Implicit Goal Conditioning via Value Disaggregation

Shashwat Saxena* **Mehul Goel*** **Sreyas Venkataraman***

Sarvesh Patil **Max Simchowitz**

Carnegie Mellon University
{ssaxena2,mehulgoel,sreyasv,sarveshp,msimchow}@andrew.cmu.edu
Code: https://github.com/simchowitzlabpublic/vadar-public

Abstract

Long-horizon tasks are important for modern AI and robotics, yet pose a challenge for reinforcement learning. A historically popular approach has been to decompose a task into subgoals, and learn a subgoal-conditioned policy to execute each subtask in sequence. We introduce an alternative called *Value Disaggregation* (VADAR), which trains a single subgoal-independent policy to maximize a linear combination of per-subgoal critic values. Because subgoal information is needed only at *training-time*, VADAR removes the need for possibly costly or cumbersome subgoal-selection during policy deployment. Moreover, through careful experiments, we show that VADAR (1) is always competitive with, and frequently *outperforms*, natural baselines like goal-conditioning and (2) succeeds on tasks with non-sequential goal structure and multiple optimization objectives. In particular, VADAR is the **first** algorithm to solve challenging, long-horizon tasks in the ROBOCASA365 suite, even when starting from a policy pretrained from behavior cloning with *near-zero* initial task success. Taken together, VADAR's successes suggest that subgoal information benefits actor-critic reinforcement learning by enhancing the efficacy of *critic learning*, whereas policy goal-conditioning is often unnecessary.

1 Introduction

Learning to complete long-horizon, multi-step tasks is a fundamental challenge across a number of core AI domains, from robot [Gupta et al., 2019], to language modeling [Wei et al., 2022], to chemical reaction design [Liu et al., 2023b]. A popular approach, often referred to as *hierarchical goal conditioning* (HGC) in reinforcement learning literature, decomposes these complex, long-horizon tasks into a set of subgoals, each of which can be independently executed by a dedicated learned policy [Gupta et al., 2019, Nasiriany et al., 2019, Levy et al., 2019, Nachum et al., 2018]

However, attempting to learn per-sub-task policies introduces a new set of challenges. Myopic completion of a given sub-task might compromise performance on future sub-tasks. Moreover, explicit decompositions often require sub-tasks to be sequential in nature, and necessitate a dedicated sub-task planner to decide which is the current subtask to be attempted during policy deployment.

Contributions. In this paper, we propose an alternative to HGC that we call **value-disaggregation** (VADAR), a framework for sample-efficient reinforcement learning of long-horizon multi-step tasks. We focus on robotic manipulation and continuous control tasks, and assume that our task can be decomposed into sub-tasks $g^1, \ldots, g^{|G|}$ whose successful completion can be reliably evaluated, as

*Equal contribution.

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

is rapidly becoming feasible with today's VLMs [Du et al., 2023]. Whereas HGC learns policies $\pi(s, g)$ that take sub-goals g as explicit input, VADAR instead learns a sub-goal-unconditioned policy $\pi(s)$, alongside a vector-valued critic that predicts value estimates for all sub-tasks simultaneously. The policy is then trained to maximize a linear combination of critic values, thereby optimizing for an aggregate performance on current-sub-goals with a focus on future tasks. Figure 1 is a visual representation of VADAR.

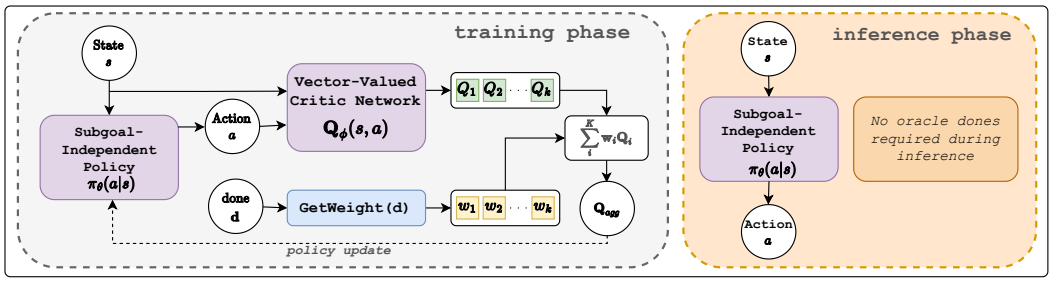


Figure 1: VADAR algorithm. We learn per-goal critics with a shared backbone, and optimize a weighted sum of these to update a subgoal independent policy, which does not require subgoals to be supplied at inference.

In Section 5.1, we evaluate VADAR across a large variety of simulated benchmarks for continuous control and robotics tasks, including the ROBOMIMIC [Mandlekar et al., 2021] and ROBOCASA365 [Nasiriany et al., 2026] testing suites, with subtask decompositions supplied. VADAR outperforms other competitive long-horizon baselines, including hierarchical goal conditioning and dense preference rewards learned from treating trajectory success as a "preference" [Brown et al., 2019]. As a testament to its efficacy, VADAR enables the first (non-VLA) BC-to-online RL method **to successfully achieve success on challenging long-horizon RoboCasa benchmarks** from a fixed state.

Moreover, VADAR removes the need for subgoal-evaluation at inference *by design*. Thus, in systems where compute-intensive VLM calls are used to delineate subgoals, these calls need only be made at *training time*, but not during policy deployments/inference/data-collection. In addition, in Section 5.2, we identify that VADAR enables 3 further benefits over traditional goal conditioning formulations. Beyond raw performance, we establish 3 key benefits of VADAR over HGC.

Benefit 1: Multi-Task Representation Learning via Vector-Valued Critics. By learning vector-valued critic networks $\mathbf{Q}_\phi(s, a) \mapsto \mathbb{R}^{|G|}$, the forward pass of the network learns to compute shared features relevant for predicting multiple subgoal Q-values simultaneously. In Section 5.2, we show that this decision alone drives better sample efficiency.

Benefit 2: Non-Myopicness via Subtask Weighting. HGC often focuses on maximizing reward/success for a current subgoal, often at the expense of subsequent subgoals (Section 5.2). In VADAR, we instead optimize a weighted combination of critic values associated with *all future subgoals*. We introduce a particular weighting we call "*Next-Future*" weighting (Table 1), which ignores the immediate current subgoal's critic values whilst maximizing the average of critics associated with all future subgoals. In Section 5.2, we show that this weighting avoids converging to myopic policies that exploit current subgoal performance at the expense of future task success.

Benefit 3: Flexibility over task-structure and auxiliary rewards/constraints. Finally, we demonstrate the flexibility of the VADAR framework compared to HGC. As noted above, by modifying the weighting of the contributions of different per-goal critics, VADAR can titrate between aggressive short-term learning and long-term optimality. VADAR can be used to incorporate additional rewards and constraints, such as obstacle avoidance, with minimal degradation of policy performance. By aggregating multiple-critics, VADAR naturally accommodates settings where subgoals have directed acyclic graph (DAG) structure in which many goals are "active" at a current time step (Appendix E).

2 Related Work

Long-horizon reinforcement learning via temporal abstraction. Long-horizon reinforcement learning is commonly addressed by introducing temporal abstraction: instead of reasoning over primitive actions, the agent reasons over extended behaviors, skills, or subgoals. Classical formula-

tions include the options framework, where temporally extended actions are specified by initiation sets, intra-option policies, and termination conditions [Sutton et al., 1999, Bacon et al., 2016]; hierarchical abstract machines [Parr and Russell, 1997]; feudal reinforcement learning [Dayan and Hinton, 1992, Vezhnevets et al., 2017]; and MAXQ value decomposition [Dietterich, 1999]. Subsequent work studies how such skills and hierarchies can be discovered from data [Fox et al., 2017, Evans and Özgür Şimşek, 2024, Cannon and Özgür Simsek, 2025, Peng et al., 2022]. While these methods differ in how subtasks are specified or learned, they use subgoal structure to organize policy execution: the learned policy must either act through a prescribed hierarchy of subtasks[Dietterich, 1999, Parr and Russell, 1997] or utilize temporally extended controllers [Bacon et al., 2016, Fox et al., 2017]. In contrast, VADAR assumes that subgoals are part of the MDP design, but does not learn or execute a hierarchy over them; subgoals are used only to construct training-time value targets for a single flat policy.

Goal-conditioned reinforcement learning. Goal conditioned RL provide a natural design for conditioning on desired outcomes/sub-goals. Instead of learning a separate controller for each task, goal-conditioned methods learn policies or value functions indexed by a goal, enabling transfer across related goal-reaching problems. Classical dynamic-goal learning first studied transfer across changing goal-reaching tasks [Kaelbling, 1993]. This idea was later extended through universal value function approximators, which learn goal-indexed value functions with function approximation [Schaul et al., 2015], and hindsight relabeling, which reuses failed rollouts as successful examples for alternative achieved goals [Andrychowicz et al., 2018]. However, conditioning a flat policy on the final task goal does not by itself resolve long-horizon credit assignment. For this reason, hierarchical goal-conditioned RL (HGC) uses goal conditioning as a local control primitive: a high-level policy, planner, or task specification provides intermediate subgoals, and a low-level goal-conditioned policy learns to reach them [Nachum et al., 2018, Levy et al., 2019, Gupta et al., 2020, Nair et al., 2018]. This reduces the effective horizon, but introduce additional dependencies: they often assume a sequential subgoal structure, treat only one subgoal as active at a time, and require the deployed policy to be supplied with the correct current goal. These issues are costly and difficult when subgoal progress and completion must be evaluated by expert rules or large vision-language models during inference. VADAR avoids this test-time dependency by learning a subgoal-agnostic policy; goal information enters only through the critic during training.

Value decomposition and reward-factorized critics. MAXQ decomposes the value of a hierarchical policy into child-task values and completion functions, reflecting the procedural structure of a subroutine hierarchy [Dietterich, 1999]. Successor representations and successor features instead factor value prediction into predictive components and reward-dependent weights [Dayan, 1993, Barreto et al., 2017]. VADAR follows this latter spirit of reward-factorized value estimation, but uses subgoal-completion signals to learn a vector-valued critic in an offline-to-online actor-critic pipeline. Closer to robotics, task-and-motion planning (TAMP) couples high-level symbolic task planners with low-level continuous motion planners over a fixed subtask vocabulary [Kaelbling and Lozano-Pérez, 2011, Garrett et al., 2021]. VADAR shares the assumption that the task admits a subgoal-level decomposition, but does not plan over symbolic actions or commit to a fixed sequencing at execution time: the decomposition enters only through the critic, and the deployed policy is a single flat, subgoal-agnostic controller.

3 Preliminaries

MDPs. We formulate our problem as a *Markov Decision Process* (MDP) $M := (S, A, P_0, P, R, \gamma)$, with states $s \in S$, actions $a \in A$, initial state distribution P_0, transition probabilities P, reward R, and discount factor $\gamma \in (0, 1)$. At each timestep t, the agent (e.g., robot) observes the state $s_t \in S$, takes an action $a_t \sim \pi(a_t \mid s_t) \in A$, transitions to the next state s_{t+1} according to $s_{t+1} \sim P(s_{t+1} \mid s_t, a_t)$ while receiving a reward $R(s_t, a_t)$. For the MDP M_{ENV}, we let $\mathbb{E}^{\pi,M}$ (resp. $\mathbb{P}^{\pi,M}$) denote the expectation (resp. probability distribution) over trajectories $(s_0, a_0, \ldots, s_T, a_T)$ with length $T + 1$, with initial state distribution $s_0 \sim P_0$ and transition operator P. The object of reinforcement learning (RL) is to train a policy to optimize the cumulative discounted return: $J^M(\pi_\theta) = \mathbb{E}^{\pi_\theta}[\sum_{t\geq 0} \gamma^t R(s_t, a_t)]$. We also recall the Q-function $Q^{\pi,M}(s, a) := \mathbb{E}^{\pi}[\sum_{t\geq 0} \gamma^t R(s_t, a_t) \mid (s_t, a_t) = (s, a)]$ and value function $V^{\pi,M}(s) := \mathbb{E}_{a\sim\pi(s)}[Q^{\pi}(s, a)]$. When M is clear from context, we can omit it from the subscript.

Value-based Reinforcement Learning. Value-based RL methods [Sutton and Barto, 2018] simultaneously learn a policy π_θ and associated Q_ϕ-function, maintaining [2] (1) a replay buffer $\mathcal{B} = \{(s, a, s', r, \texttt{done})\}$ of past states s, actions a, successor states $s' \sim P(s, a)$, rewards r, and done indicators $\texttt{done}$; (2) a policy π_θ, critic network $Q_\phi : S \times A \to \mathbb{R}$ trained from this buffer. The parameters of Q_ϕ are updated to approximate Q^{π_θ}, and $\pi_\theta(s)$ is updated to maximize critic values. Specifically, updates are sampled $(s, a, s', r) \sim \mathcal{B}$, and new actions $a' \sim \pi_\theta(s')$. Then,

1. *Critic parameters* ϕ are updated via gradient descent on a given **critic loss**

$$L_{\text{critic}}(Q_\phi \mid s, a, y^{\text{target}}) = (Q_\phi(s, a) - y^{\text{target}})^2 + L_{\text{reg}}(s', a', \pi_\theta) \tag{3.1}$$

   which measures the squared Bellman residual to the future target $y^{\text{target}} = r + \gamma Q_\phi(s', a')$, and which may also incorporate some additional regularization $L_{\text{reg}}(\cdot)$ depending on s', a', π_θ (omitted as arguments in the critic loss for simplicity).

2. *Policy parameters* θ are updated via gradient descent on a given **policy loss** $L_{\text{policy}}(\pi_\theta \mid s, Q)$ mapping from policies $\pi_\theta : S \to \Delta(A)$, states $s \in S$, and Q to $\mathbb{R}$.

Representative Algorithms: `SAC`, `FQL` and `OGPO`. We consider three representative value-based methods: Soft Actor-Critic (**`SAC`**) [Haarnoja et al., 2018b] , Flow Q-Learning (**`FQL`**) [Park et al., 2025b] and off-policy generative policy optimization (**`OGPO`**) [Patil et al., 2026] . The **`SAC`** algorithm uses a maximum-entropy policy loss $L_{\text{policy}}(\pi_\theta \mid s, Q_\phi]) = \mathbb{E}_{a \sim \pi_\theta(s)}[\alpha \log \pi_\theta(a \mid s) - Q_\phi(s, a)]$, where $\alpha > 0$ is a temperature parameter, and incorporates an entropy regularizer in the target: $L_{\text{reg}}(s', a', \pi_\theta) = -\alpha \log \pi_\theta(a' \mid s')$. **`FQL`** and **`OGPO`** are designed for generative control policies (GCPs), where the policy itself is parametrized via a multi-step generative model such as a Diffusion or flow-based generative model. This is a more careful choice of the policy loss L_{policy}, and the critic losses have no regularizer: $L_{\text{reg}} \equiv 0$. See Appendix C for details.

Sparse-Reward RL. In many tasks, such as those encountered in robot manipulation, defining reward that densely measure progress towards goal completion is challenging [Amodei et al., 2016, Krakovna et al., 2020]. Thus, we consider a *sparse-reward* defined via the *done* indicator $\texttt{done} \in \{0, 1\}$ indicating task completion. Standard practice is to set $r \leftarrow r_{\text{sparse}}(\texttt{done})$, where

$$r_{\text{sparse}}(\texttt{done}) = -1 \cdot \mathbf{I}\{\texttt{done} = 0\} + r_{\text{done}} \mathbf{I}\{\texttt{done} = 1\}, \tag{3.2}$$

and where r_{done} is a small positive constant (e.g. $r_{\text{done}} = 5$). In words, r_{sparse} penalizes each time-step until the task is not completed, and provides a small positive bonus for task success.

Offline-to-Online RL. In many robotics settings, environment interaction is expensive and a fixed dataset of demonstrations $\mathcal{D}_{\text{off}}$ is available. We follow the now-standard *offline-to-online* recipe [Nair et al., 2021]: pretrain a behavior-cloning policy π_{BC} on $\mathcal{D}_{\text{off}}$, then initialize the online policy from π_{BC} and fine-tune online, optionally mixing $\mathcal{D}_{\text{off}}$ into the off-policy updates at a fixed ratio [Ball et al., 2023]. All algorithms in our experiments share the same BC-pretraining and warm-start protocol, so comparisons isolate the online fine-tuning method.

3.1 Long-Horizon Reinforcement Learning via Hierarchical Goal-Conditioning

Long-horizon reinforcement learning tasks require many time steps to complete. Sparse-rewards make *credit assignment* - identifying which actions contribute to task success/failure - challenging. Thus, prior work introduces **hierarchical goal conditioning** (HGC)[3], which decomposes the problem into a sequence of *subgoals* $G = \{g\}_{g \in G}$. Traditional goal-conditioned RL assumes that the completion of *individual subgoals*, what we call **subgoal-dones** $\texttt{done}_g(s) := \mathbf{I}\{g \text{ is completed at state } s\}$, can be evaluated at both training time time and during evaluation. Traditionally, $\texttt{done}_g$ have been hand-specified, but modern VLMs can now evaluate $\texttt{done}_g$'s specified by natural language queries [Chen et al., 2026]. For simplicity, in the body of this paper we assume these goals satisfy a linear order with $g^1 \prec g^2 \cdots \prec g^{|G|} \in G$, where $g^{|G|}$ denotes final task completion and g^1 is the first goal (Appendix E) easily removes this sequentiality). Under this sequentiality condition, we define the current goal $g^{\text{curr}} = \min\{g : \texttt{done}_g = 0\}$, where $\min$ is take with respect to the order $\prec$.

[2] We defer critic ensembling and parameter exponential moving averages to Appendix B.3.

[3] This same practices goes by other names in other communities; see Section 2 for further discussion.

