VINE: Taming Generative Control Policies for Reinforcement Learning

Rushuai Yang[1,2] **Zhuo Han**[1] **Houlin Li**[1] **Hecheng Wang**[1]
Zhichao Wu[1] **Rui Zhang**[1] **Zhaowei Zhang**[3] **Zihong Chen**[1]
Xiaohan Yan[1] **Chiming Liu**[1,†] **Yi Chen**[2] **Wei Shan**[1,†] **Maoqing Yao**[1,†]
[1]AgiBot [2]The Hong Kong University of Science and Technology [3]Peking University
[†]Corresponding Author

Abstract: Flow-matching policies have emerged as an effective policy parameterization for robot learning. They iteratively generate actions from noise, enabling highly expressive modeling of complex and multimodal action distributions. However, prior works observed that scaling these policies with value-gradient reinforcement learning (RL) often leads to training instability. Existing methods attribute this instability to iterative generation and therefore avoid end-to-end value-gradient optimization by sacrificing iterative generation, high expressiveness, or value-gradient optimization. Contrary to prior belief, we show the instability does not stem from iterative generation itself, but from the vanilla sampling strategy originally designed for behavior cloning, which becomes brittle under value-gradient RL. Motivated by this insight, we propose VINE, an RL-oriented sampling method that enables stable end-to-end value-gradient optimization for flow-matching policies. Instead of following a single flow trajectory, VINE reconstructs a new interpolation state at every denoising step, creating a stable differentiable path for value-gradient propagation while remaining compatible with the original flow-matching denoising process. As a result, VINE preserves the expressiveness and iterative generation of flow-matching without sacrificing end-to-end value-gradient optimization. Despite performing end-to-end backpropagation through all ten denoising steps, VINE achieves stable policy improvement and consistently outperforms state-of-the-art RL methods on the OGBench offline RL benchmark and real-world robotic manipulation task. Videos are available on our website: https://agibottech.github.io/vine.

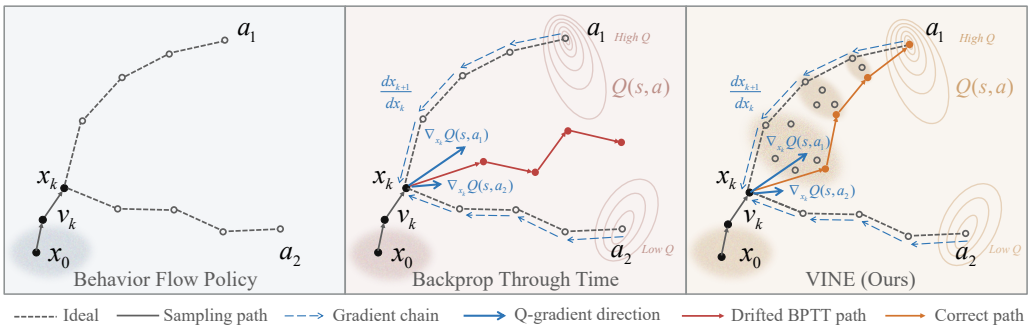

Figure 1: **Left:** A flow policy trained by behavior cloning fits the multi-modal data distribution. **Middle:** Directly backpropagating the critic gradient through the denoising steps (value BPTT) destabilizes the trajectory. **Right:** VINE produces a stable denoising trajectory that supports value-gradient BPTT toward a_1.

1 Introduction

In robotic learning, generative control policies such as diffusion and flow-matching models have achieved remarkable success for behavior cloning in a wide range of manipulation and control tasks [1, 2, 3, 4]. In general, they start from randomly sampled noise, and iteratively refine the noise using a learned velocity or score field until an executable action is produced. By combining structured noise injection with iterative refinement, these policies can represent complex and multimodal action distributions with substantially greater expressive power than conventional policy parameterizations [2]. Because human demonstrations are not always optimal and do not directly align with reward maximization, prior work has begun to further improve these generative control policies with reinforcement learning [5, 6, 7, 8, 9, 10]. In this paradigm, the policy is trained to maximize expected returns, using a learned value function to estimate future returns as the optimization signal. Yet prior work has found that such approaches suffer from severe training instability when value-gradient optimization is applied to highly expressive generative policies [11, 12, 13, 14, 15].

To mitigate this problem, existing work generally attributes this instability to the iterative generation process, and therefore avoids propagating value gradients through the full denoising trajectory [16, 17]. Specifically, prior methods typically adopt one of three strategies: (1) freezing the parameters of the generative control policy and using external guidance signals to steer the denoising dynamics [14, 18, 15, 12, 19]; (2) using (residual) Gaussian policy or distilling multi-step iterative generation into a one-step policy [13, 20]; and (3) discarding the critic's action gradient entirely and using scalar value estimates as weighting signals only [21, 22, 23]. While these approaches improve training stability, they all introduce fundamental compromises. Strategy (1) sacrifices expressiveness. It does not fully utilize representational capacity of generative model and the learnable component used for RL is restricted. Strategy (2) removes iterative generation, despite iterative refinement being widely recognized as a key advantage of generative policies. Strategy (3) sacrifices value-gradient optimization, which is generally regarded as more efficient [13, 24]. In essence, existing methods often lead to suboptimal performance without addressing challenge arising from the combination of high expressiveness, iterative generation, and value-gradient optimization in reinforcement learning for generative control policies [20, 25].

In this paper, we revisit the source of training instability in reinforcement learning for generative control policies. We identify one important bottleneck as the noise sampling process, which is poorly aligned with the optimization dynamics of reinforcement learning. Existing noise sampling strategies are primarily designed for behavior cloning. They aim to reconstruct a fixed action distribution, with expressiveness mainly reflected in how well noise can be denoised into a static, often multimodal, data distribution [2]. While this design is highly effective for supervised distribution modeling, reinforcement learning requires more than reconstructing a fixed distribution. During RL fine-tuning, the policy is repeatedly optimized through value queries from a learned critic, whose gradients encourage the policy to shift probability mass toward higher-value actions [12, 26]. This requires the iterative sampling process to transition smoothly among different high-value modes rather than simply reproduce the behavior distribution. A sampling process designed for static multimodal reconstruction is therefore forced to support a dynamic, value-driven mode-shifting process, creating the instability observed in value-gradient fine-tuning. This observation motivates a reinforcement-learning-oriented redesign of the sampling process.

Our primary contribution is an RL-oriented sampling method, which we call VINE, for fine-tuning generative control policies with end-to-end value gradients while simultaneously preserving their expressive iterative sampling structure. VINE replaces the standard single-noise flow trajectory with a sequence of reconstructed interpolation states and intermediate noise injection, providing a stable differentiable path for value-gradient backpropagation through the sampler. Our analysis shows that this reconstruction keeps every network query compatible with the original flow-matching training semantics. As a result, VINE can be applied to pretrained flow-matching policies by modifying only the sampler implementation. Empirically, we evaluate VINE on challenging offline RL benchmarks and real-world control task, where it achieves state-of-the-art performance while improving the robustness of value-gradient fine-tuning.

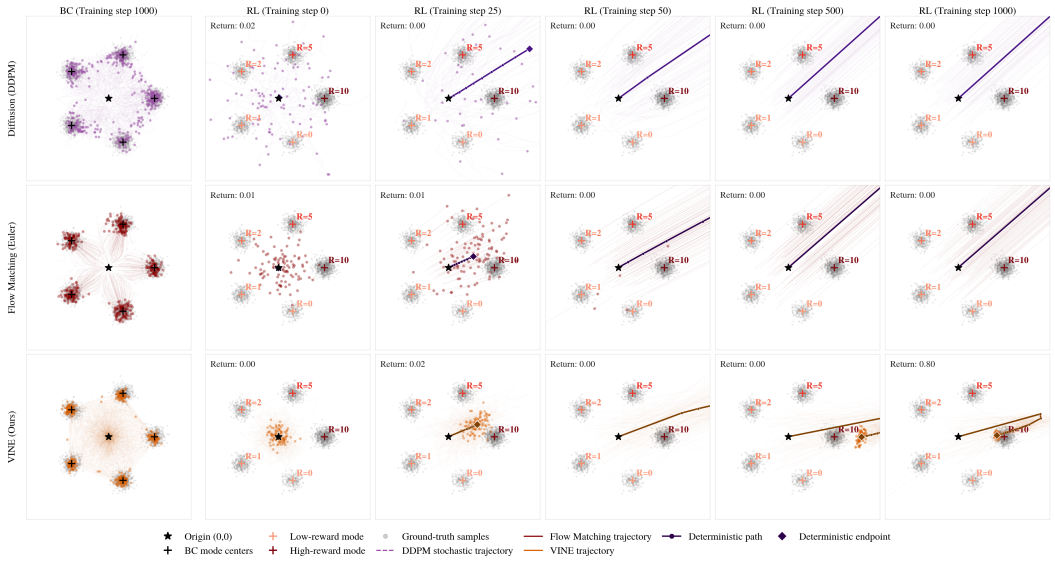

Figure 2: **Toy simulation of probability paths induced by different samplers under behavior cloning and offline value-gradient fine-tuning. Left:** Under behavior cloning, all samplers recover the multimodal data distribution, but Euler trajectories form isolated petal-like paths, DDPM explores broadly with noisy endpoints, and VINE provides structured exploration and broader state coverage. **Right:** After assigning different rewards (R) to the clusters and fine-tuning with value gradients, standard diffusion and Euler flow-matching samplers are prone to persistent directional errors, whereas VINE enables later refinement steps to correct early deviations and steer the trajectory toward high-value action regions.

