Jump Start Your Policy Learning with Lessons from 160,000 Training Runs

Nabil Omi[1]* **Eric Bae**[1] **Chung Yik Edward Yeung**[1] **Siddhartha Sen**[2] **Ali Farhadi**[1]
[1]University of Washington [2]Microsoft

Abstract

Reliable progress in offline policy learning depends on a small set of methodological practices, including careful reporting, well-tuned baselines, and evaluation across diverse conditions. Prior work has noted that results can be sensitive to reporting choices, hyperparameter tuning, and dataset properties independently, but these sources of variability have not been systematically investigated at the scale needed to understand how they shape conclusions. To address this gap, we present a large-scale empirical study of offline reinforcement and imitation learning, training over 160,000 policies across 114 datasets. At this scale, no algorithm dominates: aggregate performance across top methods is often close, but the leaders differ substantially across environments. We find that proper hyperparameter tuning frequently reshuffles perceived algorithm rankings; and simple baselines, including behavior cloning, are often stronger than is commonly assumed after extensive tuning. We also study hyperparameter transfer and sensitivity across environments, identifying a simple strategy that generalizes well. From these analyses we distill practical recommendations, and release JumpStart: a resource suite of trained models, per-model hyperparameter and reward data, strong baselines across all environments, and a website to make retrieval and analysis trivial. Together, these resources aim to make offline policy learning research more reliable and to open new directions for work beyond the scope of this study.

Website | **Code** | **Models**

1 Introduction

Reinforcement Learning (RL) studies how an agent can learn to act by interacting with an *environment* to maximize long-term reward. In *online* RL, the agent improves through repeated interaction, collecting new data as it updates its policy. In many real applications, however, such interaction is expensive, slow, or unsafe, which motivates learning from fixed datasets instead. Two common approaches in this setting are Offline Reinforcement Learning (ORL), which uses reward-labeled, pre-collected *offline* data to optimize a policy without further environment interaction, and Imitation Learning (IL), which learns to reproduce behavior directly from demonstrations. In this paper, we use the term *offline policy learning* to refer broadly to both ORL and IL, since our focus is on the shared empirical question of how reliably we can train and compare policies from pre-collected *datasets*. We define the combination of an environment and the pre-collected data as a *task*.

That promise has driven a lot of algorithmic progress, but it has also made the empirical side of the field harder than it sometimes looks. Offline policy learning results can shift substantially depending on data distributions, hyperparameter tuning, and even seemingly minor reporting choices, and small-scale studies often do not make those sensitivities visible. As a result, even when papers evaluate on standard benchmarks, it can still be surprisingly difficult to tell whether a method is

*Corresponding author: nabilomi@cs.uw.edu

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

genuinely better or whether the comparison itself is doing some of the work. Resolving this requires scale: diverse tasks to show where and when algorithms succeed or fail, and more tuning runs per method to ensure each is evaluated near its best.

Prior empirical work already hints at many of these sensitivities. Kang et al. [1] show that outcomes can depend heavily on implementation and evaluation details such as reward normalization, initialization, policy parameterization, layer normalization, learning-rate schedules, and the underlying data distribution, while other studies show that model scale [2], dataset exploration and trajectory quality [3], and problem properties such as reward sparsity, horizon, stochasticity, and human versus synthetic data [4] can all change which methods appear strongest. In practice, the literature is also hard to compare cleanly: papers often evaluate on different task subsets, use different aggregation rules or numbers of evaluation episodes, or omit numerical tables. Results are also frequently copied across papers in cases where they likely should not be, for instance when underlying environment versions differ (e.g., v0 vs. v2), a point raised by Kostrikov et al. [5]. What this calls for is not yet another benchmark, but a unifying set of strong baselines, tuning data, and trained models on the environments the field already uses.

To tackle these problems, we present a large-scale empirical study of offline policy learning, training over $160{,}000$ policies across 114 datasets spanning a diverse range of environments, algorithms, and tasks. Our primary contributions are:

- **Scale.** A large-scale empirical study of offline policy learning, training over $160{,}000$ policies across 114 datasets, designed to make sensitivity to methodological choices visible.
- **Findings on reporting and baselines.** We show that incomplete reporting and under-tuned baselines can obscure how much of a paper's reported gain actually comes from the proposed algorithm; that proper hyperparameter tuning frequently reshuffles algorithm rankings; and that simple baselines, including behavior cloning, are often stronger than commonly assumed. We also find that no single algorithm dominates: aggregate performance across top methods is often close, and the leader differs substantially across environments.
- **Hyperparameter transfer and sensitivity.** A characterization of which algorithms are sensitive to which hyperparameters, along with simple transfer strategies that generalize well.
- **Practical recommendations.** Concrete and data-backed guidance for evaluation, tuning, and baseline construction that anyone can adopt.
- **JUMPSTART.** A resource suite of trained models, per-model hyperparameter and reward data, strong baselines across all tasks, and an interactive website that lets users reproduce and extend the figures in this paper, including a configurable leaderboard whose ranking criteria can be redefined per use case (we present several illustrative configurations in the appendix). The name reflects the goal: a researcher writing a new offline-policy-learning paper should not have to spend a significant percentage of their time setting up baselines, environments, evaluation infrastructure, and reporting code, and JUMPSTART provides those upfront. Trained models and source code will be released upon acceptance; the anonymized raw results dataset and the website are available now as a compressed ZIP file in supplementary material (usage instructions in Appendix B), and the website is designed to be extended by the community as new methods and environments are added.

2 Background and Related Work

In this section, we briefly review the background needed to place our study in context. We first summarize the datasets and benchmarks used in offline policy learning, then outline the main algorithm families we study, and finally discuss prior empirical studies on what makes algorithms work or fail, along with related issues in evaluation, reporting, and tuning.

Algorithms. We study a mix of imitation learning and offline RL methods spanning direct supervised policies, sequence models, generative policies, and value-based methods. Behavior Cloning (BC) fits actions directly with supervised learning. Action Chunking with Transformers (ACT; 6) predicts short action chunks to reduce compounding error in manipulation settings. VQ-BeT (7) uses vector-quantized latent actions to capture multimodal behavior. Decision Transformer (DT; 8) models trajectories autoregressively and conditions on the remaining reward within a trajectory. Diffusion [9]

uses diffusion on actions to represent multimodal trajectories. Implicit Q-Learning (IQL; 5) improves a policy through implicit value estimation without querying out-of-distribution actions. Conservative Q-Learning (CQL; 10) regularizes value estimates conservatively to stabilize offline Q-learning. ReBRAC (11) builds on TD3+BC with a set of specific design choices, including layer normalization, a deeper critic, a higher discount factor, and decoupled actor and critic penalties. Batch-Constrained deep Q-learning (BCQ; 12) constrains candidate actions to remain close to the dataset support.

Empirical studies. Andrychowicz et al. [13] train *on-policy* RL agents while varying more than 50 implementation choices, and find that low-level details such as policy initialization, observation normalization, and action-distribution parameterization often matter as much as the high-level algorithm. Engstrom et al. [14] similarly find that implementation details explain much of PPO's advantage over TRPO. Orsini et al. [15] reach a similar conclusion for adversarial imitation learning, and additionally show that algorithm rankings shift substantially between synthetic and human demonstrations. Paine et al. [16] carry out an early study of hyperparameter selection in offline RL, finding that final-policy quality depends strongly on hyperparameter choices that cannot be reliably selected from offline data alone; their setup predates current benchmarks but the underlying tension still applies. Adkins et al. [17] quantify dependence on per-environment tuning and the number of hyperparameters needed to recover near-peak performance in PPO variants, though this is in online reinforcement learning. Kang et al. [1] ablate 20 low-level implementation choices across CQL, CRR, and IQL on D4RL and RL Unplugged, showing that many reported gains come from these choices rather than the core algorithm and that outcomes depend strongly on the data distribution. Cetin et al. [2] argue that, in offline RL, scale can matter more than novelty: simple methods, scaled appropriately, can match or beat more elaborate ones on D4RL. Schweighofer et al. [3] find that dataset properties, especially exploration coverage and trajectory quality, are strong predictors of which methods work well, and Bhargava et al. [4] show that the relative ordering of offline RL and imitation-style methods depends heavily on reward sparsity, horizon, and whether demonstrations are human or synthetic.

Reporting practices and reproducibility. Inconsistent reporting has long been an issue in deep RL. Henderson et al. [18] document sensitivity to seeds, hyperparameters, and codebases, while Jordan et al. [19] propose cross-environment evaluation accounting for tuning and statistical uncertainty. Agarwal et al. [20] recommend robust aggregate metrics with uncertainty estimates; we adopt two throughout: the interquartile mean (IQM), which discards the top and bottom 25% of normalized scores before averaging, and the optimality gap, the average shortfall from a target performance level (we use expert reward). Canonical offline policy learning papers diverge in ways that make direct comparison hard. BCQ [12], BEAR [21], and BRAC [22] present reward-over-training curves rather than consolidated tables; Decision Transformer [8] reports only on a restricted D4RL subset, with a raw mean across it; ReBRAC [11] reports a raw mean and IQL [5] a total, both fragile to a single high-variance or easy environment. Results are also sometimes copied across papers when underlying environment versions differ (e.g., v0 vs. v2), as flagged by Kostrikov et al. [5].

3 Methods

3.1 Datasets

Minari follows the same recipe as D4RL, generating offline datasets from trained policies [23], and additionally includes human-collected Adroit datasets (`pen-human`, `door-human`, `hammer-human`, `relocate-human`) sourced from [24]. We use Minari rather than the original D4RL release because it is the actively maintained successor, with newer environment versions, clearer documentation, and openly available dataset-generation code. Because the underlying datasets are not bit-identical to D4RL, some absolute numbers (e.g., CQL on AntMaze) may differ from prior reports.

We made two additions. The Atari datasets in Minari were too small for the discrete-control comparisons we wanted, so we collected new ones with 100k states per game using their dataset generation code, following the Atari100k setting [25]. We also omit a number of environments whose default datasets are trivially solvable across all methods we tested. The remaining 114 datasets (57 continuous control, 57 discrete) cover a wide range of distributional shapes (Figure 9), including some near-zero-reward degenerate datasets that we filter from our analyses but still release, as they seem to result in useful policies. After filters, we land at 52 discrete and 50 continuous datasets.

Appendix C lists every dataset we use and a very brief summary of the task and how the data is collected by the original Minari developers.

Environment groups. For per-group reporting we pool environments into nine continuous-control categories and seven Atari categories. Continuous: **MuJoCo locomotion** (HalfCheetah, Hopper, Walker2d, Ant, Swimmer, Humanoid, HumanoidStandup) at *expert* (`mj·loc/exp`), *medium* (`mj·loc/med`), and *simple* (`mj·loc/sim`) quality; **MuJoCo simple** (Reacher, Pusher, InvertedPendulum, etc.) at *expert* (`mj·sim/exp`) and *medium* (`mj·sim/med`); **D4RL Adroit** (Door, Hammer, Pen, Relocate on a 24-DoF hand) at *expert* (`d4·adr/exp`), *cloned* (`d4·adr/cl`), and *human* (`d4·adr/hu`) quality; and **D4RL AntMaze** (`d4·antmz`), kept as a single bucket since all variants share its sparse-reward stitching structure. Atari (57 `atari-medium` environments): **hard-exploration** (ϵ-greedy fails, e.g. Montezuma's Revenge), **sports** (score-difference vs. opponent), **maze** (top-down corridor/pellet), **reflex** (paddle and pinball), **shooter** (side-scrolling and fixed-screen), **platformer** (climb/jump), and **vehicle** (driving and tank). Full membership is in Appendix E.

