ENSEMBLE VALUE-BASED REINFORCEMENT LEARNING THROUGH THE LENS OF MODE CONNECTIVITY

Anonymous authors
Paper under double-blind review

ABSTRACT

Ensemble methods are a common solution for improving the stability and performance of value-based reinforcement learning (RL) algorithms. However, conventional ensembles operate in the output space and require maintaining multiple models during inference, resulting in inference cost that grows linearly with the ensemble size. In this work, we study the geometric structure of the objective landscape in value-based RL through the lens of mode connectivity and analyze how trained Q-networks relate in parameter space. Our analysis reveals conditions under which weight-space ensembling becomes effective, how linear connectivity can emerge between Q-networks, and which RL-specific conditions can break this connectivity. Building on these insights, we propose *Q-Soup*, a weight-averaging approach that merges multiple Q-networks trained under identical or diverse configurations into a single model. Empirical results on the Atari benchmark show that Q-Soup achieves performance comparable to traditional output-space ensembles while maintaining a single model at inference time. To the best of our knowledge, this work presents the first systematic investigation of mode connectivity and weight-space ensembling in value-based RL.

1 INTRODUCTION

Ensemble learning in value-based Reinforcement Learning (RL) (Sutton & Barto, 1998; Mnih et al., 2013; 2015) is a common strategy for improving stability and performance, in which multiple value estimators are trained and their predictions combined to produce more reliable decisions. Ensembles have been shown to reduce estimation errors (Lan et al., 2020; Liang et al., 2022) and enhance exploration (Osband et al., 2016; Chen et al., 2017; Lee et al., 2021) by leveraging multiple Q-networks. However, ensemble-based approaches typically require maintaining several models during inference to aggregate their predictions during action selection, increasing computational and memory costs. Moreover, such methods combine predictions in the output space, leaving largely unexplored how the underlying Q-networks relate to one another in parameter space.

Recent studies on the geometry of neural network loss landscapes reveal that independently trained models are often not isolated solutions, but instead lie within connected regions of the parameter space. This phenomenon, known as *mode connectivity*, shows that distinct solutions can be connected by *non-linear* low-loss paths in weight space (Garipov et al., 2018; Draxler et al., 2018). While independently trained models exhibit barriers along direct linear paths, *linear mode connectivity* can emerge under particular training conditions (Frankle et al., 2020). These findings led to a growing line of research exploiting such geometric properties for practical model combination. For instance, model averaging methods such as Stochastic Weight Averaging (Izmailov et al., 2018) and Model Soups (Wortsman et al., 2022) demonstrate that averaging the parameters of multiple trained models can produce a single model with improved generalization and performance. These developments suggest that understanding the relationships between models in parameter space can provide a powerful alternative to traditional output-space ensembling. However, in RL, such geometric perspectives have primarily been studied in the context of policy networks (Gaya et al., 2022; 2023; Hendawy et al., 2025), while the connectivity between Q-networks for weight-space ensembling remains largely unexplored.

In this work, we address this gap by systematically studying the geometric structure of the objective landscape in value-based RL and analyzing how ensembles of Q-networks relate in parameter space.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

This analysis reveals when *linear mode connectivity* may emerge, when weight-space ensembling is effective in RL, and which RL-specific conditions may cause it to fail. Leveraging this geometric perspective, we investigate whether parameter averaging can provide an efficient alternative to traditional output-space ensembles, which require maintaining multiple models at test time. The main contributions of this work are: (i) an analysis of the value-based RL objective through the lens of mode connectivity, characterizing when Q-networks become linearly connected, when weight-space ensembling is effective, and which RL-specific factors can break this connectivity; (ii) a model averaging approach for value-based RL, termed *Q-Soup*, that averages multiple Q-networks trained under *identical* or diverse training configurations into a single model; and (iii) an extensive empirical evaluation showing that Q-Soup achieves performance comparable to output-space ensembles with a single model at inference across multiple value-based RL methods, and that it can also be applied to existing ensemble-based methods whose members are trained simultaneously.

2 Related Work

Ensemble learning is widely used in RL, particularly in value-based methods where function approximation introduces bias and variance in value estimates. Early work by Wiering & Van Hasselt (2008) demonstrated that combining multiple value estimators improves policy robustness. In deep RL, Osband et al. (2016) proposed Bootstrapped Deep Q-Network (DQN), which maintains Q-networks trained on bootstrapped data to capture epistemic uncertainty and encourage exploration. Similarly, ensemble variance has been used to construct upper confidence bounds for action selection (Chen et al., 2017). Ensembles have also been applied to reduce estimation bias. For instance, Maxmin Q-learning (Lan et al., 2020) computes targets using the minimum across multiple Q-functions, thereby mitigating overestimation bias. More recently, Hendawy et al. (2026) ensemble the online and target networks during training to produce more up-to-date value estimates. Overall, these approaches demonstrate that ensemble diversity improves the stability and performance of value-based RL agents. However, they operate primarily in the Q-value (output) space (Agarwal et al., 2020) and provide limited insight into how Q-networks interact in parameter space. Moreover, they typically require storing several models during inference.

Mode connectivity (Garipov et al., 2018; Draxler et al., 2018; Mirzadeh et al., 2020) studies the geometric structure of neural network loss landscapes and the relationships between different solutions in weight space. Early work by Garipov et al. (2018) and Draxler et al. (2018) showed that independently trained networks can often be connected by simple *non-linear* paths along which the loss remains low. Subsequent studies further investigated the existence of *linear* connections between solutions. Frankle et al. (2020) demonstrated that networks sharing part of their training trajectory can be linearly connected without significant performance barriers. Similarly, Neyshabur et al. (2020) observed that solutions sharing a pretrained initialization often lie within the same connected region of parameter space. Mode connectivity has also been explored in multi-task and continual learning settings. Mirzadeh et al. (2020) showed that solutions obtained from multi-task and sequential training remain linearly connected when sharing the same initialization, and leveraged this observation to design methods that constrain new solutions to remain connected to previous ones. **Mode connectivity in RL** has received comparatively limited attention. Gaya et al. (2022) learn a linear subspace of policies in parameter space, where convex combinations of anchor policies produce diverse yet high-performing behaviors for efficient adaptation to changing environment dynamics. Similarly, Gaya et al. (2023) introduce the Continual Subspace of Policies (CSP), which incrementally expands a policy subspace to address the scalability–plasticity trade-off while mitigating catastrophic forgetting. In addition, Hendawy et al. (2025) exploit mode connectivity in multi-task RL by enforcing a linear connection between single-task policies and a shared multi-task policy, enabling knowledge sharing and improving robustness across tasks. However, existing studies investigate mode connectivity exclusively among policy networks. To the best of our knowledge, this work provides the first systematic investigation of mode connectivity in value-based RL, focusing on how Q-networks are connected in weight space.

Model averaging is a widely used technique for combining multiple models into a single solution without increasing inference cost. Classical examples include Stochastic Weight Averaging (SWA) (Izmailov et al., 2018), which averages model parameters along an optimization trajectory to obtain flatter solutions with improved generalization. More recently, Model Soups (Wortsman et al., 2022) demonstrated that averaging differently configured models can produce a single model that matches or outperforms the individual ones. This idea has gained significant momentum in

the large language model (LLM) domain, where model merging techniques combine specialized models through weight-space operations such as parameter averaging or task arithmetic (Matena & Raffel, 2022; Ilharco et al., 2022). Inspired by these developments, we study weight averaging in value-based RL and analyze how weight-space geometry enables effective Q-network averaging, achieving ensemble-level performance without maintaining multiple models at inference time.

3 PRELIMINARIES

Reinforcement Learning. We formulate the problem as a Markov Decision Process (MDP) (Puterman, 1990), denoted by $< \mathcal{S}, \mathcal{A}, P, r, \mu, \gamma >$. Here, $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ the transition distribution, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward function, μ the initial-state distribution, and $\gamma \in [0, 1)$ the discount factor. A policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ assigns a probability distribution over actions for each state. Its action-value function is $Q^\pi(s, a) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s, a_0 = a]$. The goal is to maximize the expected discounted return $J(\pi) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)]$. Value-based RL methods learn an action-value function that directly determines the policy. Q-learning estimates the optimal action-value function Q^* using the Bellman optimality equation (Bellman, 1957), $Q^*(s, a) = \mathbb{E}[r(s, a) + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a')]$. The optimal policy selects actions greedily, $\pi^*(s) = \text{argmax}_{a \in \mathcal{A}} Q^*(s, a)$. In deep RL, the action-value function is approximated by a neural network $Q_{\boldsymbol{\theta}}(s, a)$. DQN (Mnih et al., 2013; 2015) stabilizes Q-learning with neural networks using replay buffers and target networks. More recently, Gallici et al. (2025) proposed Parallelised Q-Network (PQN), which removes both mechanisms and instead uses layer normalization (Ba et al., 2016) and λ-returns (Sutton, 1988). Algorithms 1 and 2 detail both training procedures.

Mode connectivity describes when distinct neural network solutions can be connected by continuous paths in parameter space that preserve high performance (or low loss). While commonly studied along one-dimensional paths, the concept extends to higher-dimensional subspaces, such as simplices. We parameterize such subspaces using a set of solutions $\{\boldsymbol{\theta}_i\}_{i=1}^k$, referred to as *anchors*. Points within their simplex are given by $\boldsymbol{\theta}(\boldsymbol{\alpha}) = \sum_{i=1}^k \alpha_i \boldsymbol{\theta_i}$, where $\alpha_i \geq 0$ and $\sum_{i=1}^k \alpha_i = 1$. For two anchors, this reduces to the linear path $\boldsymbol{\theta}(\alpha) = (1 - \alpha)\boldsymbol{\theta}_1 + \alpha\boldsymbol{\theta}_2$ with $\alpha \in [0, 1]$. To assess mode connectivity, we adapt the barrier metric of (Frankle et al., 2020) to the RL setting as the Performance Barrier Depth D. Here, $D = 0$ indicates that performance does not fall below the weaker endpoint along the path. We report D normalized by the mean performance of the anchors.

Definition 3.1 (Performance Barrier Depth) *Given the objective J and a path $\boldsymbol{\theta}(\cdot)$, $D(\boldsymbol{\theta}(\cdot))$ is defined as:*

$$D(\boldsymbol{\theta}(\cdot)) = \min\left[J(\boldsymbol{\theta}(0)), J(\boldsymbol{\theta}(1))\right] - \min_{\alpha \in [0,1]} J(\boldsymbol{\theta}(\alpha)). \tag{1}$$

4 ENSEMBLE VALUE-BASED RL VIA MODE CONNECTIVITY

In this section, we analyze the geometric structure of the objective landscape in value-based RL and examine the relationship between Q-networks in parameter space. Before presenting our analysis, we highlight a well-known yet important observation from the literature that motivates the use of ensembles in value-based RL.

Ensembling models in the output space is a widely adopted technique in RL, particularly for value-based methods where function approximation introduces estimation bias and variance. A large body of work demonstrates that ensembles can significantly improve both performance and training stability. Since independently trained models often converge to different local optima, their prediction errors tend to be only weakly correlated. Aggregating their predictions therefore exploits this diversity, smoothing estimation noise and reducing the risk that the greedy maximization step amplifies isolated overestimation. The following observation (4.1) formalizes this effect, showing that averaging action-value estimates directly usually leads to an improved policy, hence higher return.

Observation 4.1(Ensembling in Output Space). We observe that ensembling $Q_{\boldsymbol{\theta}_1}$ and $Q_{\boldsymbol{\theta}_2}$ in output space, by averaging their Q-values $Q_{\text{avg}} = \frac{1}{2}(Q_{\boldsymbol{\theta}_1}(s, \cdot) + Q_{\boldsymbol{\theta}_2}(s, \cdot))$ often increases J, the expected return of the policy of the ensembled Q-function. This is attributed to functional diversity, i.e. two models making different errors that can be mitigated through averaging.

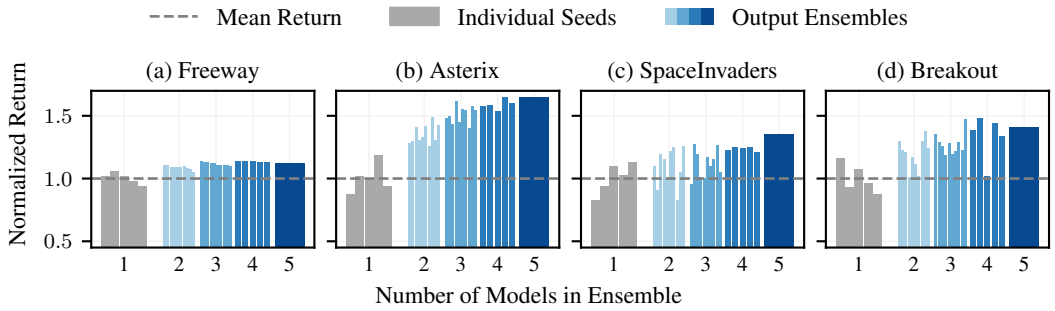

Figure 1: Empirical demonstration of how ensembling in output space can increase return across four MinAtar environments. We show the normalized returns of five independently trained DQN agents (grey) and all their output-space ensemble combinations (blue shades). Returns are normalized by the mean return of the five individual seeds.

The observation is empirically supported by the results shown in Fig. 1. In this experiment, we train agents on four environments from the MinAtar benchmark (Young & Tian, 2019) using DQN as the learning algorithm. Consistent with Observation 4.1, aggregating multiple Q-value estimates overall improves the expected return of the agent. For some environments, the benefit of increasing the ensemble size is clearly visible. In a few cases, strong disagreement between models can instead lead to performance degradation. A practical limitation of output-space ensembling, however, is that *it requires all ensemble members to be maintained and evaluated at inference time*.

To address this limitation, one could instead consider ensembling in the weight space by averaging the parameters of multiple models. This would eliminate the need to keep and execute all ensemble members at inference time. However, weight-space averaging is generally ineffective when applied to independently trained networks. The optimization landscape of neural networks is highly non-convex and contains numerous local optima that are often isolated from one another. In other words, these solutions are rarely linearly connected. This phenomenon has been extensively documented in supervised learning (Garipov et al., 2018; Draxler et al., 2018), where linear interpolation between independently trained networks typically passes through regions of significantly higher loss.

The limitations of weight-space ensembling are illustrated in Fig. 2. In this experiment, we construct a two-dimensional plane containing the parameter vectors of three independently trained Q-networks. The heatmaps in Fig. 2 show the performance across this plane, revealing that the high-return solutions are separated by regions of substantially lower return. In particular, the midpoints between these parameter solutions exhibit very poor performance. This observation provides empirical evidence that the optimization landscape is highly non-convex and that independently discovered optima are not linearly connected. These findings are summarized in Observation 4.2. This observation naturally raises two questions: *Q1. Is weight averaging always ineffective? If not, Q2. Under what conditions can it succeed?*

Observation 4.2(Ensembling in Weight Space). We observe that ensembling $Q_{\boldsymbol{\theta}_1}$ and $Q_{\boldsymbol{\theta}_2}$ in weight space, by averaging their parameters $Q_{\boldsymbol{\theta}_{\text{avg}}} = Q_{\frac{1}{2}(\boldsymbol{\theta}_1+\boldsymbol{\theta}_2)}(s,\cdot)$, fails. The resulting model performs significantly worse than either of the original models, indicating that solutions are not linearly connected in weight space, and that the optimization landscape is highly non-convex with multiple isolated minima.

Weight averaging can be effective under specific geometric conditions in the optimization landscape. In particular, weight averaging becomes viable when multiple local optima are linearly connected, forming a flat region in parameter space. To investigate this setting, we design a subspace learning approach that targets such flat optima, inspired by Wortsman et al. (2021). The proposed method, named SubQ, learns a set of Q-networks whose parameters define the vertices of a simplex in weight space. During training, interpolation weights are sampled to construct models within this simplex, which are then optimized using the RL objective. This procedure encourages the entire subspace to maintain strong performance, forming a flat region in parameter space. More details on SubQ and its adaptation to various RL algorithms are provided in Appendix A.3.

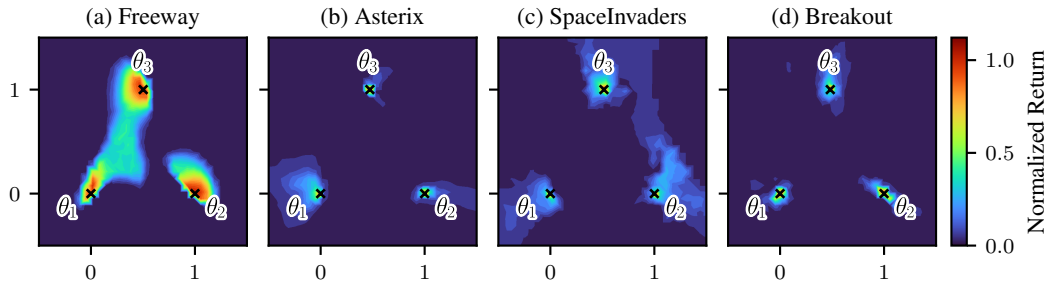

Figure 2: Empirical demonstration of the parameter landscape around three independently trained DQN agents ($\times$) across four MinAtar environments. We construct a two-dimensional plane defined by the parameters of the three agents and show the normalized returns of the corresponding policies across this plane. Returns are normalized by the mean return of the three individual agents.

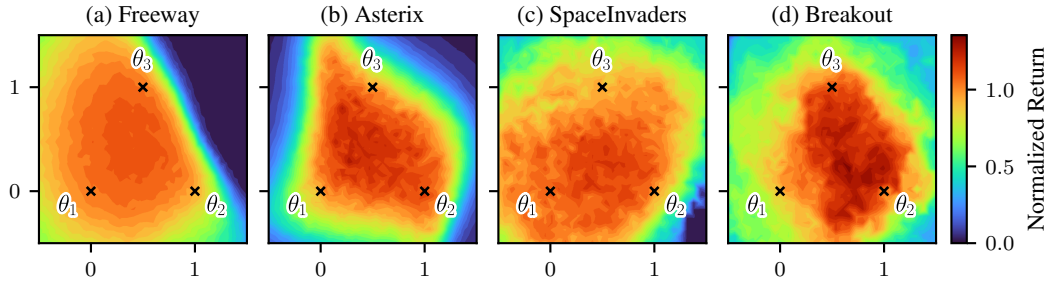

Figure 3: Empirical demonstration of the flat regions induced by simplex subspaces learned with SubQ (*SubDQN*) across four MinAtar environments. We show subspaces defined by three trained agents ($\times$), over which the heatmaps show the normalized return of the corresponding policies. Returns are normalized by the mean return of the three individual agents.

To empirically validate the existence of such flat regions, we visualize the parameter landscape by constructing the simplex subspace defined by the learned Q-network parameters, as illustrated across several MinAtar environments in Fig. 3. When weight averaging is performed over parameters located on the periphery of a flat optimum, the resulting model lies near the center of this region. This centralization can produce smoother action-value estimates and typically yields higher expected returns than the individual models. Consequently, averaging weights within a connected flat region can improve the overall performance of the agent. We summarize this finding in Observation 4.3, while the empirical comparison between the weight-averaged model and the individual Q-networks is shown in Fig. 4.

