**92.37**	102.1	0.93

TABLE IV: Training objective ablations. (Locom.=Locomotion, Act. div.=Action diversity)

The ablations show that the method depends first and foremost on conditioning-aware neighborhoods. Removing keying and computing similarity in full trajectory space causes performance to collapse, confirming that the central

failure mode is not lack of model capacity but a misaligned notion of neighborhood under conditioning. Repulsion is also essential: when it is removed, performance collapses and the action-diversity statistic explodes, indicating uncontrolled drift rather than useful multimodality. Excluding self-negatives provides a smaller but still consistent gain, suggesting that once repulsion is present, its quality matters even if it is not the dominant factor. Drift normalization is less critical–performance remains strong without it and can even improve slightly on Maze2D–but normalization improves robustness across settings and makes the training signal better behaved overall. Finally, using a single temperature degrades Maze2D, which is consistent with our intuition that long-horizon goal-conditioned planning benefits from neighborhoods defined at multiple scales.

C. Results: Dexterous manipulation and high-DoF control

We now turn to a much harder setting: dexterous manipulation, where small action errors are amplified by contacts, underactuation, and high-dimensional control.

1) Adroit results: The manipulation results make the benefit of KDP more pronounced than in locomotion. Unlike locomotion, where smooth dynamics often allow strong fast-policy baselines to dominate, Adroit requires the sampled plan to remain precise under contact and over multiple coupled degrees of freedom. In this setting, the one-step drift objective appears to align better with the control problem than iterative denoising of the entire trajectory window.

Task	Diffuser [1]		KDP	
	Score	T(s)	Score	T(s)
pen-clone	10.7	1.634	**53.4**	**0.020**
door-clone	56.7	1.598	**61.8**	**0.024**
hammer-clone	53.1	1.532	**79.6**	**0.034**
relocate-clone	56.2	1.685	**62.8**	**0.036**

TABLE V: Adroit results.

Task	I	Diff.	KDP
adroit-pen	1	38%	57%
adroit-pen	8	41%	73%
adroit-hammer	1	47%	62%
adroit-hammer	8	41%	63%
adroit-relocate	1	35%	65%
adroit-relocate	8	64%	71%

TABLE VI: Action chunking.

Two insights matter here (refer Table V). First, the gain is not just faster than Diffuser, it is faster and better on tasks where poor action structure is immediately punished. Second, the improvement is consistent with the way KDP is trained: the action block is explicitly emphasized in the regression objective, so the model is encouraged to learn plans whose control-relevant portion is sharp and usable at execution time. In high-DoF manipulation, that appears to be a more favorable bias than repeatedly refining the full window online. These results are also important for deployment—planning delays on the order of seconds are often unacceptable regardless of final normalized score, because contact transitions and object motion evolve on much faster timescales.

Ablation: *Action chunking.* Action chunking provides a second lens on trajectory quality (refer Table VI). If a sampled window is only trustworthy at its very first action, then executing multiple actions before replanning should quickly degrade performance. If, instead, the sampled trajectory is locally coherent, modest chunk sizes should improve stability by filtering high-frequency planning noise and reducing replanning overhead. The chunking trends support the latter interpretation for KDP. Moderate chunk sizes improve performance, which suggests that KDP's windows are not merely optimized for the first action token. This is the behavior one would want before moving to hardware: the planner should not collapse when sensing, actuation, and planning are slightly out of sync.

D. Real-world hardware experiments

We finally evaluate KDP on two real-world domains. We consider two representative settings with different failure modes: (i) *navigation*, using a Crazyflie 2.0 micro-quadrotor to fly from start to goal through cluttered indoor space, and (ii) *manipulation*, using an SO-100 arm for tabletop pick-place and stacking tasks.

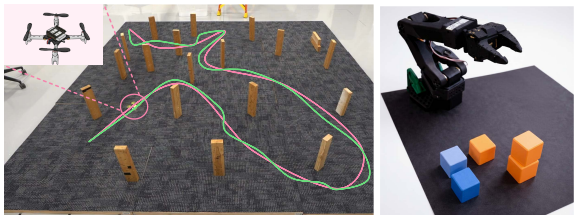

Fig. 4: Qualitative real-time hardware results. Left: Crazyflie navigation trajectories for KDP (pink) and Diffuser (green). Right: SO-100 manipulation snapshots; the full behavior is best seen in the supplementary videos.

