Appendices

A Pseudocode

A.1 Pseudocode for policy switching online

Algorithm 1 Policy switching during fine-tuning

1: Compute σ_D using data-set
2: s_0 = env.reset()
3: **for** environment step $t = 0, 1 \ldots$ **do**
4: Sample RL action: $a_{RL} \sim \pi_{\phi}^{RL}(\cdot|s_t)$
5: Sample BC action: $a_{BC} \sim \pi_{\theta}^{BC}(\cdot|s_t)$
6: $Q_{RL} = Q_{\psi}^{\text{med}}(s, a_{RL}) - f(\sigma_D) * \sigma_Q(s, a_{RL})$
7: $Q_{BC} = Q_{\psi}^{\text{med}}(s, a_{BC})$
8: **if** $Q_{RL} \geq Q_{BC}$ **then**
9: $a_t = a_{RL}$
10: **else**
11: $a_t = a_{BC}$
12: **end if**
13: $a_t \leftarrow a_t + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \sigma_\epsilon), -c, c)$
14: Act in environment: $r_t, s_{t+1} \leftarrow \text{env.step}(a_t)$
15: Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
16: Sample N transitions $\{(s, a, r, s')\} \sim \mathcal{D}$
17: Use procedure defined by lines 4-12 to select $a' \in \{a'_{RL}, a'_{BC}\}$
18: $a' \leftarrow a' + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \sigma_\epsilon), -c, c)$
19: Sample a set $\mathcal{B}$ of two distinct indices from $\{1, 2, \ldots, N\}$
20: y $\leftarrow r_i + \gamma \min_{j \in \mathcal{B}} Q_{\psi'_j}(s', a')$
21: Update critics

$$\psi_j \leftarrow \arg\min_{\psi_j} N^{-1} \sum (y - Q_{\psi_j}(s, a))^2$$

22: **if** t mod d **then**
23: $\nabla_\phi J(\phi) = \frac{1}{N} \sum \nabla_a Q_{\psi}^{\min}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
24: where $Q_{\psi}^{\min}(s, a) = \min_j Q_{\psi_j}(s, a)$
25: **end if**
26: **end for**

A.2 Pseudocode for policy switching offline

Algorithm 2 Policy switching

Compute σ_D using dataset
Train RL policy π_{RL} and BC policy π_{BC} independently using dataset
s_0 = env.reset()
for environment step $t = 0, 1 \ldots$ **do**
  Sample RL action: $a_{RL} \sim \pi_{RL}(\cdot|s_t)$
  Sample BC action: $a_{BC} \sim \pi_{BC}(\cdot|s_t)$
  Compute uncertainty adjusted critic value for reinforcement learning action:

$$Q_{RL} = Q_{\text{med}}(s, a_{RL}) - f(\sigma_D) * \sigma_Q(s, a_{RL})$$

  Compute average value estimate for BC action

$$Q_{BC} = Q_{\text{med}}(s, a_{BC})$$

  if $Q_{RL} \geq Q_{BC}$ **then**
    $a = a_{RL}$
  else
    $a = a_{BC}$
  end if
  Act in environment: $r_t, s_{t+1} \leftarrow$ env.step(a)
end for

B Experimental details

B.1 Hardware and software specifications

Hardware We run all tests on a single GeForce RTX3080 and provide the details of computational cost for training each of the algorithms of our method on hopper medium in Table 1. We also provide the computational cost of TD3-BC-N, IQL and Cal-QL as a comparison.

Software We use the following software versions

- Python 3.10.0
- Pytorch 2.3.0+cu121
- Gym 0.26.2
- mujoco-py 2.1.2.14

B.2 Baselines

Cal-QL We use the JAX implementation of Cal-QL[1] provided by the original authors in recreating the results the offline online results used for the AntMaze datasets. We

[1] https://github.com/nakamotoo/Cal-QL

	Runtime (s/epoch*)	GPU Mem. (MiB)
TD3-N (N=10)	77	334
BC	38.2	294
TD3-BC-N (N=10)	78.2	334
IQL	138.9	530
Cal-QL	420.6	806