Value-Based RL with Hierarchical Goal Conditioning. Algorithm 1 depicts how value-based RL methods can be modified to incorporate HGC. These modifications of standard value-based RL, depicted in blue, are: policies and critics take the current goal $g_t = g^{\text{curr}}$ as an argument (Line 2), so that in particular, the policy needs g_t to select actions (Line 5). This necessitates that g_t is tracked in the replay buffer (Line 6), is used to determine the critic target (Line 9), and features in both the policy and critic loss (Lines 10, 11).

Algorithm 1 Hiearchical Goal Conditioning RL (HGC-RL).

1: **Parameters:** discount factor γ; learning rate η
2: **Initialize** critic networks $Q_\phi : S \times A \times G \to \mathbb{R}$, policy π_θ: $S \times G \to \Delta(A)$, buffer $\mathcal{B} \leftarrow \emptyset$
3: **for** episode = 1, 2, ... **do**
4: **while** episode not done **do**
5: $a_t \leftarrow \pi_\theta(s_t, g_t)$, transition $s_{t+1} \sim P(s_t, a_t)$ ▷ $g_t = g^{\text{curr}} := \min\{g : \texttt{done}_g = 0\}$)
6: $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s_t, a_t, s_{t+1}, r_t, g_t)\}$, $r_t \leftarrow r_{\text{sparse}}(\texttt{done}_{g_t})$
7: **for** number of updates **do**
8: Sample $(s, a, s', r_g, g) \sim \mathcal{B}$, $a' \sim \pi_\theta(s')$
9: $y_g^{\text{target}} = r_g + \gamma Q_\phi(s', a', g)$ ▷ Target for current goal
10: $\phi \leftarrow \phi - \eta \nabla_\phi L_{\text{critic}}(Q_\phi(\cdot, \cdot, g) \mid s, a, y_g^{\text{target}})$.
11: $\theta \leftarrow \theta - \eta \nabla_\theta L_{\text{policy}}(\pi_\theta(\cdot, g) \mid s, a, Q_\phi(\cdot, \cdot; g))$

The Limitations of Hierarchical Goal-Conditioning Though HGC reduces effective problem horizon, it suffers from a number of limitations: (1) it often requires goals to have a sequential structure, with only one active goal at a time (2) optimizing for only a current goal can be overly myopic if future actions depend subtly on how a current goal is completed, such as in robotic pick and place tasks. (3) accurate goals must be supplied to the policy at test-time, which is prohibitive if sub-goal completions requires expert engineering or larger models to evaluate. These limitations are explored and verified in Section 5.2.

An Alternative: Dense Preference-Based Reward. An alternative approach to mitigating the credit assignment problem in sparse reward settings is to incorporate a dense, preference based reward [Ouyang et al., 2022] . We defer further details to Appendix D. **Limitations:** dense rewards are inherently imperfect proxies for desired behavior, and optimizing them often leads to reward hacking or specification gaming, where agents exploit loopholes in the reward definition rather than accomplishing the intended task [Amodei et al., 2016, Krakovna et al., 2020].

4 The Value Disaggregation (VADAR) Framework

We now introduce Value Disaggregation (VADAR, Algorithm 2), a framework for subgoal-driven long-horizon RL that addresses the limitations of goal conditioning described in Section 3.1. Our **key insight** is that using multi-goal critics enables extraction from a subgoal independent policy. This facilitates greater flexibility when balancing short- and long-term policy performance.

Similar to goal-conditioning, VADAR learns to predict *per-subgoal critic values* (Line 2). However, in VADAR, these are predicted as a joint vector, which is then aggregated using subgoal completion indicators to produce a scalar value for updating a *subgoal-independent policy*. Because this aggregation (Line 12) occurs only at training time, the deployed policy requires no subgoal information at inference, eliminating the test-time overhead of goal-conditioned RL. Whereas HGC can be overly myopic [Robert et al., 2023], strategically weighting per-subgoal critic values in the aggregation step balances current and future goal completion. Finally, VADAR requires minimal modification to standard off-policy RL pipelines, wherein only the critic architecture (scalar $\to$ vector output) and the corresponding aggregation strategies change. We now walk through Algorithm 2 in detail, highlighting the components unique to VADAR (shown in color in the pseudocode and below). We give a detailed **practitioners guide** to using VADAR in Appendix A.

Algorithm 2 Value Disaggregation (VADAR)

1: **Parameters:** discount factor γ; learning rate η; TD clipping parameter c_{clip}.
2: **Initialize** multi-goal critic $\mathbf{Q}_\phi : S \times A \to \mathbb{R}^G$; policy π_θ: $S \to \Delta(A)$; replay buffer $\mathcal{B} \leftarrow \emptyset$.
3: **for** episode = 1, 2, ... **do**
4: **while** episode not done **do**
5: $a_t \leftarrow \pi_{\bar{\theta}}(s_t)$, transition $s_{t+1} \sim P(s_t, a_t)$
6: Evaluate subtask completion vector: $\mathbf{d}_t = \{\texttt{done}_g(s_{t+1})\}_{g \in G} \in \{0,1\}^G$
7: $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s_t, a_t, s_{t+1}, \mathbf{r}_t, \mathbf{d}_t\}$, where $\mathbf{r}_t \leftarrow \sum_g r_{\text{sparse}}(\mathbf{d}_t[g])$,
8: **for** number of updates **do**
9: Sample $(s, a, s', \mathbf{r}, \mathbf{d}) \sim \mathcal{B}$, $a' \sim \pi_\theta(s')$
10: $\mathbf{y}^{\text{target}} = \min\{\mathbf{r} + \gamma \mathbf{Q}_\phi(s', a'),\ c_{\text{clip}} \cdot r_{\text{done}}\}$ ▷ multi-goal targets, entrywise min
11: $\phi \leftarrow \phi - \eta \nabla_\phi \sum_{g \in G} L_{\text{critic}}(\mathbf{Q}_\phi[g] \mid s, a, y_g^{\text{target}})$. ▷ multi-goal critic loss
12: $Q_{\text{agg}}(\cdot, \cdot) \leftarrow \langle \mathbf{w}, \mathbf{Q}_\phi(\cdot, \cdot) \rangle$, where $\mathbf{w} \in \mathbb{R}^G \leftarrow$ GETWEIGHTING($\mathbf{d}$)
13: $\theta \leftarrow \theta - \eta \nabla_\theta L_{\text{policy}}(\pi_\theta \mid s, a, Q_{\text{agg}})$

1. Vector-valued multi-goal critic. Where standard RL maintains a scalar critic $Q_\phi : S \times A \to \mathbb{R}$, VADAR maintains a multi-goal critic $\mathbf{Q}_\phi : S \times A \to \mathbb{R}^G$ that maps each state-action pair to a $|G|$-dimensional vector of Q-values. The g-th entry $\mathbf{Q}_\phi[g](s, a)$ estimates the value of the state-action pair under the MDP induced by the per-subgoal reward r_g. Crucially, all $|G|$ entries share a single network backbone, enabling representation sharing across subgoals: features learned for predicting one subgoal's completion are available to all others, improving learning efficiency (Section 5.2). Appendix L shows the vector-valued critic can be interpreted simply as a structured instantiation of successor features [Barreto et al., 2017], where each feature corresponds to subgoal performance.

2. Per-subgoal rewards and the critic update At each environment step, VADAR evaluates a vector of subgoal completion indicators $\mathbf{d}_t = \{\texttt{done}_g(s_{t+1})\}_{g \in G} \in \{0,1\}^G$ and computes a corresponding reward vector $\mathbf{r}_t = r_{\text{sparse}}(\mathbf{d}_t)$ entrywise via the sparse reward function (Eq. (3.2), Line 6). This converts a sparse task reward into $|G|$ per-subgoal rewards (Line 10), each of which provides signal as soon as its corresponding subgoal is completed. The tuple $(s_t, a_t, s_{t+1}, \mathbf{r}_t, \mathbf{d}_t)$ is stored in the replay buffer. Given a sampled transition $(s, a, s', \mathbf{r}, \mathbf{d})$ from the buffer and a new action $a' \sim \pi_\theta(s')$, the vector of TD targets is computed entrywise:

$$\mathbf{y}^{\text{target}} = \min\{\mathbf{r} + \gamma\, \mathbf{Q}_\phi(s', a'),\ c_{\text{clip}} \cdot r_{\text{done}}\}, \tag{4.1}$$

where the min is applied entrywise. The critic is updated on the cumulative loss across all subgoals: $\sum_{g \in G} L_{\text{critic}}(\mathbf{Q}_\phi[g] \mid s, a, y_g^{\text{target}})$. After a subgoal is completed, the agent keeps collecting r_{done} at every step, so without the clip the Q-value would accumulate to $r_{\text{done}}/(1-\gamma)$. The clip caps the target at $c_{\text{clip}} \cdot r_{\text{done}}$ (Line 10), capping critic values under completed subgoals.

3. Policy update via value weighting. VADAR gives us the flexibility to weight the contributions of different critics during policy extraction via GETWEIGHTING (Line 12):

$$Q_{\text{agg}}(s, a) = \langle \mathbf{w},\ \mathbf{Q}_\phi(s, a) \rangle = \textstyle\sum_{g \in G} w_g \cdot \mathbf{Q}_\phi[g](s, a), \qquad \mathbf{w} \leftarrow \text{GETWEIGHTING}(\mathbf{d}). \tag{4.2}$$

The policy is then updated to maximize Q_{agg} via the policy loss $L_{\text{policy}}(\pi_\theta \mid s, Q_{\text{agg}})$. Because the policy $\pi_\theta : S \to \Delta(A)$ is never conditioned on g (*subgoal agnostic*), subgoal information is not needed at test time. A good weighting function provides strong training signal for completing current goals while also improving behavior for future goals.

Table 1: Value-weighting variants used for policy extraction in VADAR.

Config.	Weight w_g	Interpretation
Current-myopic	$\mathbb{I}[g = g^{\text{curr}}]$	Only the current subtask contributes.
Current-future	$\mathbb{I}[g \succeq g^{\text{curr}}]$	All incomplete subtasks contribute.
Next-myopic	$\mathbb{I}[g = g^{\text{next}}]$	Focus on the next subtask, gated by the current subtask.
Next-future	$\mathbb{I}[g \succeq g^{\text{next}}]$	Incomplete subtasks with incomplete parents contribute.

Weighting choices for sequential goals. As in Section 3, first assume that goals are ordered as $1, \ldots, |G|$, with goal g^i serving as a prerequisite for goal g^{i+1}, denoted $g^i \prec g^{i+1}$. (see Appendix E for the general case. Given a vector of completion indicators $\mathbf{d}$, recall that $g^{\text{curr}} = \inf\{g : \mathbf{d}_g = 0\}$

denotes the current unfinished goal. We also define the next goal as $g^{\text{next}} := \min\{g^+, g^{|G|}\}$, where $g^+ := \inf\{g' : g' \succ g^{\text{curr}}\}$; again, min/inf is taken with respect to the goal order $\prec$, and $\succeq$ in Table 1 denotes the negation of $\prec$.

Table 1 summarizes four choices for the weighting function along two axes. The first axis distinguishes *myopic* weightings, which optimize only a single goal, from *future-aware* weightings, which include all relevant future goals. The second axis distinguishes *current* weightings, which include the current goal, from *next* weightings, which only optimize for goals after the current goal. These choices are illustrated in Figure 2. In our experiments, future-aware weightings generally outperform myopic ones by propagating training signal from later subtasks. Moreover, next-future weighting can outperform current-future weighting in settings where the policy otherwise over-optimizes current goal completion at the expense of later subtasks.

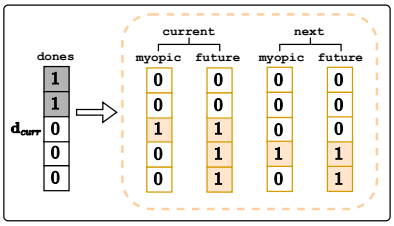


Figure 2: Subgoal weighting used by GETWEIGHTING.

Generalization to non-sequential tasks. The use of a weighting function enables a direct extension to subgoals with a DAG structure, where certain goals can be completed in parallel, and where multiple goals must be completed before unblocking a subsequent one; Figure 8 depicts such a structure for the Transport task in the ROBOMIMIC Benchmark Suite [Mandlekar et al., 2021]. In Appendix E, we describe a formalism for extending the weighting in Table 1 to the DAG setting.

Generalization to multi-objective losses. VADAR does not require that the disaggregated value functions correspond to rewards associated with subgoal long-horizon task. Any collection of independently valuable rewards, such as safety constraints or velocity penalties, can be included in the fitting of the multi-objective critics. We validate the utility of VADAR for such multi-objective problems in Section 5.3.

Iterative Critics. In principle, VADAR is compatible with critics that yield their value estimate via iterative computation [Agrawalla et al., 2025]. Appendix N explores this possibility in more detail.

VADAR: Key Design Decisions

- **Vector-valued critic with shared backbone.** A single network $\mathbf{Q}_\phi : S \times A \to \mathbb{R}^{|G|}$ produces one Q-value per subgoal, sharing features across subgoals.
- **Subgoal-agnostic policy.** Subgoal information enters only through the critic at training time; deployed policy $\pi_\theta(a \mid s)$ needs no goal selection/conditioning at inference.
- **Weighted aggregation for policy extraction.** Per-subgoal Q-values are aggregated into a single scalar signal Q_{agg}; the next-future weighting is our default and balances current and future subgoal completion.
- **Reward clipping at $c_{\text{clip}} \cdot r_{\text{done}}$.** The per-subgoal TD targets are capped at $c_{\text{clip}} \cdot r_{\text{done}}$, which prevents post-completion accumulation to $r_{\text{done}}/(1-\gamma)$. This *removes the need for gating of the critic loss* by completed subgoals.

5 VADAR Outperforms HGC and Dense-Reward Baselines

We evaluate VADAR on a suite of long-horizon manipulation benchmarks. Each method is pretrained via behavior cloning on offline data, stopped early at a low success rate so all methods begin fine-tuning from a comparable starting point, and then fine-tuned online for a fixed number of environment steps. Per-task descriptions and hyperparameters are detailed in Appendix F.

Baselines. We study VADAR with two different weightings (Table 1) - VADAR-NF, using *next-future* (plotted in a solid line) and VADAR-CF, which use *current-future* (plotted in a dotted line). We compare against 3 alternatives, each using the same base-algorithm per plot: (1) **Hierarchical Goal Conditioning** (HGC), where both the policy and critic are conditioned on the current subgoal g, where g is set to the next incomplete subgoal at every step (Algorithm 1). (2) **Dense Preference Reward**, denoted Dense, where we adopt best-practices for a learning dense reward model via preferences (described in Appendix D), where the reward models are pretrained with with additional

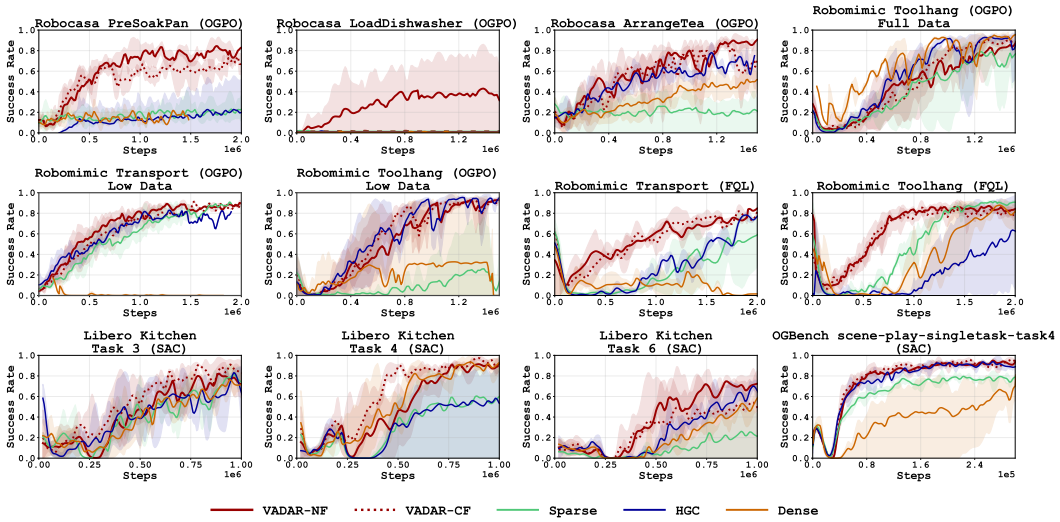


Figure 3: Success Rate plots of our comparisons of VADAR with the HGC and natural sparse reward policy baselines across ROBOCASA365, ROBOMIMIC, LIBERO and OGBENCH for our three instantiations: **SAC**, **FQL** and **OGPO**.

rollouts on a richer dataset of successful and failed runs in order to steelman their performance. (3) The original base-algorithm that using sparse reward with no goal information, denoted as "Sparse".

Environments. We evaluate on four benchmark suites covering varying task horizons, difficulties. Choices of subgoals and their dependency structure are detailed in Appendix F: **(i) ROBOCASA365** [Nasiriany et al., 2026] contains our longest-horizon, most challenging tasks: PRESOAKPAN (place a pan in the sink, place a sponge over the pan, then turn on the faucet), LOADDISHWASHER (open the dishwasher, load two dishes, then close it) and ARRANGETEA (pick the kettle from the counter and place it on the tray, then pick the mug from the cabinet and place it on the tray and then close the cabinet doors.) Next, **(ii) ROBOMIMIC** [Mandlekar et al., 2021] contains two contact-rich, medium-horizon tasks: TRANSPORT, a bimanual task in which two arms coordinate to lift a lid, transfer a hammer between containers, and dispose of trash; and TOOL-HANG, a single-arm precision task that requires inserting a hook into a base frame and hanging a wrench on it. Next, **(iii) LIBERO** [Liu et al., 2023a] contributes three multi-stage kitchen tasks from the `LIBERO-10` suite. We use `KITCHEN_SCENE3` (turn on stove and place moka pot), `KITCHEN_SCENE4` (place bowl in drawer and close it), and `KITCHEN_SCENE6` (place mug in microwave and close it). Finally, **(iv) OGBENCH** [Park et al., 2025a] contributes `scene-play-singletask-task4`, in which the agent must unlock a drawer, open it, pick up a cube, and place it inside; our suite of experiments verify that the benefits of VADAR persist in long-horizon variants of classical RL domains.