For both domains, we report task success (Succ.), median time-to-success over successful trials (TTS), the achieved replanning frequency and time of the outer closed-loop controller (Replan, E2E), and measured planner latency per control cycle (Planner).

Domain	Method	Succ. (%)	TTS (s)	Replan (Hz)	Planner p50	E2E p50
Navigation	Diffuser	92	12.5	4.1	242	312
Navigation	**KDP**	94	12.0	38	8	26
Manipulation	Diffuser	88	35	3	311	410
Manipulation	**KDP**	90	32	17	9	47

TABLE VII: Real-world closed-loop results on a Crazyflie and an SO-100 arm. Setup: Diffuser (K=64, T=20); KDP (K=64).

The advantage of KDP is not merely that it preserves task success while using fewer NFEs, but that it moves generative planning into a qualitatively different operating regime. On both platforms, KDP matches or slightly improves closed-loop task performance while sustaining substantially higher replanning rates and dramatically lower control-cycle latency. This matters because real-world failures are often caused less by the nominal quality of a single plan than by the fact that the plan becomes stale before it can be refreshed. The results therefore validate our central claim that amortizing refinement into training retains the benefits of trajectory-level generative planning while making the planner reactive enough for real-time deployment.

VI. Conclusion

We introduced 🔑 Keyed Drifting Policies (KDP), a one-step conditional trajectory generator trained with a keyed drift field. The central design choice is *conditioning-aware neighborhoods*: drift weights are computed in a compact key space aligned with the condition, while drift updates act in full trajectory space. Combined with attraction–repulsion updates, self-negative masking, constraint masking, and a stop-gradient drifted target, KDP avoids the action-collapse failure mode observed under naïve full-window similarity. Empirically, KDP achieves competitive or improved performance on standard offline RL planning benchmarks while dramatically reducing planning latency relative to diffusion sampling. Beyond simulation, we demonstrated real-time deployment on hardware in navigation and manipulation settings, where KDP sustains substantially higher replanning rates with comparable task success. *Limitations:* goal-conditioned and sparse-reward domains remain challenging.

References

[1] M. Janner, Y. Du, J. Tenenbaum, and S. Levine, "Planning with diffusion for flexible behavior synthesis," in *International Conference on Machine Learning*, 2022.

[2] A. Ajay, Y. Du, A. Gupta, J. Tenenbaum, T. Jaakkola, and P. Agrawal, "Is conditional generative modeling all you need for decision-making?" *arXiv preprint arXiv:2211.15657*, 2022.

[3] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song, "Diffusion policy: Visuomotor policy learning via action diffusion," *The International Journal of Robotics Research*, 2025.

[4] G. Zhou, S. Swaminathan, R. V. Raju, J. S. Guntupalli, W. Lehrach, J. Ortiz, A. Dedieu, M. Lázaro-Gredilla, and K. Murphy, "Diffusion model predictive control," *arXiv preprint arXiv:2410.05364*, 2024.

[5] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Advances in Neural Information Processing Systems*, 2020.

[6] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," *arXiv preprint arXiv:2010.02502*, 2020.

[7] Z. Ding and C. Jin, "Consistency models as a rich and efficient policy class for reinforcement learning," *arXiv preprint arXiv:2309.16984*, 2023.

[8] T. Salimans and J. Ho, "Progressive distillation for fast sampling of diffusion models," *arXiv preprint arXiv:2202.00512*, 2022.

[9] Y. Song, P. Dhariwal, M. Chen, and I. Sutskever, "Consistency models," in *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[10] S. Luo, Y. Tan, L. Huang, J. Li, and H. Zhao, "Latent consistency models: Synthesizing high-resolution images with few-step inference," *arXiv preprint arXiv:2310.04378*, 2023.

[11] C. Lu, H. Chen, J. Chen, H. Su, C. Li, and J. Zhu, "Contrastive Energy Prediction for Exact Energy-Guided Diffusion Sampling in Offline Reinforcement Learning," in *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[12] H. Chen, C. Lu, C. Ying, H. Su, and J. Zhu, "Offline reinforcement learning via high-fidelity generative behavior modeling," *arXiv preprint arXiv:2209.14548*, 2022.

