Investigating the Value of Metadata-Conditioning in VLA Pretraining

Nicolas Espinosa-Dice
Cornell University
ne229@cornell.edu

Anthony Liang
Toyota Research Institute
anthony.liang@tri.global

Paarth Shah
Toyota Research Institute
paarth.shah@tri.global

Gokul Swamy
Carnegie Mellon University
gswamy@cmu.edu

Abhishek Gupta
University of Washington
abhgupta@cs.washington.edu

Abstract: Vision-language-action (VLA) models are typically pretrained on high-quality teleoperation data, which is limited in both volume and coverage. Non-expert data is far more plentiful, but standard behavioral cloning (BC) degrades when trained on it, so this data is usually filtered out before pretraining. We investigate metadata-conditioning as an alternative: labeling each trajectory with cheap metadata, such as its outcome or speed, and conditioning the policy on these labels so that the data is kept and the choice of what to imitate is made at inference time. In a controlled in-domain study, we show that metadata-conditioned pretraining benefits from added non-expert data because it enables *inference-time mode selection*: the ability to imitate some modes in the data, such as high-quality trajectories, while avoiding others, such as mistakes. We perform a careful ablation to show that metadata-conditioned pretraining's in-distribution performance gain cannot be attributed to the addition of in-domain failure data. With multi-source and cross-domain data, quality-based filtering is no longer sufficient: metadata-conditioning makes better use of out-of-domain non-expert data than both BC and filtering by selecting high-quality, possibly cross-domain, modes at inference time. Together, these results position metadata-conditioning as a simple way to pretrain VLAs on all available data without committing to a fixed data curation decision at train-time.

Keywords: VLA, metadata-conditioning

1 Introduction

Vision-language-action (VLA) models are typically pre-trained on high-quality teleoperation data, which is expensive to collect and thus limited in volume. Teleoperation data is also generally performed near-optimally and with relatively consistent behavior, limiting its coverage. Yet, the VLA training pipeline itself produces additional data beyond teleoperation: supervised fine-tuning, online interactive learning, planner-generated trajectories, and evaluations all leave behind diverse sources of non-expert, sub-optimal data.

Most approaches to pretraining VLAs on non-expert data filter it in some way, either dropping it entirely, dropping the failures, or keeping everything. However, filtering has two primary downsides. It wastes limited robot data, and it fixes a data curation decision at training time, even though the best curation strategy is rarely known in advance and instead depends on the test-time environment and objective. Changing the filtering strategy requires re-running the full pretraining job, raising the natural quesiton of whether how to pretrain on non-expert data without committing to a dataest curation decision at train-time.

In this paper, we study an alternative way to incorporate non-expert data in VLA pretraining, metadata-conditioning, which labels every episode with readily available episode-level metadata, such as its return, speed, and domain, and conditions the policy on these labels by appending them to the VLA's language input. Training requires no filtering or curation: the policy trains on all data, and selecting a metadata value at inference time selects which behavior in the data to imitate. Metadata-conditioning thus removes both downsides of filtering: it wastes no data, and it strictly generalizes filtering, since the VLA learns a family of policies in a single pretraining run. Metadata-conditioning has recently been applied to large-scale industrial VLA pretraining, where it improved zeroshot performance over an unconditioned baseline [1], but the source of its improvement remains unclear. For instance, does metadata-conditioned pretraining benefit from the inclusion of *failure* (i.e., low-quality) data, or does its benefit dissipate when properly controlled for data quality?

In this paper, we investigate the value of metadata-conditioning for pretraining VLAs on non-expert data. Our contributions are as follows.

(1) Metadata-conditioned pretraining enables inference-time mode selection. In a controlled experiment with in-domain data and in-distribution evaluations, we show that metadata-conditioned pretraining benefits from additional non-expert data because metadata-conditioning enables *inference-time mode selection*: the policy can imitate some modes in the data, such as high-quality trajectories, while avoiding others, such as mistakes. When controlled for data quality, however, filtering-based pretraining recovers this improvement without the failure data, indicating that **the added failure data does not account for metadata-conditioning's in-domain benefit**.

(2) Metadata-conditioned pretraining makes better use of cross-domain data. With cross-domain pretraining data, we show that metadata-conditioning makes better use of out-of-domain data than standard BC and outperforms filtering-based approaches such as `FilteredBC`, which filters on data quality. Our ablations attribute this benefit to selecting high-quality, possibly cross-domain, modes at inference time rather than to the additional low-quality in-domain data.

2 Related Work

Conditioned Policies. Metadata-conditioning has appeared in several forms across imitation learning and reinforcement learning, most commonly as return-conditioned policies, including upside-down reinforcement learning [2, 3, 4], decision transformers and other sequence models [5, 6], and return- or success-conditioned behavioral cloning [7, 8]. Because policies already condition on state, metadata-conditioning can be viewed as augmenting the state with additional information. Goal-conditioned reinforcement learning [9, 10, 11, 12] and hindsight relabeling [13, 14, 15] can likewise be viewed as forms of metadata-conditioning, where the conditioning variable is a goal reached in hindsight rather than an episode-level label. Recently, metadata-conditioning has been applied to large-scale industrial VLA pretraining, where conditioning on data source and quality improved zeroshot performance [1]. These works treat conditioning as a policy-learning algorithm or report its effect in aggregate. We instead study metadata-conditioning as a recipe for pretraining VLAs on mixed-quality, multi-domain data, and ask which property of the data, and which capability of the conditioned policy, the benefit comes from.

Learning From Mixed-Quality Data. Using non-expert data without being pulled toward its mistakes is a long-standing problem. Weighted imitation methods reweight or filter transitions by estimated advantage or return before cloning them [16, 17, 18], inverse RL methods extrapolate beyond suboptimal demonstrations [19], and offline RL learns a value function from the full dataset and extracts a policy that improves on the behavior that generated it [20, 21, 22, 23], including with diffusion and flow policies [24, 25, 26]. At the scale of VLA pretraining, the dominant lever is instead data curation: choosing which datasets to include and how to weight them [27, 28], with large models trained on curated expert mixtures [29, 30, 31, 32]. All of these approaches decide at training time which data to trust and how much. Metadata-conditioning keeps every trajectory and its label, and

defers the decision of what to imitate to inference time, so one pretraining run supports many curation choices.