* 1 epoch = 10,000 gradient steps

Table 1: Computational cost of each method

use the same hyper-parameters suggested by the authors for generating results for the umaze and umaze-diverse tasks. For the MuJoCo tasks the authors do not specify hyper-parameters, in this case we opt to use the CQL hyper-parameters specified in the original paper for the MuJoCo tasks and kept any additional hyper-parameters introduced by Cal-QL hyper-parameters fixed. For the MuJoco tasks we use the implementations provided in Clean Offline Reinforcement Learning[2] (CORL) repository. We take the offline **CQL** scores for MuJoCo and AntMaze datasets directly from CORL. **IQL** We use the JAX implementation for IQL[3] provided by the original author to generate the offline score and fine tuning curves for both AntMaze and MuJoCo tasks. **O3F** We directly use O3F ontop of our implementation of TD3-N to generate fine-tuning curves. **AWAC** We use the implementation provided in CORL alongside the hyper-parameters recommended by the original authors to generate fine-tuning curves. **TD3-BC-N** We use our own implementation and use the hyper-parameters recommended by the authors for all datasets both offline and during online fine-tuning.

B.3 Policy switching hyperparameters

We list all hyperparameters used for our policy switching technique that include those required for training the base RL and BC algorithm as well as the policy switching hyperparameters in Table 2. For AntMaze, we specify the hyperparameter m=[1,0.5] for policy switching as we use a value of m=1 for all datasets bar AntMaze large-play where we use a value of 0.5.

[2]https://github.com/tinkoff-ai/CORL

[3]https://github.com/ikostrikov/implicit_q_learning

	Hyperparameter	Value
TD3-N Hyperparameters	Optimiser	Adam
	Critic learning rate	5e-3
	Actor learning rate	5e-3
	Mini-batch size	256
	Target update rate	1e-3
	Policy noise	0.2
	Policy noise clipping	(-0.5,0.5)
	Policy update frequency	2
	Ensemble size	10
TD3-N Architecture	Critic hidden dim	256
	Critic hidden layers	2
	Critic activation function	ReLU
	Actor hidden dim	256
	Actor hidden layers	2
	Actor activation function	ReLU
BC hyperparameters	Optimiser	Adam
	BC learning rate	5e-4
BC Architecture	Hidden dim	256
	Hidden layers	2
	Activation function	ReLU
Policy switching hyper-parameters	AntMaze	m = [1, 0.5]
		$\alpha = 3300$
		$\gamma = 0.999$
	MuJoCo	m = 1
		$\alpha = 0.075$
		γ = 0.99

Table 2: List of hyperparameters used.

C Additional Experiments

C.1 Hyperparameter tuning

We investigate the effects of adjusting hyper-parameters. Our main aim throughout tuning was to keep the value of $m = 1$ and adjusting the value of α We provide the results of adjusting the hyper-parameters on the MuJoCo dataset in Table 3.

From the results we see that lower values of α have a slight preference to using more RL than higher values of α and as a result do better on med-rep datasets and worse on exp datasets.

C.2 Gaussian BC vs Vanilla BC

We observe slightly better performance when using a gaussian BC (BC) agent over vanilla BC (VBC) agent. In theory both should converge to the same solution we look

	0.025	0.075	0.15
hopper-med-rep	99.9	99.7	99.8
hopper-med	104.1	101.8	101.2
hopper med exp	72.4	88.6	100.5
hopper exp	25	37.7	53.2
halfcheetah-med-rep	62.1	61.4	60.8
halfcheetah-med	64.1	63	62.3
halfcheetah-med-exp	97.6	93.6	98.5
halfcheetah-exp	82	94.1	88.2
walker2d-med-rep	87.3	86.9	86.5
walker2d-med	99.4	98.2	97
walker2d-med-exp	116.4	116	115.3
walker2d-exp	96.5	105.4	100.3

Table 3: MuJoCo scores for different values of α. Scores are averaged over 5 seeds.