Base-algorithm instantiations. We instantiate all above baselines with three value-based RL algorithms. Full details are given in Appendix C. On the most challenging long-horizon environments (ROBOCASA365), we use **OGPO** [Patil et al., 2026], a state-of-the-art offline-to-online RL algorithm for flow-based policies. On the medium-horizon ROBOMIMIC tasks, we evaluate on both **OGPO** and the popular, though less performant **FQL** [Park et al., 2025b], demonstrating that the benefits of VADAR apply to both. Because **OGPO** is considerably more sample efficient that than **FQL** from sparse reward when all BC data are used, we also test **OGPO** in a regime with limited BC data, denoted **low data**, on ROBOMIMIC; details are described in Table 10. Lastly, we demonstrate VADAR's applicability to non-generative policies with the seminal Soft Actor Critic (**SAC**) algorithm [Haarnoja et al., 2018b] on OGBENCH and LIBERO.

5.1 VADAR outperforms natural baselines

We depict the performance of VADAR and all baseline methods in Figure 3. VADAR consistently performs among the best across all tasks, and strictly dominates baselines on the most challenging tasks. On 2-of-3 evaluated ROBOCASA365 tasks VADAR is the only method to achieve non trivial success, and strictly dominates the baselines in all three (Figure 3, top row). On ROBOMIMIC with **OGPO**, we find that VADAR and HGC obtain similar performance, both with low- and full-data;

whereas sparse fails in the low-data regime on TOOL-HANG, and dense struggles throughout. However, using the weaker **FQL** algorithm algorithm, VADAR drastically outperforms baselines.While using **SAC** on LIBERO, VADAR consistently matches or outperforms the baselines, with the performane gap widening with the increasing task difficulty / horizon length. We note that even in scenarios where VADAR and HGC achieve comparale performace, VADAR *may still be preferable due to removing the need for goal-selection at inference time.* Finally, on OGBENCH, VADAR and HGC achieve comparable performance, with VADAR again preferable for the inference-time reasons noted above, while sparse and dense both trail slightly.

Overall, VADAR-NF and VADAR-CF have comparable performance. VADAR-NF often has greater training stability and reaches higher reward, due to the absence of a current-goal critic in the weighting which can be over-exploited. We attribute VADAR-NF's performance on ROBOCASA365-LOADDISHWASHER due to this property. In contrast, VADAR-CF can exhibit slightly quicker training (LIBERO-Task 4) due to more aggressive exploitation of current goals.

5.2 Why VADAR outperforms HGC: the bottleneck of critic learning

Next, we identify the sources of the improved performance of VADAR related to HGC. Our key finding is that the major bottleneck in long-horizon RL is **critic learning**, and that *learning multi-critics suffices to reap the sample efficiency benefits of subtask decomposition without goal-conditioning the policy, while **enhancing** performance due to non-myopic optimization on future subtasks.*

First, we introduce 3 "Steelman" variants, which interpolate the design decisions between VADAR and HGC. Full details are given in Table 2. We can arrive at HGC from VADAR via 3 modifications: (a) learning a goal-conditioned, real-valued critic (rather than an all-goal, vector-valued one) (2) optimizing for the current-subtask and (3) conditioning the policy on the current goal. In Steelman 1 (dotted) , we replace the critic goal-conditioned critic with a vector-valued critic, as in VADAR, but keep the other components the same. Then, Steelman 2 (dashed) and Steelman 3 (solid) use current-myopic and next-future policy extraction, as in VADAR, but retain goal-conditioning of the policy. Finally, VADAR-CF/VADAR-NF remove policy goal-conditioning altogether.

Table 2: Steelman variants bridging HGC and VADAR-NF. Each row flips one of three design axes from the row above: the critic shape, the policy-extraction weighting (Table 1), or whether the policy is goal-conditioned. The gap between adjacent rows then isolates the contribution of a single component.

Variant	Critic	Policy weighting	Policy
HGC (reference)	$S\times A\times G\to\mathbb{R}$	current-myopic	$\pi(a \mid s, g)$
Steelman 1	$S\times A\to\mathbb{R}^{\lvert G\rvert}$	current-myopic	$\pi(a \mid s, g)$
Steelman 2	$S\times A\to\mathbb{R}^{\lvert G\rvert}$	current-future	$\pi(a \mid s, g)$
Steelman 3	$S\times A\to\mathbb{R}^{\lvert G\rvert}$	next-future	$\pi(a \mid s, g)$
VADAR-CF (ours)	$S\times A\to\mathbb{R}^{\lvert G\rvert}$	current-future	$\pi(a \mid s)$
VADAR-NF (ours)	$S\times A\to\mathbb{R}^{\lvert G\rvert}$	next-future	$\pi(a \mid s)$

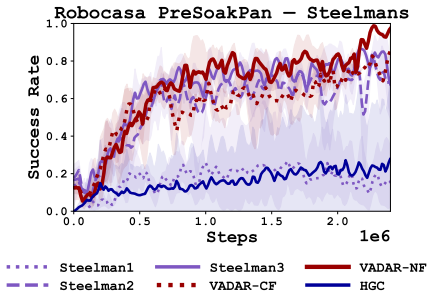


Figure 4: Steelman sweep on ROBOCASA365 PRESOAKPAN with the **OGPO** backbone.

Examining Figure 4, we see that simply moving to a vector valued critic (Steelman 1) does not salvage HGC's performance; however, using non-myopic policy extraction, Steelmen 2 & 3, approaches the performance of VADAR, with current-future slightly under-performing next-future in both cases. This strongly suggests that **non-myopic policy extraction** (Table 2) is an **essential** design decision responsible for VADAR's performance, and suggests that *traditional goal-conditioning is often limiting due to its insistence on optimizing for only the current subtask .*

While Figure 4 shows that vector-critics alone do not suffice to improve HGC's performance, we now establish that **coupled multi-subtask**

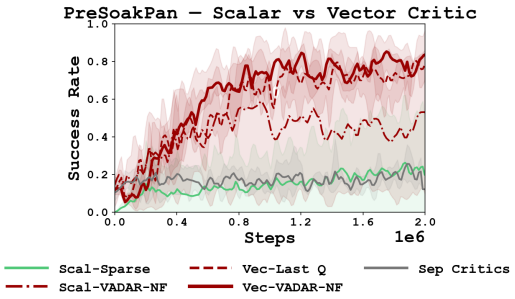


Figure 5: Critic ablation on VADAR with **OGPO** instantiation on ROBOCASA365 PRESOAKPAN

critics are the essential design decision. We consider a sequence of four ablations that interpolate between VADAR-NF and standard (non-goal conditioned) sparse reward: (1) sparse is the standard, sparse-reward RL, (2) VADARLAST-Q uses a VADAR $S \times A \to \mathbb{R}^G$ critic, but only relies on the *last* subtask critic value for policy extraction (3) SCALAR-VADAR instead uses next-future gating, but changes critic architecture to map $S \times A \times G \to \mathbb{R}$ to be a scalar, extracting the vector of critic values by concatenating $(Q_\phi(s, a, g))_{g \in G}$ and (4) separate critic is an extreme variant of SCALAR-VADAR, where per-subtask critics are implemented as separate networks. Surprisingly, we find that VADARLAST-Q *nearly matches* the performance of VADAR-NF, despite *never using intermediate subtasks for policy extraction*. Therefore, we conclude that VADAR's performance comes from the combination of non-myopicness in policy extraction (accomplished either by next-future gating, or more extremely, last-Q extraction) with multi-goal critics. We hypothesize that multi-goal critic training enables the network representations to simulate the Bellman backup operation; see Appendix M for details.

5.3 VADAR for Multi-Objective Optimization

We conclude by demonstrating one final capability of VADAR: increasing learning efficiency under multi-objective optimization. Specifically, we consider a setting where a reward function is decomposed into G groups $r = \sum_{g \in G} r^g$. Whereas traditional RL uses an *single critic* for the aggregate reward r, we propose to use VADAR to learn *separate per-reward critics*, which are aggregated after TD fitting in the policy extraction step.

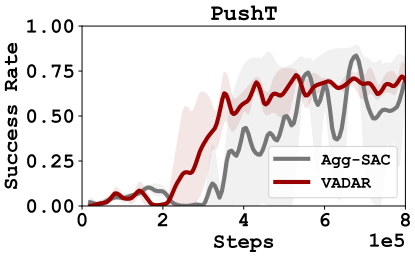


Figure 6: Multi-Objective Optimization using VADAR on a variant of `PushT` created: PUSHT-OBSTACLE. Here the goal of the agent is to align the T with the given position whilst avoiding contact with the obstacle

To test VADAR in this setting, we design PUSHT-OBSTACLE, a variant of the popular env `PushT` [Chi et al., 2023] which adds a fixed circular obstacle to be avoided, and include both a sparse task sparse *task* reward for placing the T-block on the goal pose r_t^{task}, and a dense *safety* reward that penalizes proximity to the obstacle r_t^{safety}. Full reward definitions are in Appendix G. We instantiate VADAR with **SAC** and with $G = 2$ heads (one per signal), using a constant weighting for the aggregated critic in policy extraction: $Q_{\text{agg}} = \mathbf{Q}_\phi[\text{task}] + \lambda_{\text{safety}} \mathbf{Q}_\phi[\text{safety}]$. We compare against standard RL with aggregated reward $r_t = r_t^{\text{task}} + \lambda_{\text{safety}} r_t^{\text{safety}}$ as a baseline(AGGREGATED-**SAC**). See Appendix G and Appendix G.5 for more details on the env and comparison of VADAR and AGGREGATED-**SAC** on PUSHT-OBSTACLE. Figure 6 shows that that VADAR outperforms pre-TD aggregated rewards, even when λ_{safety} *is chosen to maximize baseline performance*. In addition, VADAR performs well and trains stably across λ_{safety} safety, whilst AGGREGATED-**SAC** collapses.

6 Limitations and Conclusion

This work introduces VADAR, an approach to critic learning that utilizes subtask decomposition, which provides significant improvements across a wide range of policy extraction algorithms as well as challenging environment specifications such as long horizon or safety filtering. Still, a limitation of VADAR is that it defining a fixed DAG structure of subtasks, whereas other methods enable conditioning on a continuum of goal information [Black et al., 2024, Physical Intelligence et al., 2026]. We look forward to generalizing the design principles of VADAR to this continuous or combinatorially large goal spaces, such as those described by images or text, in future work.

Acknowledgments and Disclosure of Funding

This work was supported by a grant from the Toyota Research Institute (TRI) University 3.0 program. This research used the Babel computational cluster resources provided by the School of Computer Science at Carnegie Mellon University. The authors declare no competing interests.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: In introduction, we argue there are three benefits to VADAR, multi-task representation learning, non myopicness, and flexibility over the task-structure. Within Section 5 we demonstrate these characteristics and how VADAR improves over HGC

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: We discuss about the limitations and how we address them in Section 6

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

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

Answer: [N/A]

Justification: [N/A]

Guidelines:

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

4. **Experimental result reproducibility**

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

Answer: [Yes]

Justification: We provide detailed appendix sections, [Appendix D, Appendix F, Appendix G] and a clear practitioners guide (Appendix A) to reproduce the results presented in the paper.

Guidelines:

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

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

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

   Answer: [No]

   Justification: We intend to submit the code upon acceptance to the conference, after we get a chance to polish it up and strip the code of any identifying information.

   Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: The hyper parameters for all the results presented throughout the paper are thoroughly explained in the appendix. Within the core of the paper, we experiment with VADAR and the baselines being run in the exact same hyper parameters for a fair comparison and note of any key details that are required.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: We run every single experiment on three independent randomly generated seeds to reduce the variance of each trial. The results reported show a line which represents the average of the seeds, and the lightly shaded area behind it is the standard deviation between seeds.

   Guidelines:

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

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: We specify the computation resources used in the experiments in Appendix O

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The work shown does not work on humans, and any data handled is either publicly available or generated by the algorithms written. Thus the data has handled ethically, to the extent of having no ethical dilemmas.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [N/A]

Justification: This paper presents work whose goal is to advance the field of machine learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

Guidelines:

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

11. **Safeguards**

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

Answer: [N/A]

Justification: The models provided do not have a high risk for misuse.

Guidelines:

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

12. **Licenses for existing assets**

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

Answer: [Yes]

Justification: We have duly cited all the authors and owners of the papers, models, benchmarks and environments we have used in our work.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.

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

13. **New assets**

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

    Answer: [N/A]

    Justification: No new assets introduced in this paper.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: No crowdsourcing/research with human subjects done in this work.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: Not a research involving human subjects.

    Guidelines:

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

16. **Declaration of LLM usage**

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

    Answer: [No]

    Justification: LLM usage was primarily comprised for editing / formatting the writing and helping debug parts of the codebase.

    Guidelines:

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

References

Bhavya Agrawalla, Michal Nauman, Khush Agrawal, and Aviral Kumar. floq: Training critics via flow-matching for scaling compute in value-based rl. *arXiv preprint arXiv:2509.06863*, 2025.

Michael S Albergo, Nicholas M Boffi, and Eric Vanden-Eijnden. Stochastic interpolants: A unifying framework for flows and diffusions. *arXiv preprint arXiv:2303.08797*, 2023.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety, 2016. URL https://arxiv.org/abs/1606.06565.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay, 2018. URL https://arxiv.org/abs/1707.01495.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture, 2016. URL https://arxiv.org/abs/1609.05140.

Philip J Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, pages 1577–1594. PMLR, 2023.

André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P Van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Kevin Black, Mitsuhiko Nakamoto, Pranav Atreya, Homer Walke, Chelsea Finn, Aviral Kumar, and Sergey Levine. Zero-shot robotic manipulation with pretrained image-editing diffusion models. In *International Conference on Learning Representations (ICLR)*, 2024.

Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952. ISSN 00063444, 14643510. URL http://www.jstor.org/stable/2334029.

Daniel Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 783–792. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/brown19a.html.

Thomas P Cannon and Özgür Simsek. Accelerating task generalisation with multi-level skill hierarchies, 2025. URL https://arxiv.org/abs/2411.02998.

Kaiyuan Chen, Shuangyu Xie, Zehan Ma, Pannag R Sanketi, and Ken Goldberg. Robo2VLM: Improving visual question answering using large-scale robot manipulation data. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2026. URL https://openreview.net/forum?id=OChorZcZnY.

Xinyue Chen, Che Wang, Zijian Zhou, and Keith Ross. Randomized ensembled double q-learning: Learning fast without a model. *arXiv preprint arXiv:2101.05982*, 2021.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 2024.

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural computation*, 5(4):613–624, 1993.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In S. Hanson, J. Cowan, and C. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992. URL https://proceedings.neurips.cc/paper_files/paper/1992/file/d14220ee66aeec73c49038385428ec4c-Paper.pdf.

Thomas G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition, 1999. URL https://arxiv.org/abs/cs/9905014.

Yuqing Du, Ksenia Konyushkova, Misha Denil, Akhil Raju, Jessica Landon, Felix Hill, Nando De Freitas, and Serkan Cabi. Vision-language models as success detectors. *arXiv preprint arXiv:2303.07280*, 2023.

Joshua B. Evans and Özgür Şimşek. Creating multi-level skill hierarchies in reinforcement learning, 2024. URL https://arxiv.org/abs/2306.09980.

Roy Fox, Sanjay Krishnan, Ion Stoica, and Ken Goldberg. Multi-level discovery of deep options, 2017. URL https://arxiv.org/abs/1703.08294.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4(1):265–293, 2021.

Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019.

Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura, editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 1025–1037. PMLR, 30 Oct–01 Nov 2020. URL https://proceedings.mlr.press/v100/gupta20a.html.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. Pmlr, 2018a.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

Leslie Pack Kaelbling. Learning to achieve goals. In *International Joint Conference on Artificial Intelligence*, 1993. URL https://api.semanticscholar.org/CorpusID:5538688.

Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *2011 IEEE international conference on robotics and automation*, pages 1470–1477. IEEE, 2011.

Victoria Krakovna, Jonathan Uesato, Vladimir Mikulik, Matthew Rahtz, Tom Everitt, Ramana Kumar, Zachary Kenton, Jan Leike, and Shane Legg. Specification gaming: The flip side of ai ingenuity. https://deepmind.google/blog/specification-gaming-the-flip-side-of-ai-ingenuity/, 2020. DeepMind Blog.

Kimin Lee, Laura Smith, Anca Dragan, and Pieter Abbeel. B-pref: Benchmarking preference-based reinforcement learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. URL https://openreview.net/forum?id=ps95-mkHF_.

Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight, 2019. URL https://arxiv.org/abs/1712.00948.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning, 2023a. URL https://arxiv.org/abs/2306.03310.

Songtao Liu, Zhengkai Tu, Minkai Xu, Zuobai Zhang, Lu Lin, Rex Ying, Jian Tang, Peilin Zhao, and Dinghao Wu. Fusionretro: molecule representation fusion via in-context learning for retrosynthetic planning. In *International Conference on Machine Learning*, pages 22028–22041. PMLR, 2023b.

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *arXiv preprint arXiv:2108.03298*, 2021.

Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals, 2018. URL https://arxiv.org/abs/1807.04742.

Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets, 2021. URL https://arxiv.org/abs/2006.09359.

Soroush Nasiriany, Vitchyr Pong, Steven Lin, and Sergey Levine. Planning with goal-conditioned policies. *Advances in neural information processing systems*, 32, 2019.

Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. Robocasa365: A large-scale simulation framework for training and benchmarking generalist robots. *arXiv preprint arXiv:2603.04356*, 2026.

NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. Gr00t n1: An open foundation model for generalist humanoid robots, 2025. URL https://arxiv.org/abs/2503.14734.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35: 27730–27744, 2022.

Chaoyi Pan, Giri Anantharaman, Nai-Chieh Huang, Claire Jin, Daniel Pfrommer, Chenyang Yuan, Frank Permenter, Guannan Qu, Nicholas Boffi, Guanya Shi, and Max Simchowitz. Much ado about noising: Dispelling the myths of generative robotic control, 2026. URL https://arxiv.org/abs/2512.01809.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. Ogbench: Benchmarking offline goal-conditioned rl, 2025a. URL https://arxiv.org/abs/2410.20092.

Seohong Park, Qiyang Li, and Sergey Levine. Flow q-learning, 2025b. URL https://arxiv.org/abs/2502.02538.

Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *Proceedings of the 11th International Conference on Neural Information Processing Systems*, NIPS'97, page 1043–1049, Cambridge, MA, USA, 1997. MIT Press.

Sarvesh Patil, Mitsuhiko Nakamoto, Manan Agarwal, Shashwat Saxena, Jesse Zhang, Giri Anantharaman, Cleah Winston, Chaoyi Pan, Douglas Chen, Nai-Chieh Huang, Zeynep Temel, Oliver Kroemer, Sergey Levine, Abhishek Gupta, Hongkai Da, Paarth Shah, and Max Simchowitz. Ogpo: Sample efficient full-finetuning of generative control policies, 2026. URL https://arxiv.org/abs/2605.03065.

Shaohui Peng, Xing Hu, Rui Zhang, Ke Tang, Jiaming Guo, Qi Yi, Ruizhi Chen, Xishan Zhang, Zidong Du, Ling Li, Qi Guo, and Yunji Chen. Causality-driven hierarchical structure discovery for reinforcement learning, 2022. URL https://arxiv.org/abs/2210.06964.

Physical Intelligence, Bo Ai, Ali Amin, Ashwin Balakrishna, Kevin Black, Danny Driess, Michael Equi, Chelsea Finn, Lachlan Groom, Karol Hausman, Brian Ichter, Sergey Levine, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, et al. $\pi_{0.7}$: a steerable generalist robotic foundation model with emergent capabilities. *arXiv preprint arXiv:2604.15483*, 2026.

Arnaud Robert, Ciara Pike-Burke, and Aldo Faisal. Sample complexity of goal-conditioned hierarchical reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 62696–62712. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/c5ed2c8acda8c3716b1b6f9c6c713aaa-Paper-Conference.pdf.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1312–1320, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/schaul15.html.

Daniel Shin, Anca D. Dragan, and Daniel S. Brown. Benchmarks and algorithms for offline preference-based reward learning, 2023. URL https://arxiv.org/abs/2301.01392.

Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction second edition. *Adaptive computation and machine learning: The MIT Press, Cambridge, MA and London*, 2018.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(99)00052-1. URL https://www.sciencedirect.com/science/article/pii/S0004370299000521.

Sreyas Venkataraman, Yufei Wang, Ziyu Wang, Navin Sriram Ravie, Zackory Erickson, and David Held. Real-world offline reinforcement learning from vision language model feedback. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2025.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning, 2017. URL https://arxiv.org/abs/1703.01161.

Yufei Wang, Zhanyi Sun, Jesse Zhang, Zhou Xian, Erdem Biyik, David Held, and Zackory Erickson. Rl-vlm-f: Reinforcement learning from vision language foundation model feedback, 2024. URL https://arxiv.org/abs/2402.03681.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Tonghe Zhang, Chao Yu, Sichang Su, and Yu Wang. Reinflow: Fine-tuning flow matching policy with online reinforcement learning. *arXiv preprint arXiv:2505.22094*, 2025.

A Practitioner's Guide to VaDar

Here we give a short hands-on reference for applying VaDar to a new task or base algorithm.

Subgoal decomposition. Most of the long horizon tasks can be decomposed naturally into subtasks which should be taked as the subgoals for VaDar. For robotics tasks, we believe that the subgoals defined by us for our environments (Appendix F) can serve as good examples for subgoal selections for a new task.

Weighting variant. From our experiments, we found VaDar-NF to be the most performative variant for robotic manipulation tasks, and recommend it as the default. VaDar-CF can train slightly faster early on when the first subgoal is hard to reach from BC initialization, at the cost of greater susceptibility to over-exploiting the current goal. The myopic variants are useful as ablations but not as defaults.

Adapting to a new base algorithm. VaDar is a wrapper over any Value Based RL Algorithm. Given any value-based RL algorithm $\mathcal{A}$, the required modifications are:

1. Change critic output dim from 1 to $|\mathcal{G}|$ (backbone unchanged); ensemble aggregation becomes entrywise.
2. Store per-step completion vectors $\mathbf{d}_t$ and per-head rewards $\mathbf{r}_t$ in the replay buffer.
3. Replace the critic loss with its summed-over-heads version; broadcast any $\mathcal{L}_{\text{reg}}$ across heads (see Appendix C.1 for more details).
4. Apply the entrywise reward clip in the TD target.

5. In the policy loss, replace $Q_\phi(s,a)$ with $Q_{\text{agg}}(s,a) = \langle w(\mathbf{d}(s)), Q_\phi(s,a)\rangle$.

The policy is *not* conditioned on subgoals. We have applied this recipe unchanged to SAC, FQL, and OGPO (Appendix C), and these serve as good references for applying VADAR to any new Value Based RL Algorithm.

B Critic Ensembling and Target Networks

This section expands the brief mention in Section 3 of critic ensembling and target exponential moving averages (EMAs), which are standard practices in modern off-policy value-based RL and which we adopt without modification in VADAR. The recipes below apply uniformly across the base instantiations used in our experiments; specific details that interact with the policy update are covered in Appendix C.

B.1 Critic Ensembles

Off-policy methods are known to suffer from *overestimation bias*: maximizing over a noisy approximation of Q^π systematically inflates the resulting target [Fujimoto et al., 2018, Chen et al., 2021]. The standard remedy is to train an ensemble of M critic networks $Q_{\phi_1}, \ldots, Q_{\phi_M} : S \times A \to \mathbb{R}$ and aggregate their predictions when forming the TD target. We modify the classical scalar-valued critic in each ensemble member to be a vector-valued critic $\mathbf{Q}_{\phi_i} : S \times A \to \mathbb{R}^{|G|}$. Concretely, given any aggregator $\text{agg}_M : \mathbb{R}^M \to \mathbb{R}$ (e.g., mean, min), we extend it to act on the M vector critics by applying it independently to each subgoal head: for every $g \in G$,

$$\big[\text{agg}_M\big(\mathbf{Q}_{\phi_1}, \ldots, \mathbf{Q}_{\phi_M}\big)\big](s,a)[g] \;=\; \text{agg}_M\Big(\mathbf{Q}_{\phi_1}[g](s,a),\; \ldots,\; \mathbf{Q}_{\phi_M}[g](s,a)\Big). \tag{B.1}$$

We use Eq. (B.1) as the meaning of "entrywise aggregation" throughout the paper, and refer back to it whenever an aggregator is applied to vector critics. The ensemble size M is taken to be 10 ($M = 10$) by default, unless otherwise specified.

B.2 Target Aggregation

The TD target $\mathbf{Q}_{\bar\phi}(s',a')$ is computed by aggregating the predictions of M *target* vector critics $\mathbf{Q}_{\bar\phi_1}, \ldots, \mathbf{Q}_{\bar\phi_M} : S \times A \to \mathbb{R}^{|G|}$, whose parameters are an EMA of the online critics (Appendix B.3). We use the ensemble *mean* as the aggregator agg_M throughout, applied entrywise per Eq. (B.1):

$$\mathbf{Q}_{\bar\phi}(s',a') \;=\; \frac{1}{M}\sum_{i=1}^{M} \mathbf{Q}_{\bar\phi_i}(s',a') \;\in\; \mathbb{R}^{|G|}. \tag{B.2}$$

The aggregated scalar $Q_{\text{targ}}(s',a') = \langle \mathbf{w}, \mathbf{Q}_{\bar\phi}(s',a')\rangle$ used downstream by the policy update (Eq. (4.2)) is recovered by an inner product with the policy-extraction weights $\mathbf{w}$ ($\mathbf{w}(s) = \text{GETWEIGHTING}(\mathbf{d}(s))$ based on the weighting strategies we described in Table 1). Note that other standard alternatives like the full ensemble minimum, or a randomized two-sample minimum in the style of REDQ [Chen et al., 2021] can also be used for the curtailing overestimation bias, but we found the entrywise mean to work consistently well across all VADAR experiments and adopt it as the default.

B.3 Target Network EMAs

To stabilize the moving Bellman target, we maintain a separate set of target critic parameters $\bar\phi_1, \ldots, \bar\phi_M$ that lag the online parameters via Polyak averaging:

$$\bar\phi_i \;\leftarrow\; (1-\tau)\,\bar\phi_i + \tau\,\phi_i, \qquad i = 1, \ldots, M, \tag{B.3}$$

with decay rate $\tau \in (0,1]$ applied at every gradient step. We use $\tau = 0.005$ throughout. When the base algorithm requires a target policy, the same Polyak update is applied to the policy parameters:

$$\bar\theta \;\leftarrow\; (1-\tau)\,\bar\theta + \tau\,\theta. \tag{B.4}$$

Whether a target policy is maintained, and how it is used, is algorithm-specific; see Appendix C.

B.4 Critic Loss with the Vector Backbone

Combining Eqs. (B.2) and (B.3) with VADAR's vector-valued formulation (Eq. (4.1)) yields the following per-ensemble-member critic loss, summed over subgoal heads:

$$L_{\text{critic}}(\mathbf{Q}_{\phi_i} \mid s, a, \mathbf{y}^{\text{target}}) \;=\; \sum_{g \in G} \big(\mathbf{Q}_{\phi_i}[g](s,a) - \mathbf{y}_g^{\text{target}}\big)^2, \qquad i = 1, \ldots, M, \tag{B.5}$$

where $\mathbf{y}^{\text{target}}$ is the clipped vector TD target from Eq. (4.1).

C Base-Algorithm Details: SAC, FQL, and OGPO

We position VADAR as a wrapper around any value-based RL algorithm that decomposes scalar critics into subgoal-based vector-valued critics $\mathbf{Q}_{\phi_i}$ with a few variations in the Q_{targ} aggregation that can be plugged into the base algorithm's policy extraction objective. Here we present the policy losses L_{policy} and any algorithm-specific implementation details for the three base algorithms used in our experiments: **SAC**, **FQL**, and **OGPO**. Section 5.1 pairs each base algorithm with a benchmark suite (**SAC** with OGBENCH and LIBERO, **FQL** with ROBOMIMIC, and **OGPO** with both ROBOMIMIC and ROBOCASA365); coupling each policy extraction method to it's correspondingly challenging benchmark(s). The descriptions below are restricted to the components that interact with VADAR.

C.1 Soft Actor-Critic (SAC)

SAC [Haarnoja et al., 2018a] is a maximum-entropy off-policy actor-critic algorithm that augments the standard return with a per-step entropy bonus $\alpha\mathcal{H}(\pi(\cdot \mid s))$. The optimal policy is the one that maximizes

$$J(\pi) \;=\; \sum_{t \geq 0} \mathbb{E}_{(s_t, a_t)} \left[R(s_t, a_t) + \alpha\, \mathcal{H}\big(\pi(\cdot \mid s_t)\big) \right], \tag{C.1}$$

where $\alpha > 0$ is a temperature controlling the entropy-reward trade-off. We use a Gaussian policy with a $\tanh$-squashing transform that bounds actions in $[-1, 1]^{\dim A}$ [Haarnoja et al., 2018a, Eq. (11)]: $a_t = \tanh(\mu_\theta(s_t) + \sigma_\theta(s_t) \odot \epsilon_t)$, with $\epsilon_t \sim \mathrm{N}(0, I)$.

Critic loss. The soft Q-function is trained against an entropy-augmented Bellman target. Plugging into the value-based RL template of Section 3 ($L_{\text{critic}}(Q_\phi \mid s, a, y^{\text{target}}) = (Q_\phi(s,a) - y^{\text{target}})^2 + L_{\text{reg}}$), the entropy bonus enters through the regularizer in the target:

$$y^{\text{target}} \;=\; r + \gamma \left[Q_{\text{targ}}(s', a') - \alpha \log \pi_\theta(a' \mid s') \right], \qquad a' \sim \pi_\theta(\cdot \mid s'), \tag{C.2}$$

so that $L_{\text{reg}}(s', a', \pi_\theta) = -\gamma\, \alpha \log \pi_\theta(a' \mid s')$ in the notation of Section 3.

Policy loss. The actor minimizes the KL-divergence between the policy and the (exponentiated) soft Q-function. Using the reparameterization trick [Haarnoja et al., 2018b, Eq. (12)], this reduces to:

$$L_{\text{policy}}(\pi_\theta \mid s, Q_\phi) \;=\; \mathbb{E}_{a \sim \pi_\theta(\cdot \mid s)} \Big[\alpha \log \pi_\theta(a \mid s) - \mathtt{sg}\left[Q_\phi(s,a) \right] \Big], \tag{C.3}$$

where $\mathtt{sg}$ is the stopgrad operator. The expectation is estimated by a single reparameterized sample per state, and the gradient flows through the Squashed Gaussian into both the policy μ_θ and σ_θ heads.

Temperature auto-tuning. We tune α to match a target entropy $\bar{\mathcal{H}} = -\dim(A)$ via the auxiliary loss [Haarnoja et al., 2018b, Eq. (18)]:

$$L(\alpha) \;=\; \mathbb{E}_{a \sim \pi_\theta(\cdot \mid s)} \left[-\alpha \log \pi_\theta(a \mid s) - \alpha \bar{\mathcal{H}} \right]. \tag{C.4}$$

Interaction with VADAR. The scalar critic Q_ϕ is replaced by the vector critic $\mathbf{Q}_\phi : S \times A \to \mathbb{R}^{|G|}$. The Bellman target in Eq. (C.2) becomes a $|G|$-vector with the same entropy regularizer broadcast across subgoal heads,

$$\mathbf{y}^{\text{target}} \;=\; \mathbf{r} + \gamma \left[\mathbf{Q}_{\bar{\phi}}(s', a') - \alpha \log \pi_\theta(a' \mid s')\, \mathbf{1}_{|G|} \right], \qquad a' \sim \pi_\theta(\cdot \mid s'), \tag{C.5}$$

where $\mathbf{Q}_{\bar{\phi}}$ are the target vector critic as defined in Appendix B.3 and the per-ensemble-member critic loss is summed over heads (Appendix B):

$$L_{\text{critic}}\big(\mathbf{Q}_{\phi_i} \mid s, a, \mathbf{y}^{\text{target}}\big) \;=\; \sum_{g \in G} \big(\mathbf{Q}_{\phi_i}[g](s,a) - \mathbf{y}_g^{\text{target}}\big)^2. \tag{C.6}$$

The actor sees only the aggregated scalar $Q_{\text{agg}}(s,a) = \langle \mathbf{w}, \mathbf{Q}(s,a) \rangle$, so the policy loss in Eq. (C.3) becomes

$$L_{\text{policy}}(\pi_\theta \mid s, \mathbf{Q}_\phi) \;=\; \mathbb{E}_{a \sim \pi_\theta(\cdot|s)}\Big[\alpha \log \pi_\theta(a \mid s) - \mathtt{sg}(\langle \mathbf{w}, \mathbf{Q}_\phi(s,a) \rangle)\Big]. \tag{C.7}$$

The temperature loss remains unchanged.

C.2 Flow Q-Learning (FQL)

FQL [Park et al., 2025b] is an off-policy method designed for expressive flow-matching policies. Its key idea is to avoid backpropagating through the multi-step ODE integration of a flow policy by training a separate, expressive *one-step* policy that is simultaneously (i) regularized toward a frozen behavioral-cloning (BC) flow policy via a distillation loss and (ii) optimized to maximize the critic.

Architecture. **FQL** maintains three networks: a critic Q_ϕ, a BC flow policy $\mu_\theta(s,z)$ that defines a flow-matching velocity field $v_\theta(t,s,x^t)$, and a one-step policy $\mu_\omega(s,z) : S \times \mathbb{R}^{\dim A} \to A$ that maps Gaussian noise $z \sim \mathrm{N}(0, I_{\dim A})$ directly to an action.

BC flow loss. The BC flow policy is trained with the standard flow-matching objective on the offline dataset $\mathcal{D}$ [Park et al., 2025b, Eq. (5)]:

$$\mathcal{L}_{\text{Flow}}(\theta) \;=\; \mathbb{E}_{\substack{(s,a^1)\sim\mathcal{D},\, x^0 \sim \mathrm{N}(0,I) \\ t \sim \mathrm{Unif}([0,1])}}\Big[\big\| v_\theta(t,s,x^t) - (a^1 - x^0) \big\|_2^2\Big], \quad x^t = (1-t)x^0 + t\,a^1. \tag{C.8}$$

The corresponding stochastic policy $\pi_\theta(a \mid s)$ is the push-forward of $\mathrm{N}(0,I)$ under the ODE $\mu_\theta(s,\cdot)$ defined by v_θ. Crucially, this loss is the only training signal seen by the flow policy: μ_θ is never updated against Q_ϕ.

Critic loss. The critic is trained with the standard TD residual but using the *one-step* policy μ_ω in the bootstrap action [Park et al., 2025b, Sec. 3]:

$$L_{\text{critic}}(Q_\phi) \;=\; \mathbb{E}_{(s,a,r,s')\sim\mathcal{D},\, z\sim\mathrm{N}(0,I)}\Big[\big(Q_\phi(s,a) - r - \gamma\, Q_{\text{targ}}(s', \mu_\omega(s',z))\big)^2\Big]. \tag{C.9}$$

where $Q_{\text{targ}} = Q_{\bar{\phi}}$ is the EMA target critic (Appendix B.3).