Inference-Time Steering of Generative Policies. A complementary line improves a fixed generative policy at inference time rather than through its weights, by guiding or re-ranking its samples with a value function [33] or by scaling search over the sampling process [34, 35]. These methods require an external value model or additional samples at every step. Metadata-conditioning provides a form of inference-time steering that needs neither: selecting a metadata value changes which mode of the pretraining data the policy imitates, through the same language input the VLA already consumes.

3 Metadata-Conditioned VLA Pretraining

3.1 Problem Setting

Markov Decision Process. We consider a finite-horizon, partially observable Markov decision process (MDP), $(\mathcal{S}, \mathcal{A}, \mathcal{O}, P, O_\omega, r, \rho_0, T)$, with state space $\mathcal{S}$, action space $\mathcal{A}$, observation space $\mathcal{O}$, transition function $P(s' \mid s, a)$, starting-state distribution μ_0, horizon T, and terminal, binary success reward $r(s_T) \in \{0, 1\}$. The return of an episode $\xi = (\mathbf{o}_0, \mathbf{a}_0, \ldots, \mathbf{o}_T)$ is $R(\xi) \in \{0, 1\}$. Each task is specified by a language instruction ℓ. At each step t, the policy receives an observation $\mathbf{o}_t = \left[\mathbf{I}_t^1, \ldots, \mathbf{I}_t^n\right]$ of n camera images (and optionally proprioception) and outputs an action chunk $\mathbf{A}_t = \mathbf{a}_{t:t+H-1}$, a sequence of H actions.

VLA Policy. The learned policy $\pi_\theta(\mathbf{A}_t \mid \mathbf{o}_t, \ell)$ is a VLA with a VLM backbone f_ϕ and a flow-matching action head v_ψ, with $\theta = (\phi, \psi)$. The backbone encodes the images and the text prompt into a single embedding $\mathbf{z}_t = f_\phi(\mathbf{o}_t, \ell)$, on which the action head conditions.

We train the action head with flow matching [36, 37, 38]. Given a noising step $\tau \in [0, 1]$ and initial noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, we construct a noised action by interpolating linearly between the clean action and the noise, $\mathbf{A}_t^\tau = \tau \mathbf{A}_t + (1 - \tau)\boldsymbol{\epsilon}$. The action head regresses the velocity along this path:

$$\mathcal{L}_{\mathrm{FM}}(\theta;\, \mathbf{o}_t, \ell, \mathbf{A}_t) \;=\; \mathbb{E}_{\tau,\, \boldsymbol{\epsilon}}\Big[\, \| v_\psi(\mathbf{A}_t^\tau, \tau, f_\phi(\mathbf{o}_t, \ell)) - (\mathbf{A}_t - \boldsymbol{\epsilon}) \|_2^2 \Big]. \tag{1}$$

At inference, we sample a chunk by drawing initial noise and integrating the learned velocity field with the forward Euler method.

Pretraining Objective. Given a dataset of episodes $\mathcal{D}$, each consisting of observations, actions, and a task instruction, VLA pretraining minimizes the flow-matching loss over observation and action-chunk samples,

$$\mathcal{L}_{\texttt{BC}}(\theta;\, \mathcal{D}) \;=\; \mathbb{E}_{\xi \sim \mathcal{D},\, t}\left[\, \mathcal{L}_{\mathrm{FM}}(\theta;\, \mathbf{o}_t, \ell, \mathbf{A}_t)\right]. \tag{2}$$

Standard VLA pretraining learns from expert teleoperation demonstrations, which we denote $\mathcal{D}_{\mathrm{E}}$. We additionally consider non-expert data $\mathcal{D}_{\mathrm{S}}$ during pretraining, collected both in simulation and through real-world interaction.

To evaluate performance, we measure a policy's *success rate*, $J(\pi) = \mathbb{E}[R(\xi)]$, on a variety of manipulation tasks, spanning both single-arm and bimanual tasks. We also investigate the post-training potential of pretrained checkpoints by measuring pass@k (see Appendix A).

3.2 Metadata-Conditioned Behavioral Cloning (`MetaBC`)

Metadata. Every episode is tagged with episode-level metadata $m = (m_1, \ldots, m_J)$, where each m_i is a discrete attribute (e.g., episode length, return, data source) that is known or easily computed at training time. Restricting metadata to the episode level is not strictly necessary, but it labels every step of an episode with the same metadata, including the steps that precede the outcome, and it keeps inference-time metadata strategies simple and flexible.

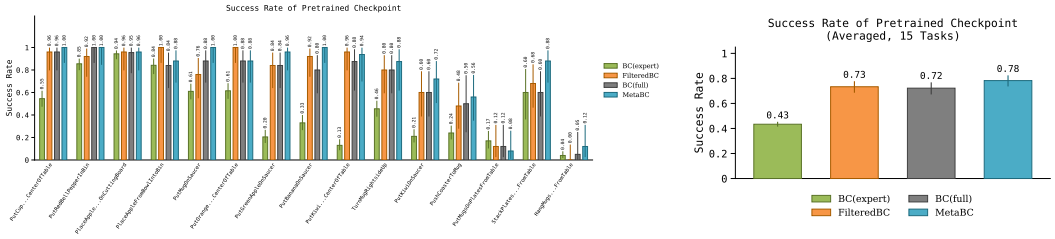


Figure 1: **Pretraining on in-domain data.** We pretrain VLAs on a dataset consisting of expert teleoperation demonstrations and non-expert policy rollouts, containing both successes and failures. Pretraining data is collected in simulation, and evaluation is performed in simulation. While incorporating non-expert data improves the performance of `MetaBC`, pretraining on data with failures filtered approximately recovers the performance gain.