Observation 4.3(Weight Averaging in Flat Regions). We observe that averaging Q-network parameters $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_k$ located on the periphery of a connected flat region typically results in higher expected return than their average performance, i.e., $J(\frac{1}{k}\sum_{i=1}^{k}\boldsymbol{\theta}_i) > \frac{1}{k}\sum_{i=1}^{k} J(\boldsymbol{\theta}_i)$, where $J(\boldsymbol{\theta})$ denotes the expected return of the policy induced by $Q_{\boldsymbol{\theta}}$.

The previous analysis answers our questions *Q1* and *Q2*, demonstrating that weight averaging can be effective and identifying the geometric conditions under which it succeeds. This naturally leads to a follow-up question: *Q3. When can Q-networks naturally become linearly connected in parameter space?* We are motivated by prior work in supervised learning (Frankle et al., 2020; Mirzadeh et al., 2020), which shows that linear connectivity can emerge when trained models share portions of their optimization trajectories. This suggests that models are more likely to be linearly connected when they originate from the same pretrained initialization.

Motivated by these findings, we investigate whether this condition is sufficient for Q-networks to exhibit linear connectivity in online RL, despite challenges such as *non-stationarity* and *bootstrapping*. To study this, we consider the same set of MinAtar environments and let models share an

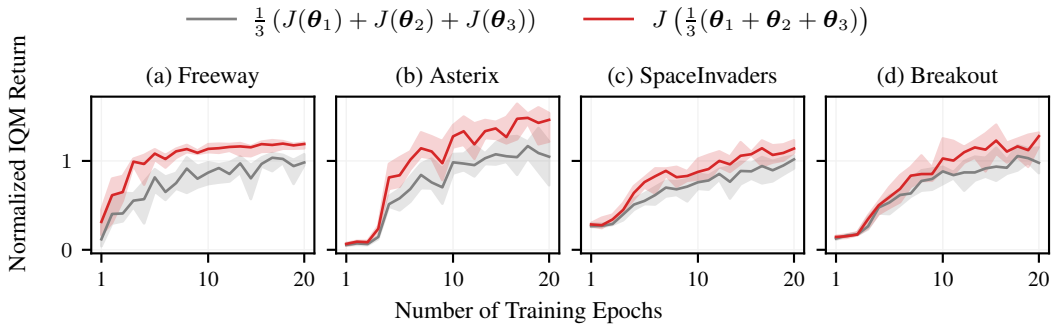

Figure 4: Empirical demonstration of the flat region observation across four MinAtar environments. We show the normalized return of the weight-averaged model (red) and the average return of the three individual agents (grey) over training, reported as the IQM with a 95% confidence interval across five seeds. Returns are normalized by the mean return of the three individual agents.

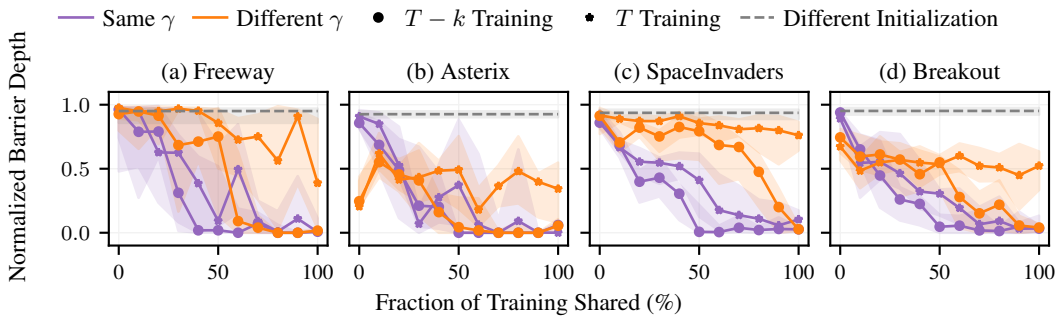

Figure 5: Normalized barrier depth as a function of the fraction of training shared for DQN across four MinAtar environments, reported as the IQM with a 95 % confidence interval across five seeds. After sharing k training steps, two models are trained independently either for the remaining $T - k$ steps or for an additional T steps. We compare runs using the same or different discount factors γ after splitting, with independently initialized models shown as a baseline.

initial segment of the training trajectory before proceeding to independent training processes with different sources of SGD noise. Concretely, we first train a DQN agent for a number of training steps and then use the resulting Q-network weights to initialize two separate training runs. As in Fig. 5, we evaluate linear mode connectivity using the performance barrier depth metric described in Sec. 3. We show that linear connectivity emerges after the models share a portion of the training trajectory, even in the presence of the non-stationarity and bootstrapping of online value-based RL, consistent with observations in supervised learning. These findings answer question *Q3*, demonstrating that linear connectivity observed in supervised learning also emerges in value-based RL.

We also investigate several RL-specific factors that may affect this behavior, as shown in Sec. D. In particular, connectivity often breaks when the two training runs use different discount factors after diverging. Since the discount factor defines the return objective and hence the value function, the two models continue learning different problems. As mode connectivity is defined w.r.t. the objective landscape, the resulting solutions are therefore no longer optimized within the same landscape, despite sharing part of their training trajectory. The overall finding is stated in Observation 4.4.

Observation 4.4(Linear Mode Connectivity). We observe that linear connectivity emerges when Q-networks share part of their optimization trajectory. Models trained independently from shared parameters $\boldsymbol{\theta}^t$ converge to linearly connected solutions, but this connectivity can break when they use different discount factors and thus optimize different return objectives.

5 Q-SOUP

Building on the analysis presented in the previous section, we now introduce a weight-space ensembling approach for value-based RL. In particular, Observation 4.3 shows that averaging Q-network parameters within a flat optimum can improve policy performance, while Observation 4.4 identifies when linear connectivity can emerge between Q-networks. Our method leverages these properties to ensemble Q-networks directly in parameter space, improving performance without requiring multiple Q-networks at inference time.

5.1 MODEL AVERAGING

The mechanics of our approach are as follows. To obtain linearly connected solutions, we follow Observation 4.4 by training an agent for several epochs. The resulting pretrained Q-network parameters, denoted $\boldsymbol{\theta}_{\text{pre}}$, are used to initialize independent finetuning runs with different SGD noise (random seeds). This produces a set of Q-networks with parameters $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_k$ that are linearly connected in parameter space. Following Observation 4.3, averaging these parameters yields a model with improved expected return. The averaged parameters $\boldsymbol{\theta}_{\text{avg}}$ define a single Q-network used to derive the (ϵ-)greedy policy during inference: $a^* = \arg\max_a Q_{\boldsymbol{\theta}_{\text{avg}}}(s, a)$ where $\boldsymbol{\theta}_{\text{avg}} = \frac{1}{k}\sum_{i=1}^{k} \boldsymbol{\theta}_i$.

5.2 HETEROGENEOUS MODEL AVERAGING AND PROXY Q-SOUP

RL algorithms involve many interacting components and tunable hyperparameters, making it challenging to identify a single configuration that performs well across tasks or environments. Inspired by the Model Soups framework proposed in Wortsman et al. (2022), we extend homogeneous model averaging to exploit diversity across training configurations. Instead of finetuning multiple Q-networks with *identical* settings (homogeneous), we allow each run to use a *different* configuration (heterogeneous), including RL-specific parameters such as the target update frequency in DQN or the λ-return coefficient in PQN. The diversity in configurations can introduce more variation in the learned solutions than in the homogeneous setting, potentially providing more diverse ingredients for weight averaging. We refer to our model averaging approach as *Q-Soup*. Each finetuned Q-network represents an *ingredient* of the soup, and the final model is obtained by *uniformly* averaging their parameters as described above.

However, some RL-specific factors can break linear connectivity. In Observation 4.4, we observe that using different discount factors across finetuned models can introduce a performance barrier, whose depth varies depending on which discount factors are used and which environment is considered. Wortsman et al. (2022) account for differences in ingredient quality with a greedy model soup, incrementally adding models only when they improve accuracy on a held-out dataset. In RL, however, evaluating each ingredient requires additional environment interactions, which can be costly or restricted. We therefore propose *Proxy Q-Soup*, a variant of our approach, which uses training returns as a proxy for ingredient quality without additional evaluation. Over a fixed window, we first aggregate the training returns of each ingredient and then standardize the resulting scores across ingredients using Z-scores to identify weak ingredients that may compromise linear connectivity. Based on these scores, we filter ingredients and compute soft weights for parameter averaging, reducing the influence of weak ingredients that may have diverged from the flat region during finetuning. More details on Proxy Q-Soup are provided in Sec. A.4.

6 EXPERIMENTAL RESULTS

We empirically evaluate Q-Soup on higher-dimensional RL problems and verify that the conditions described in Observations 4.3 and 4.4 hold. We compare Q-Soup with traditional output-space ensembling, demonstrating competitive performance with a single model at inference. Furthermore, we investigate how Proxy Q-Soup mitigates performance degradation when varying discount factors break linear connectivity. Finally, we demonstrate that Q-Soup can be integrated with existing ensemble-based methods to reduce their inference overhead.

As a benchmark, we consider 15 Atari games (Bellemare et al., 2013), following Graesser et al. (2022) due to their diverse human-normalized scores. We evaluate off-policy and on-policy value-based RL methods, namely DQN (Mnih et al., 2013) and PQN (Gallici et al., 2025). We further

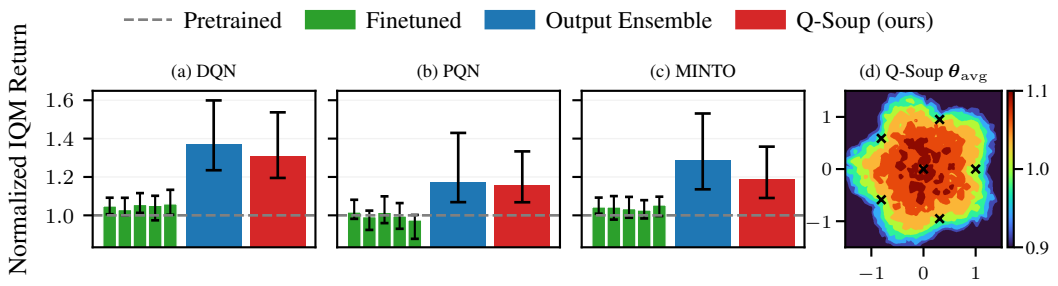

Figure 6: Empirical results of Q-Soup in the *homogeneous* setting across algorithms (a)–(c), showing normalized IQM returns for five finetuned models (green), an Output Ensemble (blue), and Q-Soup (red) relative to the pretrained baseline (grey dashed line). Results are aggregated over five pretrained seeds, with 95% confidence intervals. (d) Two-dimensional performance landscape showing the connected high-performing region of the finetuned models and Q-Soup $\boldsymbol{\theta}_{\text{avg}}$ near its center.

validate our method with the recent RL algorithm MINTO (Hendawy et al., 2026). To evaluate ensembling in weight and output spaces, we train five independent agents for each game. According to Observation 4.4, weight averaging is feasible when models are finetuned from a shared pretrained initialization. Therefore, for each trained Q-network, we initialize five additional finetuning runs from its parameters to obtain ensembles satisfying this condition.

Homogeneous setting. All finetuned models use the same configuration and hyperparameters as the pretrained model, differing only in SGD noise (random seeds). In Fig. 6(a)-(c), we evaluate Q-Soup with DQN, PQN, and MINTO, respectively. We report normalized return relative to pretrained performance, averaged across all 15 Atari games and five pretrained seeds. We compare Q-Soup against the individual finetuned models and the Output Ensemble. Across all three algorithms, Q-Soup outperforms the individual models and approaches the Output Ensemble at a fraction of its inference cost. Q-Soup improves over the average finetuned model by 24.4%, 15.6%, and 14.3% for DQN, PQN, and MINTO, respectively, and surpasses the best individual finetuned model by at least 12.9%. Notably, this improvement for PQN occurs even though finetuning alone provides no average improvement over pretraining. Relative to the Output Ensemble, Q-Soup trails by at most 7.7%, recovering 60%–89% of the performance gap between the average finetuned model and the ensemble, while requiring a single model instead of five. To better understand how the finetuned Q-networks relate to one another in weight space and to their parameter average, we project the anchors onto a two-dimensional plane in Fig. 6(d). The anchors lie within a connected high-performing region, with performance increasing toward their average $\boldsymbol{\theta}_{\text{avg}}$, consistent with Observation 4.3.

Heterogeneous setting. As discussed in Sec. 5.2, finetuned models can use different hyperparameter configurations. In Fig. 7(a)-(d), we evaluate Q-Soup by varying one hyperparameter at a time and keeping all others fixed to the pretrained configuration. For DQN, we vary the target update frequency and n-step return; for PQN, the λ-return coefficient and number of rollouts. Q-Soup improves over the average finetuned model by 18.4%–22.9% across the four settings. More importantly, simply averaging models trained with different hyperparameter values outperforms even the best individual model selected in hindsight by 6.5%–18.6%, despite including substantially weaker ingredients (e.g., $n_{\text{rollout}} = 8$). Q-Soup also closely matches the Output Ensemble for PQN and recovers 63%–77% of the ensemble's improvement over average finetuning for DQN, while requiring only one-fifth of the model memory at inference.

Observation 4.4 shows that varying the discount factor can break linear connectivity between finetuned models, causing Q-Soup to degrade substantially relative to the Output Ensemble in Fig. 8(a). Proxy Q-Soup, introduced in Sec. 5.2, addresses this failure mode without additional environment interactions, improving Q-Soup by approximately 55% and substantially closing the performance gap to the Output Ensemble. The same proxy can select the best finetuned configuration without evaluation and improve the Output Ensemble by approximately 6%. Nevertheless, the selected model remains approximately 8% behind Proxy Q-Soup, while the Output Ensemble requires all models at inference.

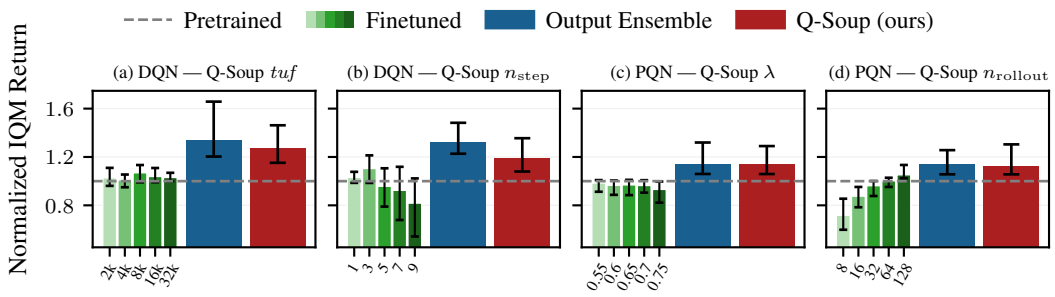

Figure 7: Empirical results of Q-Soup in the *heterogeneous* setting with DQN (a)-(b) and PQN (c)-(d), showing normalized IQM returns for five individual finetuned models using different hyperparameter values (green shades), an Output Ensemble (blue), and Q-Soup (red) relative to the pretrained baseline (grey dashed line). Results are aggregated over five pretrained seeds, with 95% confidence intervals. X-axis ticks indicate the hyperparameter values.

While Q-Soup *independently* finetunes models from a shared initialization, existing ensemble-based methods typically train multiple models *simultaneously*. To investigate whether Q-Soup can reduce the inference overhead of such methods, we consider Maxmin DQN (Lan et al., 2020), which mitigates estimation bias by taking the minimum Q-value across ensemble members. To enable weight averaging, we initialize its members from a common pretrained DQN rather than random initialization. Although this could reduce diversity and weaken its ability to mitigate overestimation, Fig. 8(b) shows that shared initialization maintains overall performance while producing a less conservative ensemble that avoids failures in some environments (Fig. 17). More importantly, Q-Soup matches both Maxmin DQN and the corresponding mean Output Ensemble by averaging their parameters into a single model.

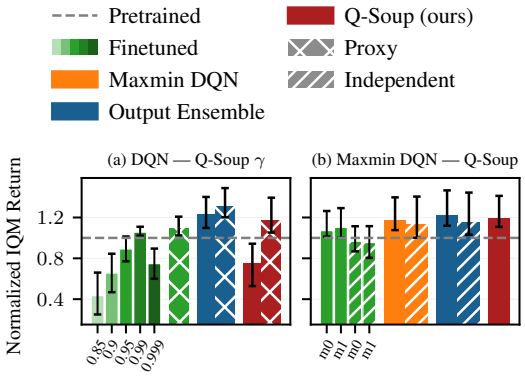

Figure 8: (a) Heterogeneous discount-factor setting comparing Q-Soup with proxy-based weight averaging, output ensembling, and model selection. (b) Q-Soup with Maxmin DQN, comparing shared pretrained and independent initialization. Results follow the same evaluation protocol as the preceding figures.

7 Conclusion

We studied the geometric structure of the objective landscape in value-based RL through the lens of mode connectivity. We analyzed how independently trained Q-networks relate in parameter space and investigated when weight-space ensembling can be effective. In particular, we addressed three key questions: Is weight averaging inherently ineffective in value-based RL? Under what conditions can it succeed? When do Q-networks naturally become linearly connected, forming flat regions in weight space? Our findings lead to Q-Soup, a weight-averaging approach that matches the performance of traditional output-space ensembles while avoiding the linear growth in inference cost with the number of models. Moreover, we have highlighted what can break linear mode connectivity in RL (i.e., the discount factor) and propose a variant of Q-Soup that relies on training returns as a proxy to identify bad models that have diverged from the flat region. To the best of our knowledge, this work provides the first systematic study of mode connectivity and weight-space ensembling for value-based RL. As a limitation, Proxy Q-Soup assumes that linear connectivity is preserved among some ingredients; complete loss of connectivity may limit its effectiveness. Our study is empirical. A theoretical study would complete our investigation of the geometry of the landscape and the performance bounds between weight-space and output-space ensembling. More broadly, we believe that our study opens new avenues for ensemble learning in value-based RL by leveraging geometric insights into parameter space.

AI USE STATEMENT

In this work, we used generative AI tools to (i) aid and polish the writing of the manuscript, (ii) assist with literature retrieval and discovery, and (iii) assist with the implementation of our methods and experiments. We did not use generative AI tools for other applicable tasks requiring disclosure, and the remaining required-disclosure tasks are not applicable to this work.

For writing, generative AI tools were used to improve the clarity, grammar, and presentation of text initially written by the authors. For literature research, they were used to identify potentially relevant related work, with all suggested references manually verified by the authors. For implementation, generative AI tools were used to assist with writing and modifying code for our methods and experiments. All AI-assisted code was reviewed and verified by the authors, and all AI-assisted text was reviewed and further modified where necessary.

We take responsibility for the final content of this work, including all text, claims, code, and other artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have taken several measures to support the reproducibility of our results. We provide pseudocode and detailed descriptions of Q-Soup, its variant Proxy Q-Soup, and their integration with the underlying value-based RL algorithms in the appendix. In addition, we provide a detailed description and pseudocode for SubQ, used to study Observation 4.3, in Sec. A.3. The appendix also details the network architectures, hyperparameter configurations, and training protocols used for pretraining and fine-tuning across the Atari and MinAtar benchmarks. In particular, we adopt the default hyperparameter configuration of PQN from Gallici et al. (2025) and the Dopamine hyperparameters (Castro et al., 2018) for DQN, MINTO, and Maxmin DQN, with any modifications made for our experiments explicitly documented. We further document the procedure used to construct and evaluate the performance landscapes in Sec. B.9, including the mapping of the fine-tuned models and their parameter average into the two-dimensional representation. For empirical evaluation, we report results across multiple random seeds and provide per-environment breakdowns in Appendix B. Together, these details are intended to facilitate independent reproduction and further validation of our findings. Finally, we will release an open-source implementation for all experiments upon acceptance.