3.2 Model training

Our pipeline is built on d3rlpy [26] for verified algorithm implementations and Ray Tune [27] for distributed tuning. We completed full tuning runs with BC, BCP (BC with percentile-based data filtering, also called BC%, a common baseline), BCQ, CQL, and DT across all 114 continuous and discrete datasets; VQ-BeT, ACT, IQL, and ReBRAC run on 57 continuous control environments. We use the LeRobot implementations of ACT and VQ-BeT[2] with minor modifications to integrate them with d3rlpy. For Diffusion Policy, we use a lightly modified implementation from the original paper's code release[3]. Across the broader project, including prototyping and failed runs, we estimate on the order of 500,000 GPU-hours on 512–768 V100 equivalents; exact accounting is difficult since jobs were preemptible and overlapped on each GPU.

Training budgets. Value-based methods (CQL, BCQ, IQL, ReBRAC) use 500,000 gradient steps on continuous environments and 250,000 on discrete; BC, BCP, and DT use 50 epochs. ACT and Diffusion use 100 epochs each, and VQ-BeT uses 300 total split among stages. These budgets were chosen via an exponentially increasing sweep on a small set of testing tasks (3 MuJoCo HalfCheetah datasets and all of D4RL AntMaze) under default and four random hyperparameter configurations, picking the smallest budget at which rewards plateau. We did not sweep budgets per (algorithm, dataset) pair, since that would have added another major axis at substantial compute cost; we spent the budget on broader hyperparameter tuning and dataset coverage instead.

Evaluation. For all evaluations, we use the final checkpoint evaluated across 100 episodes, and record the final total return at the termination/truncation point prior to averaging and normalization. We normalize returns as $(R_{\text{policy}} - R_{\text{random}})/(R_{\text{expert}} - R_{\text{random}})$, where R_{expert} and R_{random} are the mean returns of the dataset-generating policy and a random policy, respectively. Minari provides a method to recover the environment used to collect each dataset, but sometimes omits wrappers (e.g., resizing or frame stacking), as in Atari. In these cases, we recover the missing wrappers from Minari's data-generation scripts.

Hyperparameter tuning. We use two optimizers: random search [28] and the Tree-structured Parzen Estimator (TPE), which fits densities to better and worse observed configurations to bias future trials toward promising regions [29]; TPE is run through Ray Tune's `OptunaSearch` wrapper [30]. Search spaces are in Appendix A and a full breakdown of model counts is in Table 3.

4 Results

Our models support far more analysis than we can include here, so rather than cataloging every pattern across 160,000+ policies we focus on three threads most useful for making offline policy learning research more informative, comparable, and practical. Section 4.1 examines how aggregate evaluations and rankings behave across choices of metric, task subset, and tuning budget, and shows that aggregates hide more than they show. Section 4.2 looks at how hyperparameter sensitivity is distributed across algorithms and at how a single transfer point can serve as a strong default.

[2] ACT implementation; VQ-BeT implementation.

[3] Diffusion Policy code release.

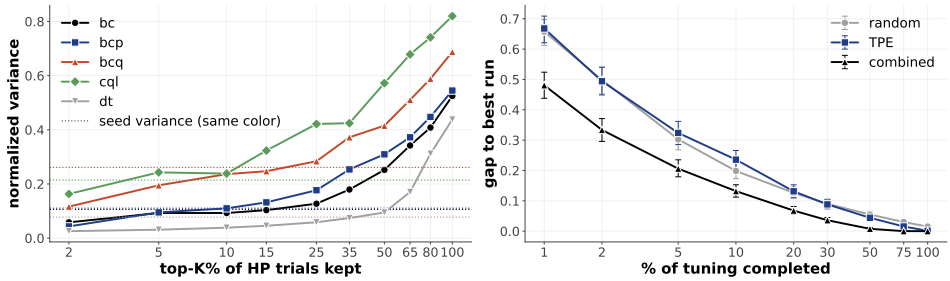

Figure 1: (Left) Hyperparameter variance dominates seed variance: coefficient of variation across all configurations is several times that across seeds. Restricted to algorithms supporting every environment in our suite. Dotted horizontal lines show the seed-variance baseline for each algorithm, using the same color as its curve. (Right) Gap to the best run (IQM and 95% bootstrap CI), $(s_{\max} - s^{(p)}_{\text{best}})/s_{\max}$, vs. the fraction p of the tuning budget consumed, for random search, TPE, and their union. $s_{\max}$ is the maximum over all available trials per (algorithm, environment) group; $s^{(p)}_{\text{best}}$ is the best score in the first fraction p of each tuner's trials.

4.1 Aggregate evaluations and ranking stability

Aggregates can hide more than they show. Which algorithm looks best depends materially on choice of metric, choice of task subset, and choice of tuning budget. We treat each as a separate failure mode and recommend evaluation practices that make the resulting ambiguity legible.

Tuning budget. The coefficient of variation across hyperparameter configurations is on average $5.05\times$ that across seeds, and exceeds seed CoV in 92.1% of (algorithm, task) groups; the seed variance of the top configurations is comparable to overall seed variance (Figure 1, left). Rather than running many seeds per configuration, we sample the top few hyperparameter configurations as our point estimate. Most algorithms reach a low gap to the best run after 25–50 trials, beyond which additional trials produce gains within seed variance (Figure 1, right); the choice of tuner does not appear to be a significant factor at this scale. We accordingly recommend running at least 25–50 configurations and reporting clear tuning suggestions (e.g. as in ReBRAC [11]). As Section 4.2 shows, predicting which hyperparameter values a new task will need is non-trivial, so leaving tuning to the reader is unlikely to recover comparable results.

Choice of aggregate metric. The standard recommendation in *online* deep RL is to summarize many-task performance with the interquartile mean (IQM) [20], but in offline policy learning IQM trims away tasks rather than just noise: per-run variance is much smaller than in online RL, and the tails of the score distribution are typically the hardest tasks rather than outlier seeds. Plain means have the opposite failure mode, with a small number of easy high-reward runs dominating the aggregate; this is the metric most prior work in offline policy learning reports, and it systematically underweights the difficult tasks where the reward gap to the expert is largest. We therefore use the optimality gap, the average shortfall of the per-task normalized score from expert, $\text{OG} = \mathbb{E}_e[\max(0, 1 - r)]$ [20], which weights difficult tasks by how much room there is to improve. Empirically, optimality-gap rankings agree more reliably across task subsets than IQM rankings (Figure 3).

Choice of evaluation subset. A single "which algorithm is best" answer presumes the ranking is a stable property of the algorithms; in practice it is partly a property of the evaluation slate. We quantify this with bootstrapped Kendall's τ between the ranking on a random task subset and the ranking on the full set, where $\tau = 1$ is full agreement, $\tau = 0$ is chance, and $\tau = 0.5$ corresponds to roughly 75% of algorithm pairs ordered consistently—equivalently, one in four pairs flipped. First, *how many* tasks matter, though not as much as one would hope: at 10 continuous datasets τ sits around 0.5 with wide residual uncertainty, and tightens only slowly as the subset grows (Figure 3, left); discrete tasks transfer better, partly because fewer algorithms support that regime so there are fewer pairs to disagree on. *Which* tasks are chosen matters at least as much: in a controlled transfer within the three families that admit a clean (task $\times$ group) grid (MuJoCo locomotion, MuJoCo simple, D4RL Adroit), structured selection beats random subsets of matched size along both axes, but τ still plateaus near 0.5 (Figure 3, middle). The takeaway is not that rankings are noisy and need more samples; it is that the ranking is genuinely regime-dependent, and a single overall ranking compresses away structure a

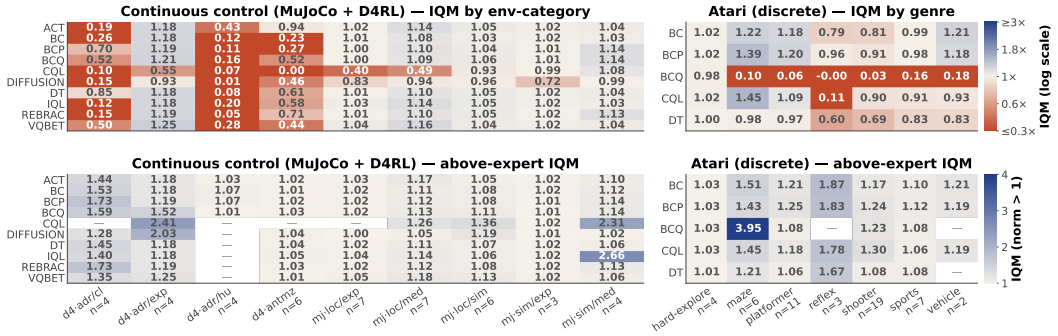

Figure 2: Per-group performance for each algorithm (IQM, higher is better). Continuous tasks are grouped by dataset type, discrete by game category.

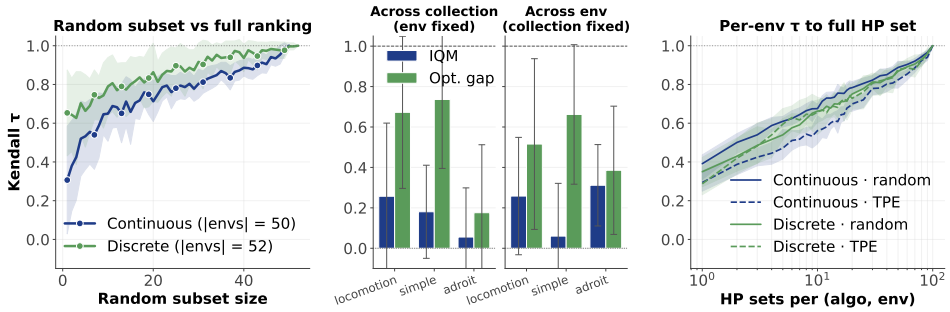

Figure 3: Stability of algorithm rankings. (Left) Bootstrapped Kendall's τ vs. full-set ranking as subset size grows. (Middle) Cross-task ranking agreement within the three families with a clean (task $\times$ group) grid; τ across collection regimes with task fixed vs. across task with group fixed. Agreement is similar along both axes and exceeds random subsets of matched size (3–5 tasks), though uncertainty is high; optimality gap transfers more reliably than IQM. (Right) Average per-task ranking vs. tuning budget; rankings shift with more tuning.

practitioner actually needs—motivating the per-regime view in Figures 2 and 5 and the per-category reporting recommended in Section 5.

Standout findings. A few patterns are robust across the per-group view in Figures 2 and 5 and the aggregate view in Figure 4. *ACT* and *DT* perform consistently well overall, with above-average performance except when using the Adroit human demonstrations. *BC* and *BCP*, often reported as weak baselines, remain competitive after extensive tuning, with BCP achieving the lowest optimality

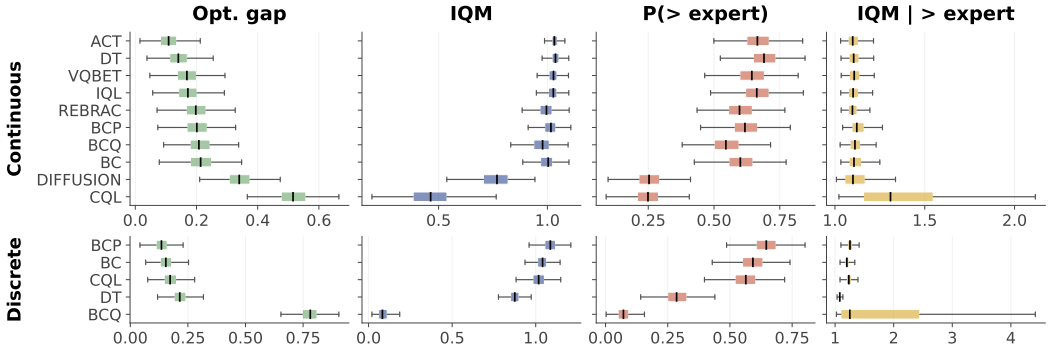

Figure 4: Aggregate performance: optimality gap (left), IQM (middle-left), probability of beating the expert (middle-right), and IQM conditioned on beating the expert (right), for continuous (top) and discrete (bottom) environments, following Agarwal et al. [20]. Raw data: Appendix F.1.