Training. `MetaBC` requires no change to the VLA's architecture or training objective. We convert the metadata labels m to text and append them to the task instruction ℓ, which yields an augmented prompt $\tilde{\ell}(\ell, m)$ (e.g., `"{ℓ} operator=rollout, domain=sim, mistake=false, speed=high"`) and a metadata-conditioned policy $\pi_\theta(\mathbf{A}_t \mid \mathbf{o}_t, \ell, m)$. `MetaBC` then trains on all available data with the augmented prompt, given by

$$\mathcal{L}_{\texttt{MetaBC}}(\theta;\, \mathcal{D}) \;=\; \mathbb{E}_{\xi\sim\mathcal{D},\, t}\left[\, \mathcal{L}_{\mathrm{FM}}\Big(\theta;\, \mathbf{o}_t,\, \tilde{\ell}(\ell, m_\xi),\, \mathbf{A}_t\Big)\right], \tag{3}$$

where m_ξ is the tag of episode ξ.

Inference. At inference time, the policy is conditioned on metadata at every step, $\pi_\theta(\mathbf{A}_t \mid \mathbf{o}_t, \ell, m)$. We denote by $m^\star$ the *optimal* metadata, the value that selects the highest-quality mode (e.g., no mistakes, fast speed). In addition to fixing one metadata value for a rollout, we may also define a *metadata strategy* ν as a distribution over metadata values, such that

$$\pi_\theta^\nu(\mathbf{A}_t \mid \mathbf{o}_t, \ell) \;=\; \mathbb{E}_{m\sim\nu}[\pi_\theta(\mathbf{A}_t \mid \mathbf{o}_t, \ell, m)]. \tag{4}$$

The metadata $m \sim \nu$ is drawn once per action chunk.

4 Experimental Results

4.1 Experimental Setup

In this paper, we examine empirically whether, and why, VLA pretraining on non-expert data with metadata-conditioning is beneficial for robot manipulation. To this end, we run careful pretraining studies that hold the model architecture, training recipe, and evaluation fixed across all methods. We exclusively vary only the pretraining data and whether the policy is conditioned on metadata. Each subsequent subsection investigates a slightly varied pretraining setting, specifically of varying data mixtures. A thorough explanation of implemention details is provided in Appendix Section A.

VLA Policy. All methods employ the same VLA architecture: we use the `Qwen3-VL-2B` vision-language backbone and a flow-matching action head, which is conditioned on the VLM's final hidden layer. The policy observes four RGB cameras (two scene, two wrist) at the current and previous timestep and outputs an 8-step chunk of relative end-effector actions for a bimanual Franka Panda. Every policy is trained with the same parameters and a fixed budget of 100M training samples.

Environment and Tasks. Our pretraining data and policy evaluations use the Drake simulator [39] on a bimanual Franka Panda tabletop station. The pretraining data consists of 41 tabletop manipulation tasks, including 20 single-arm and 21 bimanual tasks. Each task is specified by a language instruction and has a binary success criterion.

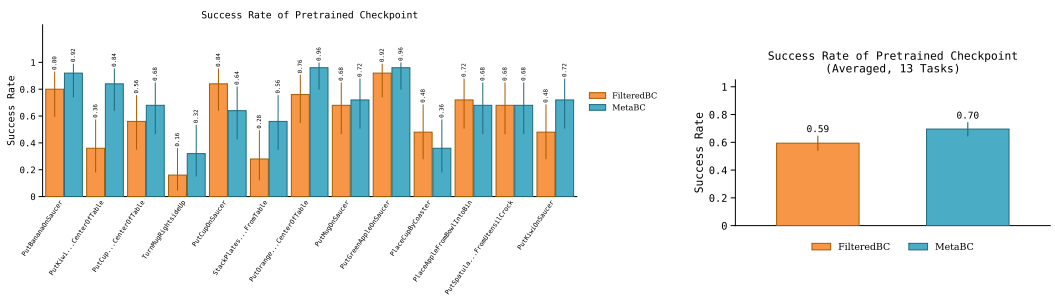


Figure 2: **Pretraining on multiple data sources.** We pretrain VLAs on a dataset consisting of real teleoperation, simulation teleoperation, policy rollouts in simulation containing both successes and failures, and planner-generated simulated trajectories. We observe that, unlike the in-domain data setting (Figure 1), `MetaBC` outperforms `FilteredBC` in the multiple data source setting. In particular, in the multiple data source setting, there may exist multiple types of behaviors or modes that all share the same success label, but some behaviors or data modes may be more useful to imitate than others. A fixed, train-time filtering strategy on task success cannot account for such careful differentiation of modes, while `MetaBC` can via its metadata.

Pretraining Data. We use four sources of pretraining: *simulated teleoperation*, consisting of human demonstrations collected in Drake with a spacemouse; *real teleoperation*, consisting leader-follower demonstrations collected on a physical robot on the same tasks; *simulated policy rollouts*, consisting of closed-loop rollouts in Drake of a previously pre-trained policy checkpoint; and *scripted planner* trajectories generated by a scripted motion planner in Drake. We use approximately 100 episodes per task of each data source. The simulated policy rollouts and the scripted planner data contain both successes and failures.

Metadata. Every episode is labeled with episodic-level information: `operator` (teleoperation, policy rollout, or planner), `mistake` (the episode's success/failure), and `speed` (a tertile of episode length). Section 4.3, that examines multiple data sources, additionally includes metadata specifying `domain` (simulation or real), `station` (the robot station), and `manipulators` (single-arm or bimanual). The labels are converted to text and appended to the task instruction (Section A.5). Unless otherwise noted, at inference-time, `MetaBC` is evaluated with *optimal* metadata $m^\star$, i.e., `mistake=false` and `speed=fast`, with the remaining fields set to match the evaluation environment.