REFERENCES

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International conference on machine learning*, pp. 104–114. PMLR, 2020.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of artificial intelligence research*, 47: 253–279, 2013.

Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G Bellemare. Dopamine: A research framework for deep reinforcement learning. *arXiv preprint arXiv:1812.06110*, 2018.

Richard Y Chen, Szymon Sidor, Pieter Abbeel, and John Schulman. Ucb exploration via q-ensembles. *arXiv preprint arXiv:1706.01502*, 2017.

Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. Essentially no barriers in neural network energy landscape. In *International conference on machine learning*, pp. 1309–1318. PMLR, 2018.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International conference on machine learning*, pp. 3259–3269. PMLR, 2020.

Matteo Gallici, Mattie Fellows, Benjamin Ellis, Bartomeu Pou, Ivan Masmitja, Jakob Nicolaus Foerster, and Mario Martin. Simplifying deep temporal difference learning. In *International Conference on Learning Representations*, 2025.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. *Advances in neural information processing systems*, 31, 2018.

Jean-Baptiste Gaya, Laure Soulier, and Ludovic Denoyer. learning a subspace of policies for online adaptation in reinforcement learning. In *International Conference of Learning Representations (ICLR) 2022*, 2022.

Jean-Baptiste Gaya, Thang Doan, Lucas Caccia, Laure Soulier, Ludovic Denoyer, and Roberta Raileanu. Building a subspace of policies for scalable continual learning. In *International Conference of Learning Representations*, 2023.

Laura Graesser, Utku Evci, Erich Elsen, and Pablo Samuel Castro. The state of sparse training in deep reinforcement learning. In *International Conference on Machine Learning*, pp. 7766–7792. PMLR, 2022.

Ahmed Hendawy, Henrik Metternich, Jan Peters, Gabriele Tiboni, and Carlo D'Eramo. It is all connected: Multi-task reinforcement learning via mode connectivity. In *Eighteenth European Workshop on Reinforcement Learning*, 2025.

Ahmed Hendawy, Henrik Metternich, Théo Vincent, Mahdi Kallel, Jan Peters, and Carlo D'Eramo. Use the online network if you can: Towards fast and stable reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*, 2022.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

Qingfeng Lan, Yangchen Pan, Alona Fyshe, and Martha White. Maxmin q-learning: Controlling the estimation bias of q-learning. *International Conference on Learning Representations*, 2020.

Kimin Lee, Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning. In *International conference on machine learning*, pp. 6131–6141. PMLR, 2021.

Litian Liang, Yaosheng Xu, Stephen McAleer, Dailin Hu, Alexander Ihler, Pieter Abbeel, and Roy Fox. Reducing variance in temporal-difference value estimation via ensemble of deep networks. In *International Conference on Machine Learning*, pp. 13285–13301. PMLR, 2022.

Michael S Matena and Colin A Raffel. Merging models with fisher-weighted averaging. *Advances in Neural Information Processing Systems*, 35:17703–17716, 2022.

Seyed Iman Mirzadeh, Mehrdad Farajtabar, Dilan Gorur, Razvan Pascanu, and Hassan Ghasemzadeh. Linear mode connectivity in multitask and continual learning. *arXiv preprint arXiv:2010.04495*, 2020.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. What is being transferred in transfer learning? *Advances in neural information processing systems*, 33:512–523, 2020.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29, 2016.

Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3 (1):9–44, 1988.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 1998.

Marco A Wiering and Hado Van Hasselt. Ensemble algorithms in reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(4):930–936, 2008.

Mitchell Wortsman, Maxwell C Horton, Carlos Guestrin, Ali Farhadi, and Mohammad Rastegari. Learning neural network subspaces. In *International Conference on Machine Learning*, pp. 11217–11227. PMLR, 2021.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International conference on machine learning*, pp. 23965–23998. PMLR, 2022.

Kenny Young and Tian Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019.

APPENDIX

A Implementation Details **14**

A.1 Deep Q-Networks (DQN) 14

A.2 Parallelized Q-Network (PQN) 14

A.3 SubQ 15

A.4 Proxy Q-Soup 17

A.5 Hyperparameters 18

B Additional Results **20**

B.1 PQN and Q-Soup on Atari 20

B.2 DQN and Q-Soup on Atari 23

B.3 MINTO and Q-Soup on Atari 26

B.4 Discount Factor and Proxy Q-Soup 27

B.5 Maxmin DQN and Q-Soup 28

B.6 Stochastic Weight Averaging and Q-Soup 29

B.7 Ablation on Ensemble Size and Number of Finetuning Steps 30

B.8 Ablation on Proxy Q-Soup Components 31

B.9 The Performance Landscape Around Q-Soup 32

B.10 Q-Soup Performance Landscapes with PQN 33

B.11 Q-Soup Performance Landscapes with DQN 38

C Geometric Analysis of Ensemble Members **43**

D Additional Observations **45**

A Implementation Details

In this section, we provide the implementation details for our RL experiments. First, we describe the core value-based RL methods considered in our experiments: DQN (Mnih et al., 2013), PQN (Gallici et al., 2025), and MINTO (Hendawy et al., 2026). Next, we formally present the algorithmic structure and training procedure of our proposed SubQ method, which is used to study Observation 4.3, followed by its adaptations to DQN and PQN, referred to as SubDQN and SubPQN, respectively. We then describe our Q-Soup variant, Proxy Q-Soup, and highlight its differences from the standard Q-Soup formulation. Finally, we detail the hyperparameter configurations and training protocols used across all evaluated environments in the Atari and MinAtar benchmark suites.

A.1 Deep Q-Networks (DQN)

Deep Q-Network (DQN) (Mnih et al., 2013; 2015) is an *off-policy*, value-based RL algorithm that combines Q-learning with deep neural networks to learn actions that maximize long-term rewards. It approximates Q-values from high-dimensional inputs and improves training stability through techniques such as a *replay buffer* and a *target network*. Alg. 1 illustrates the training procedure of DQN.

Recently, Hendawy et al. (2026) proposed a new method, named MINTO, that computes the temporal-difference (TD) target y_b using the minimum of the estimates produced by the online and target networks. MINTO preserves the stability provided by the target network while accelerating learning by incorporating more recent estimates from the online network. MINTO follows a procedure similar to DQN in Alg. 1, with the main modification being the replacement of line 9 with

$$y_b = r_b + \gamma \max_{a'} \min \left(Q_\theta(s'_b, a'), Q_{\bar{\theta}}(s'_b, a') \right), \tag{2}$$

where the resulting target value is detached from the computational graph to prevent gradient flow through the target computation.

Algorithm 1 DQN

1: Initialize online and target paramters $\bar{\theta}$, θ, an empty replay buffer $\mathcal{B}$, and $t_{\text{total}} = 0$.
2: **while** $t_{\text{total}} < n_{\text{steps}}$ **do**
3: Sample an initial state s_0 from μ
4: **for** $t = 0$ **to** n_{horizon} **do**
5: Sample an action $a_t \sim \epsilon$-greedy$(Q_\theta(s_t, \cdot))$
6: Execute action a_t in environment, observe reward r_t and next state s_{t+1}
7: Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{B}$
8: Sample a batch of B transitions $(s_b, a_b, r_b, s'_b)_{b=1}^{B}$ from $\mathcal{B}$
9: Compute the TD-target $y_b = r_b + \gamma \max_{a'} Q_{\bar{\theta}}(s'_b, a')$
10: **if** s'_b is terminal **then** $y_b \leftarrow r_b$
11: Compute the loss $\mathcal{L}(\theta) = \frac{1}{2B} \sum_{b=1}^{B} (y_b - Q_\theta(s_b, a_b))^2$
12: Obtain the gradient $\nabla_\theta \mathcal{L}$ and perform an update step
13: Every T steps update the target network $\bar{\theta} \leftarrow \theta$
14: **if** s_{t+1} is terminal **break**
15: **end for**
16: $t_{\text{total}} \leftarrow t_{\text{total}} + t$
17: **end while**

A.2 Parallelized Q-Network (PQN)

As an on-policy RL method, we employ PQN and the architecture introduced by Gallici et al. (2025). Unlike DQN and MINTO, which rely on large experience replay buffers, PQN leverages vectorized environments to collect experiences in parallel. To stabilize training, PQN incorporates LayerNorm into its network architecture. Moreover, it employs TD(λ) to compute the target values used for learning. Alg. 2 illustrates the training procedure of PQN, highlighting the key components that distinguish it from the other considered methods.

Algorithm 2 Parallelized Q-Network (PQN)

Initialize Q-network θ, and initial states s_0^i for $i \in \{1, \dots, n_{\text{envs}}\}$
$t_{\text{total}} \leftarrow 0$
while $t_{\text{total}} < n_{\text{steps}}$ **do**
  for $t = 0 \dots n_{\text{rollout}} - 1$ **do**
    Sample actions $a_t^i \sim \epsilon$-greedy$(Q_\theta(s_t^i, \cdot))$ for $i \in \{1, \dots, n_{\text{envs}}\}$
    Execute a_t^i, observe r_t^i, d_t^i, s_{t+1}^i for $i \in \{1, \dots, n_{\text{envs}}\}$
  end for
  Compute $R_t^{\lambda,i} = r_t^i + \gamma \left[\lambda R_{t+1}^{\lambda,i} + (1 - d_t^i)(1 - \lambda) \max_{a'} Q_\theta(s_{t+1}^i, a')\right]$ for all t and i
  for epoch $= 1 \dots n_{\text{epochs}}$ **do**
    Shuffle data over time t and environment i axes to form M minibatches
    for each minibatch m **do**
      $\theta \leftarrow \theta - \eta \nabla_\theta \left(\frac{1}{M} \sum_{(s_t^i, a_t^i) \in m} \left(Q_\theta(s_t^i, a_t^i) - R_t^{\lambda,i} \right)^2 \right)$
    end for
  end for
  $s_0^i \leftarrow s_{n_{\text{rollout}}}^i$ for $i \in \{1, \dots, n_{\text{envs}}\}$
  $t_{\text{total}} \leftarrow t_{\text{total}} + (n_{\text{rollout}} \times n_{\text{envs}})$
end while

A.3 SUBQ

To study the weight-space geometry underlying Observation 4.3, we introduce SubQ, a subspace-based value-based RL framework inspired by Wortsman et al. (2021). SubQ jointly optimizes a continuous subspace of Q-functions with the objective of maintaining high-performing policies throughout the subspace. By explicitly learning such a subspace, SubQ aims to target a flat region in the optimization landscape, enabling us to investigate the benefits of weight-space ensembling and weight averaging.

Specifically, the subspace is defined as a simplex whose vertices correspond to a set of learned anchor parameters $\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_k$. A point within the simplex is constructed as a convex combination of these anchors,

$$\boldsymbol{\theta}(\boldsymbol{\alpha}) = \sum_{i=1}^{k} \alpha_i \boldsymbol{\theta} i, \qquad \alpha_i \geq 0, \quad \sum_{i=1}^{k} \alpha_i = 1, \tag{3}$$

where $\boldsymbol{\alpha}$ denotes the interpolation coefficients. During training, points $\boldsymbol{\theta}(\boldsymbol{\alpha})$ are sampled from the simplex and optimized according to the objective of the underlying value-based RL algorithm. Since each sampled point is a differentiable function of the anchors, gradients evaluated at $\boldsymbol{\theta}(\boldsymbol{\alpha})$ propagate to the anchor parameters, allowing the entire subspace to be optimized jointly.

Following Wortsman et al. (2021), we additionally regularize the anchors using their mean pairwise cosine similarity (Equation 4) to discourage parameter collapse and promote diversity within the learned subspace. The specific procedures for collecting experience, constructing learning targets, and sampling points from the simplex depend on the underlying RL algorithm. We consider two instantiations of SubQ: SubDQN and SubPQN , which adapt the framework to DQN and PQN, respectively.

$$\mathcal{L}_{similarity} = \frac{2}{N(N-1)} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \cos^2(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) \tag{4}$$

A.3.1 SUBSPACE DEEP Q-NETWORK (SUBDQN)

We instantiate SubQ within the off-policy training procedure of DQN, resulting in SubDQN. During experience collection, the agent follows an ϵ-greedy policy induced by a single point $\boldsymbol{\theta}(\boldsymbol{\alpha})$ in the subspace, where $\boldsymbol{\alpha}$ is resampled episodically. Transitions are stored in the replay buffer without their collection-time interpolation coefficients. For each sampled minibatch, a new $\boldsymbol{\alpha}_b$ is independently

sampled for every transition and used to construct both the online Q-value prediction and its corresponding one-step TD target. Consequently, replayed transitions can be used to optimize different regions of the subspace irrespective of the policies under which they were originally collected. The resulting mean squared TD error, together with the cosine-similarity regularization between the anchors, is differentiated with respect to the online anchor parameters, thereby jointly optimizing the anchors that define the subspace. As in standard DQN, the target anchors are periodically updated from the online anchors. SubDQN is utilized for generating the empirical results in Fig. 3 and Fig. 4.

Algorithm 3 Subspace Deep Q-Network (SubDQN)

Initialize online anchors $\{\boldsymbol{\theta}_i\}_{i=1}^k$, target anchors $\{\bar{\boldsymbol{\theta}}_i\}_{i=1}^k \leftarrow \{\boldsymbol{\theta}_i\}_{i=1}^k$, an empty replay buffer $\mathcal{B}$, and $t_{\text{total}} \leftarrow 0$
while $t_{\text{total}} < n_{\text{steps}}$ **do**
  Sample an initial state s_0 from μ and a subspace point $\boldsymbol{\alpha}$
  for $t = 0$ **to** n_{horizon} **do**
    Sample an action $a_t \sim \epsilon$-greedy$(Q_{\boldsymbol{\theta}(\boldsymbol{\alpha})}(s_t, \cdot))$
    Execute a_t, observe r_t, d_t, and s_{t+1}
    Store transition $(s_t, a_t, r_t, d_t, s_{t+1})$ in $\mathcal{B}$
    Sample a batch of B transitions $(s_b, a_b, r_b, d_b, s'_b)_{b=1}^B$ from $\mathcal{B}$
    Sample a subspace point $\boldsymbol{\alpha}_b$ independently for each transition $b \in \{1, \dots, B\}$
    Compute $y_b = r_b + \gamma(1 - d_b) \max_{a'} Q_{\bar{\boldsymbol{\theta}}(\boldsymbol{\alpha}_b)}(s'_b, a')$ for $b \in \{1, \dots, B\}$
    $\mathcal{L}_{\text{total}} \leftarrow \frac{1}{2B} \sum_{b=1}^B \left(y_b - Q_{\boldsymbol{\theta}(\boldsymbol{\alpha}_b)}(s_b, a_b)\right)^2 + \beta \mathcal{L}_{\text{similarity}}(\{\boldsymbol{\theta}_i\}_{i=1}^k)$
    $\{\boldsymbol{\theta}_i\}_{i=1}^k \leftarrow \{\boldsymbol{\theta}_i\}_{i=1}^k - \eta \nabla_{\{\boldsymbol{\theta}_i\}_{i=1}^k} \mathcal{L}_{\text{total}}$
    Every T steps update the target anchors $\{\bar{\boldsymbol{\theta}}_i\}_{i=1}^k \leftarrow \{\boldsymbol{\theta}_i\}_{i=1}^k$
    if d_t **then break**
  end for
  $t_{\text{total}} \leftarrow t_{\text{total}} + t$
end while

A.3.2 SUBSPACE PARALLELIZED Q-NETWORK (SUBPQN)

We instantiate SubQ within the on-policy and parallelized training procedure of PQN, resulting in SubPQN. During experience collection, the agent interacts with the environment using a single point $\boldsymbol{\theta}(\boldsymbol{\alpha})$ in the subspace, where the interpolation coefficients $\boldsymbol{\alpha}$ are resampled episodically. After collecting a rollout, different $\boldsymbol{\alpha}$ values are used for each trajectory when computing the TD(λ) returns, promoting coverage of different regions of the learned subspace. Following the standard PQN procedure, the collected transitions and their corresponding λ-returns are shuffled and divided into minibatches for multiple epochs of optimization. The loss evaluated at a sampled point $\boldsymbol{\theta}(\boldsymbol{\alpha})$ is differentiated directly with respect to the anchor parameters, thereby jointly optimizing the anchors that define the subspace.

Algorithm 4 Subspace Parallel Q-Network (SubPQN)

Initialize Q-network $\{\boldsymbol{\theta}_i\}_{i=1}^{k}$, and points $\boldsymbol{\alpha}_0^i$ and initial states s_0^i for $i \in \{1, \dots, n_{\text{envs}}\}$
$t_{\text{total}} \leftarrow 0$
while $t_{\text{total}} < n_{\text{steps}}$ **do**
  for $t = 0 \dots n_{\text{rollout}} - 1$ **do**
    Sample actions $a_t^i \sim \epsilon$-greedy$(Q_{\boldsymbol{\theta}(\boldsymbol{\alpha}_t^i)}(s_t^i, \cdot))$ for $i \in \{1, \dots, n_{\text{envs}}\}$
    Execute a_t^i, observe r_t^i, d_t^i, s_{t+1}^i for $i \in \{1, \dots, n_{\text{envs}}\}$
    Update $\boldsymbol{\alpha}_{t+1}^i$ if the corresponding environment terminated
  end for
  Compute $R_t^{\lambda,i}(\boldsymbol{\alpha}_{i,t}) = r_t^i + \gamma \left[\lambda R_{t+1}^{\lambda,i}(\boldsymbol{\alpha}_{i,t}) + (1 - d_t^i)(1 - \lambda) \max_{a'} Q_{\boldsymbol{\theta}(\boldsymbol{\alpha}_{i,t})}(s_{t+1}^i, a')\right]$
  for epoch $= 1 \dots n_{\text{epochs}}$ **do**
    Shuffle data over time t and environment i axes to form M minibatches
    for each minibatch m **do**
      $\mathcal{L}_{\text{total}} \leftarrow \frac{1}{M} \sum_{(s_t^i, a_t^i) \in m} \left(Q_{\boldsymbol{\theta}(\boldsymbol{\alpha}_{i,t})}(s_t^i, a_t^i) - R_t^{\lambda,i}(\boldsymbol{\alpha}_{i,t})\right)^2 + \beta \mathcal{L}_{\text{similarity}}(\{\boldsymbol{\theta}\}_{i=1}^{k})$
      $\{\boldsymbol{\theta}\}_{i=1}^{k} \leftarrow \{\boldsymbol{\theta}\}_{i=1}^{k} - \eta \nabla_{\{\boldsymbol{\theta}\}_{i=1}^{k}} \mathcal{L}_{\text{total}}$
    end for
  end for
  $s_0^i \leftarrow s_{n_{\text{rollout}}}^i$ for $i \in \{1, \dots, n_{\text{envs}}\}$
  $t_{\text{total}} \leftarrow t_{\text{total}} + (n_{\text{rollout}} \times n_{\text{envs}})$
end while

A.4 Proxy Q-Soup

Q-Soup assigns equal weight to all finetuned ingredients. This is effective when the ingredients remain within the same connected high-performing region, but can degrade when some ingredients diverge from this region. As shown in Observation 4.4, this can occur when finetuned models use different discount factors γ. Since γ defines the return objective and hence the value function being learned, models using different discount factors are optimized under different objectives despite sharing the same pretrained initialization. This can break linear connectivity between the resulting Q-networks and make uniform parameter averaging ineffective (see Fig. 8(a)). Identifying such ingredients through evaluation would require additional environment interactions. We therefore introduce *Proxy Q-Soup*, which uses the training returns already collected during finetuning as a proxy for ingredient quality to identify and reduce the influence of ingredients that may have diverged from the connected region.