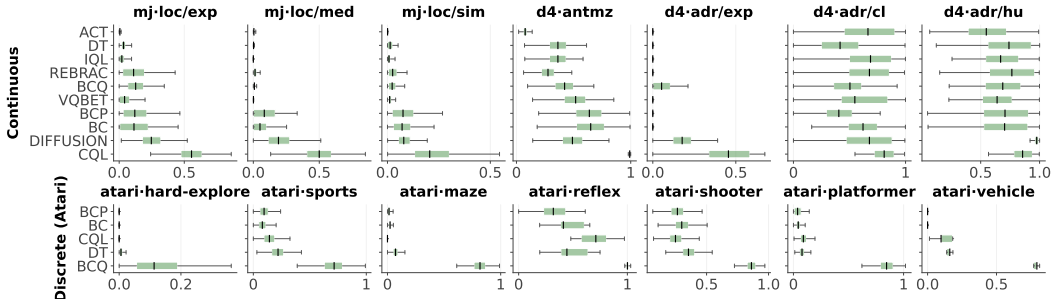

Figure 5: Algorithm performance across continuous environment subsets. Rankings shift substantially between subsets, with no single method dominating. Raw data: Appendix E.

gap on Atari. *ACT* and *VQ-BeT* rank first and third, respectively, by overall continuous-control optimality gap and are strong on the Adroit expert and human-collected datasets, matching their design intent and serving as a useful transfer result. *CQL* is the clearest case of *boom-or-bust* behavior: scoring relatively poorly on the continuous heatmap (0 on AntMaze, although it ranks third on Atari, behind BC/BCP), but conditional on beating the expert it posts some of the largest above-expert margins in the suite (above-expert IQM of 2.52 on Adroit-expert and 2.14 on `mj·sim/med`).

4.2 Hyperparameter sensitivity and transfer

The previous section showed that tuning budget materially shifts algorithm rankings; this section asks two follow-up questions. Which hyperparameters actually drive performance for each algorithm, and how consistent is the answer across environments? And once an algorithm builder has run a broad sweep, what is the right way to distill a single strong default parameter set for downstream users? We address sensitivity first, then transfer.

Which hyperparameters matter? The numbers in this section are summarized from the per-(algorithm, task) sensitivity table, which is too long to include inline; the full table is in Appendix F.2. We measure each hyperparameter's importance as its L1-normalized squared Spearman correlation with trial score within an environment (ρ_h^2, normalized so that shares sum to 1); intuitively, this is the share of explained rank variance attributable to that hyperparameter. Most tasks are sensitive to only 1–3 hyperparameters, but *which* hyperparameters those are varies substantially across tasks even within an algorithm, which is what makes peak performance hard to recover without per-task tuning.

Sensitivity concentration varies widely across algorithms. Among continuous methods, ACT and ReBRAC are the most concentrated, with a single hyperparameter (learning rate for ACT, actor BC-regularization β for ReBRAC) accounting for 53% and 62% of explained rank variance on average, respectively, and the top three covering 82% and 74%. CQL continuous assigns 27% to `alpha_threshold` and 41% to the top two, and the discrete variants of BCQ and CQL concentrate even more aggressively on actor learning rate (54% and 76%). At the other end, BCQ continuous spreads sensitivity across five hyperparameters with the top accounting for only 20%; DT sits in between.

The dominant hyperparameter matches what original papers emphasize in some cases and not others. Learning rate is on top for BC, ACT, and the discrete variants of BCP, BCQ, CQL, and DT, which is the standard emphasis. The continuous picture is messier: BCP is most sensitive to `hidden_units` and `train_percentile` (each 21%, tied with learning rate at 21%); BCQ is most sensitive to the discount factor γ rather than learning rate; and dropout is the third most important hyperparameter for BC and the fourth for BCP. The takeaway is that which hyperparameters dominate shifts substantially across algorithms and across tasks within the same algorithm, and is not always the set the original paper foregrounds.

Transfer. A practical question is how to package an algorithm for downstream users when using random search or TPE to tune hyperparameters. Grid search makes this trivial (pick the best cell across training tasks), but random and model-based search rarely place the per-task optima at the same point in the search space, so the natural "average best" does not exist. We propose a simple alternative.

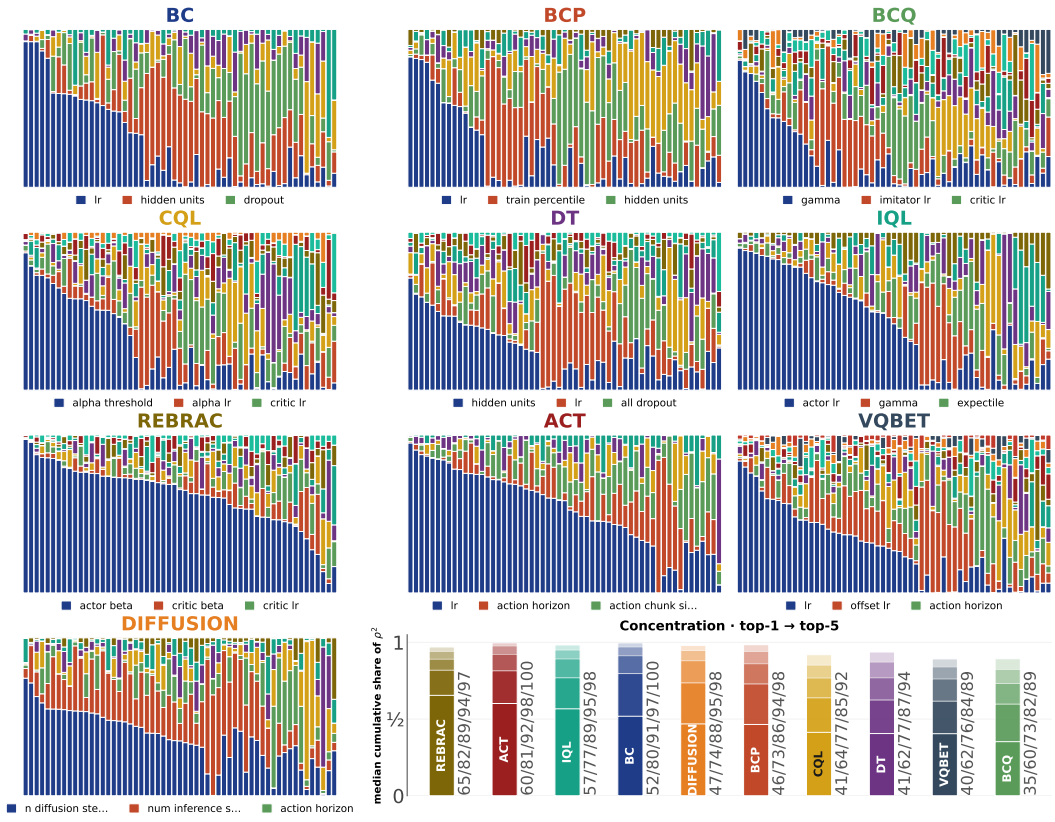

Figure 6: Hyperparameter sensitivity per algorithm. Each panel shows one stacked bar per task, partitioned by hyperparameter into segments proportional to that hyperparameter's share of explained rank variance. The top three hyperparameters by mean share are listed in each panel's legend. The right column reports per-algorithm median cumulative top-k share for $k = 1, \ldots, 5$.

Procedure. For each algorithm, we take the top 10% of trials on each training task, pool those trials, and report the per-hyperparameter median as the transfer point. This works because most tasks admit a wide range of well-performing configurations rather than a single sharp optimum: across all 683 (algorithm, task) pairs, 98.3% of median hyperparameter values fall inside the per-task top-10% Tukey fence (computed in log-space for log-scaled parameters; modal-value match for categoricals; per-algorithm range 97.4–99.3%). So the pooled median is almost never an outlier relative to what works on each task, even when no single point is optimal everywhere.

Evaluation. We evaluate using six algorithms (BC, BCQ, CQL, DT, IQL, ReBRAC) against d3rlpy [26] default values across five folds of five held-out tasks (not used for constructing defaults) each, randomly sampled, and compare the transfer point against defaults and the 200-trial sweep ceiling (Figure 7). The transfer point reduces optimality gap by approximately 25% versus library defaults ($0.551 \rightarrow 0.412$) without any per-task tuning, and starts to close the gap against full tuning (0.218).

5 Discussion

Imagine you have an idea for an offline policy learning method, some encouraging pilot results, and you want to turn it into a useful paper. The artifacts and lessons here are meant to help with the steps in between.

Is the pilot real? Pilot results can look better than they are when the baselines they beat are under-tuned, and without a sense of near-peak performance it is hard to tell whether a given reward number is good at all. The released JumpStart models address both: a pilot improvement reads as more credible against a baseline tuned to convergence on the same task.

Figure 7: **Top:** Optimality gap (Agarwal et al. [20], $\gamma = 1$) over all <algorithm, task> pairs, with 95% confidence intervals. **Bottom:** per-algorithm coverage, the mean percentage of individual hyperparameters whose transfer value lies inside the per-task top-10% Tukey fence ($Q1 - 1.5\,IQR$ to $Q3 + 1.5\,IQR$).



	BC	BCQ	CQL	DT	IQL	ReBRAC	overall
% in fence	99.3	98.1	97.9	97.4	98.5	99.0	*98.3*

Picking tasks worth running on. Section 4.1 showed there is no free lunch: different methods win in different regimes, and a single overall ranking is not stable at the dataset sizes commonly used. Breadth alone is not the goal; what matters is which regimes the method is meant to help with. With 114 tasks to draw from, a paper can scope its evaluation to the categories that probe its claim and report per-category results so the reader can see where the method helps and where it does not.

Tuning. Most algorithms are sensitive to only 1–3 hyperparameters, but *which* ones varies substantially across algorithms and tasks, so sensitivity is worth measuring rather than guessing. After roughly 25–50 trials per (algorithm, task) pair—beyond which gains fall within seed variance—our sensitivity analysis identifies which hyperparameters matter, and the transfer procedure in Section 4.2 distills a clean default for downstream users. Releasing per-algorithm tuning ranges alongside the method, as in ReBRAC [11], also makes a meaningful difference for users who would otherwise be unlikely to match the paper's reported numbers.

Comparing and reporting. A new method is most convincingly framed against tuned baselines on the regime it claims, rather than older numbers from past papers. We find optimality gap a useful headline aggregate: it transfers across task subsets more reliably than IQM and is less dominated by easy tasks than the plain mean (Section 4.1). Per-category breakdowns and uncertainty over the ranking itself also help, since even at 10 continuous tasks, $\sim 25\%$ of algorithm pairs end up ordered the wrong way relative to the full set.

After the paper. The JumpStart website (Appendix B) hosts a configurable leaderboard with reconfigurable ranking criteria and task subsets, and new methods, environments, and tuning data can be added over time. Authors can submit results directly using the database format in Appendix B.1, without relying on *our* training code, as long as they use the same datasets or contribute new ones evaluated alongside our methods.