2 Preliminaries

2.1 Reinforcement Learning and Actor-Critic Training

We consider a Markov Decision Process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \gamma, R, \mu)$, where $\mathcal{S}$ is the state space, $\mathcal{A} = \mathbb{R}^d$ is the continuous action space, $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition function, $\gamma \in [0, 1)$ is the discount factor, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\mu \in \Delta(\mathcal{S})$ is the initial state distribution. We denote by $\mathcal{D} = \{(s_i, a_i, r_i, s'_i)\}_{i=1}^{|\mathcal{D}|}$ the data buffer used for actor-critic training. In offline RL, $\mathcal{D}$ is fixed and pre-collected; in online RL, $\mathcal{D}$ is a replay buffer that is continually expanded through environment interaction. The goal is to learn a policy $\pi_\theta : \mathcal{S} \to \Delta(\mathcal{A})$ that maximizes the expected discounted return $J(\pi_\theta) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(s_k, a_k)\right]$. To optimize this objective from sampled transitions in $\mathcal{D}$, prior work commonly adopts behavior-regularized actor-critic frameworks [27, 28, 29]. These methods are simple to implement yet empirically strong, and have been shown to achieve competitive performance on standard offline RL benchmarks [29, 30, 31, 13]. A minimalist instantiation minimizes the following critic and actor losses:

$$\mathcal{L}_{\text{critic}}(\phi) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(Q_\phi(s, a) - r - \gamma Q_{\bar{\phi}}(s', \pi_\theta(s'))\right)^2\right], \tag{1}$$

$$\mathcal{L}_{\text{actor}}(\theta) = -\mathbb{E}_{s\sim\mathcal{D}}\left[Q_\phi(s, \pi_\theta(s))\right] + \alpha\, \mathbb{E}_{(s,a)\sim\mathcal{D}} F(a, \pi_\theta(s)), \tag{2}$$

where Q_ϕ is the critic parameterized by ϕ, which estimates the expected return of taking action a in state s, and $Q_{\bar{\phi}}$ is the corresponding target critic used for stable bootstrapping. The function $F(a, \pi_\theta(s))$ measures the discrepancy between the dataset action a and the policy action $\pi_\theta(s)$; The actor loss maximizes the critic-estimated value of the policy action, regularized by a behavior-cloning penalty that limits deviation from the dataset action. Its optimization requires differentiating through the policy π_θ, allowing the critic's action gradient $\nabla_a Q_\phi(s, a)$ to update the policy parameters. For one-step policies, such as Gaussian actors whose network output directly parameterizes the final action a, actor-critic training is well studied. However, for multi-step generative policies,

the critic's action gradient must be backpropagated through the entire generation process, making policy optimization substantially more challenging, which we discuss next.

2.2 Generative Control Policies

Generative control policies represent the action distribution by transforming a simple noise source into an executable action through an iterative generation process conditioned on the state s. Two representative constructions of this paradigm are flow matching [32, 33, 34] and diffusion models [35, 36]. Their multi-step generation process gives policies enough expressiveness to model complex and multimodal action distributions, making them particularly attractive for robot learning from diverse demonstrations [2]. Flow matching parameterizes a state-conditioned policy with a time-dependent velocity field $v_\theta(x, t;\, s)$ that transports noise into actions. In the continuous-time view, this transport is described by

$$\mathrm{d}\hat{X}_t = v_\theta(\hat{X}_t, t;\, s)\,\mathrm{d}t, \qquad \hat{X}_0 \sim \mathcal{N}(0, I_d). \tag{3}$$

The velocity field is trained with the conditional flow matching objective [32]:

$$\mathcal{L}_{\mathrm{FM}}(\theta) = \mathbb{E}_{t\sim\mathcal{U}[0,1],\, z\sim\mathcal{N}(0,I),\, a\sim p_1(\cdot|s)} \left[\|v_\theta(x_t, t;\, s) - (a - z)\|^2\right], \tag{4}$$

where $x_t = t\,a + (1-t)\,z$ is the noisy interpolation between a data action a and noise z at time t. At inference, the standard flow policy generates an action by discretizing the ODE into K Euler steps from initial noise $x_0 \sim \mathcal{N}(0, I_d)$:

$$x_{k+1} = x_k + \tfrac{1}{K}\, v_\theta(x_k, t_k;\, s), \qquad a = x_K. \tag{5}$$

Thus, the same initial noise sample is refined along a single denoising trajectory until it becomes the final action. Diffusion policies also perform iterative denoising, gradually transforming noise into actions through a sequence of intermediate states. While they are often trained as score- or noise-prediction models with a prescribed noise schedule, suitable parameterizations can rewrite them as velocity-field models [36, 37]. We adopt the flow-matching formulation because it gives a direct velocity-field view and typically enables faster generation with fewer steps. Nevertheless, VINE is a sampling-level redesign and can be applied beyond flow matching to iterative generative policies such as diffusion policies.

2.3 The Instability Challenge of Applying RL on Generative Control Policies.

When a policy is parameterized as a generative control policy, value-gradient optimization requires backpropagation through multiple denoising steps. Recall that rewards are assigned to state-action pairs, and the critic trained by Eq. (1) evaluates completed actions in the action space. In standard actor-critic training, $Q_\phi(s, a)$ assigns value to a final action a. However, in a generative control policy, this action is produced through an internal iterative denoising process rather than a single network evaluation. For a flow policy, the final action is the endpoint $a = x_K$ of the Euler sampler in Eq. (5). Updating the velocity network with the actor objective therefore requires propagating the critic's action gradient through every denoising step. By the chain rule, the actor gradient can be written as

$$\nabla_\theta Q_\phi(s, a) = \sum_{k=0}^{K-1} \tfrac{1}{K}\, \nabla_a Q_\phi(s, a) \frac{\partial x_K}{\partial x_{k+1}} \frac{\partial v_\theta(x_k, t_k;\, s)}{\partial \theta}, \qquad a = x_K. \tag{6}$$

This expression shows that each velocity update is driven by a gradient derived from the final action-value and is propagated backward through all subsequent denoising steps. As discussed in prior work, such direct value-gradient optimization is often unstable for multi-step generative policies [38]. Existing methods therefore avoid the full BPTT problem either by removing $\nabla_a Q_\phi$ from the policy update or by shortening the generation horizon K [6, 14, 13]. In contrast, a key insight is that the chain fundamentally depends on the sampler, which determines the denoising path. We therefore next discuss how to construct a stable denoising path for actor updates.

Algorithm 1 VINE: Actor-Critic Training

function GENERATE(s)
  $\hat{a}_0 \sim \mathcal{N}(0, I_d)$
  for $k = 0, \ldots, K-1$ **do** ▷ Iterative generation
    $t_k \leftarrow k/K$
    ~~$x_k \leftarrow x_k + \frac{1}{K} v_\theta(x_k, t_k;\, s)$~~ ▷ Replace Euler Method
    $z_k \sim \mathcal{N}(0, I_d)$
    $\hat{x}_k \leftarrow t_k\, \hat{a}_k + (1 - t_k)\, z_k$
    $\hat{a}_{k+1} \leftarrow \hat{x}_k + (1 - t_k)\, v_\theta(\hat{x}_k, t_k;\, s)$
  end for
  return $\hat{a}_K$
end function
while not converged **do**
  Collect transitions with GENERATE(s) and add to $\mathcal{D}$ ▷ Optionally for online RL
  Sample batch $\{(s, a, r, s')\} \sim \mathcal{D}$
  $a' \leftarrow$ GENERATE(s')
  Update ϕ to minimize $\mathbb{E}[(Q_\phi(s, a) - r - \gamma Q_{\bar{\phi}}(s', a'))^2]$ ▷ Train critic Q_ϕ
  $\hat{a}_K \leftarrow$ GENERATE(s)
  Update θ to minimize $-Q_\phi(s, \hat{a}_K) + \alpha \|\hat{a}_K - a\|^2$ ▷ Train velocity v_θ via BPTT
end while
return policy $\pi_\theta(s) \equiv$ GENERATE(s)

3 VINE: Value-gradient Iterative Noise Exploration

To make value-gradient propagation through the sampling chain more stable, we reconsider how the sampler transports one intermediate state to the next. The key observation is that flow-matching training adds noise to each target action by sampling many noise levels t and noise draws z. The velocity field therefore learns to reach the same action from many possible paths, whereas the Euler ODE sampler in Eq. (5) follows only one of them. This gives us the freedom to construct a different sampling path. VINE uses this freedom to build an RL-oriented sampler: at each step, it reconstructs a noisy intermediate state around the current action estimate, then applies the same velocity field to produce a refined estimate. Recall that in standard flow-matching training, noise is introduced by interpolating between a ground-truth action and a Gaussian noise sample: $x_t = t\, a + (1 - t)z$. If we have a current action estimate $\hat{a}_k$ predicted before the k denoising step, the natural way to inject noise while staying consistent with flow matching is to use the same interpolation form:

$$\hat{x}_k = t_k\, \hat{a}_k + (1 - t_k)\, z_k, \quad z_k \sim \mathcal{N}(0, I_d). \tag{7}$$