One-step policy loss. The one-step policy is trained against an actor loss with two terms: a Q-maximization term and a distillation term that pulls $\mu_\omega(s,z)$ toward the BC flow policy's output $\mu_\theta(s,z)$ at the same noise z [Park et al., 2025b, Eqs. (7),(9)]:

$$L_{\text{policy}}(\pi_\omega \mid s, Q_\phi) \;=\; \underbrace{\mathbb{E}_{\substack{s\sim\mathcal{D},\, z\sim\mathrm{N}(0,I) \\ a^\pi = \mu_\omega(s,z)}}\big[-Q_\phi(s,a^\pi)\big]}_{\text{Q loss}} \;+\; \alpha_{\text{FQL}} \underbrace{\mathbb{E}_{\substack{s\sim\mathcal{D} \\ z\sim\mathrm{N}(0,I)}}\big[\|\mu_\omega(s,z) - \mu_\theta(s,z)\|_2^2\big]}_{\text{distillation loss}}. \tag{C.10}$$

Gradients of the distillation term are stopped at μ_θ, so the BC flow policy is not affected.

Interaction with VADAR. Whem applying VADAR on top of **FQL**, the scalar critic gets replaced by the vector shared critic: $\mathbf{Q}_\phi : S \times A \to \mathbb{R}^{|G|}$. The TD bootstrap becomes a $|G|$-vector,

$$\mathbf{y}^{\text{target}} \;=\; \mathbf{r} + \gamma\, \mathbf{Q}_{\bar{\phi}}\big(s', \mu_\omega(s',z)\big), \qquad z \sim \mathrm{N}(0,I), \tag{C.11}$$

and the per-ensemble-member critic loss is summed over heads:

$$L_{\text{critic}}\big(\mathbf{Q}_{\phi_i} \mid s, a, \mathbf{y}^{\text{target}}\big) \;=\; \mathbb{E}_{(s,a,\mathbf{r},s')\sim\mathcal{D},\, z}\Big[\sum_{g \in G} \big(\mathbf{Q}_{\phi_i}[g](s,a) - \mathbf{y}_g^{\text{target}}\big)^2\Big]. \tag{C.12}$$

Similar to before, we define the online aggregate $Q_{\text{agg}}(s,a) := \langle \mathbf{w}, \mathbf{Q}_\phi(s,a) \rangle$ (online critic, matching the **FQL** convention of Eq. (C.10)). The Q-loss term in Eq. (C.10) then becomes

$$L_{\text{policy}}(\pi_\omega \mid s, Q_{\text{agg}}) \;=\; \underbrace{\mathbb{E}_{\substack{s\sim\mathcal{D},\, z\sim\mathrm{N}(0,I)\\ a^\pi=\mu_\omega(s,z)}}\big[-Q_{\text{agg}}(s,a^\pi)\big]}_{\text{Q loss}} \;+\; \alpha_{\text{FQL}}\,\mathcal{L}_{\text{distill}}(\omega;\theta). \tag{C.13}$$

The distillation term is unchanged. The deployed policy is the one-step μ_ω, requiring no goal information at inference (in line with VADAR's subgoal-agnostic property).

C.3 Off-Policy Generative Policy Optimization (OGPO)

OGPO [Patil et al., 2026] is an off-policy actor-critic for generative control policies (GCPs), like flow matching and diffusion, that side-steps the instabilities of backpropagation through the flow ODE by applying a zero-order PPO-style update to the denoising chain itself. VADAR pairs with **OGPO** on our most challenging long-horizon benchmarks (ROBOMIMIC, and ROBOCASA365). We summarize the policy loss here and note that **OGPO** follows the same aforementioned critic ensembling and EMA strategies (Appendix B).

Setup. A flow-based GCP samples an action $a_{t,0}$ via a K-step denoising chain $a_{t,K} \to a_{t,K-1} \to \cdots \to a_{t,0}$, where each step samples $a_{t,k-1} \sim \pi_\theta(\cdot \mid s_t, a_{t,k})$. The likelihood of a full chain factorizes as

$$\pi_\theta(a_{t,K:0} \mid s_t) \;=\; \prod_{k=1}^{K} \pi_\theta(a_{t,k-1} \mid s_t, a_{t,k}). \tag{C.14}$$

OGPO treats the denoising trajectory $a_{t,K:0}$ as a sequence in an inner "denoising MDP" embedded inside the environment MDP, and applies PPO at the level of this inner MDP only.

Annealed importance sampling ratio. The reference policy is the EMA target $\pi_{\bar\theta}$ (Appendix B.3). The policy ratio is the product of per-step ratios over the entire chain:

$$\omega_\theta \;:=\; \prod_{k=1}^{K} \frac{\pi_\theta(a_{t,k-1} \mid s_t, a_{t,k})}{\pi_{\bar\theta}(a_{t,k-1} \mid s_t, a_{t,k})}. \tag{C.15}$$

OGPO Policy Updates **OGPO** performs policy extraction via a PPO-style zeroth order policy gradient update that considers the entire flow-denoising chain as an MDP. The terminal action $a_{t,0}$ is the only action evaluated against the target critic. **OGPO** draws a *group* of $N_{\texttt{group}}$ denoising trajectories $a_{K:0}^{(i,j)}$ from $\pi_{\bar\theta}(\cdot \mid s^{(i)})$ per state and uses the group mean as a value baseline:

$$\hat V^{(i)} \;\leftarrow\; \frac{1}{N_{\texttt{group}}} \sum_{j=1}^{N_{\texttt{group}}} Q_{\text{targ}}\big(s^{(i)}, a_0^{(i,j)}\big), \qquad \hat A \;=\; Q_{\text{targ}}(s_t, a_{t,0}) - \hat V. \tag{C.16}$$

This GRPO-style baseline obviates the need for a separately learned value network.

Policy loss. The PPO-clipped surrogate loss is applied over the entire flow denoising chain as:

$$L_{\text{policy}}(\pi_\theta \mid s, Q_{\text{targ}}) \;=\; -\mathbb{E}_{\substack{s_t\sim\mathcal{B}\\ a_{t,K:0}\sim\pi_{\bar\theta}(\cdot|s_t)}}\Big[\min\big(\omega_\theta\,\hat A,\; \text{clip}(\omega_\theta, 1-\epsilon, 1+\epsilon)\,\hat A\big)\Big], \tag{C.17}$$

where ϵ is the PPO clip parameter. Because both ω_θ and the denoising trajectories can be computed in parallel without the need for environmental interactions, the loss is fully parallelizable across the $N_{\texttt{batch}} \times N_{\texttt{group}}$ batch.

Noise injection for flow policies. Evaluating ω_θ requires per-step likelihoods $\pi_\theta(a_{t,k-1} \mid s_t, a_{t,k})$ to be non-singular. Following ReinFlow [Zhang et al., 2025], we replace the deterministic Euler step with a Gaussian-noised update $a_{t,k-1} \sim \mathrm{N}(v_\theta(a_{t,k}, k/K, s_t),\, \sigma_k^2 I)$, and retain the the marginal path-preserving correction of Albergo et al. [2023] so that the per-step SDE marginals match standard ODE flow matching rollouts.

Interaction with VADAR. VADAR composes with **OGPO** by replacing the scalar Q_{targ} in the advantage computation with the aggregated $Q_{\text{agg}} = \langle \mathbf{w}, \mathbf{Q}_{\bar{\phi}}(s, a) \rangle$ from Eq. (4.2). The PPO surrogate, group baseline, and noise-injection scheme are otherwise unchanged. For our experiments, unless otherwise stated, we use the **OGPO+** variant (due to its superior performance).

D Dense Preference Reward: Algorithms and Implementation Details

We implemented dense prefence rewards for the same policy backbones as VADAR: **OGPO**, **SAC**, **FQL**. Our PBRL implementation was heavily inspired by [Shin et al., 2023, Wang et al., 2024, Venkataraman et al., 2025] . We used trajectory-wise preferences to train a state-based reward model independently for each task. Trajectory preferences were autogenerated and fit using the Bradley–Terry [Bradley and Terry, 1952] model. When training the reward model, we uniformly sample from the generated preference dataset. **We found that the most reliable preference signal was binary trajectory success.** To ensure we used the most accurate preference signal, we experimented with alternatives in Table 3, but found them to be inaccurate.

Table 3: Alternative auto generated preference signals tested for PBRL.

Signal	Observed issue
$\sum_{s \in \tau} V^*(s)$	Uses the value function V^* from an optimal policy π^*, but produced noisy preferences because V^* gave poor estimates outside the distribution induced by π^*.
$\sum_{t, s_t \in \tau} \gamma^t r_{\text{sparse}}(s_t)$	Monte Carlo returns depend on the policy used to generate the trajectory, making the induced reward signal noisy and not policy-invariant at the state level.
$\sum_{r_{\text{dense}} \in \tau} r_{\text{dense}}$	Uses r_{dense} as defined from the QIPM main algorithm, but it preferred longer trajectories, which introduced a harder and less reliable preference signal for the reward model to learn.

Offline Training PBRL was provided privileged information, a diverse set of trajectories to kick off it's reward training from, 100 trajectories collected from $(\pi_{\text{EARLY}}, \pi_{\text{MID}}, \pi_{\text{LATE}})$, where these were checkpoints collected during the training of an optimal policy, π^*. This dataset combined with the original optimal success dataset provided on average 500 trajectories to pre-train the reward model from, with different levels of optimality and a diverse range of failure modes. **We saw that this was required for PBRL to show signs of success**, as demonstrated in Figure 7. This is a significant setback compared to VADAR, which is able to operate on far more limited datasets.

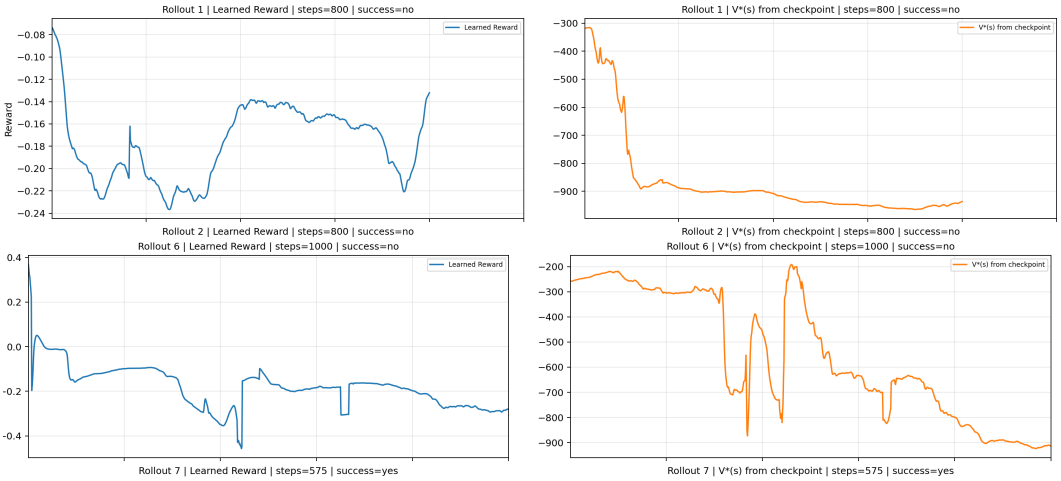


Figure 7: Comparison of success-only (top) and mixed (bottom) dataset based reward rollouts. We see that the success-only dataset is unable to accurately estimate failure cases, not having a clear signal for reward. Both models were using preference based on V^*

Online Training After offline reward-model pretraining, we continue updating the dense reward model during online RL following [Wang et al., 2024, Lee et al., 2021]. The reward model is trained using the following iterative loop: the current reward model is used to train the policy, the resulting policy collects new trajectories, these trajectories are converted into additional synthetic preference comparisons, and the reward model is fine-tuned on the offline and online preference dataset.

During online RL, the environment reward is replaced by the learned state-only reward model $\hat{r}_\theta(s)$. The underlying RL algorithm is otherwise unchanged: **OGPO**, **SAC**, and **FQL** use their standard actor and critic losses, but with rewards given by $\hat{r}_\theta$ instead of the environment reward.

Every K_{int} environment steps, the reward model is fine-tuned on the union of offline and online preference pairs using the same Bradley–Terry objective as in offline pretraining. The reward model gradually adapts to the on-policy state distribution induced by the improving policy, while still being regularized by the original offline preference data.

After each reward-model update, we relabel the live RL replay buffer with the updated reward model: $r_t \leftarrow \hat{r}_\theta(s_t)$. This step makes the online reward update immediately affect subsequent policy learning.

Hyper Parameters Table 4 mentions the primary hyper parameters used to train the model.

Table 4: Reward-model hyperparameters used for PBRL experiments.

Hyperparameter	Value
Reward model	
ensemble size	3
model architecture	MLP of $(512, 512, 512, 512)$
learning rate	3×10^{-4}
training epochs	50
segment size	`episode`
Online PBRL schedule	
RM update interval	5,000 env steps
maximum online feedback	10,000 preference pairs
feedback buffer	FIFO

E DAG Extension for Non-Sequential Subtasks

In the sequential setting, subtasks must be completed in a fixed linear order. However, many real-world tasks have more complex structures where multiple subtasks can be pursued simultaneously for optimal behavior. Moreover, some subtasks can form a precondition boundary for others, where policies can be rolled out in the future subtasks only if the preconditions are satisfied. VADAR can naturally be applied to a litany of these subtask dependencies often represented as directed acyclic graphs (DAGs) with each node corresponding to an individual Q-function. This provides the core benefits of value disaggregation to rich dependency structures dominant in long-horizon RL settings.

Illustrative Example: Robomimic Transport. To illustrate this, consider the robomimic Transport task, a multi-robot task where two robot arms must coordinate to transport a payload from one bin to another while managing trash removal. The task decomposes into 7 interdependent subtasks forming a DAG:

1. `lid_removed`: Lid removed from start bin
2. `payload_grasped`: Payload grasped from start bin (Robot 0)
3. `payload_lifted`: Payload lifted above bins (Robot 0)
4. `trash_grasped`: Trash grasped from target bin (Robot 1)
5. `trash_in_bin`: Trash placed in trash bin (Robot 1)
6. `payload_handed_off`: Payload transferred from Robot 0 to Robot 1
7. `payload_in_target`: Payload placed in target bin (SUCCESS)

The dependencies form two parallel execution paths: $(1 \to 2 \to 3)$ for payload handling by Robot 0, and $(4 \to 5)$ for trash management by Robot 1. The handoff step (6) requires both paths to complete, and the final success (7) depends on the handoff, an illustration is provided in Figure 8.

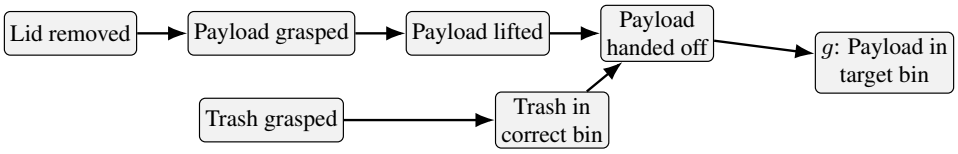


Figure 8: Transport Subgoal Structure.

DAG Formalism. Recall from Section 3.1 that the subgoals G are partially ordered, where $g^i \prec g^j$ means that g^i must be completed before g^j. We represent this structure with a DAG $\mathcal{D} = (G, E)$, with

$$(g^i, g^j) \in E \iff g^i \prec g^j \text{ and } \nexists\, g^k \in G \text{ such that } g^i \prec g^k \prec g^j.$$

This graph is acyclic: a directed cycle would imply $g^i \prec g^i$ for some subgoal g^i, contradicting the prerequisite relation. The linear ordering from Section 3.1 is the special case in which the DAG is a chain: every subgoal except the initial one has exactly one parent. We define the following graph functions:

$$\mathrm{pred}(g) \coloneqq \{p \in G : (p, g) \in E\} \qquad \mathrm{desc}(S) \coloneqq \{g \in G : \exists s \in S \text{ such that } s \preceq g\}.$$

Gate Weights for DAG Structures. For a DAG, the "current" subgoals are the incomplete goals whose prerequisites have all been completed:

$$G^{\mathrm{curr}}(\mathbf{d}) \coloneqq \{g \in G : \mathbf{d}_g = 0 \text{ and } \mathbf{d}_p = 1 \ \ \forall p \in \mathrm{pred}(g)\},$$

We also define the terminal subgoals and the immediate successors of the current frontier as

$$G^{\mathrm{term}} \coloneqq \{g \in G : \nexists\, h \in G \text{ such that } (g, h) \in E\},$$

$$G^{\mathrm{next}}(\mathbf{d}) \coloneqq \{g \in G : \exists\, p \in G^{\mathrm{curr}}(\mathbf{d}) \text{ such that } (p, g) \in E\} \cup \left(G^{\mathrm{curr}}(\mathbf{d}) \cap G^{\mathrm{term}}\right).$$

For g^{next}, we handle the edge case where an element from g^{curr} is a terminal node, putting those nodes into the set. Using these sets, the DAG value-weighting variants are given in Table 5, provided that we are given $\mathbf{d}$.

Table 5: DAG value-weighting variants used for policy extraction in VADAR.

Config.	**Weight** w_g	**Interpretation**
Current-myopic	$\mathbb{I}[g \in G^{\mathrm{curr}}(\mathbf{d})]$	Only currently available subtasks contribute.
Current-future	$\mathbb{I}[g \in \overline{\mathrm{desc}}(G^{\mathrm{curr}}(\mathbf{d}))]$	Current subtasks and their descendants contribute.
Next-myopic	$\mathbb{I}[g \in G^{\mathrm{next}}(\mathbf{d})]$	Only direct successors of current subtasks contribute.
Next-future	$\mathbb{I}[g \in \overline{\mathrm{desc}}(G^{\mathrm{next}}(\mathbf{d}))]$	Next subtasks and their descendants contribute.