6 Broader impacts

The most likely positive effect of this work is methodological: lowering the cost of fair baseline comparison should reduce the rate at which incorrect or stale results propagate through the offline policy learning literature. Better baselines and clearer per-regime evaluation also matter downstream, since offline RL and imitation learning are increasingly applied in safety-critical domains (robotics, healthcare, industrial control) where misleading benchmark results have real consequences. Negative impacts include the substantial compute footprint of the study itself ($\sim 500{,}000$ GPU-hours), which we hope is offset by the community reusing released models rather than retraining from scratch.

7 Limitations

Our suite covers ten algorithms, chosen to span the most widely used offline policy learning methods of the past few years across both offline RL (BCQ, CQL, DT, IQL, ReBRAC) and imitation/sequence-

modeling approaches (BC, BCP, Diffusion, ACT, VQ-BeT). Several recent methods are not yet included, including model-based offline RL, and goal-conditioned approaches; we built on d3rlpy, Ray Tune, and Minari, and made deliberate design choices throughout to keep the framework easy to extend. Our findings about ranking stability, sensitivity, and transfer should generalize to those methods, but we cannot verify this directly until they are added. Second, our hyperparameter search spaces are still bounded (although still very permissive); a method whose true optimum lies outside our ranges could be undersold by our analysis.

8 Conclusion

We presented a large-scale empirical study in offline policy learning, training over $160{,}000$ policies across 114 datasets to characterize how reporting choices, hyperparameter tuning, and task selection shape conclusions. No algorithm dominates across regimes, simple baselines are often stronger than assumed, and aggregate rankings are unstable enough that per-category reporting is worth the effort. Alongside these findings we release JumpStart, with the goal of making future offline policy learning research easier to do well and easier to compare cleanly.

Acknowledgments and Disclosure of Funding

We gratefully acknowledge the University of Washington for its support of this research. The authors declare no competing interests.

References

[1] Bingyi Kang, Xiao Ma, Yirui Wang, Yang Yue, and Shuicheng Yan. Improving and benchmarking offline reinforcement learning algorithms. *arXiv preprint arXiv:2306.00972*, 2023.

[2] Edoardo Cetin, Andrea Tirinzoni, Matteo Pirotta, Alessandro Lazaric, Yann Ollivier, and Ahmed Touati. Simple ingredients for offline reinforcement learning. *arXiv preprint arXiv:2403.13097*, 2024.

[3] Kajetan Schweighofer, Andreas Radler, Marius-Constantin Dinu, Markus Hofmarcher, Vihang Patil, Angela Bitto-Nemling, Hamid Eghbal-zadeh, and Sepp Hochreiter. A dataset perspective on offline reinforcement learning. *arXiv preprint arXiv:2111.04714*, 2021.

[4] Prajjwal Bhargava, Rohan Chitnis, Alborz Geramifard, Shagun Sodhani, and Amy Zhang. When should we prefer decision transformers for offline reinforcement learning? *arXiv preprint arXiv:2305.14550*, 2023.

[5] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *CoRR*, abs/2110.06169, 2021.

[6] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *CoRR*, abs/2304.13705, 2023.

[7] Seungjae Lee, Yibin Wang, Haritheja Etukuru, H. Jin Kim, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Vq-bet: Behavior generation with latent actions. In *International Conference on Machine Learning (ICML)*, 2024.

[8] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[9] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *CoRR*, abs/2303.04137, 2023.

[10] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[11] Denis Tarasov, Vladislav Kurenkov, Alexander Nikulin, and Sergey Kolesnikov. Revisiting the minimalist approach to offline reinforcement learning. *CoRR*, abs/2305.09836, 2023.

[12] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration, 2019. URL https://arxiv.org/abs/1812.02900.

[13] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters in on-policy reinforcement learning? a large-scale empirical study, 2020. URL https://arxiv.org/abs/2006.05990.

[14] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep RL: A case study on PPO and TRPO. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=r1etN1rtPB.

[15] Manu Orsini, Anton Raichuk, Léonard Hussenot, Damien Vincent, Robert Dadashi, Sertan Girgin, Matthieu Geist, Olivier Bachem, Olivier Pietquin, and Marcin Andrychowicz. What matters for adversarial imitation learning?, 2021. URL https://arxiv.org/abs/2106.00672.

[16] Tom Le Paine, Cosmin Paduraru, Andrea Michi, Caglar Gulcehre, Konrad Zolna, Alexander Novikov, Ziyu Wang, and Nando de Freitas. Hyperparameter selection for offline reinforcement learning, 2020. URL https://arxiv.org/abs/2007.09055.

[17] Jacob Adkins, Michael Bowling, and Adam White. A method for evaluating hyperparameter sensitivity in reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 37, pages 124820–124842, 2024. doi: 10.52202/079017-3964. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/e1cadf5f02cc524b59c208728c73f91c-Abstract-Conference.html.

[18] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. doi: 10.1609/aaai.v32i1.11694. URL https://ojs.aaai.org/index.php/AAAI/article/view/11694.

[19] Scott Jordan, Yash Chandak, Daniel Cohen, Mengxue Zhang, and Philip Thomas. Evaluating the performance of reinforcement learning algorithms. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4962–4973. PMLR, 2020. URL https://proceedings.mlr.press/v119/jordan20a.html.

[20] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. arXiv:2108.13264.

[21] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction, 2019. URL https://arxiv.org/abs/1906.00949.

[22] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning, 2019. URL https://arxiv.org/abs/1911.11361.

[23] Omar G. Younis, Rodrigo Perez-Vicente, John U. Balis, Will Dudley, Alex Davey, and Jordan K Terry. Minari, September 2024. URL https://doi.org/10.5281/zenodo.13767625.

[24] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. 2018. URL https://arxiv.org/abs/1709.10087.

[25] Łukasz Kaiser, Mohammad Babaeizadeh, Piotr Miłoś, Błażej Osiński, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-based reinforcement learning for atari. In *International Conference on Learning Representations (ICLR)*, 2020. arXiv:1903.00374.

[26] Takuma Seno and Michita Imai. d3rlpy: An offline deep reinforcement learning library. *Journal of Machine Learning Research*, 23(315):1–20, 2022. URL https://www.jmlr.org/papers/v23/22-0017.html.

[27] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E. Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training, 2018. URL https://arxiv.org/abs/1807.05118.

[28] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[29] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24 (NeurIPS)*, pages 2546–2554, 2011.

[30] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[31] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2021. URL https://arxiv.org/abs/2004.07219.

[32] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[33] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033, 2012.

[34] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019. arXiv:1910.11956.

[35] Cathy Wu, Aboudy Kreidieh, Kanaad Parvate, Eugene Vinitsky, and Alexandre M. Bayen. Flow: Architecture and benchmarking for reinforcement learning in traffic control. In *arXiv preprint arXiv:1710.05465*, 2017.

[36] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio López, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Conference on Robot Learning (CoRL)*, 2017.

[37] 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, 2021. URL https://arxiv.org/abs/2108.03298.

[38] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

[39] Cong Lu, Philip J. Ball, Tim G. J. Rudner, Jack Parker-Holder, Michael A. Osborne, and Yee Whye Teh. Challenges and opportunities in offline reinforcement learning from visual observations. 2023. URL https://arxiv.org/abs/2206.04779.

[40] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2023. arXiv:2306.03310.

[41] Rafael Rafailov, Kyle Beltran Hatch, Anikait Singh, Laura Smith, Aviral Kumar, Ilya Kostrikov, Philippe Hansen-Estruch, Victor Kolev, Philip J. Ball, Jiajun Wu, Chelsea Finn, and Sergey Levine. D5RL: Diverse datasets for data-driven deep reinforcement learning. *Reinforcement Learning Journal*, 5:2178–2197, 2024. arXiv:2408.08441.

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

[43] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[44] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018. URL https://arxiv.org/abs/1801.01290.

[45] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[46] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013. ISSN 1076-9757. doi: 10.1613/jair.3912. URL http://dx.doi.org/10.1613/jair.3912.

[47] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, and Jeff Braga. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms, 2021. URL https://arxiv.org/abs/2111.08819.

[48] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures, 2018. URL https://arxiv.org/abs/1802.01561.

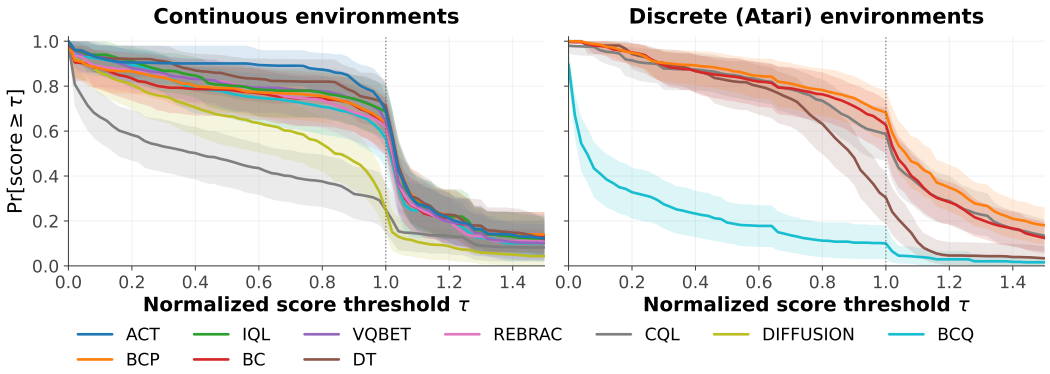

Figure 8: Survival curves across the algorithms in continuous (left) and discrete (right) environments, following Agarwal et al. [20].

A Tuning Details

We list the hyperparameter search ranges used during tuning. Algorithms that support both continuous- and discrete-action environments are given separate ranges for each; algorithms restricted to continuous control appear only in Table 1. The Decision Transformer (DT) uses a single search space shared across both environment types and is listed once, under continuous.

A.1 Continuous environments

Table 1: Hyperparameter ranges searched on continuous-action environments.