We denote the reconstructed interpolation state by $\hat{x}_k$ to distinguish it from the Euler state x_k, the noise z_k is sampled independently at each step. We take $0 \le t_1 < \cdots < t_K \le 1$ to be a monotonically increasing time schedule, so that $\hat{x}_k$ is progressively dominated by the current action estimate $\hat{a}_k$ as k grows. Note that $\hat{x}_k$ is a fresh noisy interpolation state used as the network input, rather than a state obtained in Eq. (5). The remaining question is how to recover an action estimate from this noisy state during RL rollout, where the ground-truth endpoint is unavailable. From a probabilistic view, the flow-matching objective trains v_θ to predict the conditional velocity from a noisy interpolation state. Under the optimal velocity $v^\star$, the transformed quantity $x_t + (1 - t)v^\star(x_t, t;\, s)$ estimates the posterior mean of the final action. Applying this closed form to $\hat{x}_k$ yields,

$$\hat{a}_{k+1} = \hat{x}_k + (1 - t_k)\, v_\theta(\hat{x}_k, t_k;\, s), \tag{8}$$

Starting from an initial action estimate $\hat{a}_0 \sim \mathcal{N}(0, I_d)$, we recursively compute the interpolation state with Eq.(7) and endpoint prediction with Eq.(8) so the final action becomes $a = \hat{a}_K$. This sampling method has two practical consequences. First, each network query remains compatible with the flow-matching training semantics: the input is a noisy interpolation state, and the velocity field is used in its form. Thus, VINE changes the sampling path without requiring a new generative objective or an auxiliary guidance model; we provide a more formal justification in Appendix A.1.

Second, fresh noise is injected locally around the current action estimate at every step. This makes stochasticity act as structured local exploration during refinement, rather than letting a single initial noise sample determine the entire action trajectory as in Eq. (5). Moreover, since t_k is monotonically increasing, the noise weight $(1 - t_k)$ decreases across steps, moving the sampler from coarse stochastic exploration toward finer deterministic refinement of the final action.

To understand the sampling process, we visualize this behavior with a toy simulation in noise space in Fig. 2. Under behavior cloning, both the flow-matching and VINE are trained using the same flow-matching objective in Eq. (4), while the DDPM baseline is trained with the standard diffusion loss. At inference, we sample many initial points from $\mathcal{N}(0, I)$ and trace their trajectories using three samplers: Euler, a DDPM sampler, and VINE. Euler trajectories form isolated petal-like paths with few bridges between modes. DDPM explores more broadly, but its stochastic transitions introduce endpoint-prediction error. In contrast, VINE traverses the full pentagon with bridging paths between neighboring modes, yielding broader state coverage while preserving accurate endpoints. This coverage is useful under critic-gradient fine-tuning, where the desired action may shift from one mode to another.

4 Related Work

Gaussian and one-step actors for RL. Classical actor-critic methods typically parameterize the policy with a Gaussian, deterministic, or residual actor, making value-gradient optimization straightforward because the critic's action gradient is backpropagated through a single network evaluation [27, 28, 29, 39]. This simplicity makes Gaussian-style actors strong baselines for offline RL, including behavior-regularized methods [40, 41]. Recent work applies a similar principle to generative policies by shortening the sampling chain, using consistency models, shortcut models, or one-step flow policies before applying value-gradient optimization [42, 43, 13, 44, 45]. These approaches improve stability by simplifying the policy computation graph, but they sacrifice the iterative refinement and expressive multimodal modeling that motivate generative control policies.

RL fine-tuning of diffusion and flow policies. Diffusion and flow-matching policies have been widely adopted in robot learning and reinforcement learning because they can represent complex multimodal action distributions [2, 46, 3, 47]. The central challenge is how to optimize these multi-step generative policies with a learned critic [48, 49]. Existing methods differ mainly in where the value signal is applied relative to the denoising chain. *Around-the-chain methods* avoid direct BPTT through the full sampler by constructing objectives around the denoising process. Advantage- or value-weighted methods avoid critic action gradients by reweighting generative-model objectives with scalar critic values [50, 51, 52]. Other methods convert value, energy, or guidance signals into denoising objectives or guidance terms, steering generation without directly optimizing through the complete sampling chain [14, 19, 53, 54]. Adjoint matching [55] transports terminal gradients backward to provide step-wise supervision, and QAM [12] applies this idea to offline RL. Related variants make different approximation and complexity trade-offs [56, 57, 58, 59]. *Outside-the-chain methods* improve actions after or outside the base sampling process through value-guided selection [22, 20], residual corrections in action or noise space [60, 61, 18], or gradient-based action editing [62]. These methods can improve sampled actions, but the value signal acts outside the internal denoising dynamics. *Through-the-chain methods* directly differentiate through the full denoising process [63, 64, 65, 11]. This is the most direct way to use critic action gradients, but prior work has found it unstable for expressive multi-step policies. In contrast, VINE keeps the full iterative chain and supports stable value-gradient BPTT by redesigning the noise injection process, rather than bypassing the chain, shortening it, or applying value signals only outside it.

5 Experiments

We conduct experiments to study how well VINE optimizes generative control policies on both benchmark tasks and real-world manipulation. Concretely, we ask three research questions: (1) Can

	Gaussian	Around the Chain				Outside the Chain							Through the Chain		
Task Category	ReBRAC	FQL	FAWAC	IFQL	QAM	DAC	QSM	CGQL	CGQL-M	CGQL-L	DSRL	FEdit	FBRAC	BAM	VINE
antmaze-large	94	76	17	36	81	88	91	76	71	65	61	58	2	84	**99**
antmaze-giant	57	0	0	1	18	16	15	0	4	3	3	2	0	1	**76**
humanoidmaze-medium	69	68	24	86	67	83	83	60	42	62	53	22	39	60	**87**
humanoidmaze-large	17	9	0	24	11	0	10	5	6	6	3	3	0	5	**46**
scene-sparse	65	78	38	84	97	68	86	38	74	88	**99**	62	50	98	73
puzzle-3x3-sparse	79	70	3	**100**	**100**	68	53	48	**100**	90	87	99	0	56	95
cube-double	9	46	2	11	64	35	56	38	41	45	**74**	40	0	47	4
cube-triple	1	3	0	0	3	5	3	**8**	**8**	**8**	1	2	0	3	1
all (40 tasks)	49	44	10	43	55	45	50	34	43	46	48	36	12	44	**60**

Table 1: **Offline RL results on OGBench.** We follow the QAM evaluation protocol [12]. Around the Chain denotes methods that update the generative control policy with gradient information but avoid BPTT through the full sampler; Outside the Chain denotes methods that freeze the generative policy and optimize actions externally; Through the Chain denotes methods that directly perform BPTT through the sampler. VINE achieves better performance on 40 tasks in total.

VINE optimize policies effectively compared to representative offline RL methods on OGBench? (2) Can VINE make flow-matching stable for real-world online RL? (3) Why VINE helps?

5.1 Experimental Setup

Offline RL setting. We first evaluate VINE in the offline RL setting on eight domains from OGBench, each containing five tasks: antmaze-large, antmaze-giant, humanoidmaze-medium, humanoidmaze-large, scene-sparse, puzzle-3x3-sparse, cube-double, and cube-triple. The antmaze and humanoidmaze domains test long-horizon navigation from diverse offline trajectories, while the scene, puzzle, and cube domains require solving sparse-reward manipulation-style tasks. For offline RL baselines, we compare against representative Gaussian, flow, and diffusion policy methods, grouped by how they propagate value-gradient updates through or around the sampling chain. These include ReBRAC [29], FQL and FAWAC [13], IFQL [22], QAM [12], DAC [19], QSM [14], CGQL variants [53], DSRL [18], FEdit [20], FBRAC [13], and BAM [55, 12].

Real-world online RL setting. We further evaluate VINE on a real-world socket insertion task to assess whether stable value-gradient optimization can transfer to physical robotic manipulation. In this task, the robot must pick up a plug from the table and insert it into a socket, requiring precise contact-rich control under tight insertion tolerances. We evaluate each method over 20 trials and report the success rate, wall-clock fine-tuning time, and the fraction of trajectories requiring human intervention. We compare against representative real-world reinforcement learning baselines, including SAC-Flow [11], Hil-SERL [7], DSRL [18], RLT [24], and EXPO [20]. All methods receive images and proprioceptive states as inputs and directly predict low-level actions or action chunks. We divide the baselines into two categories according to their training paradigms. DSRL, RLT, and EXPO leverage pretrained policies. Specifically, we initialize these methods from $\pi_{0.5}$ and perform BC warm-up using 15 pre-collected human demonstrations before online RL. In contrast, Hil-SERL, SAC-Flow, and VINE are trained without pretrained initialization. For Hil-SERL, SAC-Flow, and VINE, we use a ResNet-10 encoder to extract visual features. SAC-Flow follows flow-matching objective but employs a transformer-based action head, whereas Hil-SERL and VINE use a three-layer MLP action head for single action prediction. For fair comparison, SAC-Flow, and VINE all use 10 denoising steps during action generation.