Baselines. We compare `MetaBC` against three BC variants that share its architecture, hyperparameters, and training budget. `BC(expert)` trains on the expert teleoperation data alone, and `BC(full)` trains on all data, expert and non-expert, without metadata. `FilteredBC` trains on the successful episodes of all data, which is a proxy for data quality-based train-time filter. `BC(full)` and `MetaBC` therefore see exactly the same data and differ only in metadata-conditioning, while `BC(expert)`, `FilteredBC`, and `BC(full)` differ only in how the non-expert data is filtered.

Evaluation. We evaluate every pre-trained checkpoint zeroshot, with frozen weights and the base task instruction, via closed-loop control in Drake in the expert data's visual domain. Each policy is rolled out on initial scenes that are disjoint from those in the pretraining data, and all policies within an experiment are evaluated on the same scenes.

4.2 The addition of in-domain failure data does not drive metadata-conditioning's in-distribution performance improvement.

We first consider a simple, controlled setting, where the pretraining data consists of expert teleoperation and non-expert policy rollouts collected on the same tasks, environments, simulators, and

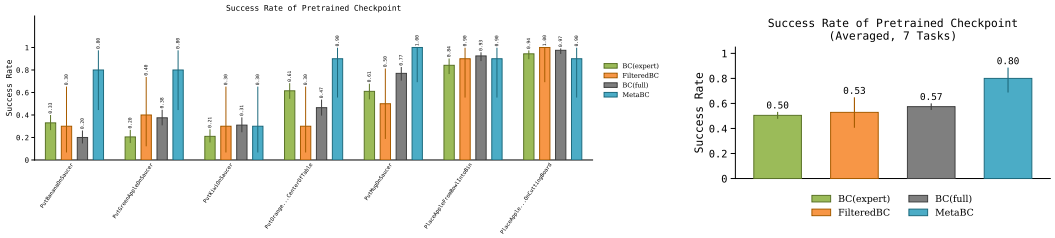


Figure 3: **Pretraining on cross-domain data.** We investigate the value of metadata-conditioning on cross-domain pretraining data. We pretrain VLAs on a dataset consisting of expert teleoperation demonstrations and non-expert policy rollouts, containing both successes and failures, as in the in-domain setting (Figure 1), but the non-expert rollouts are collected on a shifted visual domain, while evaluation is performed in the expert data's visual domain. We observe that, unlike the in-domain setting, `MetaBC` outperforms both the `BC` baselines and `FilteredBC`, indicating that it makes better use of out-of-domain non-expert data. In particular, at evaluation time the images resemble expert states, so the `BC` baselines implicitly sample from the expert data's mode, while `MetaBC` can additionally condition on the non-expert mode via its metadata and draw on the useful action signal in the out-of-domain data.

domains. Both datasets therefore come from the same MDP and differ only in the state-visitation distributions induced by their data-generating policies.[1]

From Figure 1, we observe that adding non-expert data improves metadata-conditioning's performance. However, `FilteredBC`, which filters the training data by task success and trains only on the successful episodes of both the expert and non-expert data, approximately recovers `MetaBC`'s performance improvement, suggesting that low-quality in-domain data does not drive metadata-conditioning's performance gain on in-distribution evaluation.

4.3 Metadata-conditioning enables inference-time mode selection.

Next, we broaden our pretraining study to consider pretraining data with more sources and domains. The previous experiment deliberately used a controlled setting with two sources, expert teleoperation and non-expert policy rollouts, to isolate the change of adding non-expert data (failure data, in particular). In practice, non-expert data may have many sources, such as planner-generated trajectories, and spans both the real world and simulation. In this experiment, we pretrain on data consisting of real-world teleoperation demonstrations, simulated teleoperation demonstrations, simulated policy rollouts, and planner-generated simulated trajectories. Of the 40 tasks, 35 have real-teleop, sim-teleop, and sim-rollout data, 6 of which also have planner data, and the remaining 5 have only real-teleop data.

From Figure 2, we observe that when pretraining on multiple data sources, filtering based on data quality is no longer sufficient to match the performance of `MetaBC`. The key difference in this experiment is the additional data sources, which can contribute distinct behaviors, or modes, under the same success label. Scripted or planner-generated data, for example, can complete the task while being less realistic for the policy to follow, since it may be non-Markovian or rely on privileged state information. Real teleoperation (leader-follower) and simulated teleoperation (spacemouse) may also behave differently on the same tasks. `MetaBC`, however, is able to perform *inference-time mode selection*: that is, `MetaBC` can selectively condition on the data modes that are best suited for the task objective. Unlike `FilteredBC`, which only considers task success as the mode to filter on, `MetaBC`

[1] From a theoretical perspective, adding non-expert data from a task with no existing expert-data coverage should improve performance, since the performance of a policy on a task with no data coverage will be approximately random. Thus, any non-expert data on the new task that is better than random should improve performance. However, the expected benefit of adding sub-optimal data from a task with expert-data coverage is less clear, since it depends on the data's coverage, the quality of the sub-optimal data, the generalization ability of the VLA, and optimization and finite-sample errors.

can instead leverage all of its metadata to filter modes at inference-time, resulting in performance improvement over the `FilteredBC` baseline.

4.4 Metadata-conditioning enables better cross-domain data usage.

The previous results showed that `MetaBC` can use its metadata to imitate certain modes at inference time—filtering low-quality in-domain modes during in-distribution evaluation. In this experiment, we examine whether `MetaBC` can leverage the inference-time mode selection to better utilize out-of-domain non-expert data.

In practice, non-expert data often comes from a different domain than the expert data—for example, from robot stations with different lighting. To study this setting, we maintain the same setting as Section 4.2—where pretraining data consists of expert teleoperation demonstrations and non-expert policy rollouts, but we shift the visual domain under which the non-expert policy rollouts are collected. The non-expert data thus still provides useful action signal, containing both successes and failures, but its visual domain is out of distribution relative to the expert data and the evaluation environments.