at a comparison of the results

	VBC	BC	TD3-N+VBC(PS)	TD3-N+BC(PS)
hopper-med-rep	29.2	26.6	99.7	99.7
hopper-med	53.9	51.5	102.6	101.8
hopper-med-exp	60	51.3	82.4	88.6
halfcheetah-med-rep	109.9	110.6	20.7	37.8
halfcheetah-med	41.8	41.9	63	63
halfcheetah-med-exp	64.4	60.1	100.6	93.6
halfcheetah-exp	92.6	92.9	81.1	94.1
walker2d-med-rep	21.5	23.5	86.7	86.9
walker2d-med	67.8	71.8	95.3	98.2
walker2d-med-exp	95.5	107.7	116	115.9
walker2d-exp	108.1	108.1	99.5	105.44
antmaze-umaze	72	86	87.7	93.7
antmaze-umaze-diverse	73	87.3	91.7	94
antmaze-medium-play	0	0	63.3	63.3
antmaze-medium-diverse	0	0	68.3	71
antmaze-large-play	0	0	47.3	41
antmaze-large-diverse	0	0	48.3	49.7

Table 4: Comparison of results when using Vanilla BC (VBC) and Gaussian BC (BC) directly with the dataset and within our policy switching method. Scores are averaged over 5 seeds for MuJoCo datasets and 3 seeds for AntMaze datasets.

The results in Table 4 indicate that there is a similar level of improvement when using a Gaussian BC or vanilla BC agent within our policy switching method. For results in the main paper we opted to report the results from integrating the Gaussian BC agent.

C.3 All datasets for offline to online fine-tuning

We provide the full results for offline to online fine-tuning across all datasets here. Figure 1 presents the fine-tuning curves for all MuJoCo datasets and Figure 2 shows how the proportion of BC and standard deviation evolves during fine-tuning for the same MuJoCo datasets. Figure 3 and 4 are the equivalent plots for the AntMaze datasets.

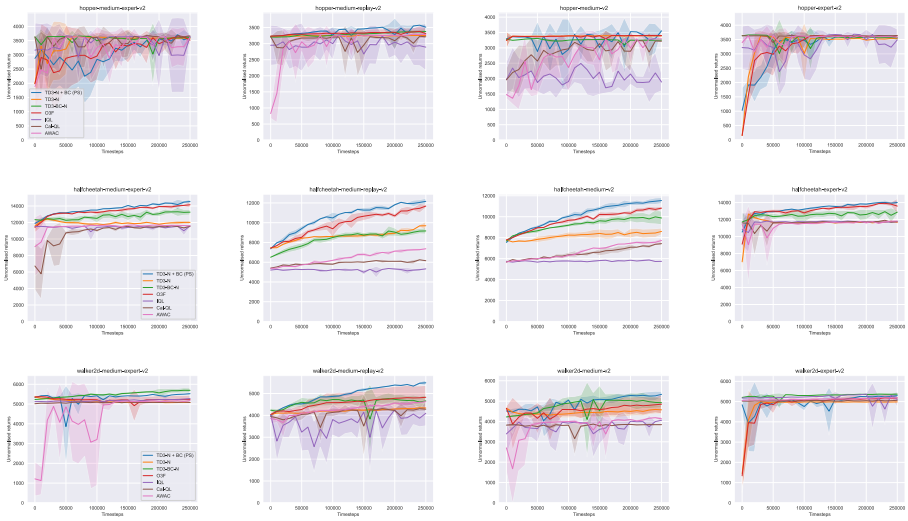

Figure 1: Fine tuning curves for all datasets available for the MuJoCo environment in the D4RL suite. All algorithms use 250k training steps evaluated every 10k steps across 5 seeds and 10 episodes per seed