F Environment Details: Per-Task Descriptions: Subgoals, Dones, and DAG Structure

We list the subgoals used by VADAR and HGC on each benchmark task, taken directly from the simulator wrappers in our codebase. For each task we list (i) the high-level objective, (ii) the per-subgoal completion indicators $\texttt{done}_g(s)$ as evaluated from low-level simulator state (no VLM calls are used for `done` evaluation), and (iii) the prerequisite structure used to gate subgoal completion. Subgoal completion is evaluated with *monotonic completion* semantics: once $\texttt{done}_g$ becomes true and all prerequisites are satisfied, $\texttt{done}_g$ remains true for the rest of the episode. The default prerequisite structure is a sequential chain (g^i requires $g^1, \ldots, g^{i-1}$); tasks that override this with an explicit DAG are noted below. See Appendix E for an in-depth explanation of the DAG structure.

F.1 ROBOCASA365

Our longest-horizon tasks come from the ROBOCASA365 [Nasiriany et al., 2026] suite. Each task involves multiple sequential phases of long-horizon kitchen manipulation. For our experiments, we use the state-based environments of ROBOCASA365 composite tasks (having multiple subtasks to be done in order to achieve success), which do not involve the movement of the robot's base. Concretely, we evaluate on the following ROBOCASA365 tasks:

PRESOAKPAN. The agent must place a pan in the sink, place a sponge inside the pan in the sink, and then turn on the faucet. This task is decomposed into 5 sequential subgoals:

1. $g^1 =$ `pan_grasped_lifted`: pan (`obj1`) is grasped and lifted to height $z > 0.88$ m.
2. $g^2 =$ `pan_in_sink`: pan is contained inside the sink basin (`obj_inside_of(obj1, sink)`).
3. $g^3 =$ `sponge_grasped_lifted`: sponge (`obj2`) is grasped and lifted to height $z > 0.88$ m.
4. $g^4 =$ `sponge_in_pan_and_sink`: sponge is inside the sink, with xy-distance to pan < 0.25 m and vertical offset $z_{\text{sponge}} - z_{\text{pan}} \in (-0.05, 0.20)$ m.
5. $g^5 =$ `water_on`: faucet is on, both pan and sponge are in the sink, and the gripper is far from both objects (matches the environment's original success check).

Final task success is $\texttt{done}_{g^5}$.

LOADDISHWASHER. The agent must load two dishes (a cup and a bowl) into the dishwasher rack and close the dishwasher. Decomposed into 5 sequential subgoals:

1. $g^1 =$ `cup_grasped`: cup (`dish0`) is grasped (`check_obj_grasped`).
2. $g^2 =$ `cup_in_dw`: cup is in contact with the dishwasher rack.
3. $g^3 =$ `bowl_grasped`: bowl (`dish1`) is grasped.
4. $g^4 =$ `bowl_in_dw`: bowl is in contact with the dishwasher rack.
5. $g^5 =$ `dw_closed`: cup in rack *and* bowl in rack *and* dishwasher door closed (angular tolerance 0.05).

Final task success is $\texttt{done}_{g^5}$.

ARRANGETEA : This is the third ROBOCASA365 task we evaluate on. In this task, the agent must pick up the kettle and place it on a tray on the kitchen counter, then pick up a mug from inside the kitchen cabinet and then place it next to the kettle on the tray, and then close the hinged cabinet that initially holds the mug. This task is decomposed into 6 natural subgoals:

1. $g^1 =$ `kettle_grasped`: kettle is grasped (`check_obj_grasped(kettle)`).
2. $g^2 =$ `kettle_on_tray`: kettle is contained on the tray (`check_obj_in_receptacle(kettle, container)`).
3. $g^3 =$ `kettle_on_tray` & `mug_grasped`: kettle still on tray *and* mug is grasped.
4. $g^4 =$ `kettle_on_tray` & `mug_on_tray`: kettle still on tray *and* mug is now on the tray.
5. $g^5 =$ `both_on_tray` & `right_door_closed`: both objects on tray *and* the right cabinet door is closed.
6. $g^6 =$ `both_on_tray` & `cab_closed`: both objects on tray *and* the full hinged cabinet is closed.

Final task success is $\texttt{done}_{g^6}$ which matches the original success of the ARRANGETEA task.

F.2 ROBOMIMIC

ROBOMIMIC [Mandlekar et al., 2021] contributes two contact-rich, medium-horizon tasks.

TRANSPORT. A bimanual task in which two arms coordinate to lift a lid, transport a payload between bins, and dispose of trash. The codebase exposes 7 subgoal indicators; the user-facing description of the DAG is given in Appendix E. The 7 subgoal checks are:

1. g^1 = `lid_removed`: lid handle has slid leftward by more than 0.08 m from its initial x-position ($x_{\text{lid}} < x_{\text{lid}}^{\text{init}} - 0.08$).
2. g^2 = `payload_grasped`: Robot 0's gripper is in contact with the payload geoms (`_check_grasp`).
3. g^3 = `payload_lifted`: payload z-position exceeds the table height by > 0.10 m.
4. g^4 = `trash_grasped`: Robot 1's gripper is in contact with the trash geoms.
5. g^5 = `trash_in_bin`: `transport.trash_in_trash_bin` flag is true.
6. g^6 = `payload_handed_off`: Robot 1 is grasping the payload *and* Robot 0 is not.
7. g^7 = `payload_in_target`: `transport.payload_in_target_bin` flag is true.

Structure: two parallel chains ($g^1 \to g^2 \to g^3$) and ($g^4 \to g^5$), both prerequisites for g^6, which is itself a prerequisite for g^7 (Figure 8). Final task success is $\texttt{done}_{g^7}$.

TOOL-HANG. A single-arm precision task: insert a hook frame into a base stand, then hang a wrench on it. Decomposed into 4 sequential subgoals:

1. g^1 = `frame_grasped`: gripper is grasping the frame and the frame is lifted > 0.02 m above the table (`_check_grasp_frame`).
2. g^2 = `frame_inserted`: frame is assembled into the stand (`_check_frame_assembled`).
3. g^3 = `frame_inserted_and_wrench_grasped`: frame is still inserted *and* the wrench is grasped and lifted > 0.02 m above the table.
4. g^4 = `wrench_attached`: wrench is hung on the inserted frame (`_check_tool_on_frame`).

The final success is $\texttt{done}_{g^4}$ which matches the success of the original TOOL-HANG env.

F.3 OGBENCH

Next we use the `scene-play-singletask-task4` (`put_in_drawer`) from OGBENCH [Park et al., 2025a]. In this task, the agent needs to first unlock the drawer (by pressing a button), then open the drawer, then lift the cube and place it inside the drawer and finally close the drawer. The subgoal decomposition is as follows:

1. g^1 = `press_button_0`: button 0 state is set to pressed (`_cur_button_states[0] == 1`); this unlocks the drawer.
2. g^2 = `open_drawer`: drawer slide x-position < -0.12 (slide range $[-0.16, 0]$).
3. g^3 = `block_in_drawer`: block xyz-position satisfies the env's `_is_in_drawer` predicate.
4. g^4 = `close_drawer`: $|x_{\text{slide}}| < 0.04$ (drawer back to nearly fully closed).

Final task success is $\texttt{done}_{g^4}$ which is coonsisitent with the origianl success.

F.4 LIBERO

We evaluate VADAR on 3 multi stage kitchen tasks from the LIBERO-10 [Liu et al., 2023a] suite. These are: `KITCHEN_SCENE3`, `KITCHEN_SCENE4`, `KITCHEN_SCENE6`. We use the following subgoal decomposition for these tasks:

KITCHEN_SCENE3 (turn on the stove and place the moka pot). BDDL goal: (Turnon flat_stove_1) ∧ (On moka_pot_1 flat_stove_1_cook_region). Decomposed into 3 sequential subgoals:

1. g^1 = stove_on: (turnon flat_stove_1).
2. g^2 = moka_pot_grasped: gripper is grasping any geom containing moka_pot_1.
3. g^3 = moka_pot_on_stove: (on moka_pot_1 flat_stove_1_cook_region).

Final task success is $\texttt{done}_{g^3}$.

KITCHEN_SCENE4 (place black bowl in bottom drawer and close it). The bottom drawer of white_cabinet_1 is open at reset. BDDL goal: (Close white_cabinet_1_bottom_region) ∧ (In akita_black_bowl_1 white_cabinet_1_bottom_region). Decomposed into 3 sequential subgoals:

1. g^1 = bowl_grasped: gripper is grasping any geom containing akita_black_bowl.
2. g^2 = bowl_in_drawer: (in akita_black_bowl_1 white_cabinet_1_bottom_region).
3. g^3 = drawer_closed: (close white_cabinet_1_bottom_region).

Final task success is $\texttt{done}_{g^3}$.

KITCHEN_SCENE6 (place mug in microwave and close it). The microwave is open at reset. BDDL goal: (In white_yellow_mug_1 microwave_1_heating_region) ∧ (Close microwave_1). Decomposed into 3 sequential subgoals:

1. g^1 = mug_grasped: gripper is grasping any geom containing white_yellow_mug.
2. g^2 = mug_in_microwave: (in white_yellow_mug_1 microwave_1_heating_region).
3. g^3 = microwave_closed: (close microwave_1).

Final task success is $\texttt{done}_{g^3}$.

F.5 PUSHT-OBSTACLE

The multi-objective PUSHT-OBSTACLE task does not use a sequential subgoal decomposition; instead, the two "heads" correspond to two independent per-step signals defined in Appendix G:

1. g^1 = task_done: block-goal coverage (intersection-over-union proxy) ≥ 0.95. Ratchets up.
2. g^2 = collision_occurred: continuous proximity cost $c_t \in [0, 1]$ around the obstacle boundary (1 inside, decaying linearly across a 40-px soft margin).

Neither head is a prerequisite for the other, so GETWEIGHTING collapses to a constant weighting $\mathbf{w} = (1, \lambda_{\text{safety}})$.

F.6 Hyperparameters

VADAR is tested with three off policy value based instantiations. Hyperparameters for each instantiation follows the best practices recommended in the corresponding original papers; we tune only the small set of task-level knobs (training-step budget, discount, BC regularization strength) needed to keep each baseline competitive on the task at hand, and we use the *same* setting across all baselines (Sparse, Dense, HGC, VADAR) on a given task so that comparisons are not confounded by tuning.

Table 6: Summary of subgoal counts and dependency structure across benchmark tasks.

Suite	Task	$\|G\|$	Structure
RoboCasa365	PreSoakPan	5	sequential chain
RoboCasa365	LoadDishwasher	5	sequential chain
RoboCasa365	ArrangeTea	6	sequential chain
RoboMimic	Transport	7	DAG, two parallel chains, joint handoff
RoboMimic	Tool-Hang	4	sequential chain
Libero	`KITCHEN_SCENE3`	3	sequential chain
Libero	`KITCHEN_SCENE4`	3	sequential chain
Libero	`KITCHEN_SCENE6`	3	sequential chain
OGBench	`scene-play-singletask-task4`	4	sequential chain
–	PushT-Obstacle (multi-objective)	2	independent heads (no prerequisites)

Table 7: Maximum episode length per environment (used both for evaluation and as the truncation horizon during online finetuning)

Environment	Max episode length
RoboCasa365 (all envs: ArrangeTea, PreSoakPan, LoadDishwasher)	1200
RoboMimic Transport	800
RoboMimic Tool-Hang	1000
Libero `KITCHEN_SCENE3/4/6`	500
OGBench `scene-play-singletask-task4`	100
PushT-Obstacle	300

Table 8: Hyperparameters shared across all tasks and backbones (**OGPO**, **FQL**, **SAC**)

Hyperparameter	Value
Optimization	
critic learning rate	3×10^{-4}
actor / BC learning rate	3×10^{-4}
batch size	256
replay buffer size	2×10^{6}
Networks	
actor MLP hidden dims	$(512, 512, 512, 512)$
critic MLP hidden dims	$(512, 512, 512, 512)$
critic ensemble size M	10
ensemble aggregation	`mean`
target EMA τ (critic)	0.05
target EMA $\bar{\tau}_\pi$ (actor)	0.05
VaDar *critic*	
critic shape	$S \times A \to \mathbb{R}^{\|G\|}$
advantage gating mode	`grandparent_future` (VaDar-NF) / `goal_future` (VaDar-CF)
Online schedule	
UTD (warmup, online)	$(1, 1)$

F.6.1 Common Hyperparameters

Most of our hyper parameters remain the same for all are experiments, unless explicitly mentioned. The detailed list of these are listed in Table 8.

F.6.2 OGPO specific hyperparameters

OGPO[Patil et al., 2026] is our flow-based backbone for the RoboCasa365 and RoboMimic suites. Table 9 lists the **OGPO** hyperparameters that are held fixed across all five **OGPO**-backed tasks; the per-task differences are then enumerated below.

Table 9: **OGPO** hyperparameters held fixed across *all* five **OGPO**-backed tasks (PRESOAKPAN, LOADDISHWASHER, ARRANGETEA, TRANSPORT, TOOL-HANG)

Hyperparameter	Value
h (action chunk length)	8
`q_warmup_steps`	20,000
`flow_steps`	10
`best_of_n`	8
critic target EMA τ	0.05

The remaining hyper parameters vary per task. For the ROBOCASA365 suite of tasks (PRESOAKPAN, LOADDISHWASHER, ARRANGETEA), we use the bc data obtained from the official VLA checkpoint [NVIDIA et al., 2025]. Since we deal with state-based environments and ROBOCASA365 environments change the layout/scenes/fixtures for every environment reset, we fix them across resets to avoid partial observability in state-based settings. For ROBOCASA365 environments, we set `bc_coeff` to 1 and `offline_steps` to $500,000$. We use around 150 demos collected from the VLA checkpoint. On ROBOMIMIC, we do two sets of experiments: (1) with the full set of the offline demos present (2) reducing the offline demonstration set to a fraction of the original (the `dataset.proportion` parameter): only this fraction of the offline trajectories is used for BC pretraining. The full list of our hyper parameter settings for ROBOMIMIC experiments on **OGPO** is present in Table 10.

Table 10: Low-data BC ablation on ROBOMIMIC. Both columns use the **OGPO** backbone with VADAR-style critic disaggregation. `dataset.proportion` controls the fraction of offline demonstrations used for BC pretraining and offline replay; all other hyperparameters remain the same.

	TRANSPORT (low)	TRANSPORT (full)	TOOL-HANG (low)	TOOL-HANG (full)
`dataset.proportion`	0.1	1.0	0.30	1.0
offline_steps	300K	300K	300K	300K
online_steps	3M	3M	3M	3M
q_warmup_steps	20K	20K	20K	20K
start_training	40K	40K	10K	10K
$\bar{\tau}_\pi$	0.005	0.005	0.005	0.005
`bc_coeff`	0.1	0.1	0.1	0.1
`clip_bc_threshold`	0.30	0.30	0.30	0.30
success buffer	true	true	true	true
`subsample_bon`	true	true	true	true

F.6.3 FQL per-task hyperparameters

FQL [Park et al., 2025b] is the secondary flow-based backbone we use on ROBOMIMIC TRANSPORT and TOOL-HANG. Table 11 lists the values used. We use the same network architecture and learning rates as **OGPO** (i.e., Table 8); We do behavior cloning using all the available data a bit longer, due to **FQL** being less performative. The detailed list of hyperparameters used for **FQL** are listed in Table 11

G Multi-Objective Optimization on PUSHT-OBSTACLE: Full Details

This appendix gives the full environment, algorithm, and training details for the multi-objective experiment in Section 5.3, together with the reward-shaping ablation that controls for the safety signal's smoothness.

G.1 Environment

PUSHT-OBSTACLE extends the standard `PushT` [Chi et al., 2024] planar block-pushing task with a static circular obstacle of radius 30 px placed at $(320, 220)$ in the 512×512 workspace. The

Table 11: Per-task hyperparameters for the **FQL** backbone on ROBOMIMIC. Values not listed match Table 8.

	TRANSPORT	TOOL-HANG
offline_steps	1M	500K
online_steps	6M	3M
q_warmup_steps	50K	50K
start_training	10K	10K
h (chunk)	8	8
γ	0.999	0.999
$\bar{\tau}_\pi$	0.005	0.005
q_agg	mean	mean
$\|G\|$, structure	7, DAG	4, chain
group_num_samples	32	32
best_of_n, flow_steps	8, 10	8, 10
α_{FQL} (distillation coeff.)	1000	1000
bc_coeff	1.0	1.0
clip_bc_threshold	0.45	0.45

agent (a circular pusher) must move a T-shaped block to a target pose while avoiding the obstacle. We use the keypoint observation variant (`pusht-keypoints-v0`) with a 5D state $(x_{\text{agent}}, y_{\text{agent}}, x_{\text{block}}, y_{\text{block}}, \theta_{\text{block}})$ and a 2D continuous action setting the target position of the PD-controlled pusher. Episodes terminate after 300 steps.

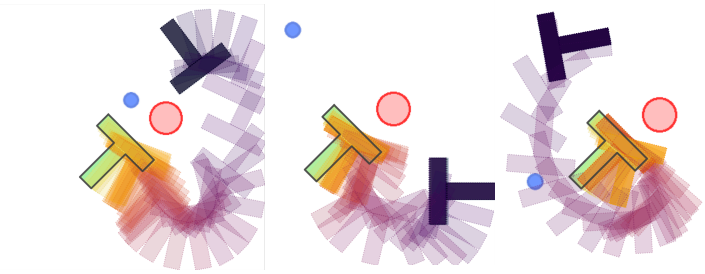


Figure 9: Depiction of our PUSHT-OBSTACLE env. The agent needs to place the T into the T shaped slot (measured by IoU $\geq 95\%$) whilst avoiding the red shaped obstacle placed in the vicinity of the final position.