Unlike in the single-domain setting, `MetaBC` outperforms both BC and `FilteredBC` with cross-domain data (Figure 3), indicating that it makes better use of out-of-domain non-expert data. Although the mechanism inside the network is hard to verify, we offer an intuition for this result. A VLA conditions on state through two inputs, the images and the language instruction, which the VLM encodes into the features that the flow-matching action head consumes. Because the non-expert data comes from a different visual domain, the VLM representations of non-expert and expert states may differ. At evaluation time, a standard VLA sees images that resemble expert states, so its action head implicitly conditions on expert-like representations and samples from the expert data's mode. `MetaBC` sees the same images as the `BC` baselines, but the metadata lets it condition on the non-expert mode as well, so it can draw on the useful action signal in the non-expert data. In summary, the results suggest that `MetaBC` is more robust to the small distribution shifts that prevent BC and `FilteredBC` from using non-expert data.

5 Discussion

In this paper, we examined the value of metadata-conditioned VLA pretraining on non-expert data. We showed that pretraining with metadata-conditioning on non-expert data improves pretrained performance, but that the inclusion of failure data specifically does not drive the performance gain. Instead, we showed that metadata-conditioning allows for inference-time mode selection, which can be used to better leverage cross-domain pretraining data. Future work may investigate the impact of metadata-conditioning on the post-training potential of the policy.

References

[1] P. Intelligence, B. Ai, A. Amin, R. Aniceto, A. Balakrishna, G. Balke, K. Black, G. Bokinsky, S. Cao, T. Charbonnier, et al. pi07: a steerable generalist robotic foundation model with emergent capabilities. *arXiv preprint arXiv:2604.15483*, 2026.

[2] J. Schmidhuber. Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*, 2019.

[3] R. K. Srivastava, P. Shyam, F. Mutz, W. Jaśkowski, and J. Schmidhuber. Training agents using upside-down reinforcement learning. *arXiv preprint arXiv:1912.02877*, 2019.

[4] J. Di Ventura, D. R. Ashley, V. Herrmann, F. Faccio, and J. Schmidhuber. Upside down reinforcement learning with policy generators. *arXiv preprint arXiv:2501.16288*, 2025.

[5] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[6] M. Janner, Q. Li, and S. Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34:1273–1286, 2021.

[7] M. Nauman, M. Cygan, and P. Abbeel. Reward-conditioned reinforcement learning. *arXiv preprint arXiv:2603.05066*, 2026.

[8] D. Russo. Success conditioning as policy improvement: The optimization problem solved by imitating success. *arXiv preprint arXiv:2601.18175*, 2026.

[9] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

[10] M. Liu, M. Zhu, and W. Zhang. Goal-conditioned reinforcement learning: Problems and solutions. *arXiv preprint arXiv:2201.08299*, 2022.

[11] S. Park, D. Ghosh, B. Eysenbach, and S. Levine. Hiql: Offline goal-conditioned rl with latent states as actions. *Advances in Neural Information Processing Systems*, 36:34866–34891, 2023.

[12] S. Park, K. Frans, B. Eysenbach, and S. Levine. Ogbench: Benchmarking offline goal-conditioned rl. *arXiv preprint arXiv:2410.20092*, 2024.

[13] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[14] B. Eysenbach, X. Geng, S. Levine, and R. R. Salakhutdinov. Rewriting history with inverse rl: Hindsight inference for policy improvement. *Advances in neural information processing systems*, 33:14783–14795, 2020.

[15] A. Li, L. Pinto, and P. Abbeel. Generalized hindsight for reinforcement learning. *Advances in neural information processing systems*, 33:7754–7767, 2020.

[16] Q. Wang, J. Xiong, L. Han, H. Liu, T. Zhang, et al. Exponentially weighted imitation learning for batched historical data. *Advances in Neural Information Processing Systems*, 31, 2018.

[17] X. B. Peng, A. Kumar, G. Zhang, and S. Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

[18] A. Nair, A. Gupta, M. Dalal, and S. Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[19] D. Brown, W. Goo, P. Nagarajan, and S. Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International conference on machine learning*, pages 783–792. PMLR, 2019.

[20] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[21] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *Advances in neural information processing systems*, 33:1179–1191, 2020.

[22] I. Kostrikov, A. Nair, and S. Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2022.

[23] S. Fujimoto and S. S. Gu. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021.

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

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

[26] T. Chen, Z. Wang, and M. Zhou. Diffusion policies creating a trust region for offline reinforcement learning. *arXiv preprint arXiv:2405.19690*, 2024.

[27] A. O'Neill, A. Rehman, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, A. Pooley, A. Gupta, A. Mandlekar, A. Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903. IEEE, 2024.

[28] J. Hejna, C. Bhateja, Y. Jiang, K. Pertsch, and D. Sadigh. Re-mix: Optimizing data mixtures for large scale imitation learning. *arXiv preprint arXiv:2408.14037*, 2024.

[29] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR, 2023.

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

[31] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[32] J. Mercat, S. Keh, K. Arora, I. Huang, P. Shah, H. Nishimura, S. Iwase, and K. Liu. Vla foundry: A unified framework for training vision-language-action models. *arXiv preprint arXiv:2604.19728*, 2026.

[33] M. Nakamoto, O. Mees, A. Kumar, and S. Levine. Steering your generalists: Improving robotic foundation models via value guidance. *arXiv preprint arXiv:2410.13816*, 2024.

[34] N. Ma, S. Tong, H. Jia, H. Hu, Y.-C. Su, M. Zhang, X. Yang, Y. Li, T. Jaakkola, X. Jia, and S. Xie. Inference-time scaling for diffusion models beyond scaling denoising steps, 2025. URL https://arxiv.org/abs/2501.09732.