Hyperparameter	Range
BC	
`learning_rate`	log-uniform $[10^{-5}, 10^{-2}]$
`weight_decay`	log-uniform $[10^{-4}, 10^{-1}]$
`clip_grad_norm`	uniform $[0.1, 2.0]$
`dropout`	uniform $[0.0, 0.3]$
`hidden_units`	choice $\{[128, 128], [256, 256], [512, 512]\}$
`policy_type`	choice {deterministic, stochastic}
BCP	
`learning_rate`	log-uniform $[10^{-5}, 10^{-2}]$
`weight_decay`	log-uniform $[10^{-4}, 10^{-1}]$
`clip_grad_norm`	uniform $[0.1, 2.0]$
`dropout`	uniform $[0.0, 0.3]$
`hidden_units`	choice $\{[128, 128], [256, 256], [512, 512]\}$
`train_percentile`	uniform $[0.0, 90.0]$
`policy_type`	choice {deterministic, stochastic}
BCQ	
`actor_learning_rate`	log-uniform $[10^{-5}, 10^{-2}]$
`critic_learning_rate`	log-uniform $[10^{-5}, 10^{-2}]$
`imitator_learning_rate`	log-uniform $[10^{-5}, 10^{-2}]$
`gamma`	log-uniform $[0.9, 0.999]$
`tau`	uniform $[0.001, 0.02]$
`n_critics`	integer $\{1, \ldots, 4\}$
`update_actor_interval`	integer $\{1, \ldots, 4\}$
`lam`	uniform $[0.5, 0.95]$
`n_action_samples`	integer $\{20, \ldots, 200\}$
`rl_start_step`	integer $\{0, \ldots, 10000\}$
`beta`	uniform $[0.1, 0.9]$
CQL	

(continued on next page)

(continued from previous page)

Hyperparameter	Range
SAC-N	
actor_learning_rate	log-uniform $[10^{-5}, 10^{-2}]$
critic_learning_rate	log-uniform $[10^{-5}, 10^{-2}]$
temp_learning_rate	log-uniform $[10^{-5}, 10^{-2}]$
alpha_learning_rate	log-uniform $[10^{-5}, 10^{-2}]$
gamma	log-uniform $[0.9, 0.999]$
tau	uniform $[0.001, 0.02]$
conservative_weight	uniform $[0.1, 10.0]$
alpha_threshold	uniform $[-1.0, 20.0]$
n_action_samples	integer $\{5, \ldots, 20\}$
n_critics	integer $\{1, \ldots, 4\}$
DT (also used for discrete environments)	
learning_rate	log-uniform $[10^{-5}, 10^{-2}]$
weight_decay	log-uniform $[10^{-4}, 10^{-1}]$
clip_grad_norm	uniform $[0.1, 2.0]$
context_size	choice $\{10, 20, 30\}$
hidden_units	choice $\{[128, 128, 128], [256, 256, 256]\}$
all_dropout	uniform $[0.0, 0.3]$
num_heads	choice $\{1, 2, 4\}$
num_layers	integer $\{2, \ldots, 5\}$
gamma	log-uniform $[0.9, 0.999]$
reward_scale	choice $\{0.0, 1.0\}$
IQL	
actor_learning_rate	log-uniform $[10^{-5}, 10^{-2}]$
critic_learning_rate	log-uniform $[10^{-5}, 10^{-2}]$
gamma	log-uniform $[0.9, 0.999]$
tau	uniform $[0.001, 0.02]$
n_critics	integer $\{1, \ldots, 4\}$
expectile	uniform $[0.5, 0.9]$
weight_temp	uniform $[1.0, 10.0]$
ReBRAC	
actor_learning_rate	log-uniform $[10^{-5}, 10^{-2}]$
critic_learning_rate	log-uniform $[10^{-5}, 10^{-2}]$
actor_beta	log-uniform $[10^{-5}, 1.0]$
critic_beta	log-uniform $[10^{-5}, 1.0]$
gamma	log-uniform $[0.9, 0.999]$
tau	uniform $[0.001, 0.02]$
n_critics	integer $\{1, \ldots, 4\}$
target_smoothing_sigma	uniform $[0.0, 0.5]$
target_smoothing_clip	uniform $[0.0, 1.0]$
VQ-BeT	
learning_rate	log-uniform $[10^{-5}, 10^{-2}]$
offset_learning_rate	log-uniform $[10^{-5}, 10^{-2}]$
action_chunk_size	choice $\{2, 8\}$
action_horizon	choice $\{1, 4\}$
vqvae_steps_ratio	uniform $[0.1, 0.4]$
commitment_weight	log-uniform $[0.01, 1.0]$
focal_gamma	uniform $[0.5, 5.0]$
offset_loss_weight	uniform $[0.1, 5.0]$
primary_code_weight	uniform $[1.0, 10.0]$
secondary_code_weight	uniform $[0.1, 3.0]$
weight_decay	log-uniform $[10^{-6}, 10^{-2}]$
clip_grad_norm	uniform $[0.1, 2.0]$
ACT	
learning_rate	log-uniform $[10^{-5}, 10^{-2}]$
kl_weight	log-uniform $[1.0, 10.0]$
action_chunk_size	choice $\{2, 8\}$
action_horizon	choice $\{1, 4\}$

(continued on next page)

(continued from previous page)

Hyperparameter	Range
`weight_decay`	log-uniform $[10^{-6}, 10^{-2}]$
`clip_grad_norm`	uniform $[0.1, 2.0]$
Diffusion	
`learning_rate`	log-uniform $[10^{-5}, 10^{-3}]$
`n_diffusion_steps`	choice $\{50, 100\}$
`num_inference_steps`	integer $\{20, \ldots, 100\}$
`action_chunk_size`	choice $\{8, 16\}$
`action_horizon`	choice $\{1, 4\}$
`weight_decay`	log-uniform $[10^{-6}, 10^{-2}]$
`clip_grad_norm`	uniform $[0.1, 2.0]$

A.2 Discrete environments

Table 2: Hyperparameter ranges searched on discrete-action environments. The DT search space is shared with the continuous setting and is reported once in Table 1.

Hyperparameter	Range
BC	
`learning_rate`	log-uniform $[10^{-5}, 10^{-2}]$
`weight_decay`	log-uniform $[10^{-4}, 10^{-1}]$
`clip_grad_norm`	uniform $[0.1, 2.0]$
`dropout`	uniform $[0.0, 0.3]$
`hidden_units`	choice $\{[128, 128], [256, 256], [512, 512]\}$
`reg_factor`	uniform $[0.0, 1.0]$
BCP	
`learning_rate`	log-uniform $[10^{-5}, 10^{-2}]$
`weight_decay`	log-uniform $[10^{-4}, 10^{-1}]$
`clip_grad_norm`	uniform $[0.1, 2.0]$
`dropout`	uniform $[0.0, 0.3]$
`hidden_units`	choice $\{[128, 128], [256, 256], [512, 512]\}$
`train_percentile`	uniform $[0.0, 90.0]$
`reg_factor`	uniform $[0.0, 1.0]$
BCQ	
`gamma`	log-uniform $[0.9, 0.999]$
`actor_learning_rate`	log-uniform $[10^{-5}, 10^{-2}]$
`n_critics`	integer $\{1, \ldots, 4\}$
`target_update_interval`	integer $\{2000, \ldots, 20000\}$
`action_flexibility`	uniform $[0.01, 0.1]$
`beta`	uniform $[0.1, 0.9]$
CQL	
`actor_learning_rate`	log-uniform $[10^{-5}, 10^{-2}]$
`gamma`	log-uniform $[0.9, 0.999]$
`n_critics`	integer $\{1, \ldots, 4\}$
`target_update_interval`	integer $\{1000, \ldots, 10000\}$
`alpha`	uniform $[0.1, 2.0]$

B JumpStart Website

The JumpStart website is a SvelteKit application that surfaces the suite of trained models and their training runs. Every page reads from a read-only SQLite database, so all the controls described in the in-site documentation take effect by re-querying that database — no caching, no client-side estimation.

Running the site from the supplementary archive. The `jumpstart-site/` directory in the supplementary materials is a self-contained copy of the application, including the two SQLite database files it depends on. To bring it up locally:

1. Install bun and Node.js 22 or later (the dev server uses the built-in `node:sqlite` module to open the database read-only).
2. Unzip the archive and `cd jumpstart-site`.
3. Run `bun install` to install dependencies. If `.env` does not already exist, copy `.env.example` to `.env`; the default `DATABASE_URL=file:local/all_results.db` matches the included database paths.
4. Confirm `local/all_results.db` (per-run scores, hyperparameters, and aggregate metrics) and `local/dataset_stats.db` (expert-return statistics used for normalization and overlays) are present in `local/`. The site derives the `dataset_stats.db` path automatically by string-replacing the filename, so the two files must sit side by side.
5. Run `bun run dev` and open `http://localhost:5173`. `bun run build` followed by `bun run preview` produces and serves a production build.

Where the documentation lives. A walkthrough of the four functional surfaces a paper-reading visitor will spend their time in — the algorithm leaderboard, the hyperparameter Spearman view, the run-level comparison tool, and the per-(algorithm, environment) drill-down — is on the running site at `http://localhost:5173/getting-started`, reachable from the "Get Started" button on the landing page. The repository's `jumpstart-site/README.md` repeats the setup instructions and documents the read-only design of the database client; the unused Drizzle scripts in `package.json` are scaffolding artifacts and can be ignored.

B.1 Database Format

This section is a reference guide to the database format we use to populate the website for researchers and practitioners to extend JUMPSTART with their own algorithms. The site (`jumpstart-site/`) reads two read-only SQLite files from `jumpstart-site/local/`:

- `all_results.db` *(required)*: per-run scores, hyperparameters, and tuner metadata. One table per algorithm.
- `dataset_stats.db` *(optional)*: expert-return statistics used for normalization and the expert overlay on the per-(algorithm, environment) page. Keyed by `environment`; no per-algorithm changes are needed.

Adding a new algorithm

To register a new algorithm `myalg`, add a single new table `myalg` to `all_results.db`. The site discovers algorithms by enumerating tables in `sqlite_master`, so no schema migration or registration step is required beyond creating the table and (optionally) classifying it in the algorithm-category map.

Required columns. Every algorithm table must define the following six columns. They are recognized as "common" columns by `src/routes/algorithms/[algo]/[env]/+page.server.ts` and are filtered out before columns are surfaced as hyperparameters.

Column	Type	Meaning
`algorithm`	`TEXT`	Algorithm name; equals the table name (e.g. `"myalg"`).
`environment`	`TEXT`	Hierarchical env id, e.g. `mujoco/halfcheetah/medium-v0`.
`score`	`REAL`	Higher-is-better return. `NULL` rows are filtered everywhere.
`time_total_s`	`REAL`	Wall-clock duration of the trial, in seconds.
`env_type`	`TEXT`	`"continuous"` or `"discrete"`. Drives env-type filters and HP-range splits.
`tuner`	`TEXT`	Search algorithm that produced the trial. See *Tuner labels* below.

A seventh column, `file_path` (`TEXT`), is also recognized as common and stripped from the hyperparameter set. It is optional and is retained only as trial provenance metadata — the site no longer parses it.

Hyperparameter columns. Every additional column on the table is treated as a hyperparameter and is auto-surfaced on the per-(algorithm, environment) page, the HP-Spearman view, and the HP-scatter API. Columns are stored as `TEXT` regardless of the underlying type so that mixed encodings (e.g. list-valued `hidden_units` like `"[128, 128]"`) round-trip without casting. Numeric values should be written as the canonical string of the float (e.g. `"0.0005"`); list/choice values should be written as a Python-style `repr` (e.g. `"[256, 256]"`, `"deterministic"`). The set of HP columns may differ between algorithms; existing examples range from 6 (**BC**) to 14 (**BCQ-continuous**). See the schemas of the released tables for references on naming conventions.

Environment naming. The `environment` string is parsed by splitting on `/`. The first segment is the dataset group (`mujoco`, `D4RL`, `atari-medium`, ...) and is used to group algorithms on the per-algorithm summary page. The remaining segments form the "leaf" name shown in tables. To remain compatible with the dataset-stats join, use the same `environment` string that appears as the primary key in `dataset_stats.db`.

Tuner labels. The leaderboard's "Split by tuner" control and the per-tuner stats table on the drill-down page read `tuner` verbatim. The recognized values are `"random"`, `"optuna"`, `"hyperopt"`, and `"other"` (the fallback for trials produced by an unrecognized search algorithm). To register a new tuner label, add it to `VALID_TUNERS` in `src/routes/api/leaderboard/+server.ts` and `src/routes/api/leaderboard/rank-curve/+server.ts`; no other site code needs to change.

If you index your trials with `scripts/index_results.py`, `tuner` is filled in automatically by substring match on the result-file path (`/optuna/`, `/random/`, `/hyperopt/`, else `"other"`). Practitioners writing their own tables should populate `tuner` directly; no path convention is required.