Training-return proxy. Let $\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_N$ denote N ingredients finetuned from the same pretrained parameters. For each ingredient i, we consider the completed training episodes $e = 1, \dots, E_i$, with length $\ell_{i,e}$ and undiscounted return $G_{i,e}$. Let $T_i = \sum_e \ell_{i,e}$ denote the total number of environment steps and $s_{i,e} = \sum_{e' < e} \ell_{i,e'}$ the step at which episode e begins. We compute the proxy score P_i from episodes starting within a window covering the final ω fraction of training:

$$P_i = \text{IQM}\left\{G_{i,e} : s_{i,e} \geq (1 - \omega) T_i\right\}. \tag{5}$$

We use $\omega = 0.1$ throughout. Using the final portion of training makes the proxy representative of the parameters being merged, while the interquartile mean (IQM) reduces sensitivity to outlier episodes. *Importantly, the proxy is used only to compare ingredients and does not require additional environment interactions.*

Z-score filtering. We standardize the proxy scores across ingredients using Z-scores:

$$z_i = \frac{P_i - \mu_P}{\sigma_P}, \qquad \mu_P = \frac{1}{N} \sum_{j=1}^{N} P_j, \qquad \sigma_P^2 = \frac{1}{N} \sum_{j=1}^{N} (P_j - \mu_P)^2. \tag{6}$$

Ingredients whose proxy score lies more than κ standard deviations below the mean are removed:

$$\mathcal{S} = \{i \in [N] : z_i \geq -\kappa\}. \tag{7}$$

Algorithm 5 Proxy Q-Soup

Require: finetuned ingredients $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N$ and training returns; window fraction ω; threshold κ; minimum size $m_{\min}$; temperature τ

for $i = 1, \ldots, N$ **do**
  $P_i \leftarrow$ IQM of returns from episodes starting within the final ω fraction of training
end for
Standardize $\{P_i\}_{i=1}^N$ to obtain $\{z_i\}_{i=1}^N$
$\mathcal{S} \leftarrow \{i : z_i \geq -\kappa\}$
if $|\mathcal{S}| < m_{\min}$ **then**
  $\mathcal{S} \leftarrow$ indices of the $m_{\min}$ largest P_i
end if
Re-standardize $\{P_i\}_{i \in \mathcal{S}}$ to obtain $\{z_i^{\mathcal{S}}\}_{i \in \mathcal{S}}$
if $\sigma_{\mathcal{S}} \approx 0$ **then**
  $w_i \leftarrow 1/|\mathcal{S}|$ for all $i \in \mathcal{S}$
else
  $w_i \leftarrow \mathrm{softmax}_{i \in \mathcal{S}} \left(z_i^{\mathcal{S}}/\tau\right)$
end if
return $\boldsymbol{\theta}_{\text{Proxy}} = \sum_{i \in \mathcal{S}} w_i \boldsymbol{\theta}_i$

We use $\kappa = 1$ in all experiments. If fewer than $m_{\min}$ ingredients remain, we retain the $m_{\min}$ ingredients with the highest proxy scores. This adaptive filtering preserves all ingredients when their training returns are comparable while removing clear low-performing outliers.

Softmax weighting. After filtering, we re-standardize the proxy scores over the retained set $\mathcal{S}$ and compute soft weights

$$w_i = \frac{\exp\left(z_i^{\mathcal{S}}/\tau\right)}{\sum_{j \in \mathcal{S}} \exp\left(z_j^{\mathcal{S}}/\tau\right)}, \qquad i \in \mathcal{S}, \tag{8}$$

where $\tau > 0$ controls the strength of the weighting. We use $\tau = 1$ in all experiments. If the proxy scores within $\mathcal{S}$ have zero variance, we instead use uniform weights $w_i = 1/|\mathcal{S}|$. The final Proxy Q-Soup is obtained as

$$\boldsymbol{\theta}_{\text{Proxy}} = \sum_{i \in \mathcal{S}} w_i \boldsymbol{\theta}_i. \tag{9}$$

Thus, Proxy Q-Soup uses training returns both to filter weak ingredients and to reduce their influence during parameter averaging, without requiring additional evaluation interactions.

The same proxy can also be applied in output space by computing

$$Q_{\text{Proxy-Ens}}(s, a) = \sum_{i \in \mathcal{S}} w_i Q_{\boldsymbol{\theta}_i}(s, a). \tag{10}$$

This provides an output-space counterpart using the same filtering and weighting procedure. The training-return proxy can also be used to select the best-performing configuration without additional evaluation, as shown in Fig. 8. Algorithm 5 summarizes Proxy Q-Soup.

A.5 Hyperparameters

As detailed in Table 1, we adopt the default hyperparameter configuration for PQN established by Gallici et al. (2025). For DQN, MINTO, and Maxmin DQN on Atari, we follow the official Dopamine hyperparameters (Castro et al., 2018), as reported in Table 2. Within each benchmark, we use the same configuration across all evaluated games.

For finetuning, we retain the configurations reported in the tables, except for the number of training steps and the hyperparameter varied in each one-dimensional sweep of the heterogeneous setting. For DQN, MINTO, and Maxmin DQN, the main results use 10 finetuning epochs, corresponding to 2.5×10^6 additional training steps. The independent Maxmin DQN variant in Fig. 8 is instead trained for the combined pretraining (1.25×10^7) and finetuning (2.5×10^6) budget, resulting in 1.5×10^7 steps. For PQN, the number of finetuning steps used for the main results is 1×10^7 on top of pretraining steps of 5×10^7 as shown in the table.

Hyperparameter	**Atari**	**MinAtar**
Total Steps	5×10^7	1×10^7
Number of Epochs	50	50
Number of Environments	128	128
Rollout Steps	32	32
Number of Minibatches	32	32
Update Epochs	2	2
Discount Factor (γ)	0.99	0.99
Eligibility Trace (λ)	0.65	0.65
Learning Rate	2.5×10^{-4}	5×10^{-4}
Max Gradient Norm	10.0	10.0
Initial Exploration (ϵ_{start})	1.0	1.0
Final Exploration (ϵ_{end})	0.001	0.05
Exploration Decay Steps	5×10^6	2×10^6
Architecture Type	CNN	CNN
LayerNorm	True	True
Features	[32, 64, 64, 512	[16, 128]
Cosine Similarity (β) (SubQ)	1.0	1.0

Table 1: Hyperparameter settings for PQN on Atari and MinAtar.

Hyperparameter	**DQN/MINTO**		**MaxminDQN**
Environment	**Atari**	**MinAtar**	**Atari**
Total Steps	1.25×10^7	5×10^6	1.25×10^7
Number of Epochs	50	50	50
Batch Size	32	32	32
Discount Factor (γ)	0.99	0.99	0.99
Learning Rate	6.25×10^{-5}	3×10^{-4}	6.25×10^{-5}
Initial Exploration (ϵ_{start})	1.0	1.0	1.0
Final Exploration (ϵ_{end})	0.01	0.05	0.01
Exploration Decay Steps	250,000	1,000,000	250,000
Replay Buffer Capacity	1,000,000	10,000	1,000,000
Update Horizon	1	1	1
Training Steps per Epoch	250,000	100,000	250,000
Data to Update	4	4	4
Initial Samples	20,000	10,000	20,000
Target Update Frequency (T)	8,000	1,000	8,000
Architecture Type	CNN	CNN	CNN
LayerNorm	False	True	False
Features	[32, 64, 64, 512]	[16, 128]	[32, 64, 64, 512]
Cosine Similarity (β) (SubQ)	1.0	1.0	–
Ensemble Size (N)	–	–	2

Table 2: Hyperparameter settings for DQN and MINTO on Atari and MinAtar, and MaxminDQN on Atari.

B ADDITIONAL RESULTS

B.1 PQN AND Q-SOUP ON ATARI

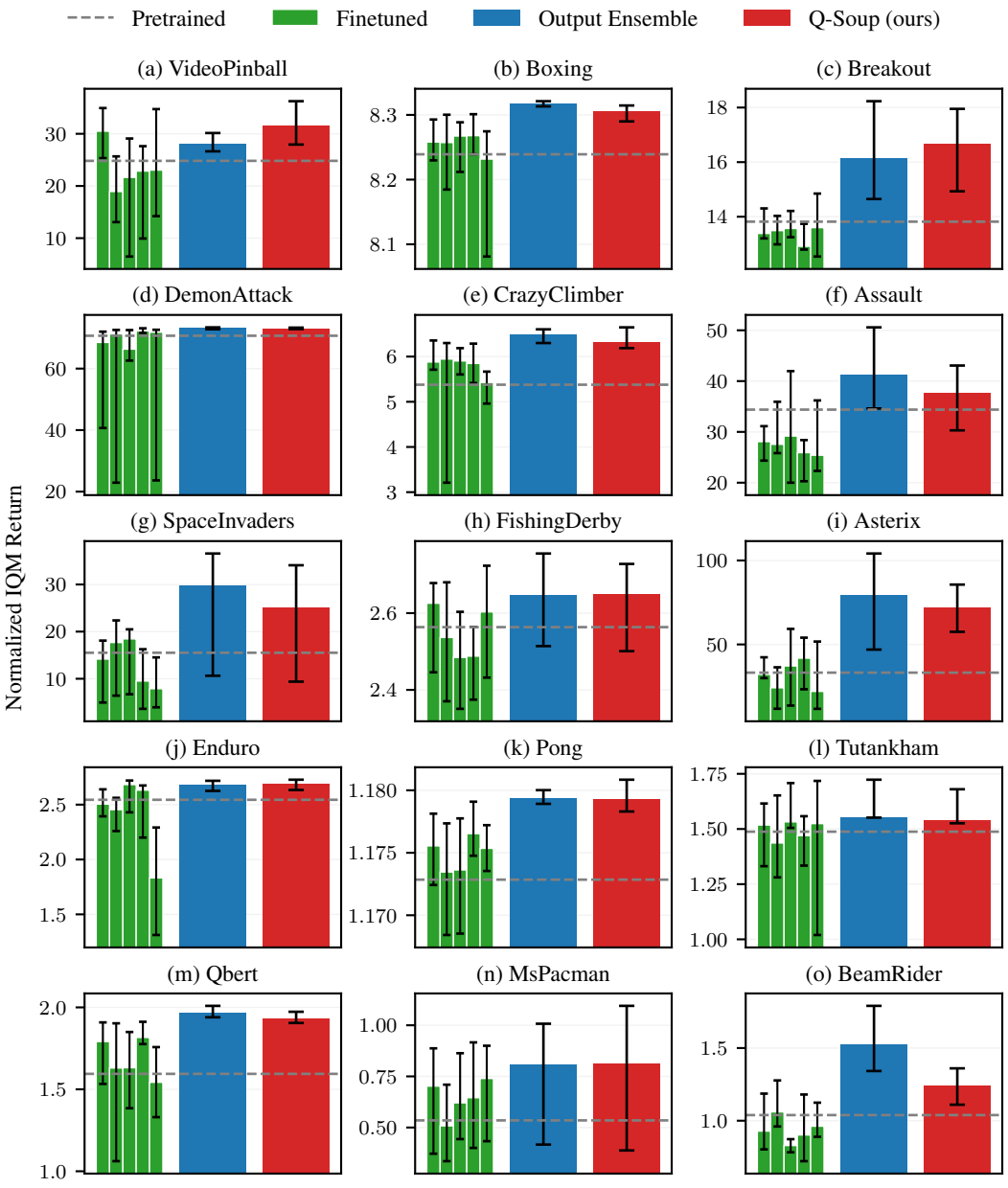

Figure 9: Individual environment results with **PQN** as the learning algorithm demonstrating the normalized IQM return for 15 Atari environments, aggregated over 5 pretrained seeds, comparing 5 individual finetuned models (green) with **homogeneous** configuration, an output ensemble (blue), and Q-Soup (red) against a pretrained baseline (grey dashed line). Evaluation is performed using 8 parallel environments with 27,000 steps each. Error bars denote the 95% confidence intervals. The normalization of the return is carried out by human-score normalization of the raw scores.

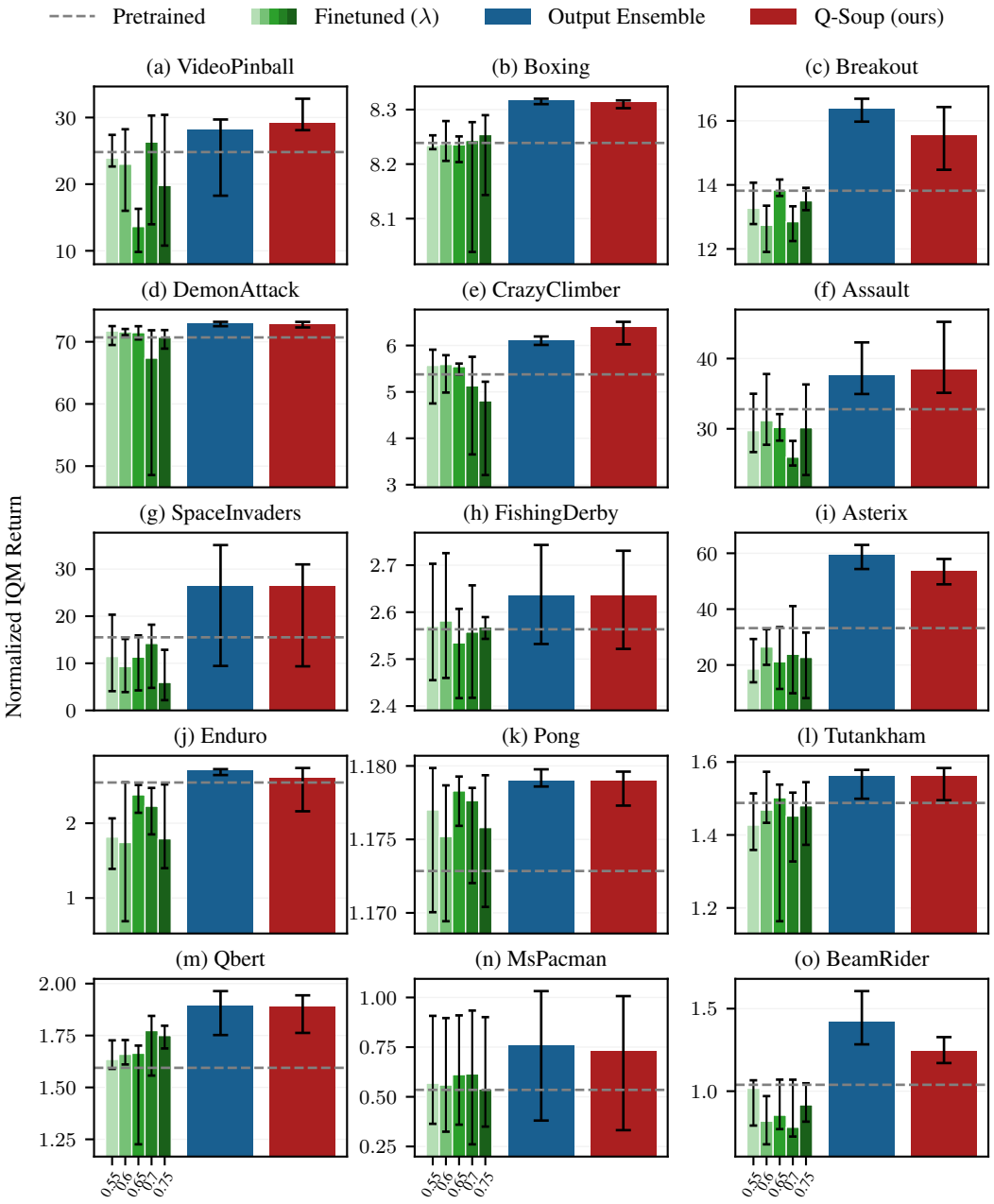

Figure 10: Individual environment results with **PQN** as the learning algorithm demonstrating the normalized IQM return for 15 Atari environments, aggregated over 5 pretrained seeds, comparing 5 individual finetuned models (green shades) with **heterogeneous** configuration of varying λ – return, an output ensemble (blue), and Q-Soup (red) against a pretrained baseline (grey dashed line). Evaluation is performed using 8 parallel environments with 27,000 steps each. Error bars denote the 95% confidence intervals. The normalization of the return is carried out by human-score normalization of the raw scores.

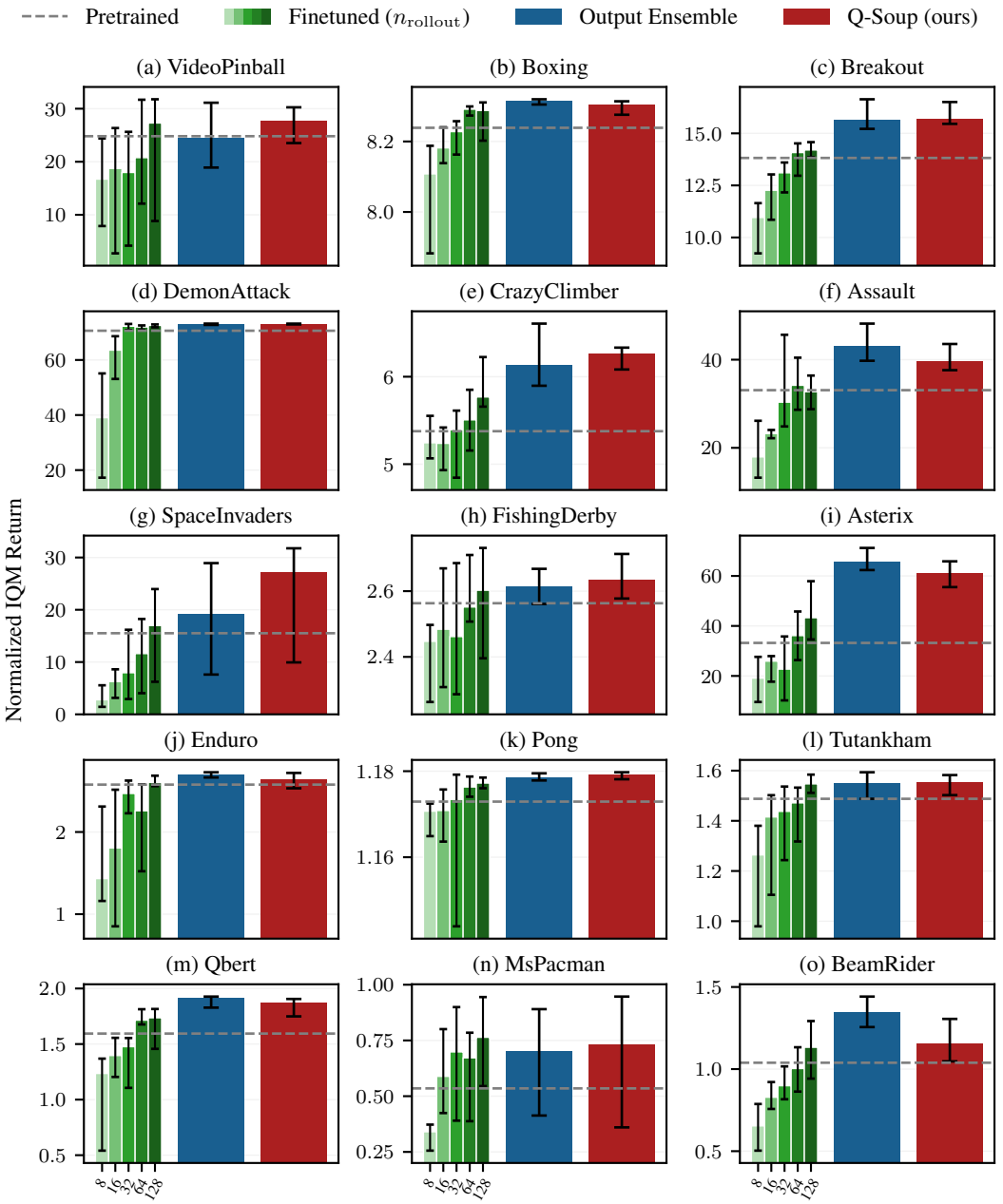

Figure 11: Individual environment results with **PQN** as the learning algorithm demonstrating the normalized IQM return for 15 Atari environments, aggregated over 5 pretrained seeds, comparing 5 individual finetuned models (green shades) with **heterogeneous** configuration of varying n_{rollout}, an output ensemble (blue), and Q-Soup (red) against a pretrained baseline (grey dashed line). Evaluation is performed using 8 parallel environments with 27,000 steps each. Error bars denote the 95% confidence intervals. The normalization of the return is carried out by human-score normalization of the raw scores.