[35] R. Singhal, Z. Horvitz, R. Teehan, M. Ren, Z. Yu, K. McKeown, and R. Ranganath. A general framework for inference-time scaling and steering of diffusion models, 2025. URL https://arxiv.org/abs/2501.06848.

[36] Y. Lipman, R. T. Chen, H. Ben-Hamu, M. Nickel, and M. Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

[37] X. Liu, C. Gong, and Q. Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

[38] Y. Lipman, M. Havasi, P. Holderrieth, N. Shaul, M. Le, B. Karrer, R. T. Chen, D. Lopez-Paz, H. Ben-Hamu, and I. Gat. Flow matching guide and code. *arXiv preprint arXiv:2412.06264*, 2024.

[39] R. Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics, 2019. URL https://drake.mit.edu.

[40] F. Chen, A. Huang, N. Golowich, S. Malladi, A. Block, J. T. Ash, A. Krishnamurthy, and D. J. Foster. The coverage principle: How pre-training enables post-training. *arXiv preprint arXiv:2510.15020*, 2025.

A Experimental and Implementation Details

A.1 VLA Policy

We implement and pre-train all VLA policies with the VLA-Foundry codebase [32]. Table 1 summarizes the architecture; the subsections below give additional detail.

Table 1: VLA policy architecture.

Component	Value
Vision-language backbone	
Pre-trained model	`Qwen/Qwen3-VL-2B-Thinking`
Trainable parameters	Full
Visual tokens per image	256
Conditioning feature	Last-layer hidden state at an appended action token ($d = 2048$)
Observation	
Cameras	4 RGB (2 scene, 2 wrist)
Frame history	2 ($t-1$, t), i.e. 8 images per observation
Image resolution	224×224 crop of a 342×256 (H $\times$ W) frame
Language	Task instruction, optionally prefixed with metadata
Action space	
Embodiment	Bimanual Franka Panda
Dimension	$20 = 2 \times (3 \text{ translation} + 6 \text{ rotation} + 1 \text{ gripper})$
Representation	Relative end-effector pose; 6D rotation
Normalization	Per-timestep, centered 1st–99th percentile, fit per data source
Action window	1 past + 8 future steps
Prediction / execution horizon	8 / 8 steps (open-loop, then replan)
Action expert	
Architecture	Causal transformer
Layers / hidden size / heads	24 / 1024 / 16
Feed-forward	SwiGLU
Positional encoding	RoPE
Normalization	LayerNorm
Conditioning	VLM feature + flow timestep embedding
Flow matching	
Interpolation path	$x_s = (1-s)\,x_0 + s\,\epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$
Timestep distribution	$s \sim \mathcal{U}\{0, \frac{1}{1000}, \ldots, \frac{999}{1000}\}$
Regression target / loss	Velocity $\epsilon - x_0$ / MSE
Input noise augmentation	Gaussian, $\sigma = 0.05$
Noisy-action clamping	$[-1.5, 1.5]$
Noise samples per observation	8 (one shared VLM forward pass)
Integration steps (inference)	8

A.1.1 Vision-Language Backbone

Policies use a `Qwen/Qwen3-VL-2B-Thinking` VLM backbone. The backbone's parameters are updated during pretraining. An action token is appended to the text, and the action head takes as input the hidden state of the final token of the last VLM layer, which has dimension 2048. Image tokens are produced by the Qwen processor, and each image contributes 256 visual tokens.

A.1.2 Observation

The observation consists of images from four cameras at the current and previous timesteps. The policy receives no proprioception unless otherwise noted.

Table 2: Cameras used as policy observations.

Role	Camera name
Scene, right	`scene_right_0`
Scene, left	`scene_left_0`
Wrist, left	`wrist_left_plus`
Wrist, right	`wrist_right_minus`

A.1.3 Action Representation

The action is a 20-dimensional chunk of relative end-effector commands for two Franka Panda arms, with the following field order:

Table 3: Layout of the 20-dimensional action vector.

Slice	Field	Dimensions
0:3	Left end-effector translation, relative	3
3:6	Right end-effector translation, relative	3
6:12	Left end-effector rotation, 6D, relative	6
12:18	Right end-effector rotation, 6D, relative	6
18:19	Left gripper	1
19:20	Right gripper	1

The window is one past step and eight future steps. The eight future steps are the action head's prediction target. At evaluation, the policy executes the eight future steps open-loop, then replans (i.e., generates a new action chunk). Actions are normalized per timestep by a centered 1st-to-99th percentile normalization fit to each data source's own statistics. Noisy actions are clamped to $[-1.5, 1.5]$.

A.1.4 Action Head

The action head is a 24-layer causal transformer with hidden size 1024, trained with flow matching. It uses 16 heads, SwiGLU feed-forward, rotary position embeddings, and layer norm. The action head is conditioned on the flow (i.e., noise) timestep embedding. Each observation is paired with eight independent noise samples. The vision-language forward pass is computed once and reused. At inference time, we integrate the flow for 8 steps.

A.2 Pretraining Data

We use the Drake simulator for simulation-based data collection and evaluation [39]. All data is collected on Franka Panda arms. Each experiment mixes a subset of the following sources.

Table 4: Unique tasks in the pretraining data.