Algorithm category (optional). The `/algorithms` listing page groups tables by family. To classify `myalg`, add an entry to `ALGO_CATEGORY` in `src/routes/algorithms/+page.server.ts`; valid values are `"Imitation Learning"`, `"Offline RL"`, and `"Other"`. Algorithms not present in the map fall through to `"Other"`, so this step is purely cosmetic.

Worked example. A minimal table sufficient for the site to render `myalg` alongside the existing algorithms:

```
CREATE TABLE myalg (
    algorithm     TEXT,   -- always 'myalg'
    environment   TEXT,   -- e.g. 'mujoco/halfcheetah/medium-v0'
    score         REAL,
    time_total_s  REAL,
    env_type      TEXT,   -- 'continuous' or 'discrete'
    tuner         TEXT,   -- 'random' | 'optuna' | 'hyperopt' | 'other'
    file_path     TEXT,   -- optional trial provenance; not parsed
    -- hyperparameter columns (TEXT, any number, any names):
    learning_rate TEXT,
    hidden_units  TEXT,
    dropout       TEXT
);

INSERT INTO myalg VALUES (
    'myalg',
    'mujoco/halfcheetah/medium-v0',
    4321.5,
    1284.7,
    'continuous',
    'optuna',
    '/scratch/results/myalg/mujoco/halfcheetah/medium-v0/result.json',
```

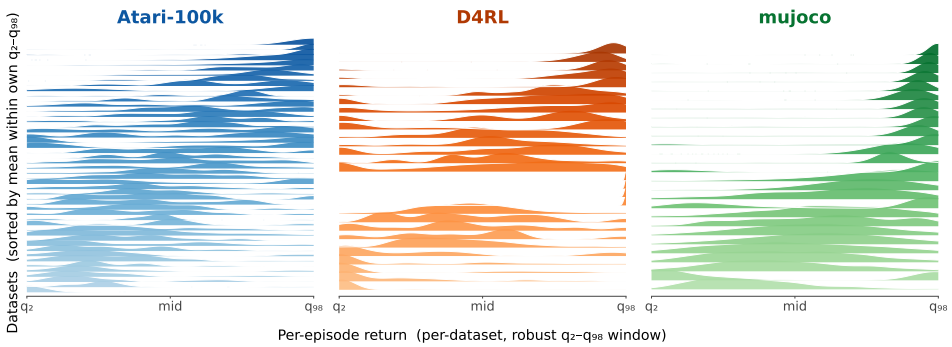

Figure 9: Dataset return distributions span many types across the Minari environments.

```
    '0.0003', '[256, 256]', '0.1'
);
```

After inserting rows, restart the dev server (`bun run dev`); the new algorithm will appear on `/algorithms`, in the leaderboard pool, in the comparison and HP-transfer views, and at `/algorithms/myalg/<env>` without further configuration.

C Datasets

A review of candidate datasets. The data used for offline policy learning can vary widely in quality, coverage, and domain, and these properties can strongly influence which algorithms perform well. The most widely used dataset is D4RL [31], which introduces a suite of tasks to evaluate offline (or data-driven) reinforcement learning. This suite consists of Maze2D and AntMaze [31], Gym-MuJoCo [32, 33], Adroit [24], FrankaKitchen [34], Flow [35], and CARLA [36]. In practice, most papers seem to remove Flow and CARLA from the set of tasks they use. Prior to D4RL, papers would generate their own non-standardized testing datasets [12, 21, 22]. Generally, these datasets are generated by training an expert for the environment with online reinforcement learning, and the resulting replay buffer or trajectories collected using a checkpoint of the expert are used to train the final models using ORL or IL. Later, Mandlekar et al. [37] introduced RoboMimic, a dataset of human-collected robot teleoperation trajectories for the RoboSuite [38] framework, and finds that batch RL performs well in the computer-generated datasets, but fails when trained on human-collected datasets. More recent benchmarks have aimed to address limitations of D4RL: Lu et al. [39] introduce V-D4RL, providing pixel-based analogues of the D4RL tasks to evaluate offline RL from visual observations; Liu et al. [40] release LIBERO, a procedurally generated suite of 130 language-conditioned manipulation tasks built around RoboSuite, designed to study lifelong knowledge transfer in imitation learning; Rafailov et al. [41] propose D5RL as a direct successor to D4RL, motivated by saturation on the original tasks, with realistic robotic manipulation and locomotion environments and heterogeneous data sources spanning scripted, teleoperated, and play-style data; and Park et al. [42] introduce OGBench, an offline goal-conditioned RL benchmark spanning 8 environment types and 85 datasets specifically designed to probe stitching, long-horizon reasoning, and stochasticity. In parallel, the Farama Foundation released Minari [23], a standardized dataset format and Python API that hosts re-implementations of D4RL and other reference offline RL datasets, partially supplanting the original D4RL codebase, which is no longer actively maintained. For ease of reproduction and maintenance purposes (which we detail further in Section 3.1), we use a 114-dataset subset of Minari, which we detail below. A visual showing the diversity of data distributions is shown in Figure 9.

D4RL. The Minari D4RL group is a faithful reproduction of the original D4RL benchmark, regenerated from the same task families with publicly released collection scripts at `https://github.com/Farama-Foundation/minari-dataset-generation-scripts`. We use the four families that have non-trivial offline-RL coverage: PointMaze, AntMaze, Adroit, and FrankaKitchen.

- **PointMaze.** A 2D force-actuated ball is driven to a goal location in one of four maze layouts (`umaze`, `medium`, `large`, `open`). Each layout has `dense` and sparse variants that differ only in the reward function: sparse pays 1 for reaching the goal and 0 otherwise, while `dense`

adds a shaped distance penalty. All transitions are produced by a PD controller that tracks waypoints generated with QIteration on the discretized maze; uniform action noise is added to widen the trajectory distribution. Once the agent reaches a goal a new random goal is sampled without resetting the agent, so trajectories form long, continuous traces.

- **AntMaze.** The same maze layouts are navigated by an 8-DoF Ant quadruped, whose low-level locomotion is provided by a goal-reaching SAC policy trained to follow the same QIteration waypoints, achieving above 80% success rate. Two coverage modes are released. In `play`, both the start position and the goal are sampled uniformly at the start of each episode. In `diverse`, both are sampled from a hand-picked set of cells chosen to maximize state coverage of the maze. Each maze size (`umaze`, `medium`, `large`) is paired with the corresponding modes; `umaze` additionally has a single fixed-goal variant.
- **Adroit.** Four dexterous-manipulation tasks on the 24-DoF Shadow Hand from Rajeswaran et al. [24]: `pen` (in-hand reorientation), `door` (latch and pull), `hammer` (drive a nail), and `relocate` (pick a ball and place it on a target). Each task ships three dataset variants. `human` contains the original 25 teleoperated human demonstrations released with the DAPG codebase [24]. `expert` is rolled out from a fully fine-tuned RL policy from the same DAPG repository (typically tens to hundreds of thousands of steps depending on task). `cloned` is constructed by training a behavior-cloning policy on the union of `human` and `expert`, rolling that BC policy in the environment, and mixing the rollouts $50{:}50$ with the source demonstrations.
- **FrankaKitchen.** A simulated 7-DoF Franka arm with a parallel gripper operating in a kitchen scene from Gupta et al. [34]; episodes require completing four target subtasks (microwave, kettle, light switch, slide cabinet). All three variants come from the original human teleoperation dataset of Gupta et al. [34], partitioned by completion structure: `complete` keeps only demonstrations that finish all four target subtasks in the canonical order; `partial` additionally includes demonstrations that finish a subset of the targets while solving other subtasks; `mixed` contains demonstrations that touch the same subtasks in arbitrary orderings, including ones that never complete any of the four target tasks.

MuJoCo. The Minari MuJoCo group covers the standard Gymnasium MuJoCo continuous-control suite (`halfcheetah`, `hopper`, `walker2d`, `ant`, `humanoid`, `humanoidstandup`, `swimmer`, `reacher`, `pusher`, `invertedpendulum`, `inverteddoublependulum`) [32, 33]. Unlike the original D4RL MuJoCo datasets, which mix random, medium-trained, and replay buffers, the Minari version releases up to three checkpoints from a single Stable-Baselines3 [43] training run per environment, each rolled out for $1{,}000$ episodes / up to $1{,}000{,}000$ transitions. The training algorithm is chosen per environment: `halfcheetah` and `humanoid` use TQC [21]; `hopper`, `walker2d`, `ant`, `humanoidstandup`, `reacher`, `pusher`, `invertedpendulum`, and `inverteddoublependulum` use SAC [44]; `swimmer` uses PPO [45]. Within an environment, the three variants correspond to checkpoints at increasing levels of training:

- `simple`: an early checkpoint. The policy is competent but clearly suboptimal (e.g. for HalfCheetah, "runs with decent speed and never falls over frequently").
- `medium`: an intermediate checkpoint, roughly halfway through training. The policy is qualitatively close to the expert on the primary task dimension but lower in cumulative return.
- `expert`: the fully converged checkpoint at the end of training. This represents an upper bound on per-env return.

Total training-step counts vary by environment (e.g. $25 \cdot 10^6$ for HalfCheetah/Hopper/Walker2d-SAC, $20 \cdot 10^6$ for Humanoid-TQC, $100 \cdot 10^6$ for Swimmer-PPO, 10^6 for InvertedPendulum-SAC), but the qualitative ordering of the three checkpoints is preserved. Not every environment ships all three variants: simpler control tasks (`pusher`, `reacher`, `invertedpendulum`, `inverteddoublependulum`, `swimmer`) release only `expert` and/or `medium`.

Atari. The `atari-medium` group covers all 57 ALE games from the standard Atari benchmark [46], each with a single `expert` dataset rolled out from a converged CleanRL PPO+IMPALA agent [47, 48]. The released Minari datasets contain only $\sim$10 episodes per game (e.g. $\sim$23k transitions for Breakout), which is too small for stable offline-RL evaluation. We therefore use a lightly modified version of the Minari collection script that draws *additional* rollouts from the same checkpoint to produce

100k transitions per game; the underlying expert policies and environment wrappers (frame stacking, grayscale, 84×84 resize) are unchanged from the upstream Minari pipeline.

D Model Count Breakdown

This appendix tallies the random-search and Optuna tuning sweeps. The remaining 1,151 models are archived without valid updated scores and excluded from those analyses.

D.1 Hyperparameter tuning trials

Every row in this subsection counts one fully-trained model: a single sampled hyperparameter configuration, trained from scratch on one (`algorithm, environment`) pair taking anywhere from a few GPU-minutes (BC, BCP) to a few GPU-hours (ACT, CQL, IQL, ReBRAC). The 114 environments split evenly into 57 continuous-action and 57 discrete-action (Atari) envs; ACT, Diffusion, IQL, ReBRAC, and VQ-BeT only run on continuous control. Per-algo trial budgets targeted 100 random + 100 Optuna per environment. Counts reflect completed trials with valid audited scores; deficits from the target include incomplete sweeps and excluded evaluations. Dashes indicate algorithms not run on that action type.

Algorithm	Continuous	Discrete	Total
ACT	10,472	—	10,472
BC	11,400	11,400	22,800
BCP	11,400	11,400	22,800
BCQ	11,400	8,687	20,087
CQL	11,400	10,021	21,421
Diffusion	9,889	—	9,889
DT	11,309	11,400	22,709
IQL	11,302	—	11,302
ReBRAC	11,401	—	11,401
VQ-BeT	8,832	—	8,832
Total	**108,805**	**52,908**	**161,713**

Table 3: Hyperparameter-tuning trial counts by algorithm and action type. Each cell is the number of independently-trained models on the relevant 57 continuous or 57 discrete environments.