B.2 DQN AND Q-SOUP ON ATARI

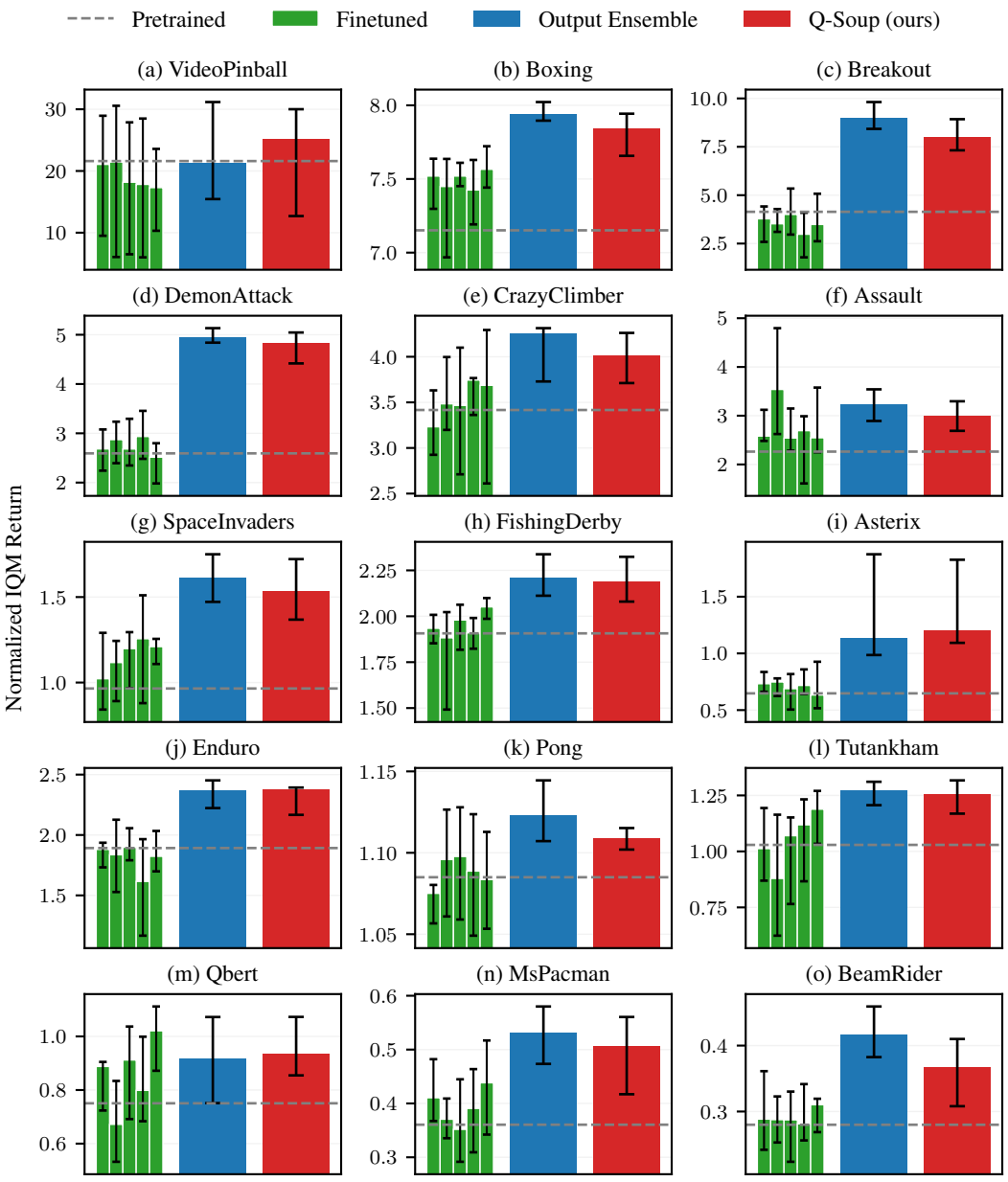

Figure 12: Individual environment results with **DQN** as the learning algorithm demonstrating the normalized IQM return for 15 Atari environments, aggregated over 5 pretrained seeds, comparing 5 individual finetuned models (green) with **homogeneous** configuration, an output ensemble (blue), and Q-Soup (red) against a pretrained baseline (grey dashed line). Evaluation is performed for 20 episodes with horizon of 27,000 steps each. Error bars denote the 95% confidence intervals. The normalization of the return is carried out by human-score normalization of the raw scores.

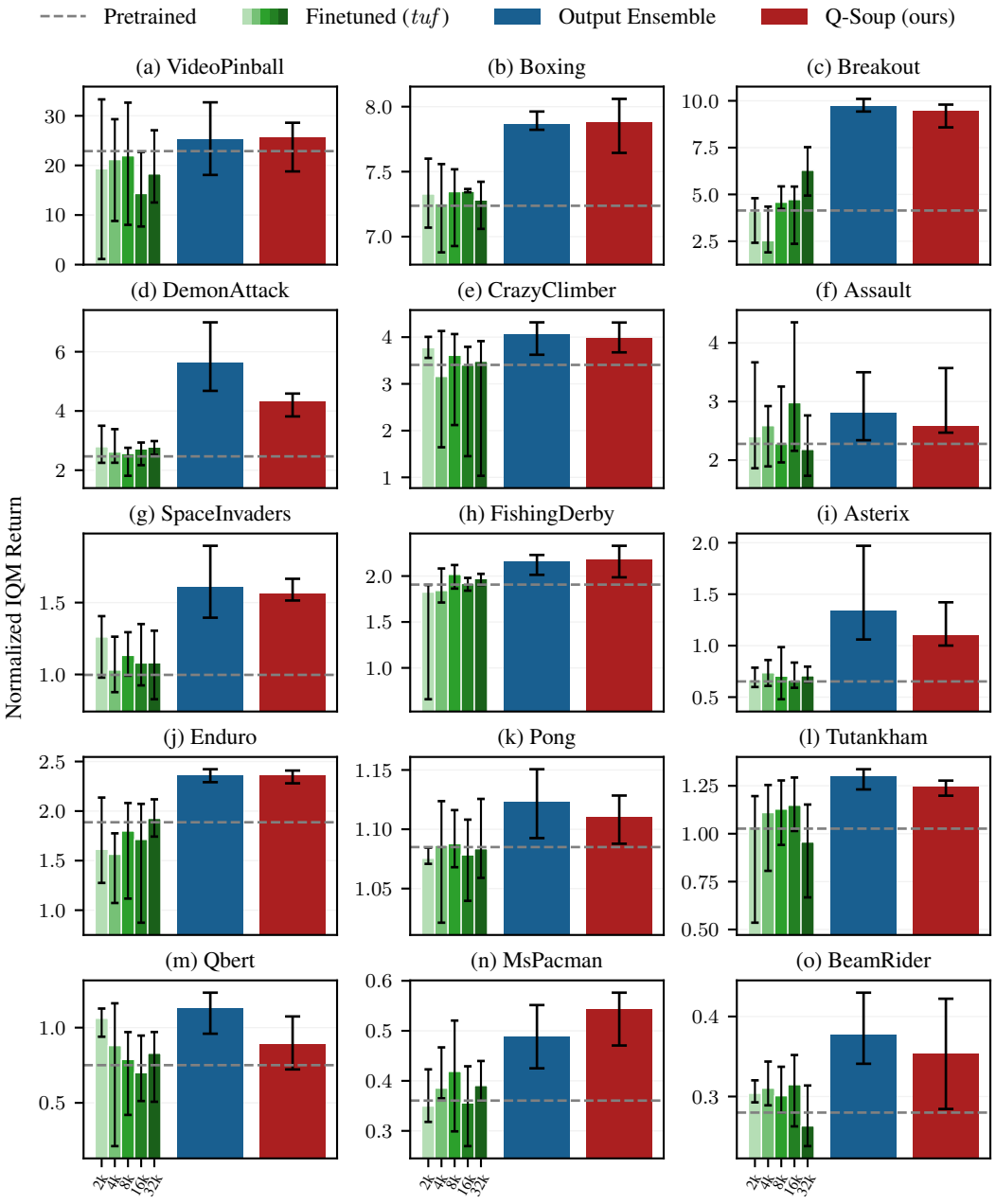

Figure 13: Individual environment results with **DQN** as the learning algorithm demonstrating the normalized IQM return for 15 Atari environments, aggregated over 5 pretrained seeds, comparing 5 individual finetuned models (green shades) with **heterogeneous** configuration of varying target update frequency (tuf), an output ensemble (blue), and Q-Soup (red) against a pretrained baseline (grey dashed line). Evaluation is performed for 20 episodes with horizon of 27,000 steps each. Error bars denote the 95% confidence intervals. The normalization of the return is carried out by human-score normalization of the raw scores.

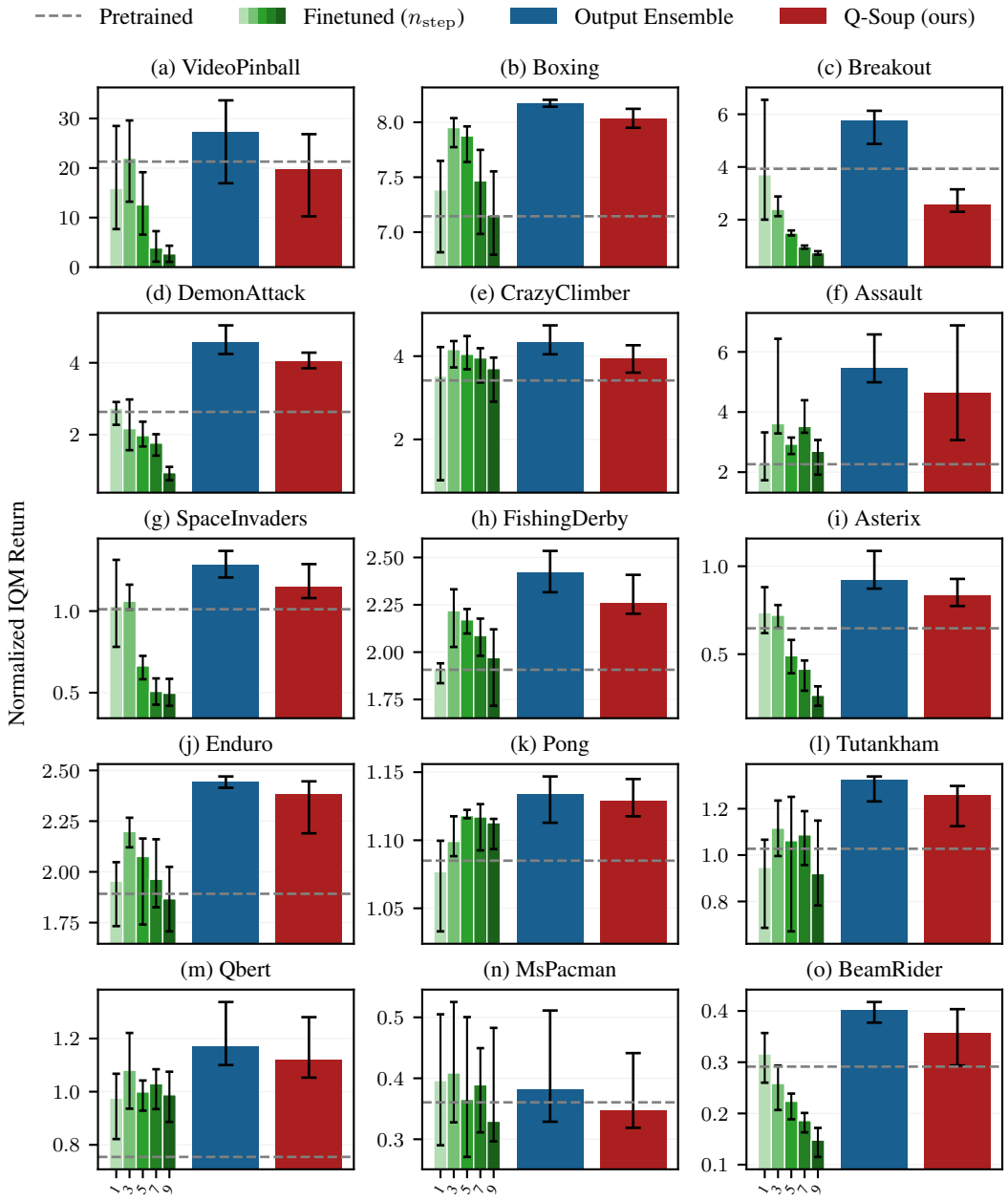

Figure 14: Individual environment results with **DQN** as the learning algorithm demonstrating the normalized IQM return for 15 Atari environments, aggregated over 5 pretrained seeds, comparing 5 individual finetuned models (green shades) with **heterogeneous** configuration of varying n-step return, an output ensemble (blue), and Q-Soup (red) against a pretrained baseline (grey dashed line). Evaluation is performed for 20 episodes with horizon of 27,000 steps each. Error bars denote the 95% confidence intervals. The normalization of the return is carried out by human-score normalization of the raw scores.

B.3 MINTO AND Q-SOUP ON ATARI

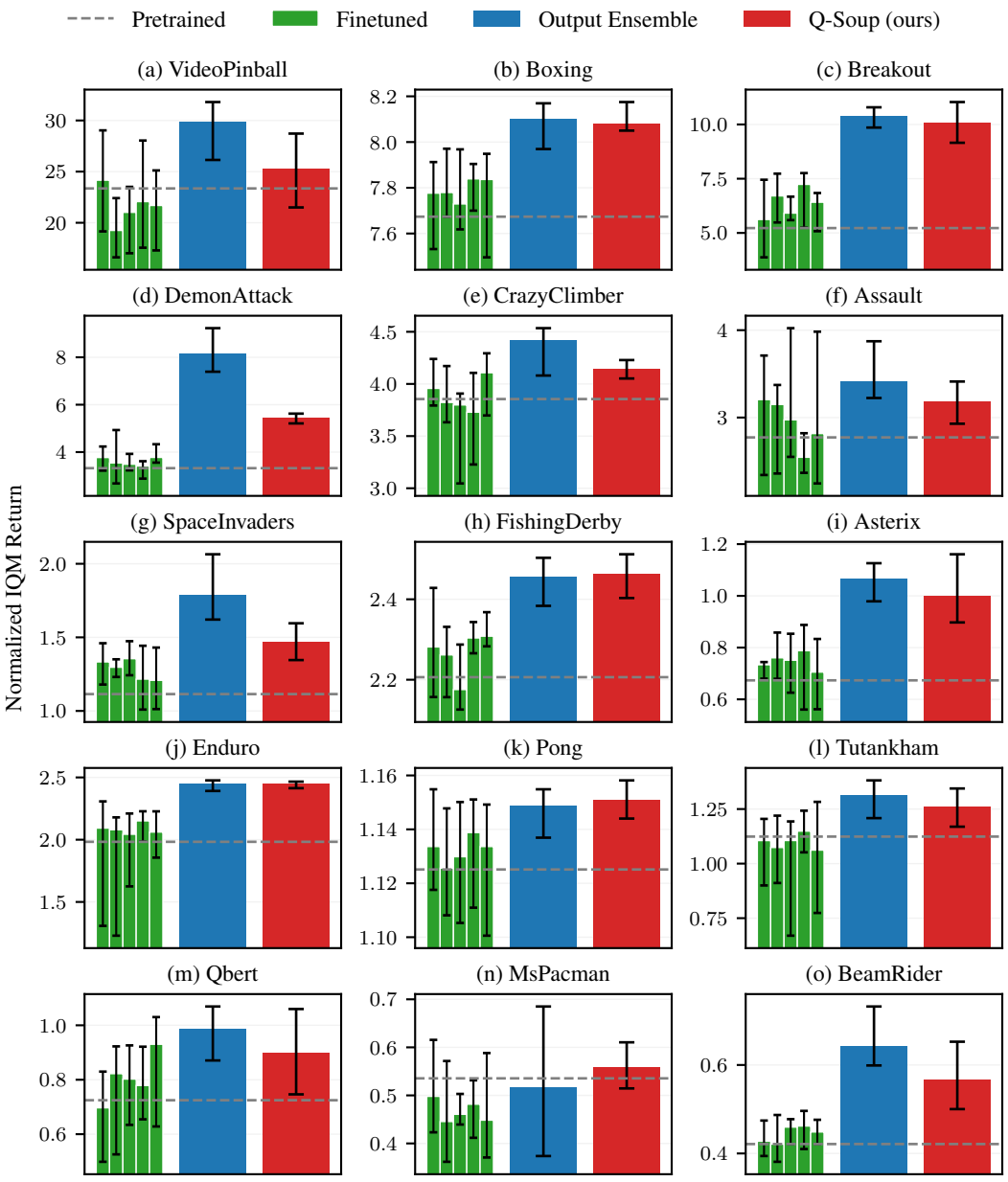

Figure 15: Individual environment results with **MINTO** as the learning algorithm demonstrating the normalized IQM return for 15 Atari environments, aggregated over 5 pretrained seeds, comparing 5 individual finetuned models (green) with **homogeneous** configuration, an output ensemble (blue), and Q-Soup (red) against a pretrained baseline (grey dashed line). Evaluation is performed for 20 episodes with horizon of 27,000 steps each. Error bars denote the 95% confidence intervals. The normalization of the return is carried out by human-score normalization of the raw scores.