5.2 Can VINE optimize effectively compared to representative methods on OGBench?

Table 1 reports results on 40 OGBench tasks across eight domains. Following QAM [12], we train each task with 12 seeds and report the mean performance with 95% confidence intervals computed by bootstrapping with 5000 samples. VINE achieves the best aggregate score, with especially large gains on long-horizon navigation domains such as `antmaze-giant` and `humanoidmaze-large`.

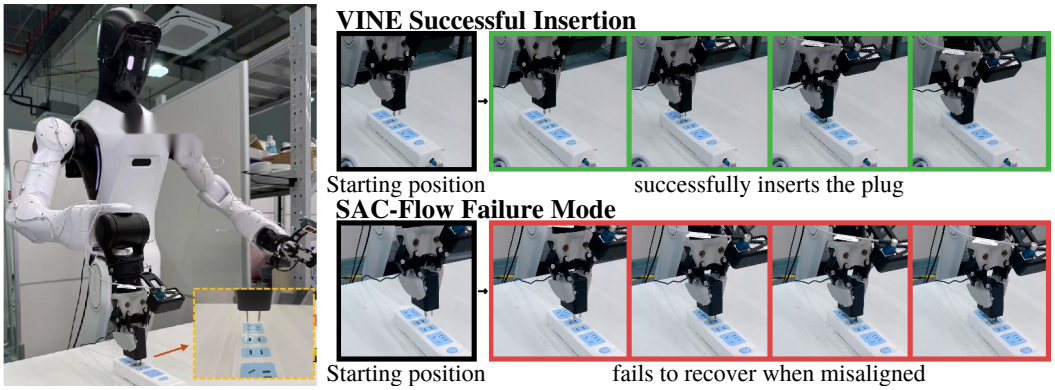

Figure 3: **Real-world online RL setting**. VINE learns an policy that insert successfully from any location. SAC-Flow fails to learn a successful policy during online RL.

Table 2: **Online RL results on a real-world human-in-the-loop manipulation task.** We evaluate plug insertion and report success rate over 20 trials, online RL fine-tuning time, and the fraction of online rollout steps requiring human intervention.

Plug Insertion	BC init.	DSRL	RLT	EXPO	SAC-Flow	Hil-SERL	VINE
Success rate (↑)	10/20	17/20	17/20	19/20	12/20	16/20	**20/20**
Time (↓)	–	50 min	50 min	50 min	20 min	20 min	**20 min**
Human-intervention (↓)	–	–	29.7%	56.3%	74.6%	55.9%	**19.2%**

These results suggest that VINE can stably optimize expressive generative policies with value gradients in offline RL.

5.3 Can VINE make flow-matching stable for real-world online RL?

We evaluate VINE on a real-world socket insertion task as shown in Figure 3, where the robot must pick up a plug and insert it into a fixed socket under tight contact tolerances. We compare against baselines under human-in-the-loop real-world RL setting [7, 24]. Human-in-the-loop means human operator intervenes and provides corrections during autonomous execution. We report success rate, training time, and human-intervention ratio in Table 2. VINE achieves the highest success rate with the shortest fine-tuning time and the lowest human-intervention ratio among the online methods.

5.4 Why VINE Helps?

We use ablations to isolate two design choices in VINE: per-step stochastic interpolation and iterative endpoint-prediction. Both are evaluated under the same actor-critic training protocol as the full method, changing only the component under study.

VINE stabilizes value-gradient backpropagation. We compare the standard Euler sampler used by flow matching with VINE. The Euler sampler samples noise only once at initialization and follows a fixed deterministic denoising trajectory throughout inference, whereas VINE reconstructs a noisy interpolation state by injecting fresh Gaussian noise, at every denoising step. As shown in Fig. 4, VINE consistently achieves substantially higher success rates across all five OGbench tasks. Moreover, VINE maintains significantly smaller and more stable backpropagated gradients throughout training, whereas the Euler sampler exhibits frequent large gradient spikes that correlate with unstable optimization. These results demonstrate that VINE stabilizes end-to-end value-gradient propagation through the denoising chain, leading to both more robust optimization and stronger policy performance.

Iterative generation improves action quality from coarse to fine. We evaluate different intermediate denoising steps with $K = 1, 4, 6, 8, 10$. Figure 5 shows that early intermediate actions are

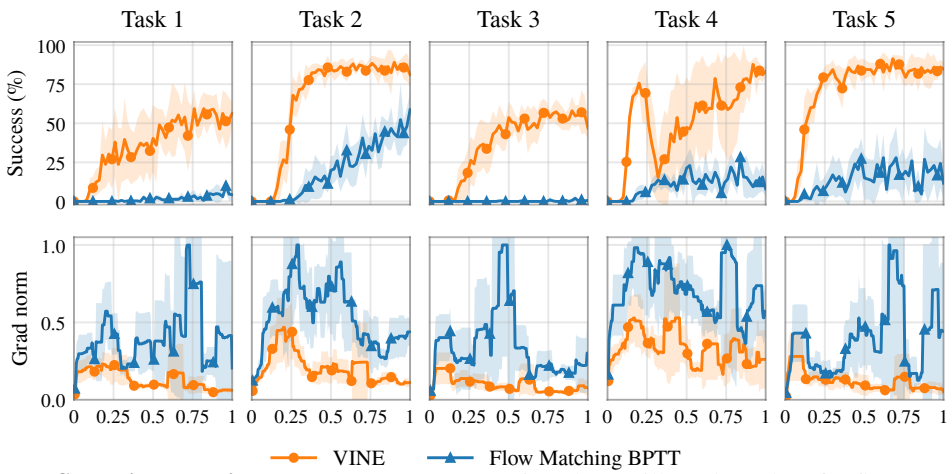

Figure 4: **Sampling ablation.** We compare VINE against the vanilla Euler solver for flow-matching on OGBench. VINE consistently achieves higher success rates and more stable backpropagated gradients.

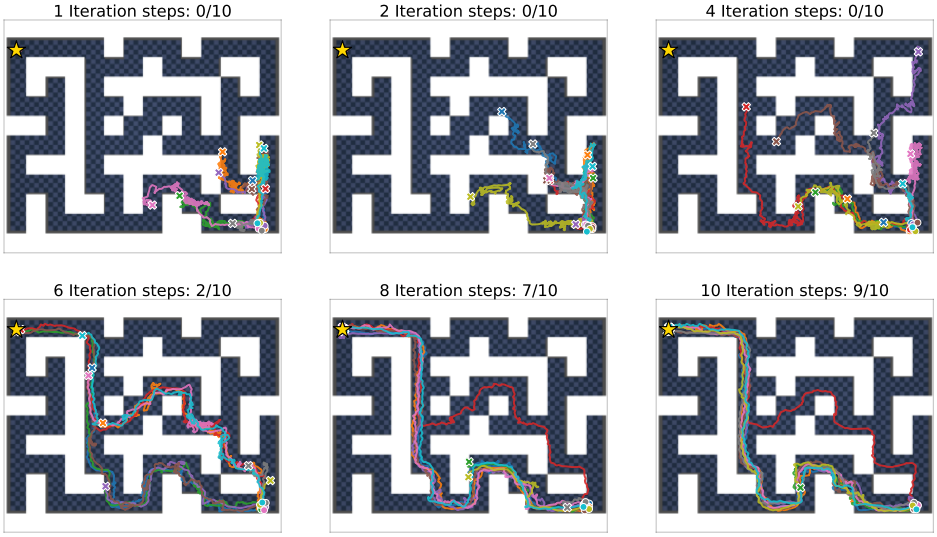

Figure 5: **Preserving the Iterative Refinement.** We generate the final action using different numbers of denoising steps K and evaluate the resulting policies on the AntMaze task. As the number of refinement iterations increases, the generated action quality improves, leading to higher task success rates.

insufficient for solving the long-horizon AntMaze task. With only 1 or 4 denoising steps, the agent largely remains near the start region and fails to reach the goal. VINE does not merely produce a good action in the first few iterations; instead, the later refinement steps are essential for transforming an initially poor action proposal into a task-completing action.

6 Conclusion

In this paper, we identify the training instability of value-gradient RL with expressive generative policies originates in part from the sampler and proposed VINE, an RL-oriented sampling method that enables stable value-gradient optimization toward expressive policies. VINE remains compatible with pretrained flow-matching policies. Empirically, VINE achieves stable policy improvement with a 10-step denoising process and consistently outperforms state-of-the-art baselines on both OGBench and real-robot task. A promising future direction is to extend VINE to large foundation models and adapt variant RL algorithms for broader validation.

References

[1] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Robotics: Science and Systems*, 2023.

[2] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 2023.