Algorithm	D4RL	atari-medium	MuJoCo	Total
ACT	5,508	—	4,964	10,472
BC	5,800	11,400	5,600	22,800
BCP	5,800	11,400	5,600	22,800
BCQ	5,800	8,687	5,600	20,087
CQL	5,800	10,021	5,600	21,421
Diffusion	5,105	—	4,784	9,889
DT	5,754	11,400	5,555	22,709
IQL	5,800	—	5,502	11,302
ReBRAC	5,801	—	5,600	11,401
VQ-BeT	4,683	—	4,149	8,832
Total	**55,851**	**52,908**	**52,954**	**161,713**

Table 4: The same trials, broken down by environment source. D4RL spans 29 envs (Adroit cloned/expert/human, AntMaze, PointMaze, and FrankaKitchen), `atari-medium` spans 57 envs (`expert-v0` variant), and MuJoCo spans 28 locomotion + non-locomotion envs.

Search algorithm split. Of the 161,713 audited trials, 83,181 come from random search and 78,532 from Optuna. Both budgets target 100 trials per (`algorithm, environment`); the gap reflects incomplete sweeps and the exclusion of trials without valid updated evaluation scores.

E Environment Subsets

The full list of continuous environment subsets are itemized below:

- **d4** · **adr/cl** — D4RL Adroit, cloned variant: `D4RL/door/cloned-v2`, `D4RL/hammer/cloned-v2`, `D4RL/pen/cloned-v2`, `D4RL/relocate/cloned-v2`.
- **d4** · **adr/exp** — D4RL Adroit, expert variant: `D4RL/door/expert-v2`, `D4RL/hammer/expert-v2`, `D4RL/pen/expert-v2`, `D4RL/relocate/expert-v2`.
- **d4** · **adr/hu** — D4RL Adroit, human-demonstration variant: `D4RL/door/human-v2`, `D4RL/hammer/human-v2`, `D4RL/pen/human-v2`, `D4RL/relocate/human-v2`.
- **d4** · **antmz** — D4RL AntMaze (all maze sizes and dataset variants): `D4RL/antmaze/umaze-v1`, `D4RL/antmaze/umaze-diverse-v1`, `D4RL/antmaze/medium-play-v1`, `D4RL/antmaze/medium-diverse-v1`, `D4RL/antmaze/large-play-v1`, `D4RL/antmaze/large-diverse-v1`.
- **mj** · **loc/exp** — Gym MuJoCo locomotion, expert variant: `mujoco/halfcheetah/expert-v0`, `mujoco/hopper/expert-v0`, `mujoco/walker2d/expert-v0`, `mujoco/ant/expert-v0`, `mujoco/swimmer/expert-v0`, `mujoco/humanoid/expert-v0`, `mujoco/humanoidstandup/expert-v0`.
- **mj** · **loc/med** — Gym MuJoCo locomotion, medium variant: `mujoco/halfcheetah/medium-v0`, `mujoco/hopper/medium-v0`, `mujoco/walker2d/medium-v0`, `mujoco/ant/medium-v0`, `mujoco/swimmer/medium-v0`, `mujoco/humanoid/medium-v0`, `mujoco/humanoidstandup/medium-v0`.
- **mj** · **loc/sim** — Gym MuJoCo locomotion, simple variant: `mujoco/halfcheetah/simple-v0`, `mujoco/hopper/simple-v0`, `mujoco/walker2d/simple-v0`, `mujoco/ant/simple-v0`, `mujoco/humanoid/simple-v0`, `mujoco/humanoidstandup/simple-v0`.
- **mj** · **sim/exp** — Gym MuJoCo non-locomotion (Reacher, Pusher, pendulums), expert variant: `mujoco/reacher/expert-v0` (filtered out), `mujoco/pusher/expert-v0` (filtered out), `mujoco/invertedpendulum/expert-v0` (filtered out), `mujoco/inverteddoublependulum/expert-v0` (filtered out).
- **mj** · **sim/med** — Gym MuJoCo non-locomotion, medium variant: `mujoco/reacher/medium-v0` (filtered out), `mujoco/pusher/medium-v0` (filtered out), `mujoco/invertedpendulum/medium-v0` (filtered out), `mujoco/inverteddoublependulum/medium-v0` (filtered out).

The full list of discrete environment subsets are itemized below. All discrete environments are drawn from the `atari-medium` dataset group with the `expert-v0` variant.

- **hard** · **explore** — sparse-reward / hard-exploration Atari games (Bellemare et al., 2016, plus Skiing): `atari-medium/freeway/expert-v0`, `atari-medium/gravitar/expert-v0`, `atari-medium/montezumarevenge/expert-v0` (filtered out), `atari-medium/pitfall/expert-v0`, `atari-medium/privateeye/expert-v0`, `atari-medium/solaris/expert-v0` (filtered out), `atari-medium/venture/expert-v0` (filtered out), `atari-medium/skiing/expert-v0` (filtered out).
- **maze** — top-down corridor / pellet navigation games: `atari-medium/alien/expert-v0`, `atari-medium/amidar/expert-v0`, `atari-medium/bankheist/expert-v0`, `atari-medium/mspacman/expert-v0`, `atari-medium/qbert/expert-v0`, `atari-medium/wizardofwor/expert-v0`.
- **platformer** — climb / jump / map-traversal games: `atari-medium/crazyclimber/expert-v0`, `atari-medium/frostbite/expert-v0`, `atari-medium/hero/expert-v0`, `atari-medium/jamesbond/expert-v0`, `atari-medium/kangaroo/expert-v0`, `atari-medium/krull/expert-v0`, `atari-medium/kungfumaster/expert-v0`, `atari-medium/namethisgame/expert-v0`, `atari-medium/roadrunner/expert-v0`, `atari-medium/tutankham/expert-v0`, `atari-medium/upndown/expert-v0`.

- **reflex** — paddle / pinball reactive-control games: `atari-medium/breakout/expert-v0`, `atari-medium/pong/expert-v0`, `atari-medium/videopinball/expert-v0`.
- **shooter** — side-scrolling and fixed-screen shooters: `atari-medium/assault/expert-v0`, `atari-medium/asterix/expert-v0`, `atari-medium/asteroids/expert-v0`, `atari-medium/atlantis/expert-v0`, `atari-medium/beamrider/expert-v0`, `atari-medium/berzerk/expert-v0`, `atari-medium/centipede/expert-v0`, `atari-medium/choppercommand/expert-v0`, `atari-medium/defender/expert-v0`, `atari-medium/demonattack/expert-v0`, `atari-medium/gopher/expert-v0`, `atari-medium/phoenix/expert-v0`, `atari-medium/riverraid/expert-v0`, `atari-medium/seaquest/expert-v0`, `atari-medium/spaceinvaders/expert-v0`, `atari-medium/stargunner/expert-v0`, `atari-medium/timepilot/expert-v0`, `atari-medium/yarsrevenge/expert-v0`, `atari-medium/zaxxon/expert-v0`.
- **sports** — one-on-one games scored by point/score difference: `atari-medium/boxing/expert-v0`, `atari-medium/tennis/expert-v0`, `atari-medium/bowling/expert-v0`, `atari-medium/icehockey/expert-v0`, `atari-medium/doubledunk/expert-v0`, `atari-medium/fishingderby/expert-v0`, `atari-medium/surround/expert-v0`.
- **vehicle** — top-down driving / tank games: `atari-medium/battlezone/expert-v0`, `atari-medium/enduro/expert-v0` (filtered out), `atari-medium/robotank/expert-v0`.

Raw numbers for Figure 5

We provide raw numbers for Figure 5 in Table 5.

	D4RL				**MuJoCo**		
Algorithm	adr/cl	adr/exp	adr/hu	antmz	loc/exp	loc/med	loc/sim
ACT	0.186	1.180	0.427	0.937	1.018	1.143	1.049
BC	0.258	1.182	0.120	0.230	1.012	1.084	1.033
BCP	0.699	1.190	0.112	0.265	1.003	1.099	1.037
BCQ	0.519	1.208	0.163	0.521	0.999	1.091	1.061
CQL	0.100	0.547	0.070	0.000	0.402	0.493	0.933
Diffusion	0.151	0.932	0.008	0.462	0.825	0.941	0.957
DT	0.852	1.181	0.082	0.609	1.007	1.096	1.046
IQL	0.125	1.183	0.204	0.585	1.029	1.141	1.047
ReBRAC	0.152	1.194	0.045	0.710	1.011	1.102	1.051
VQ-BeT	0.496	1.247	0.282	0.435	1.042	1.162	1.040
Discrete (Atari) by genre							
Algorithm	hard-expl.	maze	platf.	reflex	shooter	sports	vehicle
BC	1.018	1.224	1.176	0.794	0.812	0.988	1.207
BCP	1.018	1.394	1.202	0.961	0.912	0.984	1.176
BCQ	0.977	0.101	0.061	-0.002	0.033	0.163	0.180
CQL	1.024	1.451	1.090	0.114	0.905	0.909	0.928
DT	1.001	0.977	0.970	0.601	0.691	0.829	0.832

Table 5: Per-algorithm IQM of expert-normalized top-10% scores, by environment subset. Top: continuous environments grouped by D4RL and MuJoCo families. Bottom: discrete (Atari) environments grouped by genre.

F Comprehensive Results

Here we explain where to access the associated raw data, and some notable but lengthy tables.

F.1 Model aggregate performance

Raw numbers from Figure 4 are available in Table 6.

Algorithm	Opt. gap ↓	IQM ↑	$\Pr[s > 1]$ ↑	IQM | $s > 1$ ↑
Continuous (50 environments)				
ACT	0.111 [0.046, 0.192]	1.032 [1.003, 1.082]	0.666 [0.538, 0.779]	1.098 [1.046, 1.235]
DT	0.142 [0.068, 0.224]	1.039 [0.978, 1.090]	0.691 [0.560, 0.815]	1.101 [1.049, 1.245]
VQ-BeT	0.172 [0.086, 0.263]	1.028 [0.923, 1.081]	0.643 [0.520, 0.766]	1.106 [1.049, 1.313]
IQL	0.174 [0.093, 0.263]	1.028 [0.917, 1.077]	0.664 [0.538, 0.781]	1.099 [1.049, 1.283]
ReBRAC	0.200 [0.117, 0.297]	0.997 [0.872, 1.057]	0.599 [0.472, 0.724]	1.095 [1.046, 1.219]
BCP	0.203 [0.112, 0.304]	1.018 [0.878, 1.075]	0.620 [0.489, 0.743]	1.117 [1.058, 1.350]
BCQ	0.212 [0.126, 0.307]	0.977 [0.835, 1.048]	0.547 [0.418, 0.665]	1.110 [1.045, 1.262]
BC	0.216 [0.119, 0.321]	1.006 [0.852, 1.058]	0.601 [0.475, 0.724]	1.103 [1.044, 1.311]
Diffusion	0.342 [0.240, 0.445]	0.767 [0.589, 0.904]	0.253 [0.148, 0.369]	1.098 [1.017, 5.754]
CQL	0.517 [0.403, 0.623]	0.460 [0.260, 0.680]	0.249 [0.145, 0.369]	1.279 [1.054, 3.064]
Discrete (52 environments)				
BCP	0.138 [0.074, 0.209]	1.090 [0.982, 1.180]	0.646 [0.532, 0.759]	1.251 [1.156, 1.385]
BC	0.156 [0.090, 0.229]	1.041 [0.945, 1.119]	0.591 [0.470, 0.704]	1.199 [1.121, 1.315]
CQL	0.174 [0.103, 0.251]	1.018 [0.916, 1.113]	0.563 [0.444, 0.686]	1.236 [1.126, 1.358]
DT	0.216 [0.147, 0.286]	0.876 [0.795, 0.940]	0.287 [0.182, 0.409]	1.079 [1.047, 1.171]
BCQ	0.781 [0.692, 0.863]	0.082 [0.035, 0.179]	0.071 [0.020, 0.142]	1.162 [1.023, 16.977]

Table 6: Per-algorithm aggregate metrics on the top-10% of trials, normalized expert-relative as $(s-\text{random})/(\text{expert mean}-\text{random})$. Each cell shows the point estimate with a 95% non-parametric env-bootstrap interval ($B = 1000$, resampling environments with replacement). Algorithms within each block are sorted by Opt. gap (best first). Source for `rankings.pdf`.