Each environment step yields two scalar signals:

- *Task signal*: a sparse coverage indicator $r_t^{\text{task}} \in \{-1, +5\}$, equal to $+5$ when the block-goal IoU exceeds 95% and -1 otherwise.
- *Safety signal*: a continuous soft-margin proximity cost $c_t \in [0, 1]$ that takes value 1 if either the pusher or the block penetrates the obstacle, decays linearly to 0 across a safety margin of 10 px around the obstacle boundary, and is 0 outside the margin. The corresponding per-step safety reward used by the critic is $r_t^{\text{safety}} = -5\, c_t$.

G.2 VADAR Instantiation

We provide the full pseudocode in Algorithm 3. We instantiate VADAR on top of a **SAC** backbone with $G = 2$: head 0 tracks task completion and head 1 tracks the safety cost. The vector critic $\mathbf{Q}_\phi : S \times A \to \mathbb{R}^2$ is a 4-layer MLP of width 512 with layer norm, ensembled over 10 members and aggregated by mean. The actor is a tanh-squashed Gaussian over a 4-step action chunk, also a 4-layer MLP of width 512. The TD targets follow Eq. (4.1): the task head uses the standard reward-clipped VADAR target with cap $c_{\text{clip}} \cdot r_{\text{done}} = 15$, while the safety head uses the backup $-5\, c_t + \gamma^H Q[\text{safety}](s', a')$ with $\gamma = 0.99$ and chunk length $H = 4$. Because neither subtask is a prerequisite for the other, the policy weighting GETWEIGHTING collapses to a constant $\mathbf{w} =$

$(1, \lambda_{\text{safety}})$, yielding the aggregated objective

$$Q_{\text{agg}}(s,a) \;=\; \mathbf{Q}_\phi[\text{task}](s,a) \;+\; \lambda_{\text{safety}}\, \mathbf{Q}_\phi[\text{safety}](s,a),$$

with $\lambda_{\text{safety}} = 1.0$ in our experiments. (We also do a sweep over the values of λ_{safety} Appendix G.5). Apart from the vector-valued critic and this aggregation, the agent is parameterized as a standard Squashed Gaussian policy with learnable temperature.

Algorithm 3 VADAR for Multi-Objective RL (instantiated on PUSHT-OBSTACLE with $G = 2$).

Parameters: discount γ; learning rate η; safety weight λ_{safety}; task-head TD clip c_{clip}.
Initialize multi-objective critic $\mathbf{Q}_\phi : S \times A \to \mathbb{R}^2$ with heads $\{\text{task}, \text{safety}\}$; policy $\pi_\theta : S \to \Delta(A)$; replay buffer $\mathcal{B} \leftarrow \emptyset$.
for episode = 1, 2, … **do**
  while episode not done **do**
    $a_t \leftarrow \pi_\theta(s_t)$, transition $s_{t+1} \sim P(s_t, a_t)$
    Read both signals: task indicator $\texttt{done}[\text{task}]_t \in \{0,1\}$ and continuous proximity cost $c_t \in [0,1]$.
    Form reward vector $\mathbf{r}_t = (r_{\text{sparse}}(\texttt{done}[\text{task}]_t),\ -5\, c_t)$.
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s_t, a_t, s_{t+1}, \mathbf{r}_t, \texttt{done}[\text{task}]_t)\}$
    for number of updates **do**
      Sample $(s, a, s', \mathbf{r}, \texttt{done}[\text{task}]) \sim \mathcal{B}$, $a' \sim \pi_\theta(s')$
      $y^{\text{target}}[\text{task}] = \min\{\mathbf{r}[\text{task}] + \gamma\, \mathbf{Q}_\phi[\text{task}](s', a'),\ c_{\text{clip}} \cdot r_{\text{done}}\}$ ▷ reward clipping
      $y^{\text{target}}[\text{safety}] = \mathbf{r}[\text{safety}] + \gamma\, \mathbf{Q}_\phi[\text{safety}](s', a')$
      $\phi \leftarrow \phi - \eta \nabla_\phi \sum_{g \in \{\text{task},\text{safety}\}} L_{\text{critic}}(\mathbf{Q}_\phi[g] \mid s, a, y^{\text{target},g})$
      $Q_{\text{agg}}(\cdot,\cdot) \leftarrow \mathbf{Q}_\phi[\text{task}](\cdot,\cdot) + \lambda_{\text{safety}}\, \mathbf{Q}_\phi[\text{safety}](\cdot,\cdot)$ ▷ constant weighting
      $\theta \leftarrow \theta - \eta \nabla_\theta L_{\text{policy}}(\pi_\theta \mid s, a, Q_{\text{agg}})$

G.3 Baselines

To isolate the contribution of value disaggregation from the contribution of the underlying reward terms, we compare against a dense-reward baseline that shares VADAR's base algorithm (**SAC**), critic ensemble, network sizes, BC pretraining, and online step budget, and that consumes *the exact same per-step task and safety signals* but combines them into a single scalar reward fitted by a scalar critic. We call this AGGREGATED-**SAC** (Algorithm 4), and it combines the task and safety signals term-for-term as $r_t = r_t^{\text{task}} + \lambda_{\text{safety}}\, r_t^{\text{safety}}$, with the same $\lambda_{\text{safety}} = 1.0$. This is the most direct baseline: it differs from VADAR only in collapsing the two signals before the Bellman backup rather than after.

Algorithm 4 Dense-reward shaping baseline for PUSHT-OBSTACLE (AGGREGATED-**SAC**).

Parameters: discount γ; learning rate η; safety weight λ_{safety}.
Initialize *scalar* critic $Q_\phi : S \times A \to \mathbb{R}$; policy $\pi_\theta : S \to \Delta(A)$; replay buffer $\mathcal{B} \leftarrow \emptyset$.
for episode = 1, 2, … **do**
  while episode not done **do**
    $a_t \leftarrow \pi_\theta(s_t)$, transition $s_{t+1} \sim P(s_t, a_t)$
    Read both signals: task indicator $\texttt{done}_t^{\text{task}} \in \{0,1\}$ and continuous proximity cost $c_t \in [0,1]$
    Combine before the Bellman backup into a single scalar reward:
    $r_t \leftarrow r_{\text{sparse}}(\texttt{done}_t^{\text{task}}) + \lambda_{\text{safety}}\, (-5\, c_t)$ ▷ shared signals with Algorithm 3; same λ_{safety}
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s_t, a_t, s_{t+1}, r_t)\}$
    for number of updates **do**
      Sample $(s, a, s', r) \sim \mathcal{B}$, $a' \sim \pi_\theta(s')$
      $y^{\text{target}} = r + \gamma\, Q_\phi(s', a')$ ▷ single scalar TD target
      $\phi \leftarrow \phi - \eta \nabla_\phi L_{\text{critic}}(Q_\phi \mid s, a, y^{\text{target}})$
      $\theta \leftarrow \theta - \eta \nabla_\theta L_{\text{policy}}(\pi_\theta \mid s, a, Q_\phi)$

G.4 Training and Evaluation Protocol

All methods are pre-trained for 150,000 behavior-cloning steps on a fixed dataset of transitions collected from a previously trained **SAC** policy on the obstacle-free task, and then fine-tuned online for 800,000 environment steps. We measure the success rate in terms of *success*: block-goal coverage $\geq 95\%$ *and* no collision with the obstacle occurs the episode.

G.5 Parameter Sweep for λ_{safety}

We sweep over values of $\lambda_{\text{safety}} \in \{0.1, 1, 5, 10\}$ (Figure 10) and observe two trends. First, VADAR is far less sensitive to the choice of λ_{safety}: it reaches comparable final success rates at $\lambda = 0.1$ and $\lambda = 1$, degrades only moderately at $\lambda = 5$, and still makes meaningful progress at $\lambda = 10$. AGGREGATED-**SAC** matches VADAR at small λ but collapses to near-zero success once $\lambda \geq 5$, since the safety penalty starts dominating the aggregate reward. Second, at any fixed λ_{safety}, VADAR trains more stably than AGGREGATED-**SAC**, with narrower seed bands and fewer fluctuations during training. We conjecture that this is due to ease of composition at the Q-function level: each VADAR head fits a λ-independent target, while AGGREGATED-**SAC** absorbs λ into the target itself.

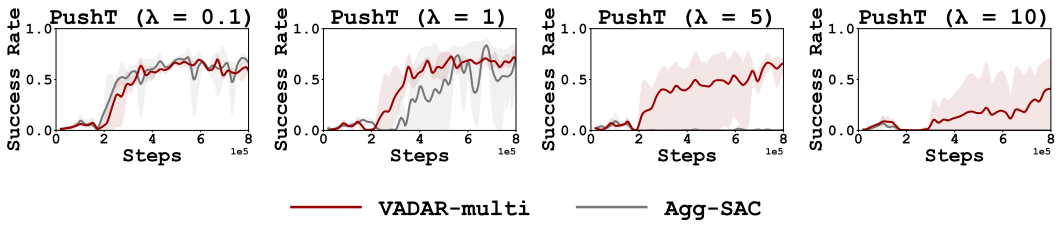


Figure 10: Plots of success rate for different values of λ_{safety} comparing both VADAR and AGGREGATED-**SAC** on our PUSHT-OBSTACLE environment. Across all the values of λ_{safety} that VADAR is more performative and stable than AGGREGATED-**SAC**

H Fixed-Weight Value Aggregation and Reward Shaping

For fixed weights and true per-goal Q-functions under a common policy, value aggregation is equivalent to evaluating an aggregated reward. Consider family of per-goal MDPs $M_g = (S, A, P_0, P, R_g, \gamma)$ which share common dynamics and discount factor, but differ in their reward R_g. For example, in the goal-conditioned setting described in Section 3.1, choosing $R_g(s, a) \leftarrow r_{\text{sparse}}(\texttt{done}_g(s))$ yields that M_g is the MDP where rewards are associated with subgoal g. In principle, however, the theory accomodates any choices of rewards. Finally, we let Q^{π, M_g} denote the true Q-function (as described in the preliminaries) associated with π under MDP M_g.

We now show that the Q-function achieved by aggregating per-goal Q-functions is equivalent to the Q-function of a policy in an MDP whose reward aggregates per-goal rewards:

Proposition H.1. *Fix a weighting vector $\mathbf{w} \in \mathbb{R}^G$. Then, for any goal-unconditioned policy $\pi : S \to \Delta(A)$, and any s, a, we have the equality*

$$Q^{\pi}_{\text{agg}}(s, a) := \sum_g \mathbf{w}[g] Q^{\pi, M_g}(s, a) = \mathbb{E}^{\pi}\left[\sum_{t \geq 0} \gamma^t R_{\mathbf{w}}(s_t, a_t) \mid (s_0, a_0) = (s, a)\right] = Q^{\pi, M_{\mathbf{w}}}(s, a), \tag{H.1}$$

where above $R_{\mathbf{w}} := \sum_g \mathbf{w}[g] R_g$ and $M_{\mathbf{w}}$ is the MDP defined by $(S, A, P_0, P, R_{\mathbf{w}}, \gamma)$

Proof. Observe that for any MDP M_g with reward R_g, we have

$$Q^{\pi, M_g}(s, a) = \mathbb{E}^{\pi}\left[\sum_{t \geq 0} \gamma^t R_g(s_t, a_t) \mid (s_0, a_0) = (s, a)\right],$$

where $\mathbb{E}^{\pi}$ is expectation under the dynamics of the MDP and policy π. As the dynamics are shared between all per-goal MDPs, we may leverage linearity of expectation to commute the expectation with the weighted summation. □

This identity applies to fixed weights and exact Q-functions. It does not establish equivalence between VADAR's state-dependent weighting and clipped, approximate critic updates and learning a scalar critic on an aggregated reward. The experiments in Section 5.2 and Appendix H.1 distinguish reward aggregation from the benefits of dense critic supervision.

Reward shaping does *not* explain Vadar's success

It might be tempting to attribute VADAR's success to reward shaping. However, in Section 5.2, we find that even using the *final subgoal critic* can lead to strong performance in many benchmarks. We conclude that VADAR's benefits come as much from representation learning induced by denser critic supervision as they do from explicit reward shaping during policy extraction.

H.1 Subgoal Reward Shaping Does *Not* Reliably Enable Long-Horizon Learning

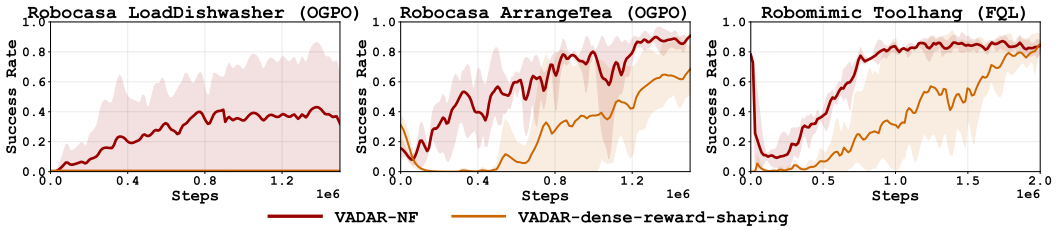


Figure 11: Subgoal reward shaping ablation. VADAR-NF against VADAR-SHAPED, which sums the per-subgoal rewards into a single shaped reward before the Bellman backup and trains a standard scalar critic on it, on ROBOCASA365 with **OGPO** and ROBOMIMIC TOOL-HANG with **FQL**.

Appendix H establishes that, for fixed weights and true Q-functions under a common policy, a weighted sum of subgoal values equals the value of the correspondingly weighted reward. This identity does not guarantee equivalent learning with state-dependent weights or clipped, approximate critics. This suggests a natural alternative to VADAR: aggregate the per-subgoal rewards before the Bellman backup, and train a standard scalar critic and policy on the resulting shaped reward. We call this variant VADAR-SHAPED.

Figure 11 shows that this equivalence does not translate to learning in practice. VADAR-SHAPED fails to learn on the harder ROBOCASA365 tasks and is substantially less sample efficient than VADAR-NF elsewhere. Since VADAR-SHAPED combines the subgoal rewards before the Bellman update, the critic is trained on a single scalar target rather than learning each subgoal value separately. Together with the VADARLAST-Q results, this suggests that VADAR's gains come from multi-resolution temporal supervision, rather than from reward shaping alone.

I Additional Baseline Comparisons

I.1 Hindsight experience replay (HER) struggles with contact-rich control

An alternative approach to mitigating the credit assignment problem in goal-conditioned RL is hindsight experience replay (HER) [Andrychowicz et al., 2018], which relabels states reached during a rollout as goals, so that the policy receives learning signal even when the task reward is never observed. HER has largely been evaluated in reaching, pick-and-place and maze settings, where any state is a reasonable goal. To test whether it applies to contact-rich manipulation settings, we evaluate HER with **SAC** on LIBERO-Task 6, with goals given by object states and relabelled from future states of the same episode, and with the same BC pretraining, online step budget and networks as our other **SAC** baselines. We also steelman HER with two variants that provide it with the same subgoal information that HGC receives: an oracle that supplies a representative goal state for each subgoal in turn, and a learned goal embedding trained to recognize subgoal completions. In both the variants, we find HER to be incapable of performing multi stage complex manipulation tasks, with the eventual success rate remaining quite close to the initial BC success rate.

J Learning with VLM-Provided Subgoal Labels

J.1 Goal-conditioned policies are computationally prohibitive with frontier model feedback

For our main experiments, the subgoal completion indicators $\texttt{done}_g$ are evaluated using privileged simulator state. In real-world settings, these intermediate subgoal labels may instead be obtained

from a VLM given visual observations and natural-language descriptions of the subgoals [Du et al., 2023].

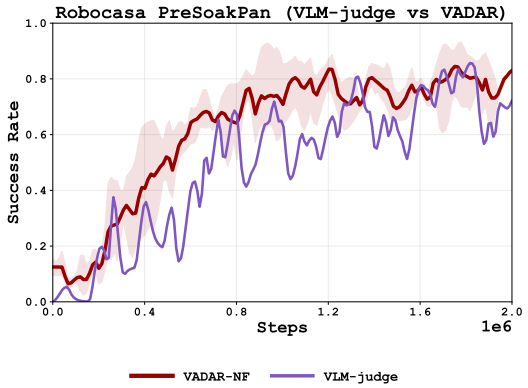


Figure 12: VADAR-NF on ROBOCASA365 PRESOAKPAN with the **OGPO** base algorithm, trained with oracle subgoal completions vs. with completions judged by a VLM (`gemini-3.6-flash`).

We test whether VADAR can learn when these intermediate training signals are provided by a VLM rather than by the simulator. On ROBOCASA365 PRESOAKPAN, we replace the oracle subgoal completion indicators with predictions from a VLM judge. The environment provides only the final task-success signal, while all intermediate subgoal supervision used by VADAR comes from the VLM. The policy and critic remain state-based, and the VLM is used only to generate training-time subgoal labels. Full details and the judge prompt are provided in Appendix K.

Figure 12 compares VADAR-NF trained with oracle and VLM-predicted subgoal completions. VLM supervision is somewhat less sample efficient early in training, but reaches a similar final success rate. These results show that VLMs can serve as training-time evaluators for the subgoal rewards used by VADAR, without introducing any VLM dependence at deployment time, while HGC needs such subgoal completion information during inference time, adding to the cost and latency of querying VLMs during deployment.

K VLM Judge Prompt

Protocol. We use `gemini-3.6-flash` as the VLM judge for the experiment in Appendix J.1. For every 8-step action chunk, the judge receives the frames at step offsets $[0, 2, 5, 7]$ from all three cameras, together with the prompt below, and returns a JSON list of $|G|$ booleans, which we use as the subgoal completion vector $\mathbf{d}_t$ for the final step of the chunk. The prompt used for ROBOCASA365 PRESOAKPAN is shown below.