[3] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[4] Physical Intelligence, A. Amin, R. Aniceto, A. Balakrishna, K. Black, K. Conley, G. Connors, J. Darpinian, K. Dhabalia, J. DiCarlo, et al. $\pi^*_{0.6}$: a vla that learns from experience. *arXiv preprint*, arXiv:2511.14759, 2025. arXiv preprint arXiv:2511.14759.

[5] K. Chen, Z. Liu, T. Zhang, Z. Guo, S. Xu, H. Lin, H. Zang, X. Li, Q. Zhang, Z. Yu, et al. $\pi_{\texttt{rl}}$: Online rl fine-tuning for flow-based vision-language-action models. *arXiv preprint*, arXiv:2510.25889, 2025. arXiv preprint arXiv:2510.25889.

[6] T. Zhang, C. Yu, S. Su, and Y. Wang. Reinflow: Fine-tuning flow matching policy with online reinforcement learning. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, pages 106282–106319. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/98d6f928497d9eaad9f320fb2db3040d-Paper-Conference.pdf.

[7] J. Luo, C. Xu, J. Wu, and S. Levine. Precise and dexterous robotic manipulation via human-in-the-loop reinforcement learning. *Science Robotics*, 10(105):eads5033, 2025.

[8] R. Yang, H. Wang, Z. Wu, C. Liu, X. Yan, X. Du, S. Yue, C. Zhang, Y. Wang, Y. Liu, L. Qi, Y. Chen, W. Shan, and M. Yao. ALOE: Action-level off-policy evaluation for vision-language-action model post-training, 2026. URL https://arxiv.org/abs/2602.12691.

[9] R. Yang, Z. Feng, T. Zhang, K. Wang, C. Zhang, L. Zhao, X. Su, Y. Chen, and J. Bian. Discover, learn, and reinforce: Scaling vision-language-action pretraining with diverse RL-generated trajectories, 2025. URL https://arxiv.org/abs/2511.19528.

[10] R. Yang, H. Wei, R. Zhang, Z. Feng, X. Chen, T. Li, C. Zhang, L. Zhao, J. Bian, X. Su, and Y. Chen. Beyond human demonstrations: Diffusion-based reinforcement learning to generate data for VLA training, 2025. URL https://arxiv.org/abs/2509.19752.

[11] Y. Zhang, S. Yu, T. Zhang, M. Guang, H. Hui, K. Long, Y. Wang, C. Yu, and W. Ding. Sac flow: Sample-efficient reinforcement learning of flow-based policies via velocity-reparameterized sequential modeling, 2026. URL https://arxiv.org/abs/2509.25756.

[12] Q. Li and S. Levine. Q-learning with adjoint matching. *International Conference on Learning Representations*, 2026.

[13] S. Park, Q. Li, and S. Levine. Flow q-learning. In A. Singh, M. Fazel, D. Hsu, S. Lacoste-Julien, F. Berkenkamp, T. Maharaj, K. Wagstaff, and J. Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 48104–48127. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/park25f.html.

[14] M. Psenka, A. Escontrela, P. Abbeel, and Y. Ma. Learning a diffusion model policy from rewards via Q-score matching. In *International Conference on Machine Learning*, 2024.

[15] P. Dhariwal and A. Nichol. Diffusion models beat gans on image synthesis. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 8780–8794. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/49ad23d1ec9fa4bd8d77d02681df5cfa-Paper.pdf.

[16] H. Xu, K. Hu, S. Sojoudi, and A. Zhang. Reinforcement learning via value gradient flow. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=JLL4VNVhM9.

[17] Z. Liu, T. Z. Xiao, C. Domingo-Enrich, W. Liu, and D. Zhang. Value gradient guidance for flow matching alignment. In *NeurIPS 2025 Workshop on Structured Probabilistic Inference & Generative Modeling*, 2025. URL https://openreview.net/forum?id=3oCOEL13OT.

[18] A. Wagenmaker, M. Nakamoto, Y. Zhang, S. Park, W. Yagoub, A. Nagabandi, A. Gupta, and S. Levine. Steering your diffusion policy with latent space reinforcement learning. *Conference on Robot Learning*, 2025.

[19] L. Fang, R. Liu, J. Zhang, W. Wang, and B. Jing. Diffusion actor-critic: Formulating constrained policy iteration as diffusion noise regression for offline reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2025.

[20] P. Dong, Q. Li, D. Sadigh, and C. Finn. EXPO: Stable reinforcement learning with expressive policies. *arXiv preprint arXiv:2507.07986*, 2025.

[21] X. B. Peng, A. Kumar, G. Zhang, and S. Levine. Advantage weighted regression: Simple and scalable off-policy reinforcement learning, 2020. URL https://openreview.net/forum?id=H1gdF34FvS.

[22] P. Hansen-Estruch, I. Kostrikov, M. Janner, J. G. Kuba, and S. Levine. IDQL: Implicit Q-learning as an actor-critic method with diffusion policies. *arXiv preprint arXiv:2304.10573*, 2023.

[23] P. Dong, C. Zheng, C. Finn, D. Sadigh, and B. Eysenbach. Value flows. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=2VyNYUVF2k.

[24] C. Xu, J. T. Springenberg, M. Equi, A. Amin, A. Esmail, S. Levine, and L. Ke. Rl token: Bootstrapping online rl with vision-language-action models. *arXiv preprint arXiv:2604.23073*, 2026.

[25] C. Zhang, Z. Wan, F. Chen, X. Yu, I. Tsang, and B. An. Gorl: An algorithm-agnostic framework for online reinforcement learning with generative policies. *arXiv preprint arXiv:2512.02581*, 2025.

[26] S. Zhang, Y. Lou, H. Cheng, Y. Guo, C. Fu, Y. Lyu, X. Zhang, H. Li, P. Wang, Z. Wang, et al. Force: Efficient vla reinforcement fine-tuning via value-calibrated warm-up and self-distillation. *arXiv preprint arXiv:2606.26006*, 2026.

[27] Y. Wu, G. Tucker, and O. Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[28] S. Fujimoto and S. S. Gu. A minimalist approach to offline reinforcement learning. *Advances in Neural Information Processing Systems*, 2021.

[29] D. Tarasov, A. Nikulin, D. Akimov, V. Kurenkov, and S. Kolesnikov. Corl: Research-oriented deep offline reinforcement learning library. *Advances in Neural Information Processing Systems*, 2023.

[30] R. Yang, C. Bai, H. Guo, S. Li, B. Zhao, Z. Wang, P. Liu, and X. Li. Behavior contrastive learning for unsupervised skill discovery. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 39183–39204. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/yang23a.html.

[31] C. Bai, R. Yang, Q. Zhang, K. Xu, Y. Chen, T. Xiao, and X. Li. Constrained ensemble exploration for unsupervised skill discovery. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 2418–2442. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/bai24d.html.

[32] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, and M. Nickel. Flow matching for generative modeling. *International Conference on Learning Representations*, 2023.

[33] X. Liu, C. Gong, and Q. Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *International Conference on Learning Representations*, 2023.

[34] M. S. Albergo and E. Vanden-Eijnden. Building normalizing flows with stochastic interpolants. *International Conference on Learning Representations*, 2023.

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

[36] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations. *International Conference on Learning Representations*, 2021.

[37] D. P. Kingma and R. Gao. Understanding diffusion objectives as the elbo with simple data augmentation. *Advances in Neural Information Processing Systems*, 2024.

[38] K. Shi, J. Shi, P. Hebbar, Z. Zhao, T. Amarnath, Y. Su, S. Bahl, and D. Pathak. Flowdpg: Deterministic policy gradient on flow matching policies for real-world manipulation, 2026. URL https://arxiv.org/abs/2606.22303.

[39] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In E. P. Xing and T. Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China, 22–24 Jun 2014. PMLR. URL https://proceedings.mlr.press/v32/silver14.html.

[40] S. Fujimoto, D. Meger, and D. Precup. Off-policy deep reinforcement learning without exploration. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2052–2062. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/fujimoto19a.html.

[41] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/c2073ffa77b5357a498057413bb09d3a-Paper.pdf.

[42] Z. Ding and C. Jin. Consistency models as a rich and efficient policy class for reinforcement learning. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Learning Representations*, volume 2024, pages 53047–53066, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/e9e1a0abc1a5b19a4aeb80dab19c82ae-Paper-Conference.pdf.

[43] T. Chen, Z. Wang, and M. Zhou. Diffusion policies creating a trust region for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 2024.

[44] N. Espinosa-Dice, Y. Zhang, Y. Chen, B. Guo, O. Oertell, G. Swamy, K. Brantley, and W. Sun. Scaling offline RL via efficient and expressive shortcut models. *arXiv preprint arXiv:2505.22866*, 2025.

[45] T. Chen, H. Ma, N. Li, K. Wang, and B. Dai. One-step flow policy mirror descent. *arXiv preprint arXiv:2507.23675*, 2025.

[46] Y. Ze, G. Zhang, K. Zhang, C. Hu, M. Wang, and H. Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. In *2nd Workshop on Dexterous Manipulation: Design, Perception and Control (RSS)*, 2024. URL https://openreview.net/forum?id=KwwJuZIBXH.