B.4 Discount Factor and Proxy Q-Soup

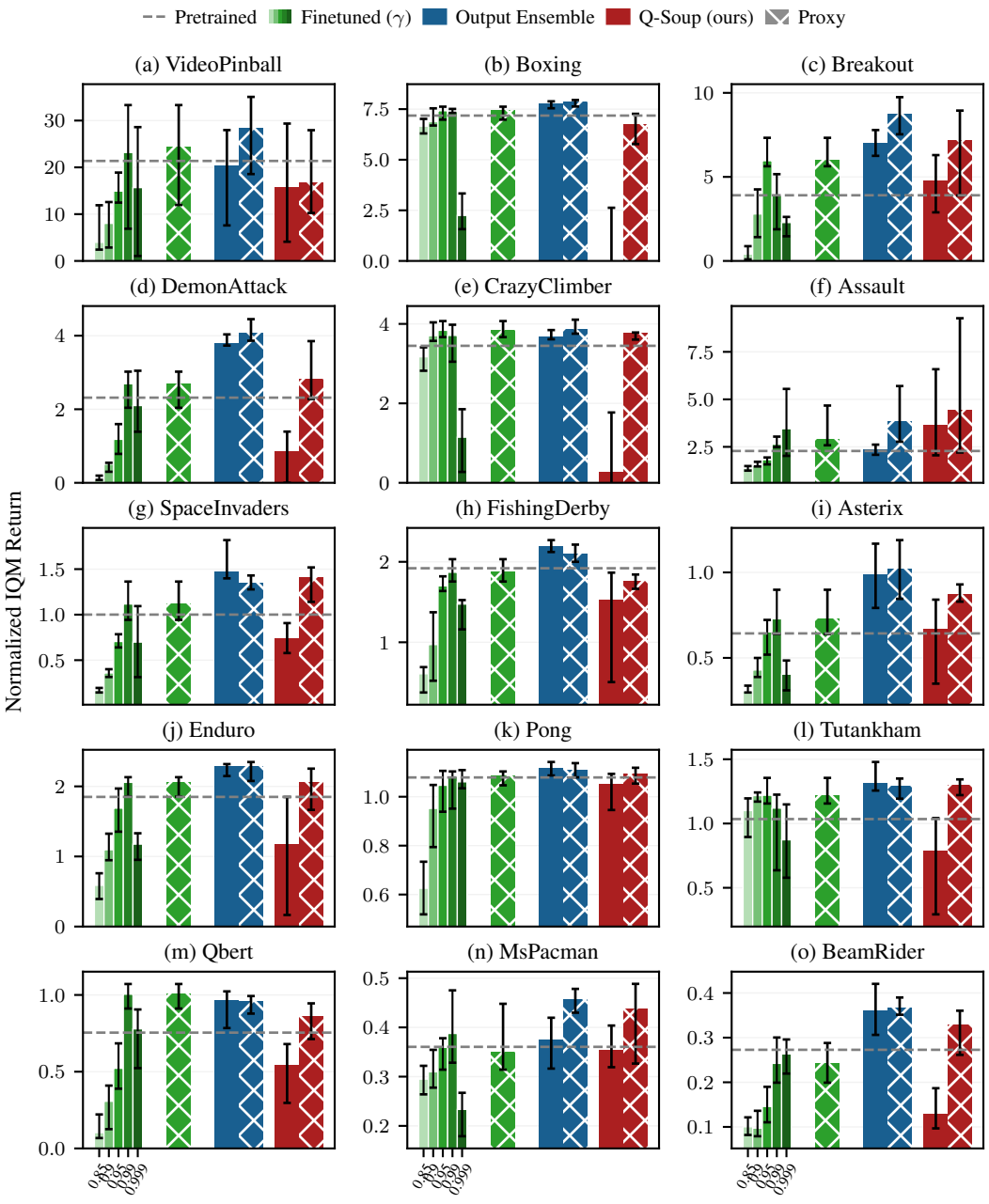

Figure 16: Individual environment results with **DQN** as the learning algorithm demonstrating the normalized IQM return for 15 Atari environments, aggregated over 5 pretrained seeds, comparing 5 individual finetuned models (green shades) with **heterogeneous** configurations of varying discount factor (γ), an Output Ensemble (blue), proxy-based Output Ensemble (blue+cross), Q-Soup (red), Proxy Q-Soup (red+cross), and proxy-selected best model (green+cross) against a pretrained baseline (grey dashed line). Evaluation is performed over 20 episodes with a horizon of 27,000 steps each. Error bars denote the 95% confidence intervals. Returns are normalized using human-score normalization of the raw scores.

B.5 MAXMIN DQN AND Q-SOUP

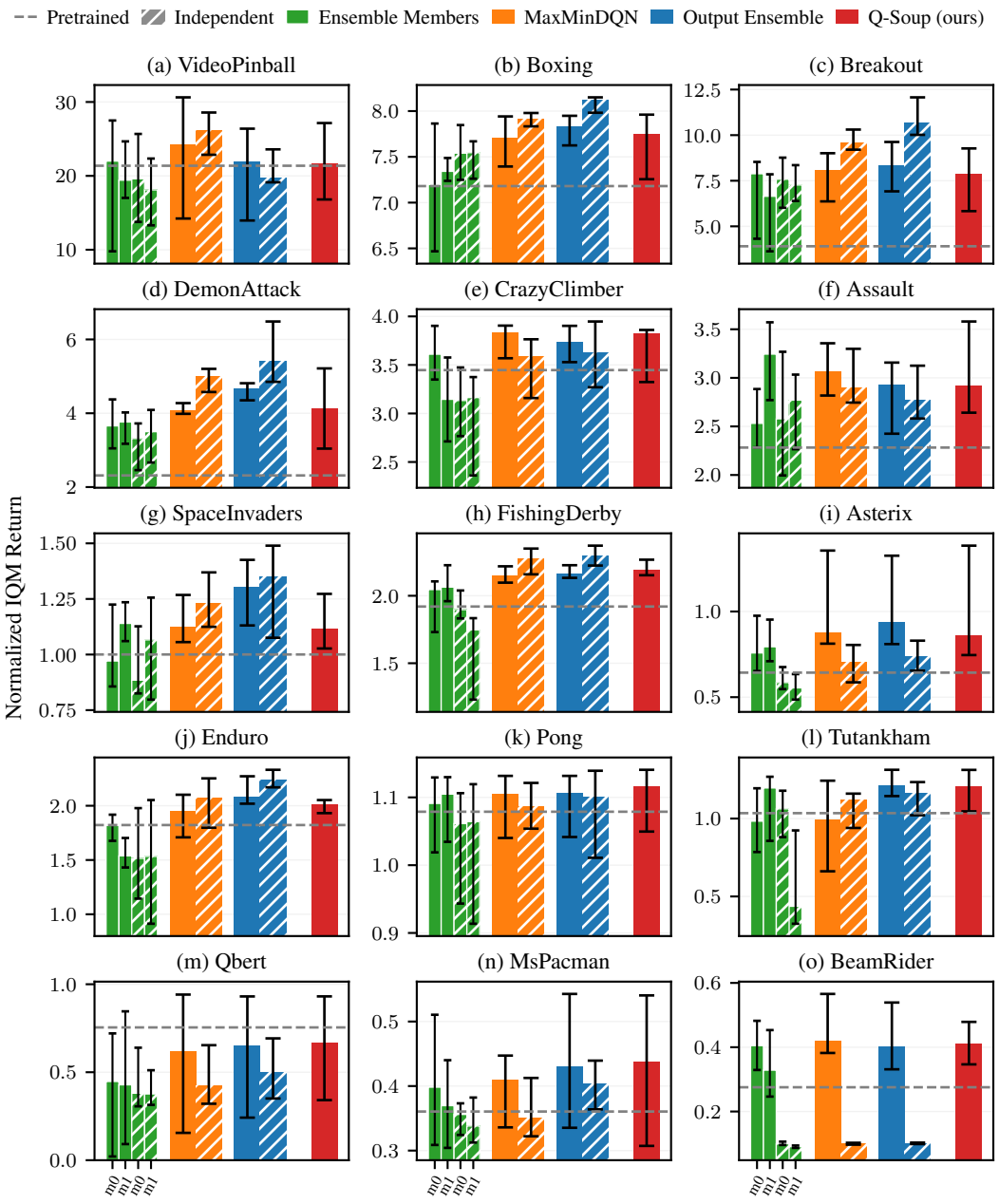

Figure 17: Individual environment results with **Maxmin DQN** as the learning algorithm demonstrating the normalized IQM return for 15 Atari environments, aggregated over 5 DQN pretrained seeds, comparing the 2 ensemble members initialized from a shared pretrained DQN (green) or independently initialized (green+stripes), Maxmin DQN with pretrained initialization (orange) or independent initialization (orange+stripes), the Output Ensemble with pretrained initialization (blue) or independent initialization (blue+stripes), and Q-Soup (red) against the pretrained baseline (grey dashed line). Q-Soup is constructed only from the pretrained-initialized ensemble members. Evaluation is performed over 20 episodes with a horizon of 27,000 steps each. Error bars denote the 95% confidence intervals. Returns are normalized using human-score normalization of the raw scores.

B.6 Stochastic Weight Averaging and Q-Soup

One prominent approach to model averaging is Stochastic Weight Averaging (SWA) (Izmailov et al., 2018), which averages model parameters along an optimization trajectory to obtain flatter solutions with improved generalization. SWA operates along a different axis from ensemble learning: it averages checkpoints along the training trajectory of a single model, whereas Q-Soup and traditional output-space ensembling combine multiple independently finetuned models. These approaches can therefore be naturally combined by first applying SWA to each finetuned model and subsequently using the resulting models as ingredients for Q-Soup or the Output Ensemble. In Fig. 18, we study this interaction by increasing the number of checkpoints k averaged by SWA. As k increases, SWA progressively improves the individual finetuned models, increasing their normalized IQM return from 1.04 at $k = 1$ to 1.23 at $k = 5$. In contrast, Q-Soup and the Output Ensemble remain relatively stable across the SWA window, suggesting that temporal averaging provides limited additional benefit once models are combined in either weight or output space. Q-Soup achieves its highest performance of 1.36 at $k = 3$, matching the Output Ensemble and remaining approximately 10% above the best-performing SWA models at $k = 5$ (1.23). At $k = 3$, Q-Soup outperforms the corresponding SWA models on all 15 games, with particularly large gains on *Breakout* (46.2%), *DemonAttack* (44.9%), and *Asterix* (34.4%). Even at $k = 5$, where SWA achieves its highest performance, Q-Soup remains approximately 6% higher while performing similarly to its no-SWA baseline ($k = 1$). These results suggest that the gains from Q-Soup are primarily driven by averaging across finetuned models, with limited additional benefit from temporal averaging within each model. Nevertheless, further investigation is needed to better understand the interaction between temporal and cross-model averaging.

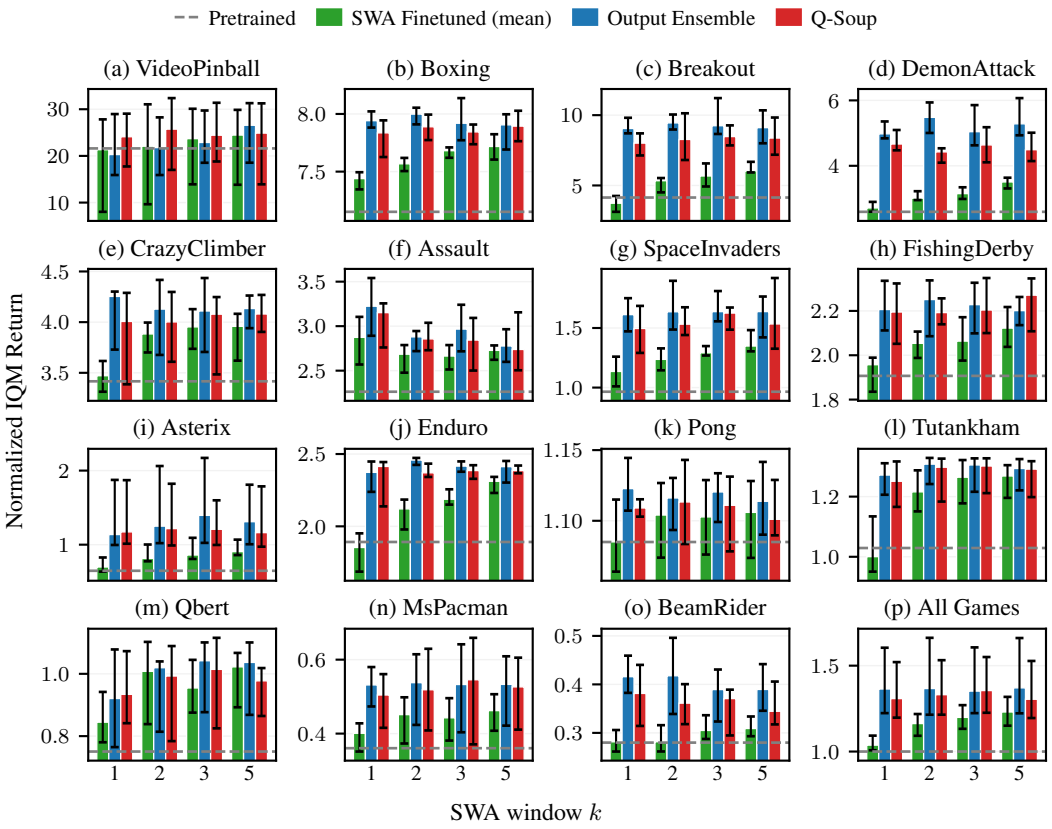

Figure 18: Empirical results of Stochastic Weight Averaging (SWA) and its effect on Q-Soup and output-space ensembling with PQN across 15 Atari environments. We compare the average SWA finetuned model (green), Output Ensemble (blue), and Q-Soup (red) as the number of checkpoints k used for SWA increases. At $k = 1$, no temporal averaging is applied. Returns are reported relative to the pretrained model as the IQM with 95% confidence intervals over five pretrained seeds.

B.7 Ablation on Ensemble Size and Number of Finetuning Steps

We now ablate several factors underlying the mechanics of our approach. Specifically, we study the effect of increasing the number of finetuned models and the number of finetuning steps. In Fig. 19 (left), we report the performance of Q-Soup, the Output Ensemble baseline, and the best individual finetuned model as we incrementally increase the number of models. Both ensembling approaches consistently outperform the best individual model; however, Q-Soup is more efficient at inference. With 10 models, the output-space ensemble requires all models to be stored and evaluated at inference, whereas Q-Soup merges them through weight averaging into a single model. In Fig. 19 (right), we vary the number of finetuning steps. Both ensembling approaches maintain a performance advantage over the best individual model as training progresses. When doubling the finetuning steps to 20M, we observe a slight drop in Q-Soup relative to the Output Ensemble. Nevertheless, Q-Soup remains on par with the Output Ensemble on average, while avoiding the need to store multiple models or select the best finetuned model through evaluation.

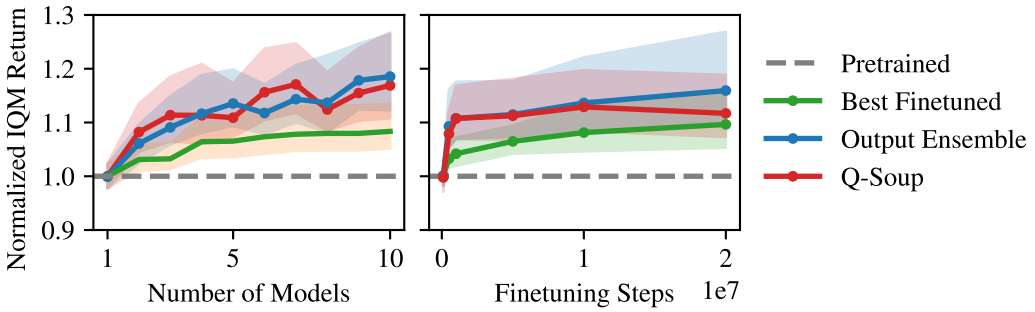

Figure 19: Ablation studies on varying the number of models used in the Q-Soup (**Left**) and the number of finetuning steps (**Right**) with PQN. Normalized IQM return is compared across the best individual finetuned model (green), an output ensemble (blue), and a Q-Soup (red), evaluated against a pretrained seed (grey dashed line). Results are aggregated over five pretrained seeds, with 95% confidence intervals.

B.8 Ablation on Proxy Q-Soup Components

We further investigate the individual contributions of Z-score filtering and softmax weighting to Proxy Q-Soup. In Fig. 20, we ablate each component in the challenging heterogeneous discount-factor setting, comparing them with standard Q-Soup and the full Proxy Q-Soup. The aggregated results show that combining both components yields the strongest overall performance. Softmax weighting accounts for most of the improvement over standard Q-Soup, while Z-score filtering provides a smaller gain when applied independently. Nevertheless, their combination provides the strongest aggregate performance, indicating that filtering weak ingredients and adaptively weighting the retained models can provide complementary benefits.

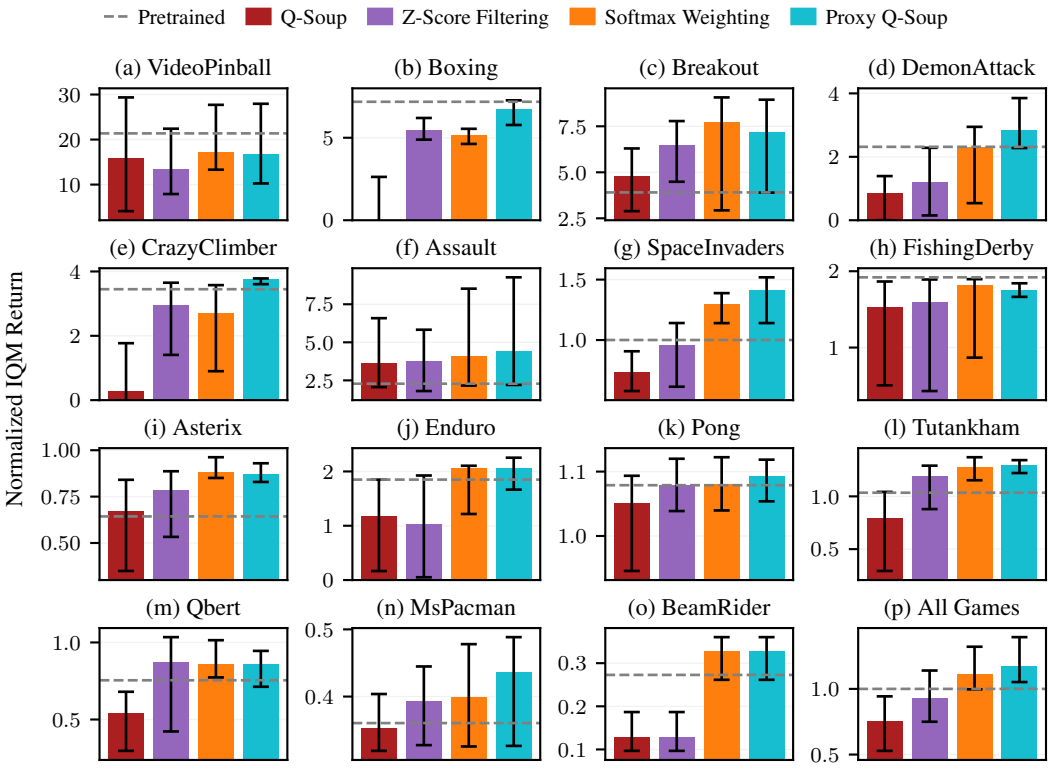

Figure 20: Ablation of Z-score filtering (purple) and softmax weighting (orange), compared with standard Q-Soup (red) and Proxy Q-Soup (cyan), which combines both components. The ablation is conducted with DQN in the heterogeneous discount-factor setting. Human-normalized returns are reported as the IQM with 95% confidence intervals across five pretrained seeds.