VLM judge prompt for RoboCasa365 PreSoakPan

```
You are a strict robot-manipulation subgoal evaluator.

Task: PreSoakPan
The input is one 8-step action chunk. It contains 4 chronological timestamps at
step offsets [0, 2, 5, 7]. Every timestamp has exactly three synchronized
camera views. Timestamps are ordered oldest to newest.

Judge whether each subgoal's completion criterion is satisfied at the FINAL
timestamp. Use the earlier timestamps as visual history to disambiguate
grasping, motion, placement, and occlusion. A transient condition such as
'gripper is holding' must still be true at the final timestamp. Do not infer
completion only from the expected task order, and do not require criteria that
are not stated below.

Subgoals:
0. pan_grasped_lifted
  Completion criterion: The gripper is holding the pan and the pan is lifted
off the counter.
1. pan_in_sink
  Completion criterion: The pan is resting inside the sink basin.
2. sponge_grasped_lifted
  Completion criterion: The gripper is holding the sponge and the sponge is
lifted up.
3. sponge_in_pan_and_sink
  Completion criterion: The sponge is inside the sink and is placed on or in
the pan.
4. water_on
  Completion criterion: The faucet water is running, the pan and sponge are in
the sink, and the gripper has moved away from both objects.

Return only strict JSON in this form: {"done":[true,false,...]}. The list must
contain exactly 5 booleans in the listed order.
```

L Connection to Successor Features

VADAR's aggregation step (Eq. (4.2)) is structurally similar to a success feature decomposition of a Q-function. Classical success features assume reward is linear in a feature map $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^{|G|}$ [Dayan, 1993, Barreto et al., 2017], $r(s,a) = \langle \mathbf{w}, \boldsymbol{\phi}(s,a) \rangle$ for some fixed $\mathbf{w} \in \mathbb{R}^{|G|}$. The Q-function then inherits the same decomposition $Q^\pi(s,a) = \langle \mathbf{w}, \boldsymbol{\psi}^\pi(s,a) \rangle$, where $\boldsymbol{\psi}$ is the successor feature, which represents the expected discounted sum of returns under policy π. Crucially as motivated by [Barreto et al., 2017], this allows efficient policy extraction from arbitrary reward weightings without retraining policies across tasks.

VADAR instantiates the reward features $\boldsymbol{\phi}(s,a)$ as per-subgoal rewards $\phi_g(s,a) = r_g(s,a)$ so the successor features are precisely the per-subgoal Q-values $\mathbf{Q}_{\phi_g}(s,a) = \boldsymbol{\psi}_g^\pi(s,a)$. The weight vector $\mathbf{w}(s) = \textsc{GetWeighting}(\mathbf{d}(s))$ is a state-dependent current subgoal completion vector. This allows VADAR to recover sufficient statistics for evaluating the entire ancestral tree of weighted objectives in a single multi-step RL training run with $\mathbf{w}(s)$ as the contraction operator.

M Multi-Goal Critics as an Implicit Bellman Backup

We expand the claim in Section 5.2 that multi-goal critic training simulates a Bellman backup in the network representations.

The classical Q-function satisfies the Bellman recursion

$$Q^\pi(s,a) = r(s,a) + \gamma\, \mathbb{E}_{s' \sim P, a' \sim \pi}\big[Q^\pi(s',a')\big], \tag{M.1}$$

and each TD update propagates information from $Q_\phi(s',a')$ one step back into $Q_\phi(s,a)$. Under a sparse final-task reward, the terminal reward must traverse the full rollout horizon T via $O(T)$ TD updates to reach the initial state.

VADAR's vector-valued critic predicts $|G|$ Q-values per (s,a) pair, each supervised by a subgoal reward r_g. Let τ_g be the timestep at which subgoal g is achieved. Head $\mathbf{Q}_\phi[g]$ receives a non-zero signal at τ_g, shrinking the Bellman backup from T to the maximum temporal gap between

consecutive subgoal completions given as $\frac{T}{|G|}$, a factor of $|G|$ reduction in the effective horizon. While HGC also benefits from this reduction in the effective temporal horizon, a lack of shared backbone prevents the learned values from the latter stage Q-values to affect the former.

VADAR's shared backbone design principle reuses the features learned by the early heads $\mathbf{Q}_\phi[g_1], \ldots \mathbf{Q}_\phi[g_{|G|-1}]$ when computing $\mathbf{Q}_\phi[g_{|G|}]$ in a single feedforward pass. Hence, the terminal head collates the the representations learned via the earlier supervision signals. In effect, the vector-valued critic architecture allows the overall network to compose Bellman-backup-like message passing through its heads alleviating the need for repeated TD updates over T. We see an empirical evidence of this phenomenon from the VADARLAST-Q experiment shown in Figure 5.

Appendix N presents an alternative view of this, where the Bellman-like computation is performed across the integration steps of an iterative critic.

N Combining VADAR with Iterative Critics

N.1 Simulating the Bellman Update via Iterative Critic Parametrizations

Standard RL critics compute Q-values in a single monolithic forward pass. In the context of VADAR, a monolithic critic produces all $|G|$ heads in a single shot, so the subgoal heads never communicate at inference time. Here we argue that combining iterative critic parameterizations with VADAR can be viewed as implementing a Bellman update at inference time.

An iterative VADAR critic proceeds in steps $j = 0, \ldots, N_{\text{iter}}$, producing an estimates $\hat{\mathbf{q}}^j \in \mathbb{R}^G$ of the critic values per step. At each step, the critic updates its estimate $\hat{\mathbf{q}}^{j+1} \leftarrow \mathbf{Q}_\phi(\mathbf{s}, \mathbf{a}, \hat{\mathbf{q}}^j, j)$. Therefore, each updated $\mathbf{q}^{k+1}$ can depend on previous estimates of critic values $\mathbf{q}^k$. Informally, this computation graph can express belief-propogation updates across critic time steps, such as the classical Bellman update [Sutton et al., 1998] which defines the Q-values. We depict this intuition in Figure 13. For training an iterative critic, we introduce a two-step minimal iterative Q-network (MIQ) inspired by Pan et al. [2026, MIP] described in Appendix N.2, which we anecdotally find more stable to train than flow-based critics [Agrawalla et al., 2025].

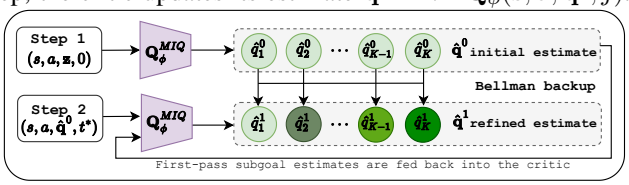


Figure 13: Iterative Critic Prediction

N.2 The MIQ Critic

VADAR's vector critic synergizes with iterative critic parameterizations: each iteration can propagate information across subgoal heads, so the second iteration's prediction for one subgoal can be conditioned on the first iteration's prediction for the others. We instantiate this with MIQ, a critic-side adaptation of the Minimal Iterative Policy of Pan et al. [2026] that retains the two-step iteration with $t^\star = 0.9$.

N.2.1 Architecture

A MIQ critic is a single MLP $Q_\phi : S \times A \times \mathbb{R} \times [0, 1] \to \mathbb{R}$ that takes a state-action pair (s, a), a scalar z that plays the role of the noise variable in a flow-matching linear interpolant, and a time index $t \in [0, 1]$. To recall the flow-matching analogy: a flow model is trained to map the linear interpolant $I_t = t\,x + (1 - t)\,z$ between Gaussian noise z and a clean target x at time $t \in [0, 1]$ back to x, so that at $t = 0$ the input is pure noise and as $t \to 1$ it approaches the target. MIP [Pan et al., 2026] retains this interpolant but supervises the network with the same target twice, at $t = 0$ and at $t = t^\star$ (with $t^\star = 0.9$), using a regression loss instead of a velocity-matching loss. MIQ is the same construction with the supervision target x instantiated as the scalar Q-target y^{target}: the flow-matching interpolant is replaced by an interpolant in Q-value space, and $z \sim \text{Unif}[-\sigma, \sigma]$ is the noise variable that lives on the noisy end of that interpolant. At inference, MIQ runs two autoregressive steps:

$$\hat{q}^0 \leftarrow Q_\phi(s, a, z, t = 0), \qquad \hat{q}^1 \leftarrow Q_\phi(s, a, \hat{q}^0, \sigma/10), t^\star), \tag{N.1}$$

i.e., the second iteration substitutes its own iteration-zero output $\hat{q}^0$ in place of an interpolated input, since $\hat{q}^0$ has been trained to approximate y^{target}.

N.2.2 Training Objective

Following Pan et al. [2026, Eq. (4.4)], MIQ supervises both iterations against the standard scalar TD target $y^{\text{target}} = r + \gamma\, Q_{\text{targ}}(s', a')$. To train the iteration-one mapping, we cannot feed the network the iteration-zero output (it is a moving target during training); instead, we feed a noise-corrupted version of the supervision signal itself. Concretely, we sample $z \sim \text{Unif}[-\sigma, \sigma]$ and form the linear interpolant

$$z_{t^\star} \;=\; (1 - t^\star)\, z \;+\; y^{\text{target}}, \tag{N.2}$$

which is a noisy copy of y^{target} (smaller noise the larger $t^\star$ is, hence $t^\star$ near 1). The MIQ critic loss for a single ensemble member is

$$L^{\text{MIQ}}_{\text{critic}}(Q_\phi \mid s, a, y^{\text{target}}, z) \;=\; \big(Q_\phi(s, a,\, z,\, 0) - y^{\text{target}}\big)^2 \;+\; \big(Q_\phi\big(s, a,\, z_{t^\star},\, t^\star\big) - y^{\text{target}}\big)^2. \tag{N.3}$$

The first term trains the iteration-zero pathway to predict y^{target} from a fully random z; the second term trains the iteration-one pathway to denoise a near-correct estimate $z_{t^\star}$ back to y^{target}. At inference we substitute the iteration-zero output $\hat{q}^0$ for $z_{t^\star}$, which is well-behaved because the iteration-zero pathway has already been trained to produce a good approximation of y^{target}.

N.2.3 Ensembling

We use an explicit ensemble of M independently parameterized MIQ networks $Q_{\phi_1}, \ldots, Q_{\phi_M}$ (`ValueMIPEnsemble` in our codebase). At each update, a single noise draw z is shared across the ensemble; each member produces its own $\hat{q}^{0,(i)}$ at iteration zero and uses that member's own $\hat{q}^{0,(i)}$ as the iteration-one input. Aggregation across the ensemble (Eq. (B.2)) is unchanged from Appendix B: we average the iteration-one predictions to form Q_{targ}.

N.2.4 MIQ with VADAR

We have seen how MIQ can be used as a general critic. Adapting it to our VADAR framework requires minimal modifications. The scalar $Q_\phi : S \times A \times \mathbb{R} \times [0,1] \to \mathbb{R}$ now becomes a vector critic: $\mathbf{Q}_\phi : S \times A \times \mathbb{R}^{|G|} \times [0,1] \to \mathbb{R}^{|G|}$. The noise variable $\mathbf{z} \sim \text{Unif}[-\sigma, \sigma]^{|G|}$ also becomes a $|G|$ dimensional vector. And the TD target is the vector $\mathbf{y}^{\text{target}} \in \mathbb{R}^{|G|}$ from Eq. (4.1), and Eq. (N.1) becomes

$$\hat{\mathbf{q}}^0 \;\leftarrow\; \mathbf{Q}_\phi(s, a,\, \mathbf{z},\, 0), \qquad \hat{\mathbf{q}}^1 \;\leftarrow\; \mathbf{Q}_\phi(s, a,\, \hat{\mathbf{q}}^0,\, t^\star).$$

Because the second pass' input is the full vector $\hat{\mathbf{q}}^0$, head g's second estimate depends on every head's previous estimate.

N.2.5 Policy Extraction via MIQ + VADAR

To validate the hypothesis that iterative critic parameterizations learn implicit Bellman updates over the subgoal DAG, we test MIQ+VADAR on LIBERO ($|G| = 3$) with **SAC** and ROBOMIMIC TRANSPORT ($|G| = 7$) with **OGPO** as the backbone. We compare the VADAR's monolithic multi-head critic against MIQ with varying Q-function update-to-data (UTD) ratios.

Results. Across the four panels in Figure 15, success rates with MIQ are broadly comparable to the monolithic critic, with a minor improvement in policy extraction for the higher critic UTD ($q = 4$) ablation. The more informative differences show up on the critic itself: in Appendix N.2 (Figure 14), per-subgoal TD residuals on LIBERO `KITCHEN_SCENE6` drop under MIQ across all three subtasks and stay flat throughout training, while the monolithic critic's TD error drifts upward late in training. The mean-Q trajectory under MIQ is more stable as hypothesized due to the implicit message passing down the subgoal DAG. We posit this being more beneficial to use-cases where critic stability even at subgoal level is a large concern.

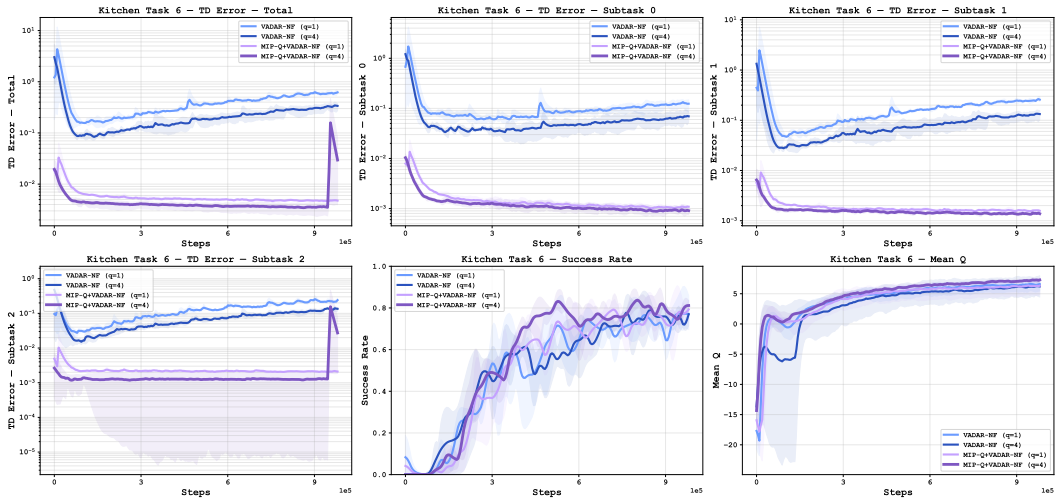


Figure 14: We compare monolithic VADAR critics with MIQ+VADAR at critic UTD $q \in \{1, 4\}$, holding the actor update frequency fixed. MIQ substantially reduces total and per-subgoal TD residuals and produces lower-variance mean-Q estimates, while maintaining comparable or slightly improved success rates.

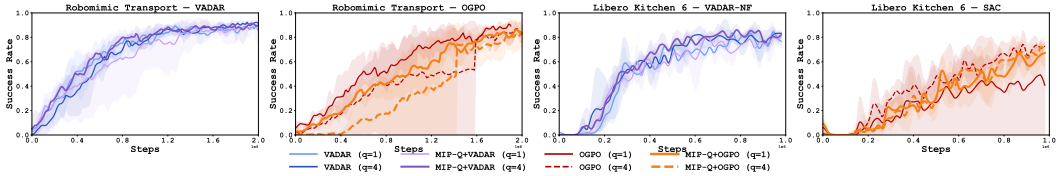


Figure 15: MIQ+VADAR (orange/purple) vs. monolithic vector critics (red/blue) at critic UTD $q = 1$ (solid) and $q = 4$ (dashed); the policy update frequency is held fixed across all runs. The largest MIQ effect is on **OGPO** TRANSPORT, where the monolithic critic degrades sharply at $q = 4$ and MIQ restores parity with $q = 1$.

This view connects directly to the VADAR-Last-Q ablation of Section 5.2, where policy extraction used only the terminal-subgoal Q-value yet matched VADAR-NF. A single forward pass through the shared multi-goal backbone implicitly couples each head's prediction to features computed for the others, so the terminal-subgoal head already inherits Bellman-like information from upstream subgoals through layer-wise message passing. MIQ makes this coupling external and recurrent — the second pass conditions every head on the full first-pass vector $\hat{\mathbf{q}}^0$, turning the implicit, depth-wise iterative computation of the monolithic critic into an explicit, two-step autoregressive refinement. The two designs therefore sit on a spectrum of how iterative computation is realized in the critic: the multi-goal backbone performs implicit message passing across its layers in a single forward pass, while MIQ adds an outer loop over the same network resulting in more stable updates while achieving similar performance to non-iterative critics.

O Experiments Compute Resources

Hardware. All training and evaluation runs were executed on an HPC cluster. Each job was allocated the equivalent of a single **NVIDIA L40S GPU (48 GB VRAM)**, 8 CPU cores, and 128 GB of system RAM. No multi-GPU or distributed training was used; every result reported in the main paper and ablations comes from a single-GPU run. Datasets, replay buffers, and checkpoints were stored on the cluster's shared scratch filesystem. Each SLURM job was subject to a 48-hour wall-clock limit. We did not use any cloud-provider compute.

Per-run compute. Per-run resource usage as logged by W&B is summarized in Table 12. CPU utilization averaged 1–3 fully utilized cores and process memory never exceeded 48 GB across any run. ROBOMIMIC and ROBOCASA365 runs frequently reached the cluster's 48-hour wall-clock cap; OGBENCH and LIBERO runs finished well inside it.

Table 12: Per-run peak GPU memory and wall-clock duration for our runs

Benchmark / environment	Peak GPU memory	Wall-clock per run
ROBOMIMIC TRANSPORT (low / full data)	4.8–6.9 GB	33.7–47.8 h
ROBOMIMIC TOOL-HANG (low / full data)	2.7–7.0 GB	25.2–47.8 h
ROBOCASA365	2.7–4.6 GB	36.9–47.8 h
OGBENCH `scene-task4`	1.6 GB	0.7 h
LIBERO `kitchen-scene-task{3,4,6}`	1.4–1.6 GB	5.2–7.7 h