[47] J. Barreiros, A. Beaulieu, A. Bhat, R. Cory, E. Cousineau, H. Dai, C.-H. Fang, K. Hashimoto, M. Z. Irshad, M. Itkina, et al. A careful examination of large behavior models for multitask dexterous manipulation. *Science Robotics*, 11(113):eaea6201, 2026. doi:10.1126/scirobotics.aea6201. URL https://www.science.org/doi/abs/10.1126/scirobotics.aea6201.

[48] Z. Zhu, H. Zhao, H. He, Y. Zhong, S. Zhang, H. Guo, T. Chen, and W. Zhang. Diffusion models for reinforcement learning: A survey, 2024. URL https://arxiv.org/abs/2311.01223.

[49] M. Uehara, Y. Zhao, T. Biancalani, and S. Levine. Understanding reinforcement learning-based fine-tuning of diffusion models: A tutorial and review, 2024. URL https://arxiv.org/abs/2407.13734.

[50] S. Zhang, W. Zhang, and Q. Gu. Energy-weighted flow matching for offline reinforcement learning. In *International Conference on Learning Representations*, 2025.

[51] B. Kang, X. Ma, C. Du, T. Pang, and S. Yan. Efficient diffusion policies for offline reinforcement learning. In *Neural Information Processing Systems*, 2023.

[52] S. Ding, K. Hu, Z. Zhang, K. Ren, W. Zhang, J. Yu, J. Wang, and Y. Shi. Diffusion-based reinforcement learning via Q-weighted variational policy optimization. In *Neural Information Processing Systems*, 2024.

[53] 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 *International Conference on Machine Learning*, 2023.

[54] K. Frans, S. Park, P. Abbeel, and S. Levine. Diffusion guidance is a controllable policy improvement operator. *arXiv preprint arXiv:2505.23458*, 2025.

[55] C. Domingo-Enrich, W. Chen, and B. Amos. Adjoint matching: Fine-tuning flow and diffusion generative models with memoryless stochastic optimal control. *arXiv preprint arXiv:2409.03698*, 2025.

[56] M. Uehara, Y. Zhao, K. Black, E. Hajiramezanali, G. Scalia, N. L. Diamant, A. M. Tseng, T. Biancalani, and S. Levine. Fine-tuning of continuous-time diffusion models as entropy-regularized control. *arXiv preprint arXiv:2402.15194*, 2024.

[57] A. Bergmeister, S. Jegelka, N. Nüsken, C. Domingo-Enrich, and J. Pidstrigach. Reinforce adjoint matching: Scaling rl post-training of diffusion and flow-matching models. *arXiv preprint*, 2025.

[58] Z. Guo, J. Sheng, D. D. Yao, and W. Tang. Improved techniques for fine-tuning flow models via adjoint matching: A deterministic control pipeline. *arXiv preprint arXiv:2605.06583*, 2026.

[59] J. Shin, D. Shin, J. Lee, J. Choi, and J. Choi. Efficient adjoint matching for fine-tuning diffusion models. *arXiv preprint arXiv:2605.11480*, 2026.

[60] L. Ankile, A. Simeonov, I. Shenfeld, M. Torne, and P. Agrawal. From imitation to refinement–residual rl for precise assembly. *arXiv preprint arXiv:2407.16677*, 2024.

[61] X. Yuan, T. Mu, S. Tao, Y. Fang, M. Zhang, and H. Su. Policy decorator: Model-agnostic online refinement for large policy model. *arXiv preprint arXiv:2412.13630*, 2024.

[62] M. S. Mark, T. Gao, G. G. Sampaio, M. K. Srirama, A. Sharma, C. Finn, and A. Kumar. Policy-agnostic RL: Offline RL and online RL fine-tuning of any class and backbone. In *Robot Learning Workshop*, 2025.

[63] Z. Wang, J. J. Hunt, and M. Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. In *International Conference on Learning Representations*, 2023.

[64] L. He, L. Shen, L. Zhang, J. Tan, and X. Wang. DiffCPS: Diffusion model based constrained policy search for offline reinforcement learning. *arXiv preprint arXiv:2310.05333*, 2023.

[65] R. Zhang, Z. Luo, J. Sjölund, T. B. Schön, and P. Mattsson. Entropy-regularized diffusion policy with Q-ensembles for offline reinforcement learning. In *Neural Information Processing Systems*, 2024.

[66] D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus), 2016. URL https://arxiv.org/abs/1606.08415.

[67] J. L. Ba. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[68] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. URL https://arxiv.org/abs/1412.6980.

A Appendix

A.1 Compatibility with Pretrained Flow Policies

A natural question is whether VINE can leverage an existing flow-matching pretrained policy. The following theorem justifies that the same iterative procedure would also apply if a pretrained FM field were used as the velocity model.

Theorem 1 (FM-VINE Consistency)**.** *Let $v^*(x, t;\, s)$ be the pointwise minimizer of the conditional flow matching loss $\mathcal{L}_{\mathrm{FM}}$ (Eq. 4). Then for any query point $x \in \mathbb{R}^d$ and time $t \in (0, 1)$,*

$$x + (1 - t)\, v^*(x, t;\, s) = \mathbb{E}[a \mid x_t = x,\, t,\, s], \tag{9}$$

regardless of how x was constructed. In particular, our endpoint prediction $\hat{a}_k = \hat{x}_k + (1 - t_k)\, v^*(\hat{x}_k, t_k;\, s) = \mathbb{E}[a \mid \hat{x}_k, t_k, s]$ is the Bayes-optimal action estimate given the noisy intermediate $\hat{x}_k$, even though $\hat{x}_k$ was not generated by the FM ODE.

This theorem implies that a flow-matching pretrained policy can be used as the velocity field in VINE—for both inference (using VINE's iterative generation) and post-training (fine-tuning with value gradients via BPTT)without changing the generation rule.

Corollary 1 (Contraction under Deterministic VINE)**.** *Suppose $p(a \mid s) = \mathcal{N}(\mu_s, \sigma_a^2 I)$. Under deterministic VINE ($z_k = 0$, $\hat{a}_0 = 0$) with optimal velocity v^*, the action estimates satisfy the recursion $\hat{a}_k = \lambda_k\, \hat{a}_{k-1} + (1 - \lambda_k)\, \mu_s$ with contraction factor*

$$\lambda_k = \frac{t_k^2 \sigma_a^2}{t_k^2 \sigma_a^2 + (1 - t_k)^2} \in (0, 1), \tag{10}$$

giving $\|\hat{a}_k - \mu_s\| = \prod_{j=1}^{k} \lambda_j \cdot \|\mu_s\| \to 0$.

Since $\lambda_k < 1$ for all $t_k < 1$, deterministic VINE contracts monotonically to μ_s. The bulk of convergence occurs at early steps where t_k is small and $(1-t_k)^2$ dominates the denominator, making $\lambda_k \approx 0$.

A.2 Proof of Theorem 1

Proof. Under the conditional flow matching objective with linear interpolation paths, the training input at time t is $x_t = (1-t)\, z + t\, a$, where $z \sim \mathcal{N}(0, I)$ and $a \sim p(a \mid s)$. The conditional velocity target is $a - z = \frac{a - x_t}{1-t}$.

The FM loss is a pointwise squared error:

$$\mathcal{L}_{\mathrm{FM}}(\theta) = \mathbb{E}_{t,z,a}\left[\left\| v_\theta(x_t, t;\, s) - \frac{a - x_t}{1 - t} \right\|^2\right]. \tag{11}$$

For any fixed query point x and time t, the pointwise minimizer of the squared loss is the conditional expectation:

$$v^*(x, t;\, s) = \mathbb{E}\left[\frac{a - x_t}{1 - t} \,\middle|\, x_t = x,\, t,\, s\right] = \frac{\mathbb{E}[a \mid x_t = x, t, s] - x}{1 - t}. \tag{12}$$

Therefore:

$$x + (1 - t)\, v^*(x, t;\, s) = x + \mathbb{E}[a \mid x_t = x, t, s] - x = \mathbb{E}[a \mid x_t = x, t, s]. \tag{13}$$

This holds for any $x \in \mathbb{R}^d$, regardless of its origin. In particular, setting $x = \hat{x}_k = t_k\, \hat{a}_{k-1} + (1 - t_k)\, z_k$ gives

$$\hat{a}_k = \hat{x}_k + (1 - t_k)\, v^*(\hat{x}_k, t_k;\, s) = \mathbb{E}[a \mid x_{t_k} = \hat{x}_k, t_k, s], \tag{14}$$

which is the Bayes-optimal action estimate under squared error loss, conditioned on the noisy intermediate $\hat{x}_k$. □

A.3 Proof of Corollary 1

Proof. Let $p(a \mid s) = \mathcal{N}(\mu_s, \sigma_a^2 I)$ and consider deterministic VINE with $z_k = 0$ and $\hat{a}_0 = 0$. The noisy intermediate is $\hat{x}_k = t_k\, \hat{a}_{k-1}$.