Task	Type	Instruction
PickAndPlaceBox	Single-arm	Pick up the block and put it in the middle of the table
PlaceCupByCoaster	Single-arm	Place the cup next to the coaster
PlaceCupOnCoaster	Single-arm	Pick up the metal cup and place it on the coaster
PushCoasterToCenterOfTable	Single-arm	Push any items in the way aside then push the coaster to the center of the table lengthwise and two squares down from the table edge closest to the robot arm bases
PushCoasterToMug	Single-arm	Push the coaster to the mug
PutBananaInCenterOfTable	Single-arm	Pick up the banana and place it near the center of the table
PutBananaOnSaucer	Single-arm	Pick up the banana and put it on the saucer
PutCupInCenterOfTable	Single-arm	Put the cup in the center of the table close to the robot.
PutCupOnSaucer	Single-arm	Put the cup on the plate
PutGreenAppleInCenterOfTable	Single-arm	Put the green apple in the center of the table
PutGreenAppleOnSaucer	Single-arm	Put the green apple on the blue saucer
PutKiwiInCenterOfTable	Single-arm	Move the kiwi mid-way between the arms
PutKiwiOnSaucer	Single-arm	Pick up the kiwi from the table and place it on the plate
PutMugOnSaucer	Single-arm	Put the mug on the plate
PutOrangeInCenterOfTable	Single-arm	Pick up the orange and put it closer to the center of the table
PutOrangeOnSaucer	Single-arm	Pick up the orange and place it on the plate
PutSpatulaInUtensilCrock	Single-arm	Put the spatula in the utensil crock
PutSpatulaInUtensilCrockFromDryingRack	Single-arm	Pick up the spatula from the drying rack and put it into the utensil crock
TurnCupUpsideDown	Single-arm	Turn the cup upside down
TurnMugRightsideUp	Single-arm	Turn the mug rightside up
BimanualHangMugsOnMugHolderFromDryingRack	Bimanual	Pick up the mug from the drying rack and hang the mug on the mug holder by the mug handle
BimanualHangMugsOnMugHolderFromTable	Bimanual	Hang the mug on the tree-shaped mug holder
BimanualLayCerealBoxOnCuttingBoardFromTopShelf	Bimanual	Take the cereal box from the top shelf and lay it flat on the cutting board
BimanualLayCerealBoxOnCuttingBoardFromUnderShelf	Bimanual	Grab the ceral box under the shelf and lay it on top of the cutting board
BimanualPlaceAppleFromBowlIntoBin	Bimanual	Pickup the green apple from the white bowl and place into the white bin
BimanualPlaceAppleFromBowlOnCuttingBoard	Bimanual	Take the apple from the bowl and place it onto the cutting board
BimanualPlaceAvocadoFromBowlOnCuttingBoard	Bimanual	Pick up the avocado from the bowl, place it on the counter, then pick it with the other arm and place in on the cutting board
BimanualPlaceFruitFromBowlIntoBin	Bimanual	Place fruit from the bowl into the bin
BimanualPlaceFruitFromBowlOnCuttingBoard	Bimanual	Pick up all pieces of fruit from the bowl, perform a hand off, and put them on the cutting board.
BimanualPlacePearFromBowlIntoBin	Bimanual	Pick up the pear from the bowl and put it into the empty bin
BimanualPlacePearFromBowlOnCuttingBoard	Bimanual	Place pear on the cutting board
BimanualPutMugsOnPlatesFromDryingRack	Bimanual	Place both mugs from the drying rack onto two separate plates, facing up
BimanualPutMugsOnPlatesFromTable	Bimanual	Place one mug on each plate
BimanualPutRedBellPepperInBin	Bimanual	Put the red bell pepper in the bin
BimanualPutSpatulaOnPlateFromDryingRack	Bimanual	Put the spatula on the plate from the drying rack
BimanualPutSpatulaOnPlateFromTable	Bimanual	Put the spatula from the table on the plate
BimanualPutSpatulaOnTableFromDryingRack	Bimanual	Take out the spatula from the drying rack and place it at the corner near the spoon
BimanualPutSpatulaOnTableFromUtensilCrock	Bimanual	Put the spatula on the table from the utensil holder
BimanualStackPlatesOnTableFromDryingRack	Bimanual	Place a plate from the drying rack on the table and stack additional plates on top
BimanualStackPlatesOnTableFromTable	Bimanual	Move one plate towards the center of the table and stack the other plate on top
BimanualStoreCerealBoxUnderShelf	Bimanual	Pickup the cereal box from the top shelf and store the cereal box in the bottom shelf

Simulation Teleoperation (v0.4.3.9). This dataset is collected from human teleoperation in Drake on 41 tasks, containing 1,644,938 samples and 7,498 episodes. There are 50 episodes for each of the eight fruit tasks, 98 episodes for the PickAndPlaceBox task, 500 episodes for each of the TurnCupUpsideDown and TurnMugRightsideUp tasks, and 200 episodes for each of the remaining 30 tasks. The data is collected on two stations: Riverway and Cabot. When the volume of the dataset is capped for a particular experiment, we keep the first episodes by session and episode index.

Real Teleoperation (v0.4.3.9). This dataset is collected from human teleoperation on the physical robot. We use the 39 tasks that have a corresponding simulation task. We use data collected from two stations: Riverway and Salem.

Policy Rollouts. The policy rollout dataset is collected from rollouts of a pre-trained policy in the Drake simulator. We collect rollout data using two checkpoints from the same pretraining run: `checkpoint3`, which completed approximately 30% of pretraining, and `checkpoint11`, which completed 100% of pretraining. The `checkpoint11` dataset contains approximately 100 episodes per task, 4,1000 episodes in total, and an average success rate of approximately 77%. The `checkpoint3` dataset contains approximately 100 episodes per task, and an average success rate of approximately 55%. Mistake labels are taken from the rollout success flag. When the volume of the dataset is capped for a particular experiment, we keep the 100 lowest-numbered existing episode indices per task, so a task whose index range has holes still contributes 100 episodes.

Scripted Planner. Finally, we also generate trajectories using a scripted planner in the Drake simulator. We generate data for six tasks, with 100 episodes for each task. We keep every collected episode, which includes some failures.

A.3 Zeroshot Evaluation

Zeroshot evaluation is closed-loop control in the Drake simulator [39] through a gRPC policy server, using a pre-trained checkpoint with frozen weights. Evaluations use initial seeds (initial scene indices) disjoint from those used to collect the pretraining data. All policies in an experiment are evaluated on the same set of seeds. Policies are evaluated for 25 episodes per task.

A.4 Training Recipe

The recipe is identical across campaigns and methods. Methods inside a campaign differ only in their pretraining data and whether they use metadata-conditioning.