B.9 The Performance Landscape Around Q-Soup

To visualize how multiple finetuned models relate to one another and to their parameter average, we construct a two-dimensional representation of weight space containing all models. Let $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N$ denote Q-networks finetuned from the same pretrained model, and let $\boldsymbol{\theta}_{\text{avg}} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{\theta}_i$ denote their Q-Soup. We place $\boldsymbol{\theta}_{\text{avg}}$ at the origin and assign each finetuned model $\boldsymbol{\theta}_i$ to the vertex $v_i = \left(\cos\frac{2\pi(i-1)}{N}, \sin\frac{2\pi(i-1)}{N}\right)$ of a regular N-gon, with indices modulo N. A point p in the sector between v_i and v_{i+1} is mapped to

$$\boldsymbol{\theta}(p) \;=\; a\,\boldsymbol{\theta}_i + b\,\boldsymbol{\theta}_{i+1} + (1-a-b)\,\boldsymbol{\theta}_{\text{avg}}, \qquad \begin{pmatrix} a \\ b \end{pmatrix} = \begin{bmatrix} v_i & v_{i+1} \end{bmatrix}^{-1} p, \tag{11}$$

such that $\boldsymbol{\theta}(v_i) = \boldsymbol{\theta}_i$ and $\boldsymbol{\theta}(0) = \boldsymbol{\theta}_{\text{avg}}$. Within the polygon, each point therefore corresponds to a convex combination of Q-Soup and two neighboring finetuned models, while points outside the polygon extrapolate beyond them.

To construct the performance landscape, we evaluate $\boldsymbol{\theta}(p)$ on an $R \times R$ grid over $[-1-m, 1+m]^2$, together with R points sampled along the segments between $\boldsymbol{\theta}_{\text{avg}}$ and each $\boldsymbol{\theta}_i$. Each point is evaluated using its human-normalized return over E episodes, using identical episode seeds across points, and the remaining values are linearly interpolated to form the heatmap. Each panel uses a color scale ranging from the 5th to the 95th percentile of its evaluated returns. We use $N = 5$, $R = 32$, $m = 0.5$, and $E = 20$. For DQN, we reduce the resolution to $R = 16$ for *Enduro* and *VideoPinball* due to their longer evaluation time.

Fig. 21 illustrates this construction and compares the landscape around the pretrained parameters $\boldsymbol{\theta}\text{pre}$ with that around Q-Soup $\boldsymbol{\theta}\text{avg}$. When centered around $\boldsymbol{\theta}\text{pre}$, the finetuned models surround a lower-performing region near the pretrained solution. In contrast, centering the same models around $\boldsymbol{\theta}\text{avg}$ reveals a connected high-performing region, with Q-Soup lying near its center. These landscapes are obtained by aggregating across all games and pretrained seeds. Specifically, Fig. 21 aggregates the individual heatmaps shown in Figs. 22–26.

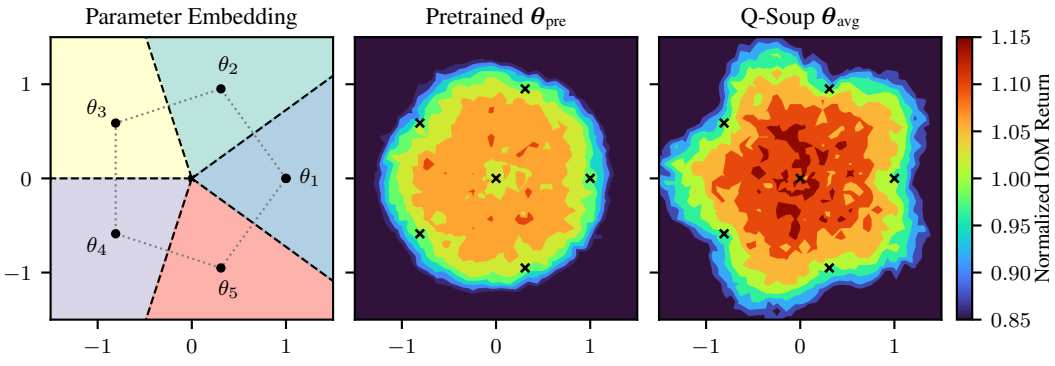

Figure 21: Projection of the finetuned weights onto a plane using the the pretrained (**Middle**) and averaged (**Right**) parameters as the midpoints. The **Left** figure shows a visualization of the projection. We report the IQM of the return normalized by the pretrained seed over 15 Atari games and 5 seeds.

B.10 Q-Soup Performance Landscapes with PQN

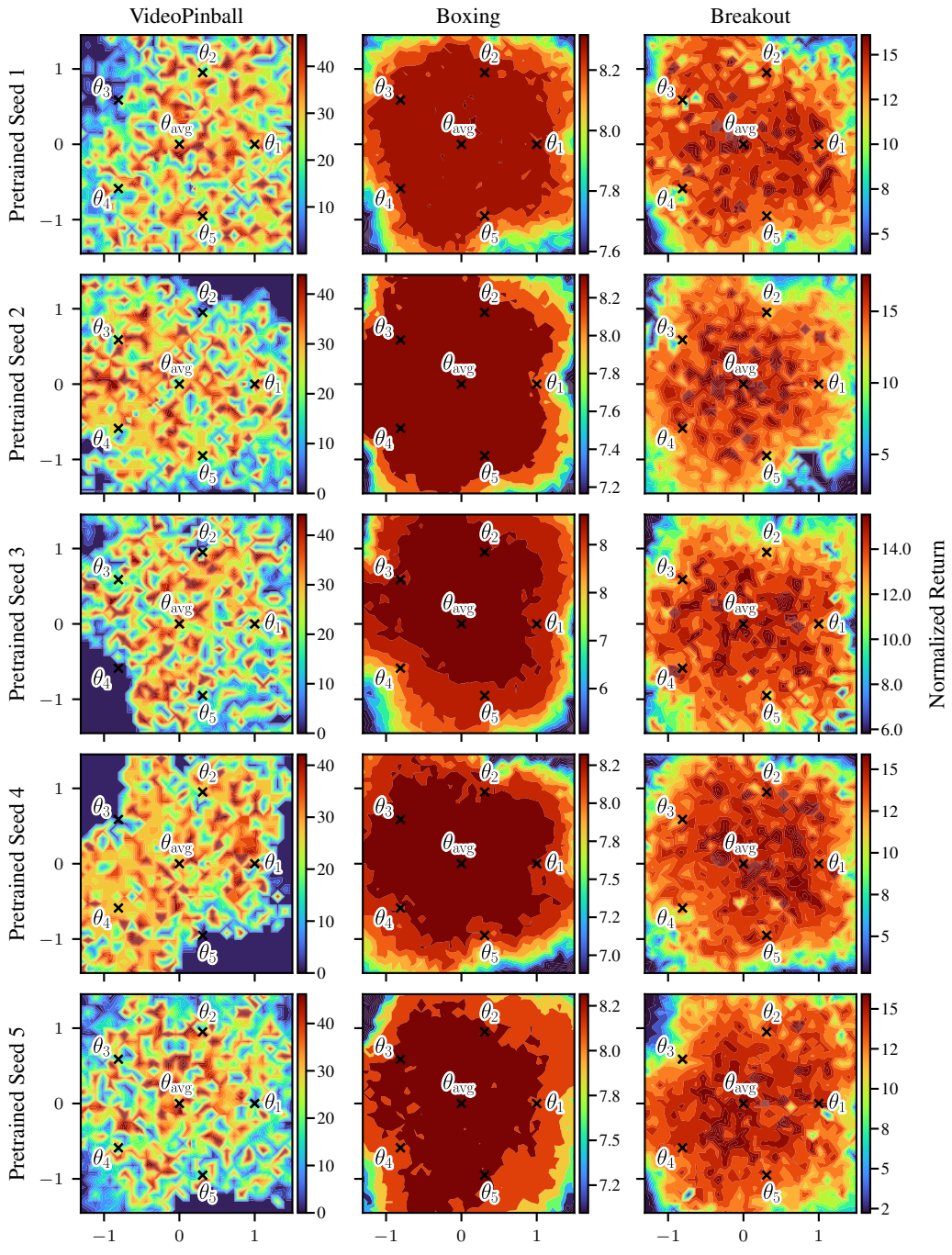

Figure 22: Performance landscapes for **PQN** in the *homogeneous* setting across *VideoPinball*, *Boxing*, and *Breakout*. Each column corresponds to an environment and each row to one of five pretrained seeds. For each seed, we project the five finetuned models $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_5$ and their parameter average $\boldsymbol{\theta}_{\text{avg}}$ onto a two-dimensional plane. Colors indicate human-normalized return, highlighting the connectivity of the finetuned models and the performance of their average. Each heatmap has a resolution of 32×32, with each point evaluated over $8 \times 27{,}000$ steps.

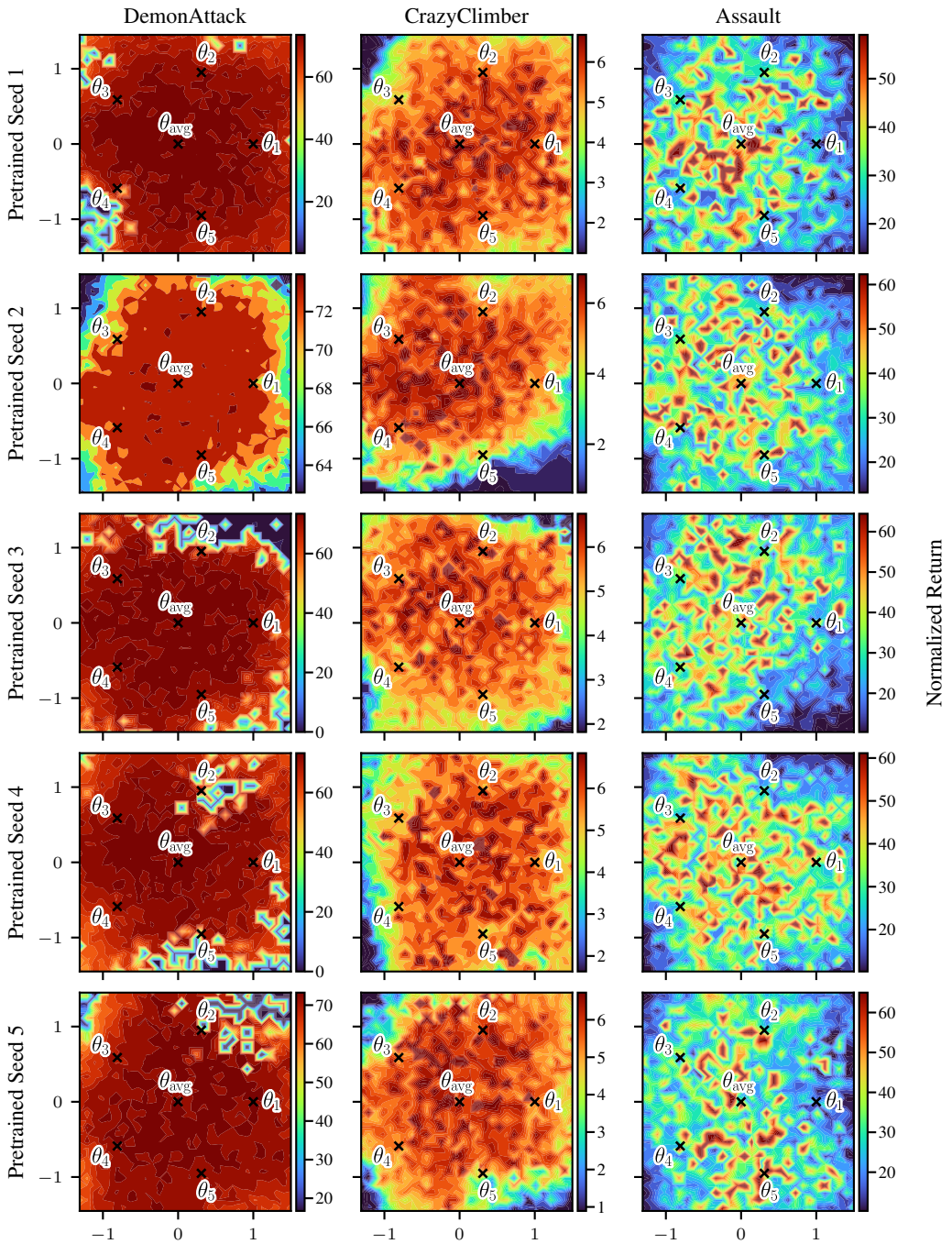

Figure 23: Performance landscapes for **PQN** in the *homogeneous* setting across *DemonAttack*, *CrazyClimber*, and *Assault*. Each column corresponds to an environment and each row to one of five pretrained seeds. For each seed, we project the five finetuned models $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_5$ and their parameter average $\boldsymbol{\theta}_{\text{avg}}$ onto a two-dimensional plane. Colors indicate human-normalized return, highlighting the connectivity of the finetuned models and the performance of their average. Each heatmap has a resolution of 32×32, with each point evaluated over $8 \times 27{,}000$ steps.

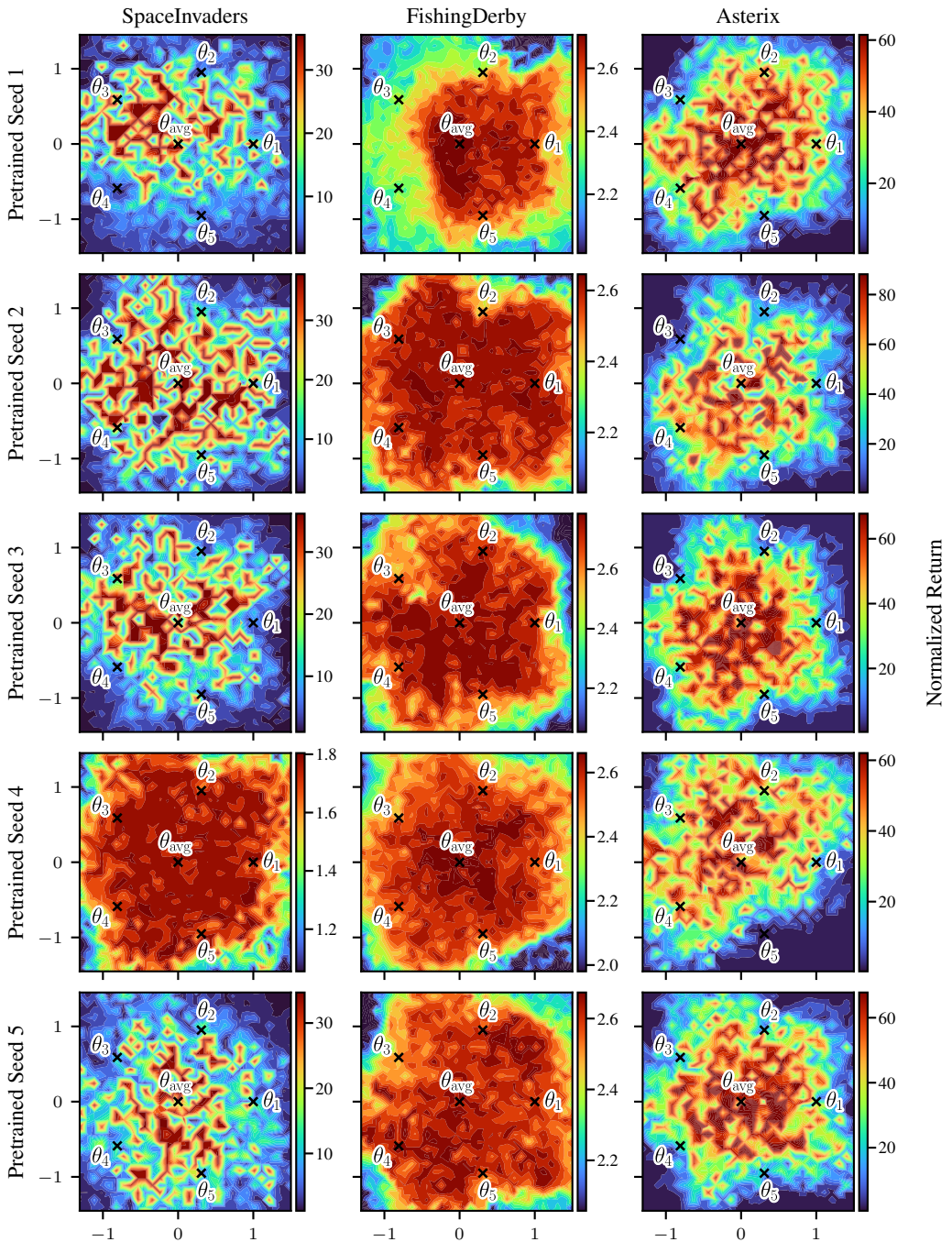

Figure 24: Performance landscapes for **PQN** in the *homogeneous* setting across *SpaceInvaders*, *FishingDerby*, and *Asterix*. Each column corresponds to an environment and each row to one of five pretrained seeds. For each seed, we project the five finetuned models $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_5$ and their parameter average $\boldsymbol{\theta}_{\text{avg}}$ onto a two-dimensional plane. Colors indicate human-normalized return, highlighting the connectivity of the finetuned models and the performance of their average. Each heatmap has a resolution of 32×32, with each point evaluated over $8 \times 27{,}000$ steps.

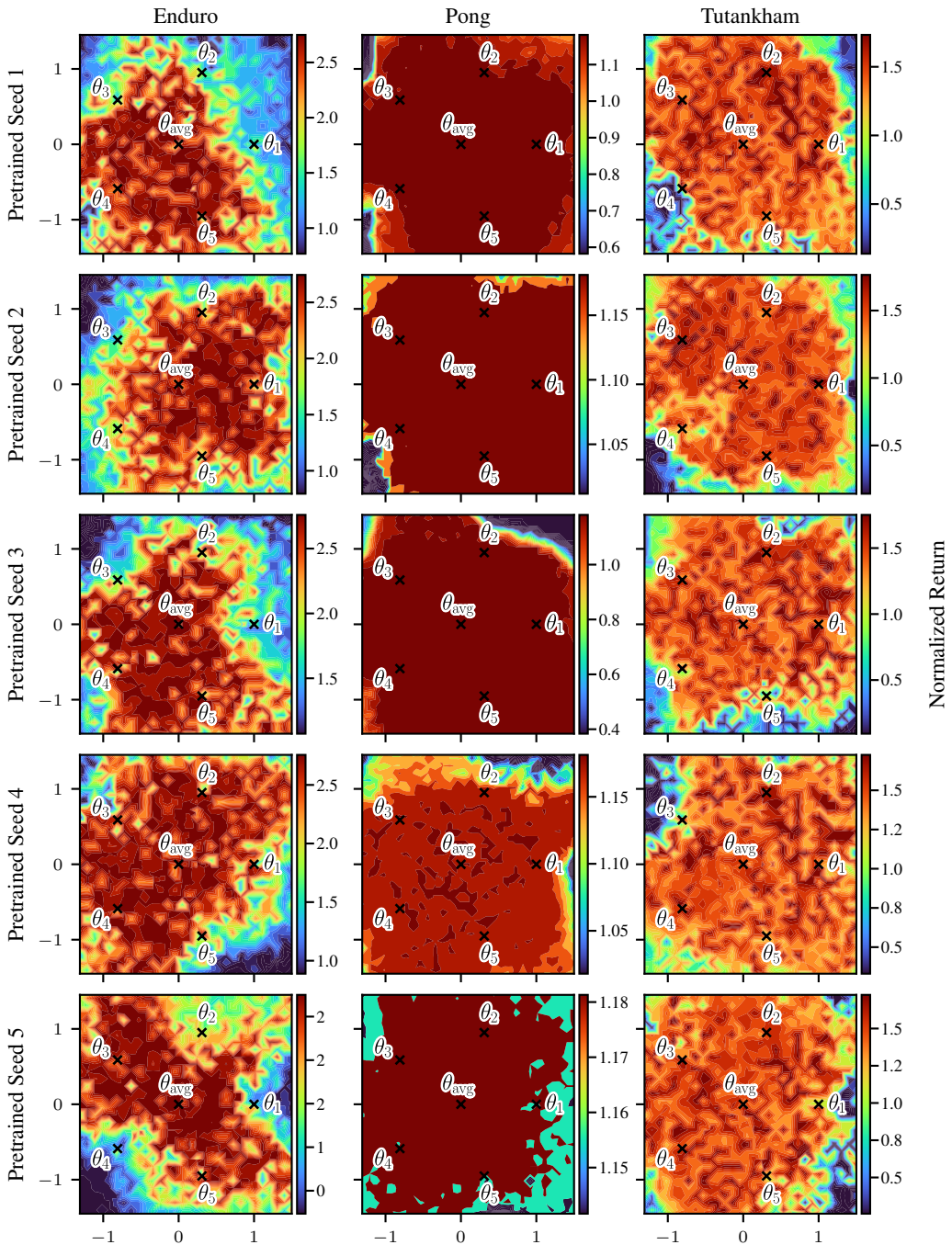

Figure 25: Performance landscapes for **PQN** in the *homogeneous* setting across *Enduro*, *Pong*, and *Tutankham*. Each column corresponds to an environment and each row to one of five pretrained seeds. For each seed, we project the five finetuned models $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_5$ and their parameter average $\boldsymbol{\theta}_{\text{avg}}$ onto a two-dimensional plane. Colors indicate human-normalized return, highlighting the connectivity of the finetuned models and the performance of their average. Each heatmap has a resolution of 32×32, with each point evaluated over $8 \times 27{,}000$ steps.