Under the linear interpolation $x_t = (1-t)\, z + t\, a$ with $z \sim \mathcal{N}(0, I)$ and $a \sim \mathcal{N}(\mu_s, \sigma_a^2 I)$, the joint (x_t, a) is Gaussian. The conditional posterior is:

$$p(a \mid x_t = x, t) = \mathcal{N}\left(\frac{t\,\sigma_a^2\, x + (1-t)^2\, \mu_s}{t^2\sigma_a^2 + (1-t)^2},\ \Sigma_{\text{post}}\right), \tag{15}$$

where we used $\mathrm{Var}[x_t] = t^2\sigma_a^2 + (1-t)^2$ (isotropic components) and the standard linear-Gaussian conditioning formula. The posterior covariance Σ_{post} is irrelevant for the mean.

By Theorem 1, endpoint prediction with optimal velocity gives:

$$\hat{a}_k = \mathbb{E}[a \mid x_{t_k} = \hat{x}_k, t_k] = \frac{t_k\, \sigma_a^2\, \hat{x}_k + (1-t_k)^2\, \mu_s}{t_k^2\sigma_a^2 + (1-t_k)^2}. \tag{16}$$

Substituting $\hat{x}_k = t_k\, \hat{a}_{k-1}$:

$$\hat{a}_k = \frac{t_k^2\, \sigma_a^2}{t_k^2\sigma_a^2 + (1-t_k)^2}\, \hat{a}_{k-1} + \frac{(1-t_k)^2}{t_k^2\sigma_a^2 + (1-t_k)^2}\, \mu_s. \tag{17}$$

Define $\lambda_k \coloneqq \frac{t_k^2\sigma_a^2}{t_k^2\sigma_a^2 + (1-t_k)^2}$. Then $\hat{a}_k = \lambda_k\, \hat{a}_{k-1} + (1-\lambda_k)\, \mu_s$.

Since $0 < \lambda_k < 1$ for all $t_k \in (0,1)$, this is a contraction toward μ_s. With $\hat{a}_0 = 0$, the error evolves as:

$$\hat{a}_k - \mu_s = \lambda_k\, (\hat{a}_{k-1} - \mu_s) = \cdots = \left(\prod_{j=1}^{k} \lambda_j\right) (\hat{a}_0 - \mu_s) = -\left(\prod_{j=1}^{k} \lambda_j\right) \mu_s. \tag{18}$$

Hence $\|\hat{a}_k - \mu_s\| = \prod_{j=1}^{k} \lambda_j \cdot \|\mu_s\| \to 0$ since each $\lambda_j < 1$. □

A.4 Implementation Details

In this section, we describe the full implementation details of VINE. We use a step count of $K{=}10$ across all tasks, with the fixed MIP grid $t \in \{0.0, 0.1, \ldots, 0.9\}$. Following the implementations in FQL, we train two Q functions to improve stability. We take the mean of the two Q values for the Q loss term in the actor objective. For the critic target, we use the minimum of the two Q values (clipped double Q-learning) on all reported OGBench domains. The actor is an MLP with hidden sizes $[512, 512, 512, 512]$ and GELU activations [66]. The critic is a twin ResNet value network with width 512 (depth 4 by default; depth 2 on `humanoidmaze` and `cube` domains), with layer normalization [67] to stabilize training. This critic backbone matches our FQL baseline for fair comparison. We train offline VINE with 1M gradient steps for state-based OGBench tasks, and evaluate the agent every 20k steps using 50 episodes. We report the average success rates following the official evaluation scheme [12],

Hyperparameters. We refer to Tables 3 and 4 for the complete list of hyperparameters.

Table 3: Hyperparameters for VINE.

Hyperparameter	**Value**
Learning rate	0.0003
Optimizer	Adam [68]
Gradient steps	1000000 (default)
Minibatch size	512
Actor MLP dimensions	$[512, 512, 512, 512]$
Critic architecture	ResNet (width 512; depth 4 / 2)
Nonlinearity	GELU [66]
Target network smoothing coefficient	0.005
Discount factor γ	0.995
Flow steps K	10
MIP time grid	$\{0.0, 0.1, \ldots, 0.9\}$
Clipped double Q-learning	True
Actor noise std (train)	1.0
BC coefficient α	Table 4

Table 4: VINE BC coefficient α for each task.

Task	**VINE** α
`antmaze-large`	10
`antmaze-giant`	10
`humanoidmaze-medium`	30
`humanoidmaze-large`	20
`scene-sparse`	300
`puzzle-3x3-sparse`	1000
`cube-double`	300
`cube-triple`	300

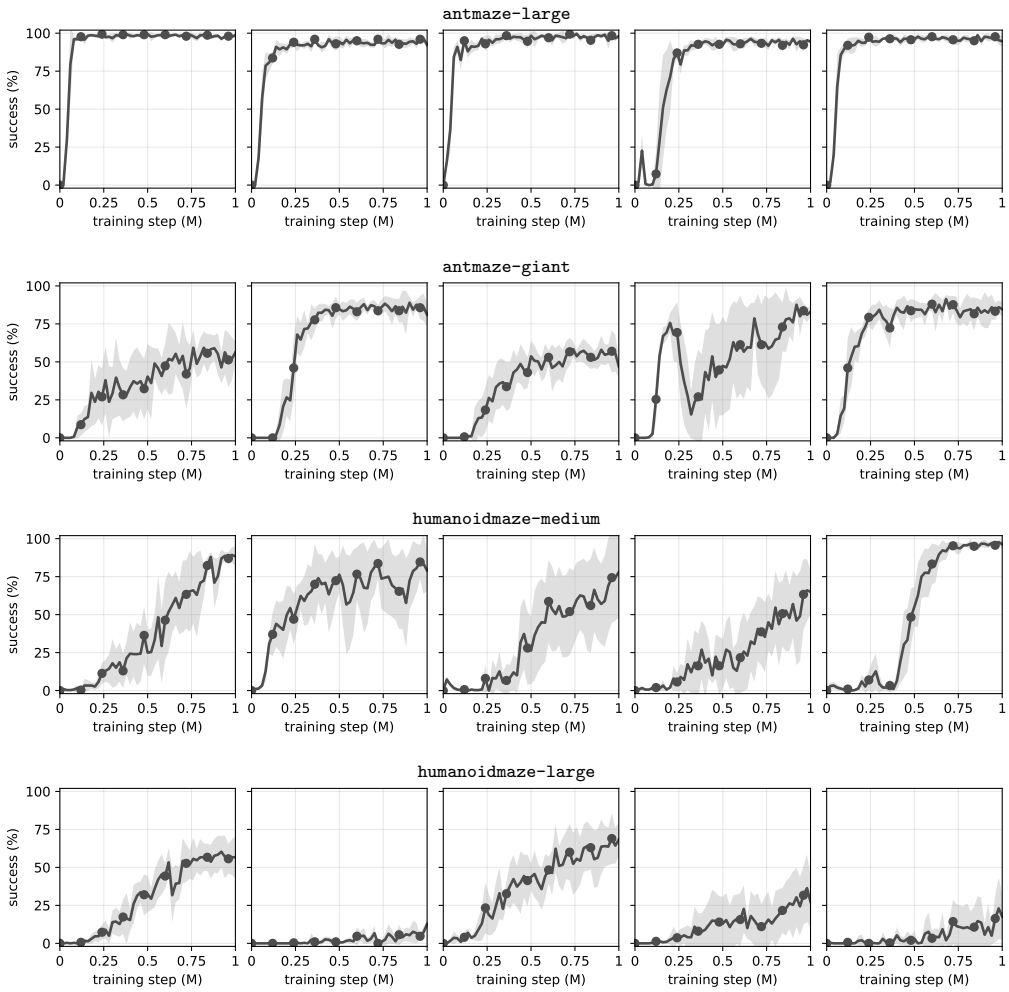

Figure 6: VINE training curves (success rate vs. training steps), maze navigation domains. Each row is one domain; the five columns are task1–task5. Curves show the mean over seeds with a 95% confidence band.