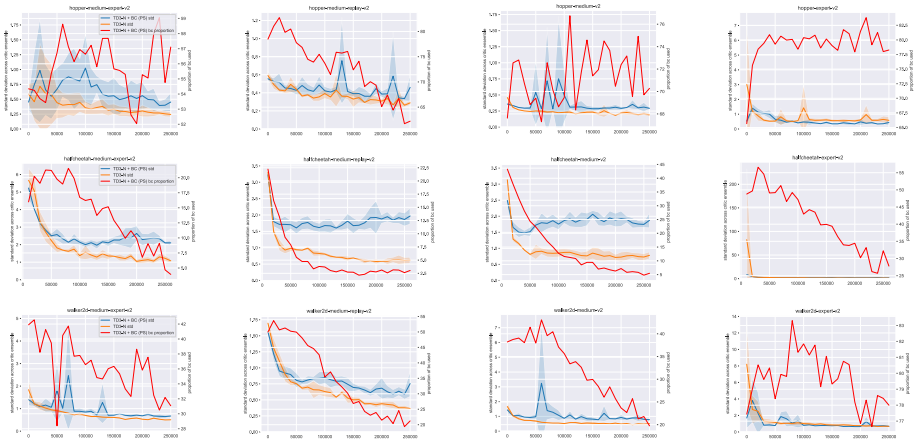

Figure 2: Corresponding evolution of critic ensemble variance and amount of behavioural cloning used for the duration of online fine-tuning for all MuJoCo datasets

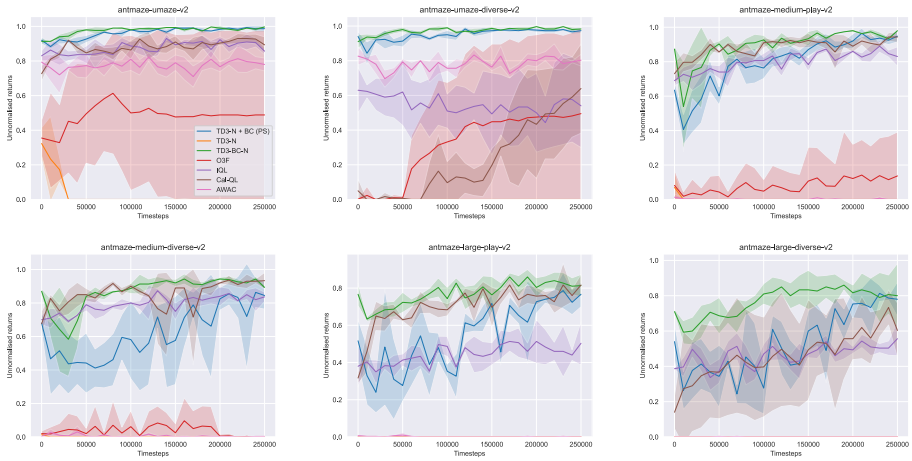

Figure 3: Fine tuning curves for all datasets available for the AntMaze environment in the D4RL suite. All algorithms use 250k training steps evaluated every 10k steps across 3 seeds and 100 episodes per seed

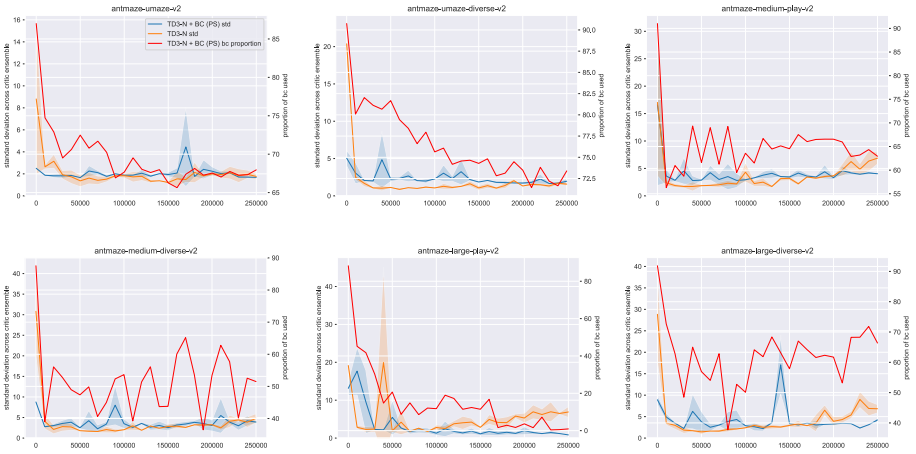

Figure 4: Corresponding evolution of critic ensemble variance and amount of behavioural cloning used for the duration of online fine-tuning for all AntMaze datasets