Table 5: Pretraining hyperparameters, shared across all experiments and methods.

Hyperparameter	Value
Optimizer	AdamW (β_1 0.9, β_2 0.95, epsilon 10^{-8})
Learning rate	5×10^{-5}, constant schedule
Warmup	1,000 steps, linear
Weight decay	10^{-8}
Gradient clip	1.0
Dropout on metadata	0.0
Precision	bfloat16 mixed precision
Batch	16 per GPU, 1,024 global
Distribution	FSDP
Nodes	4
Node type	`ml.p5.48xlarge`
GPUs	NVIDIA H100, 8 per node (32 total)
EMA	Off
Seed	42
Training budget	100,000,000 samples
Samples actually seen at completion	99,999,744
Epochs	Multiple passes are allowed; sources are re-sampled until the budget is spent

A.5 `MetaBC`'s Metadata

Table 6: `MetaBC`'s metadata fields.

Metadata	Values
`Operator`	0 = policy rollout 1 = teleoperation 2 = planner
`Domain`	0 = simulation 1 = real
`Station`	0 = cabot 1 = riverway 2 = salem
`Manipulators`	1 = single arm 2 = bimanual
`Speed`	1 = fastest 2 = middle 3 = slowest
`Mistake`	0 = episode success 1 = episode failure

`mistake` is determined by the episode's task success flag. Teleoperation is never labeled a failure. `speed` is computed from episode length, with buckets at the tertiles of raw episode length. `manipulators` is a function of the task name: 1 for single-arm tasks, 2 for bimanual tasks.

A.6 pass@k Evaluation

As a preliminary investigation of downstream post-training potential of checkpoints, we measure pass@k performance, the probability that at least one of k independent rollouts from the same initial condition succeeds. While success rate measures zeroshot performance, pass@k measures whether the policy's sampling distribution *covers* a successful behavior and is an effective proxy for post-training potential [40].

Formally, for a task ℓ and initial state s_0, let $p^{\pi}_{\text{succ}}(\ell, s_0) = \mathbb{E}[R(\xi) \mid \ell, s_0]$ be the success probability from that initial state. We measure

$$\text{pass@}k(\pi) \;=\; \mathbb{E}_{\ell,\, s_0 \sim \rho_0}\Big[\, 1 - (1 - p^{\pi}_{\text{succ}}(\ell, s_0))^k \Big]. \tag{5}$$

Given $N \geq k$ rollouts per initial condition, of which c succeed, we average the unbiased estimator $1 - \binom{N-c}{k} \big/ \binom{N}{k}$ over initial conditions and tasks [40].

pass@k is a coverage statistic on a fixed scene. We roll out the same scene n times, fixing it with the same initial seed for every attempt. If c of those attempts succeed, the unbiased estimator of the probability that at least one of k attempts succeeds is

$$\text{pass@}k \;=\; 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \tag{6}$$

for $1 \leq k \leq n$ and $0 \leq c \leq n$, and 1 when $n - c < k$. The reported curve averages this quantity over scenes and then over tasks with equal weight. Note that success rate uses one rollout per scene, while pass@1 is the mean success rate across the attempts of each scene, not the success of the first attempt alone.

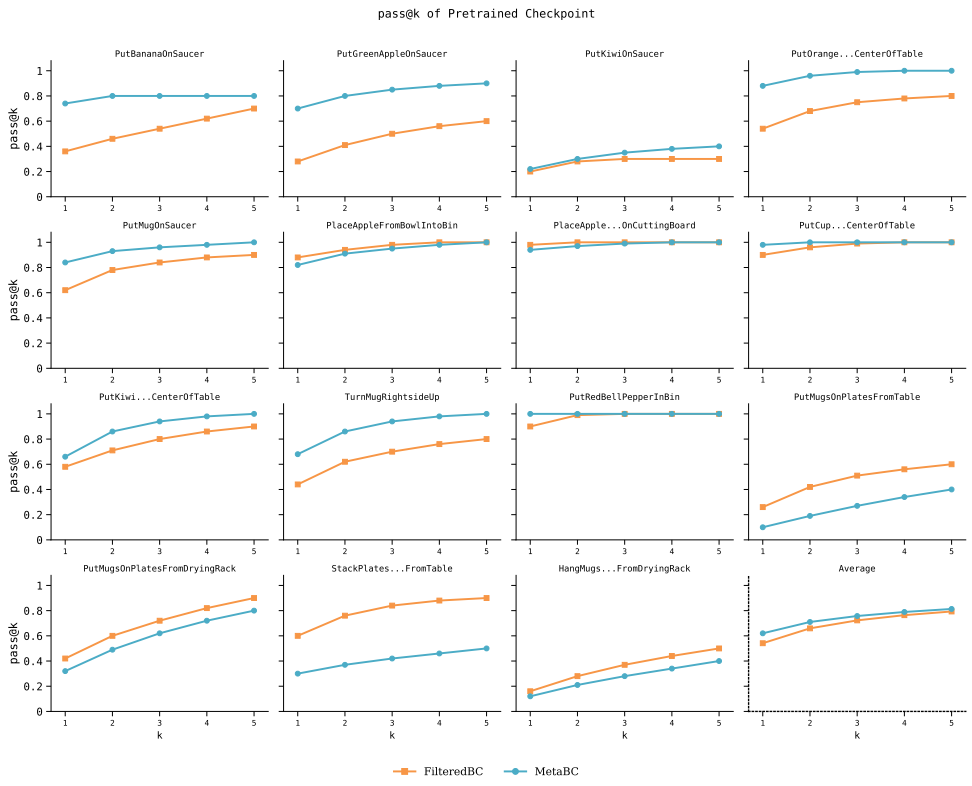


Figure 4: We evaluate the pass@k performance of `MetaBC` and `FilteredBC` from the cross-domain data experiment. We report pass@k performance with $k = 5$ and $n = 10$.