F.2 Detailed sensitivity analysis

The heatmap is provided in Figure 10. The table below (linking to it breaks the pdf) contains a full breakdown of hyper-parameter sensitivities.

algorithm	env type	hyperparameter	mean share	cumulative
BC	continuous	learning_rate	0.28	0.28
		hidden_units	0.26	0.53
		dropout	0.22	0.75
		weight_decay	0.11	0.86
		policy_type	0.08	0.93
		clip_grad_norm	0.07	1.00
	discrete	learning_rate	0.37	0.37
		dropout	0.25	0.62
		hidden_units	0.12	0.73
		reg_factor	0.10	0.84
		weight_decay	0.09	0.92
		clip_grad_norm	0.08	1.00
BCP	continuous	learning_rate	0.21	0.21
		train_percentile	0.21	0.42
		hidden_units	0.21	0.63
		dropout	0.19	0.82
		weight_decay	0.07	0.89
		policy_type	0.07	0.96
		clip_grad_norm	0.04	1.00
	discrete	learning_rate	0.34	0.34
		dropout	0.20	0.54
		train_percentile	0.13	0.67
		reg_factor	0.12	0.79
		hidden_units	0.08	0.87
		weight_decay	0.07	0.94

continued on next page

algorithm	env type	hyperparameter	mean share	cumulative
		clip_grad_norm	0.06	1.00
BCQ	continuous	gamma	0.20	0.20
		imitator_learning_rate	0.15	0.35
		critic_learning_rate	0.13	0.48
		beta	0.09	0.57
		n_critics	0.07	0.64
		lam	0.07	0.71
		actor_learning_rate	0.06	0.77
		update_actor_interval	0.06	0.83
		n_action_samples	0.06	0.89
		rl_start_step	0.06	0.94
		tau	0.06	1.00
	discrete	actor_learning_rate	0.54	0.54
		beta	0.11	0.65
		gamma	0.10	0.75
		n_critics	0.09	0.84
		target_update_interval	0.09	0.93
		action_flexibility	0.07	1.00
CQL	continuous	alpha_threshold	0.27	0.27
		alpha_learning_rate	0.14	0.41
		critic_learning_rate	0.12	0.53
		gamma	0.10	0.63
		n_critics	0.10	0.73
		temp_learning_rate	0.07	0.80
		actor_learning_rate	0.06	0.87
		n_action_samples	0.05	0.92
		tau	0.05	0.96
		conservative_weight	0.04	1.00
	discrete	actor_learning_rate	0.76	0.76
		alpha	0.09	0.85
		n_critics	0.06	0.92
		target_update_interval	0.04	0.96
		gamma	0.04	1.00
DT	continuous	hidden_units	0.25	0.25
		learning_rate	0.21	0.46
		all_dropout	0.12	0.58
		weight_decay	0.11	0.69
		num_layers	0.10	0.79
		context_size	0.07	0.86
		gamma	0.05	0.91
		clip_grad_norm	0.05	0.96
		num_heads	0.04	1.00
	discrete	learning_rate	0.52	0.52
		hidden_units	0.09	0.61
		all_dropout	0.08	0.69
		context_size	0.07	0.76
		num_layers	0.06	0.82
		clip_grad_norm	0.05	0.87
		weight_decay	0.05	0.92
		gamma	0.04	0.96
		num_heads	0.04	1.00
IQL	continuous	actor_learning_rate	0.42	0.42
		gamma	0.11	0.54

continued on next page

algorithm	env type	hyperparameter	mean share	cumulative
		expectile	0.11	0.64
		critic_learning_rate	0.10	0.74
		weight_temp	0.09	0.84
		n_critics	0.09	0.92
		tau	0.08	1.00
REBRAC	continuous	actor_beta	0.62	0.62
		critic_beta	0.06	0.68
		critic_learning_rate	0.06	0.74
		gamma	0.06	0.80
		n_critics	0.05	0.85
		target_smoothing_sigma	0.04	0.89
		target_smoothing_clip	0.04	0.93
		actor_learning_rate	0.04	0.97
		tau	0.03	1.00
ACT	continuous	learning_rate	0.53	0.53
		action_horizon	0.15	0.68
		action_chunk_size	0.14	0.82
		clip_grad_norm	0.07	0.89
		kl_weight	0.06	0.94
		weight_decay	0.06	1.00
VQBET	continuous	learning_rate	0.29	0.29
		offset_learning_rate	0.19	0.48
		action_horizon	0.13	0.60
		action_chunk_size	0.09	0.69
		focal_gamma	0.06	0.75
		commitment_weight	0.04	0.79
		secondary_code_weight	0.04	0.83
		weight_decay	0.04	0.87
		primary_code_weight	0.04	0.90
		vqvae_steps_ratio	0.03	0.94
		offset_loss_weight	0.03	0.97
		clip_grad_norm	0.03	1.00
DIFFUSION	continuous	n_diffusion_steps	0.37	0.37
		num_inference_steps	0.26	0.62
		action_horizon	0.11	0.73
		learning_rate	0.10	0.84
		weight_decay	0.06	0.90
		action_chunk_size	0.06	0.96
		clip_grad_norm	0.04	1.00

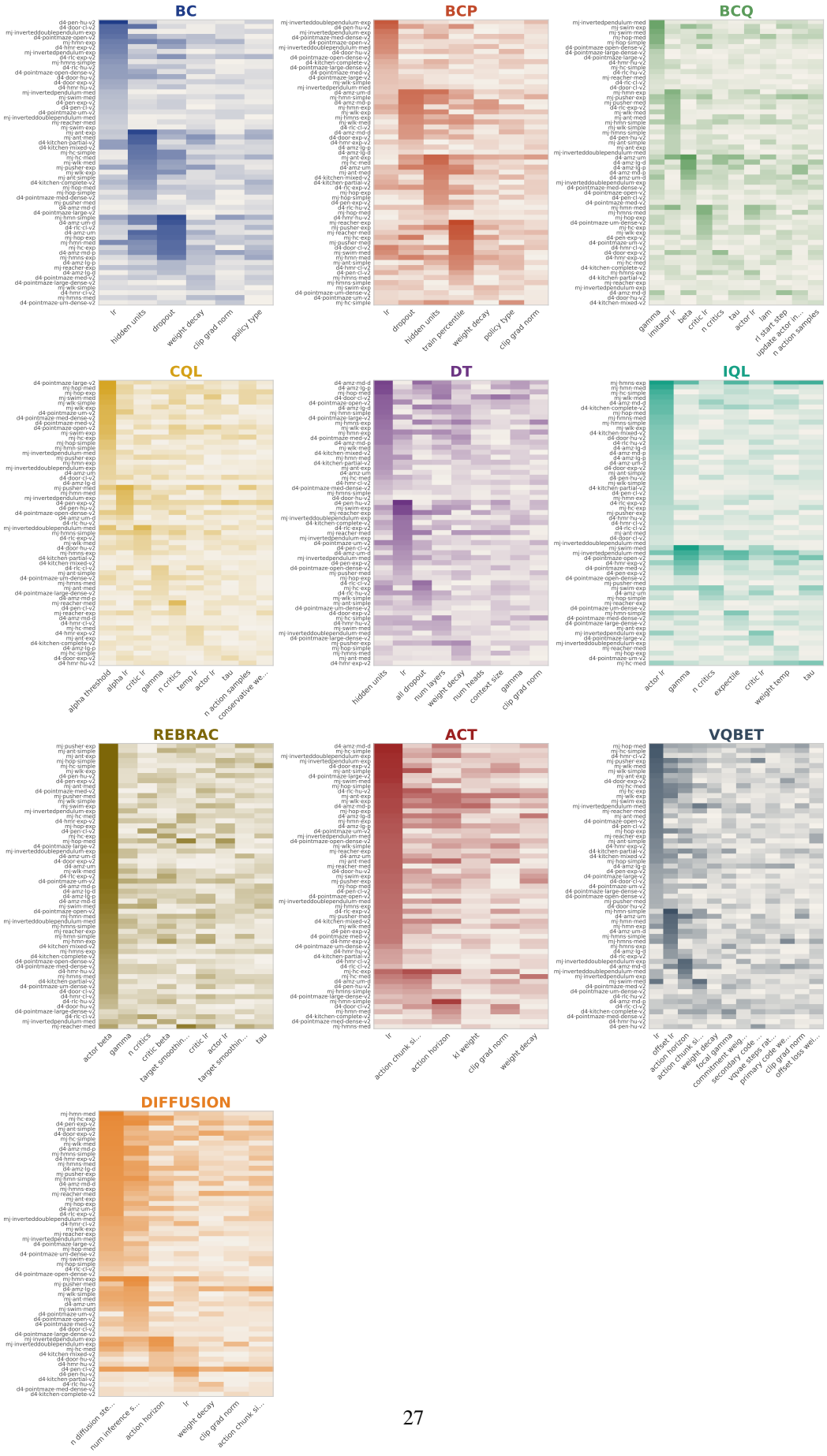

Figure 10: Heatmaps for sensitivity sorted by number of environments sensitive to a factor, then by the magnitude of the sensitivity.

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: All statements in the abstract are given a subsection, figure, table, or paragraph to address it.

   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 have provided a limitations section.

   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: No theoretical results.

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: All experimental details, raw numbers, and hyper-parameter ranges are in the appendix. We attach anonymized source code in the supplemental; datasets and website will be available upon acceptance due to risks of breaking double blind review and extreme scale making it difficult to ensure anonymity.

   Guidelines:

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

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

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

Answer: [Yes]

Justification: We have attached source code. Access to the full dataset of models and hyperparameter-reward data are too large to upload directly as supplemental material and will be fully available upon release. Currently, it is too expensive and risky to release the full dataset of models anonymously and we are not confident that some of it not contain identifying material. Data used to train the continuous models are publicly available.

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: We provide summaries in the main paper and detailed information in the appendix.

   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 have added 95CI and IQR wherever possible.

   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: While it is difficult to exactly compute the number of GPU hours, we do mention that we use a cluster with roughly 512-768 V100s available to us on a pre-emptible basis for somewhere around 500,000 GPU-hours for the entire project including failed runs, re-runs and launch overhead, and we multi-tenant training jobs. This is why our algorithm training time estimates may not be fully accurate, since we multi-tenant gpus for better overall efficiency.

   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: We have reviewed the code of ethics to make sure we are in compliance. We do not collect human data.

   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: [Yes]

Justification: We discuss in section 6

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: We do not believe there to be a risk to accessing our data or models.

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: All licenses are respected.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.

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

13. **New assets**

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

    Answer: [Yes]

    Justification: Data contains documentation.

    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: We do not crowdsource.

    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: We do not use 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: [N/A]

    Justification: We primarily use LLMs as a writing aid, they did not create original contributions to this research. LLM coding assistants were used in portions of the code.

    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.