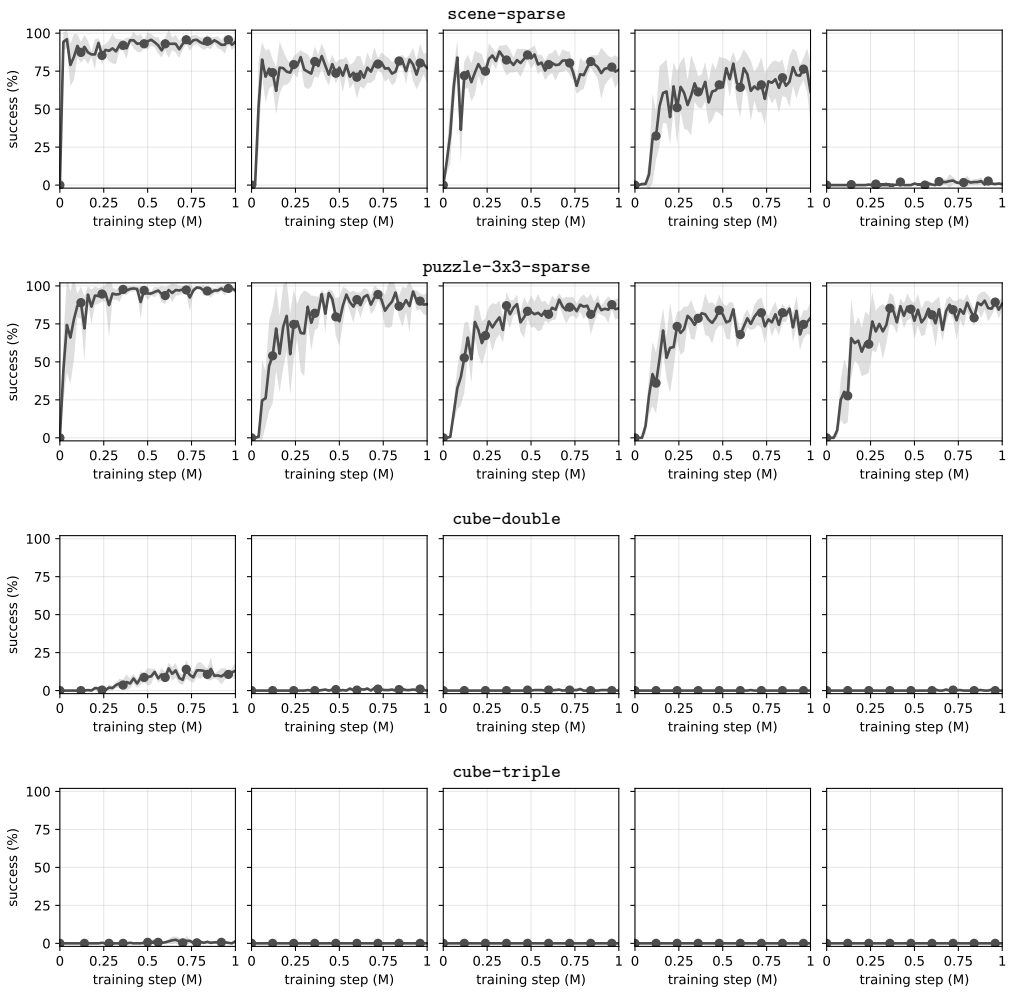

Figure 7: VINE training curves (success rate vs. training steps), manipulation domains. Each row is one domain; the five columns are task1–task5. Curves show the mean over seeds with a 95% confidence band.

			ReBRAC	FBRAC	BAM	FQL	FAWAC	CGQL	CGQL-M	CGQL-L	DAC	QSM	DSRL	FEdit	IFQL	QAM	VINE
antmaze-large			**98**	**0**	**90**	**93**	**6**	**63**	**56**	**39**	**88**	**89**	**62**	**67**	**41**	**77**	**100**
			88	**0**	**60**	**86**	**1**	**73**	**49**	**51**	**71**	**84**	**75**	**67**	**14**	**80**	**98**
			98	**12**	**94**	**59**	**40**	**91**	**90**	**79**	**98**	**98**	**81**	**65**	**54**	**89**	**100**
			94	**0**	**85**	**54**	**18**	**78**	**78**	**76**	**91**	**90**	**18**	**28**	**25**	**69**	**98**
			95	**0**	**88**	**86**	**22**	**75**	**80**	**78**	**92**	**93**	**69**	**63**	**45**	**89**	**99**
	agg.	(5 tasks)	**94**	**2**	**84**	**76**	**17**	**76**	**71**	**65**	**88**	**91**	**61**	**58**	**36**	**81**	**99**
antmaze-giant			**36**	**0**	**3**	**0**	**0**	**2**	**0**	**0**	**53**	**72**	**0**	**0**	**0**	**7**	**60**
			73	**0**	**0**	**0**	**0**	**0**	**0**	**4**	**0**	**0**	**0**	**0**	**0**	**0**	**87**
			13	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**51**
			74	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**12**	**10**	**2**	**34**	**86**
			89	**0**	**1**	**0**	**0**	**0**	**22**	**11**	**25**	**1**	**2**	**0**	**2**	**49**	**95**
	agg.	(5 tasks)	**57**	**0**	**1**	**0**	**0**	**0**	**4**	**3**	**16**	**15**	**3**	**2**	**1**	**18**	**76**
humanoidmaze-medium			**38**	**26**	**49**	**34**	**18**	**30**	**8**	**55**	**87**	**81**	**49**	**0**	**86**	**40**	**92**
			91	**78**	**69**	**95**	**44**	**78**	**99**	**93**	**96**	**96**	**91**	**39**	**92**	**97**	**99**
			83	**28**	**75**	**96**	**20**	**78**	**0**	**62**	**92**	**93**	**36**	**0**	**93**	**96**	**87**
			37	**3**	**22**	**14**	**1**	**23**	**0**	**2**	**43**	**47**	**0**	**0**	**60**	**3**	**56**
			96	**59**	**83**	**99**	**36**	**89**	**100**	**98**	**99**	**99**	**90**	**68**	**98**	**99**	**99**
	agg.	(5 tasks)	**69**	**39**	**60**	**68**	**24**	**60**	**42**	**62**	**83**	**83**	**53**	**22**	**86**	**67**	**87**
humanoidmaze-large			**36**	**0**	**5**	**8**	**0**	**2**	**0**	**2**	**0**	**10**	**14**	**8**	**36**	**6**	**71**
			1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**16**
			32	**0**	**6**	**19**	**1**	**11**	**4**	**18**	**0**	**24**	**1**	**3**	**55**	**16**	**77**
			10	**0**	**6**	**9**	**0**	**5**	**8**	**4**	**0**	**9**	**0**	**1**	**2**	**16**	**43**
			7	**0**	**11**	**9**	**0**	**5**	**16**	**9**	**1**	**7**	**2**	**1**	**29**	**19**	**25**
	agg.	(5 tasks)	**17**	**0**	**5**	**9**	**0**	**5**	**6**	**6**	**0**	**10**	**3**	**3**	**24**	**11**	**46**
scene-sparse			**98**	**66**	**100**	**99**	**62**	**79**	**100**	**100**	**100**	**100**	**100**	**95**	**93**	**100**	**98**
			90	**80**	**99**	**71**	**15**	**88**	**99**	**98**	**99**	**82**	**100**	**97**	**63**	**98**	**90**
			51	**41**	**99**	**97**	**14**	**23**	**92**	**91**	**79**	**97**	**97**	**61**	**71**	**100**	**86**
			60	**49**	**96**	**92**	**71**	**0**	**6**	**86**	**8**	**99**	**100**	**35**	**98**	**100**	**87**
			27	**17**	**96**	**33**	**27**	**0**	**74**	**66**	**53**	**50**	**100**	**24**	**95**	**87**	**5**
	agg.	(5 tasks)	**65**	**50**	**98**	**78**	**38**	**38**	**74**	**88**	**68**	**86**	**99**	**62**	**84**	**97**	**73**
puzzle-3x3-sparse			**99**	**1**	**26**	**99**	**8**	**87**	**100**	**98**	**98**	**94**	**83**	**100**	**100**	**98**	**100**
			77	**0**	**75**	**79**	**2**	**55**	**100**	**92**	**66**	**67**	**83**	**100**	**100**	**100**	**99**
			85	**0**	**78**	**83**	**1**	**24**	**100**	**87**	**54**	**5**	**83**	**98**	**100**	**100**	**93**
			62	**0**	**92**	**84**	**1**	**25**	**100**	**85**	**72**	**54**	**83**	**97**	**100**	**100**	**89**
			70	**1**	**9**	**6**	**1**	**47**	**100**	**88**	**51**	**46**	**100**	**100**	**100**	**100**	**96**
	agg.	(5 tasks)	**79**	**0**	**56**	**70**	**3**	**48**	**100**	**90**	**68**	**53**	**87**	**99**	**100**	**100**	**95**
cube-double			**29**	**0**	**84**	**81**	**8**	**55**	**50**	**62**	**36**	**67**	**90**	**77**	**16**	**85**	**16**
			6	**0**	**49**	**46**	**0**	**39**	**46**	**52**	**35**	**65**	**87**	**28**	**12**	**79**	**2**
			2	**0**	**38**	**42**	**0**	**44**	**50**	**52**	**31**	**57**	**85**	**44**	**10**	**54**	**1**
			1	**0**	**8**	**10**	**0**	**13**	**11**	**18**	**16**	**21**	**32**	**14**	**4**	**22**	**0**
			4	**0**	**56**	**50**	**0**	**42**	**48**	**41**	**56**	**69**	**76**	**39**	**11**	**82**	**0**
	agg.	(5 tasks)	**9**	**0**	**47**	**46**	**2**	**38**	**41**	**45**	**35**	**56**	**74**	**40**	**11**	**64**	**4**
cube-triple			**4**	**2**	**14**	**15**	**0**	**39**	**40**	**40**	**24**	**16**	**7**	**11**	**2**	**14**	**4**
			0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
			0	**0**	**3**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**2**	**0**
			0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
			0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	agg.	(5 tasks)	**1**	**0**	**3**	**3**	**0**	**8**	**8**	**8**	**5**	**3**	**1**	**2**	**0**	**3**	**1**
all	agg.	(40 tasks)	**49**	**12**	**44**	**44**	**10**	**34**	**43**	**46**	**45**	**50**	**48**	**36**	**43**	**55**	**60**

Table 5: **Offline results, 8 domains.**