[13] M. Deng, H. Li, T. Li, Y. Du, and K. He, "Generative modeling via drifting," *arXiv preprint arXiv:2602.04770*, 2026.

[14] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," *Advances in neural information processing systems*, 1988.

[15] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *Advances in neural information processing systems*, 2020.

[16] I. Kostrikov, A. Nair, and S. Levine, "Offline reinforcement learning with implicit q-learning," *arXiv preprint arXiv:2110.06169*, 2021.

[17] ——, "Offline reinforcement learning with implicit q-learning," *arXiv preprint arXiv:2110.06169*, 2021.

[18] S. Fujimoto and S. S. Gu, "A Minimalist Approach to Offline Reinforcement Learning," in *Advances in Neural Information Processing Systems*, 2021.

[19] Z. Wang, A. Novikov, K. Zolna, J. S. Merel, J. T. Springenberg, S. E. Reed, B. Shahriari, N. Siegel, C. Gulcehre, N. Heess, and N. de Freitas, "Critic Regularized Regression," in *Advances in Neural Information Processing Systems*, 2020.

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

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

[22] A. Argenson and G. Dulac-Arnold, "Model-based offline planning," *arXiv preprint arXiv:2008.05556*, 2020.

[23] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision Transformer: Reinforcement Learning via Sequence Modeling," in *Advances in Neural Information Processing Systems*, 2021.

[24] M. Janner, Q. Li, and S. Levine, "Offline Reinforcement Learning as One Big Sequence Modeling Problem," in *Advances in Neural Information Processing Systems*, 2021.

[25] D. Brandfonbrener, A. Bietti, J. Buckman, R. Laroche, and J. Bruna, "When does return-conditioned supervised learning work for offline reinforcement learning?" in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, 2022.

[26] Z. Wang, J. J. Hunt, and M. Zhou, "Diffusion policies as an expressive policy class for offline reinforcement learning," *arXiv preprint arXiv:2208.06193*, 2022.

[27] W. Li, X. Wang, B. Jin, and H. Zha, "Hierarchical Diffusion for Offline Decision Making," in *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[28] W. Li, "Efficient planning with latent diffusion," *arXiv preprint arXiv:2310.00311*, 2023.

[29] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-based generative modeling through stochastic differential equations," *arXiv preprint arXiv:2011.13456*, 2020.

[30] A. Vahdat, K. Kreis, and J. Kautz, "Score-based Generative Modeling in Latent Space," in *Advances in Neural Information Processing Systems*, 2021.

[31] J. Jeon and N. Park, "Spi-gan: Denoising diffusion gans with straight-path interpolations," *arXiv preprint arXiv:2206.14464*, 2022.

[32] K. Fukunaga and L. Hostetler, "The estimation of the gradient of a density function, with applications in pattern recognition," *IEEE Transactions on Information Theory*, no. 1, 1975.

[33] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002.

[34] K. Muandet, K. Fukumizu, B. Sriperumbudur, and B. SchÃ¶lkopf, "Kernel Mean Embedding of Distributions: A Review and Beyond," *Foundations and Trends in Machine Learning*, no. 1-2, 2017.

[35] Q. Liu and D. Wang, "Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm," in *Advances in Neural Information Processing Systems*, 2016.

[36] Y. Liu, P. Ramachandran, Q. Liu, and J. Peng, "Stein variational policy gradient," *arXiv preprint arXiv:1704.02399*, 2017.

[37] A. Hyvärinen and P. Dayan, "Estimation of non-normalized statistical models by score matching." *Journal of Machine Learning Research*, 2005.

[38] M. U. Gutmann and A. Hyvärinen, "Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics," *Journal of Machine Learning Research*, 2012.

[39] A. van den Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[40] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A Simple Framework for Contrastive Learning of Visual Representations," in *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[41] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4rl: Datasets for deep data-driven reinforcement learning," *arXiv preprint arXiv:2004.07219*, 2020.

[42] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," *Advances in neural information processing systems*, 2021.

[43] M. Janner, Q. Li, and S. Levine, "Offline reinforcement learning as one big sequence modeling problem," *Advances in neural information processing systems*, 2021.