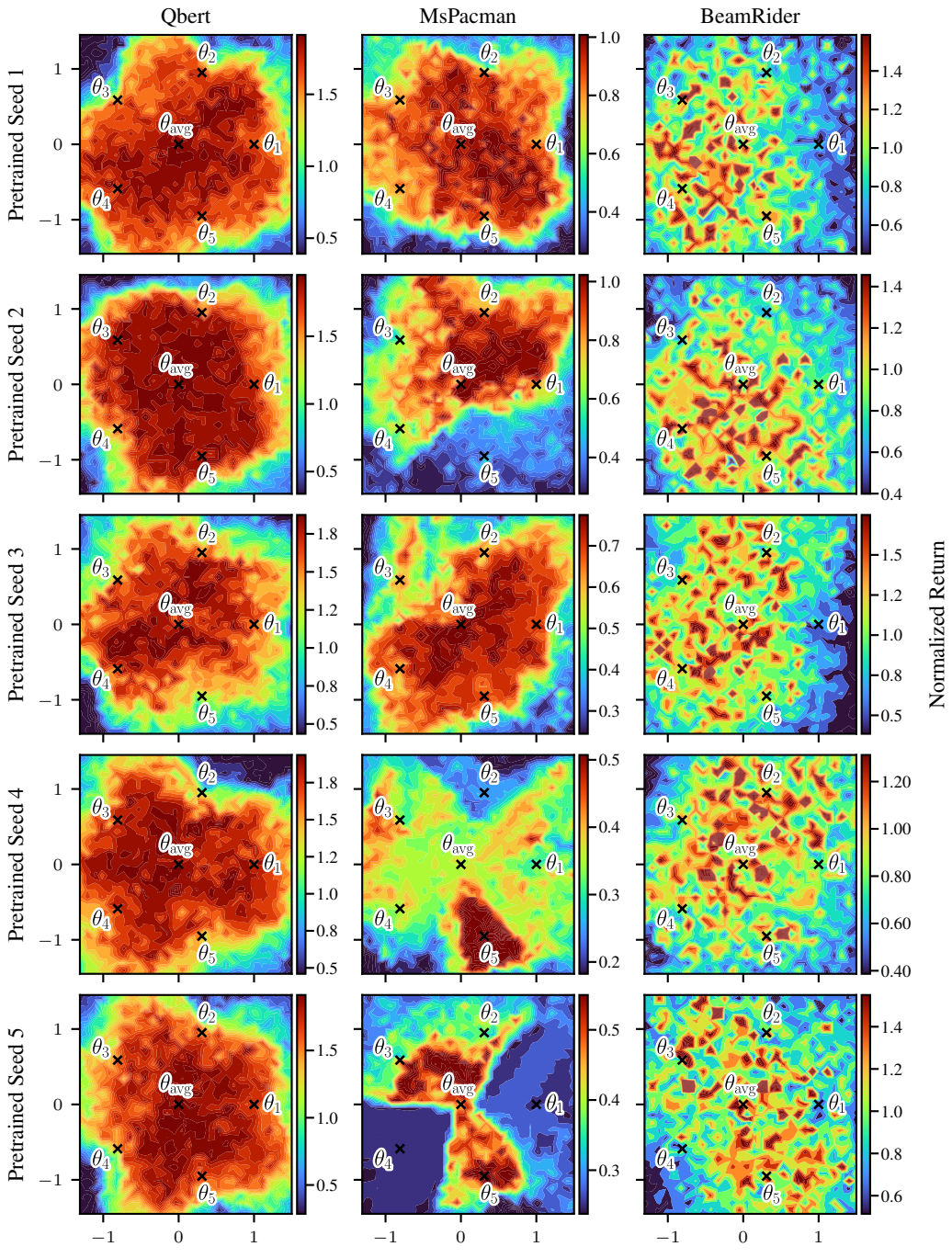

Figure 26: Performance landscapes for **PQN** in the *homogeneous* setting across *Qbert*, *MsPacman*, and *BeamRider*. Each column corresponds to an environment and each row to one of five pretrained seeds. For each seed, we project the five finetuned models $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_5$ and their parameter average $\boldsymbol{\theta}_{\text{avg}}$ onto a two-dimensional plane. Colors indicate human-normalized return, highlighting the connectivity of the finetuned models and the performance of their average. Each heatmap has a resolution of 32×32, with each point evaluated over $8 \times 27{,}000$ steps.

B.11 Q-SOUP PERFORMANCE LANDSCAPES WITH DQN

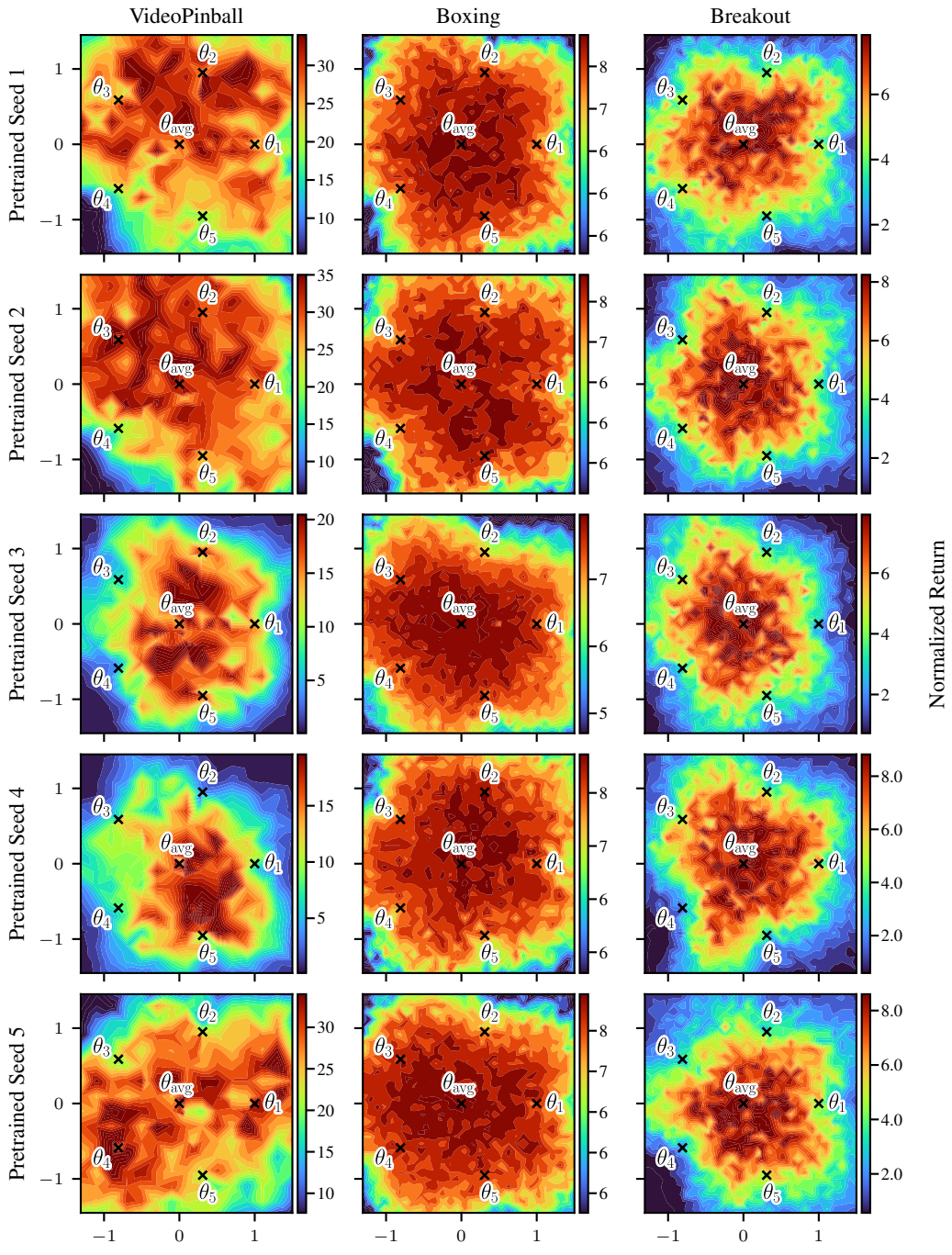

Figure 27: Performance landscapes for **DQN** in the *homogeneous* setting across *VideoPinball*, *Boxing*, and *Breakout*. Each column corresponds to an environment and each row to one of five pretrained seeds. For each seed, we project the five finetuned models $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_5$ and their parameter average $\boldsymbol{\theta}_{\text{avg}}$ onto a two-dimensional plane. Colors indicate human-normalized return, highlighting the connectivity of the finetuned models and the performance of their average. Each heatmap has a resolution of 32×32, with each point evaluated over 20 episodes of 27,000 steps each. For *VideoPinball*, we use a 16×16 resolution due to its longer evaluation time.

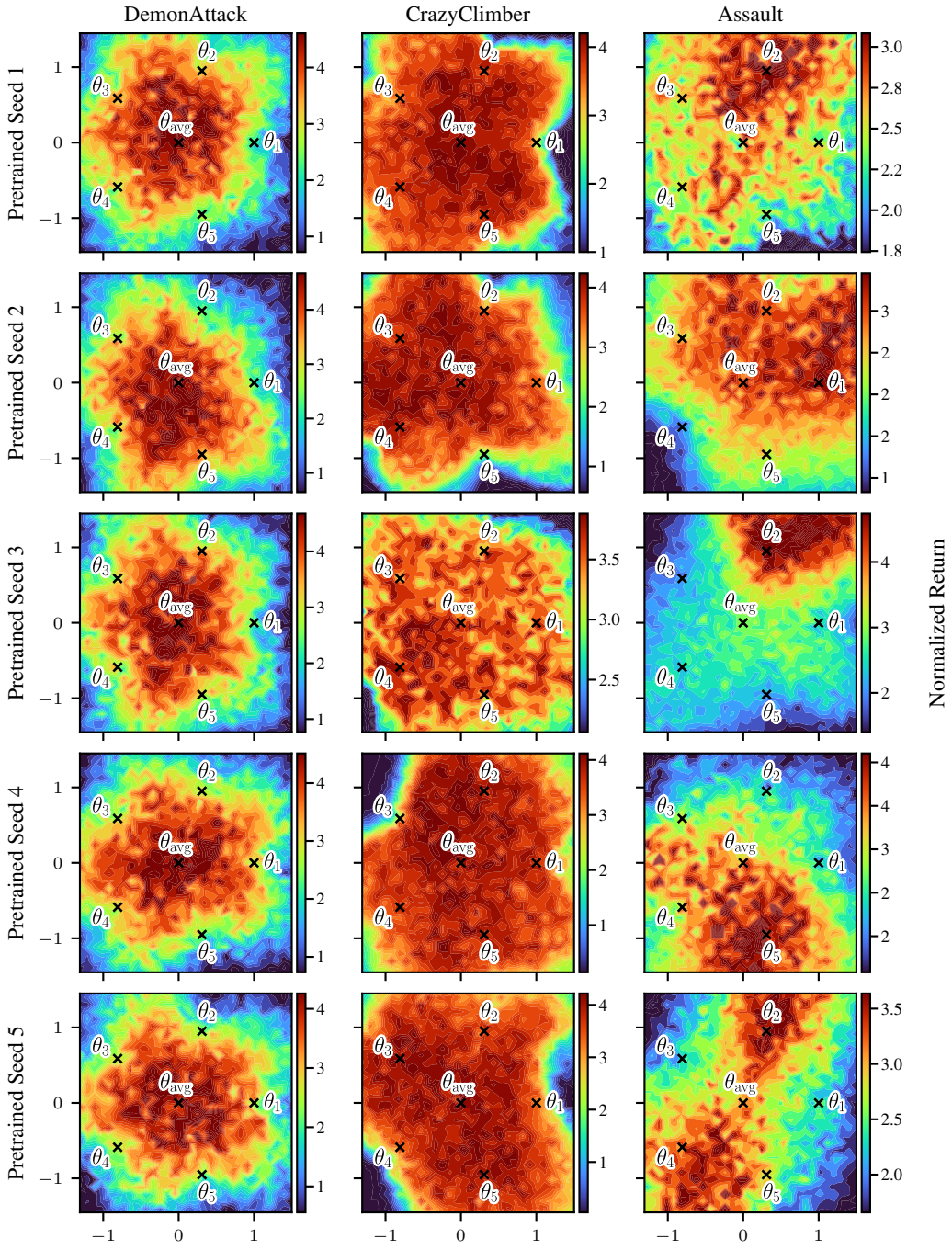

Figure 28: Performance landscapes for **DQN** in the *homogeneous* setting across *DemonAttack*, *CrazyClimber*, and *Assault*. Each column corresponds to an environment and each row to one of five pretrained seeds. For each seed, we project the five finetuned models $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_5$ and their parameter average $\boldsymbol{\theta}_{\text{avg}}$ onto a two-dimensional plane. Colors indicate human-normalized return, highlighting the connectivity of the finetuned models and the performance of their average. Each heatmap has a resolution of 32×32, with each point evaluated over 20 episodes of 27,000 steps each.

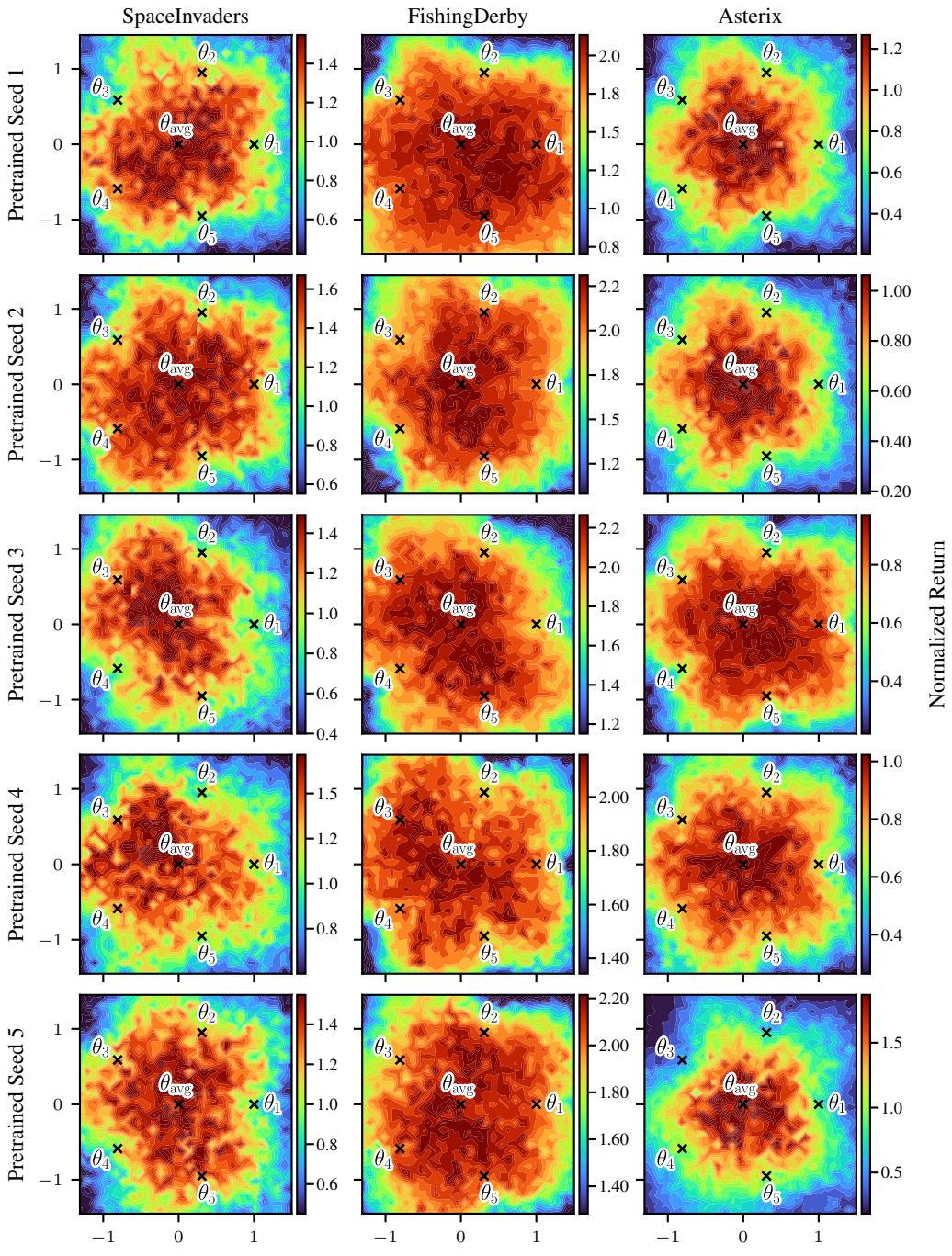

Figure 29: Performance landscapes for **DQN** in the *homogeneous* setting across *SpaceInvaders*, *FishingDerby*, and *Asterix*. Each column corresponds to an environment and each row to one of five pretrained seeds. For each seed, we project the five finetuned models $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_5$ and their parameter average $\boldsymbol{\theta}_{\text{avg}}$ onto a two-dimensional plane. Colors indicate human-normalized return, highlighting the connectivity of the finetuned models and the performance of their average. Each heatmap has a resolution of 32×32, with each point evaluated over 20 episodes of 27,000 steps each.

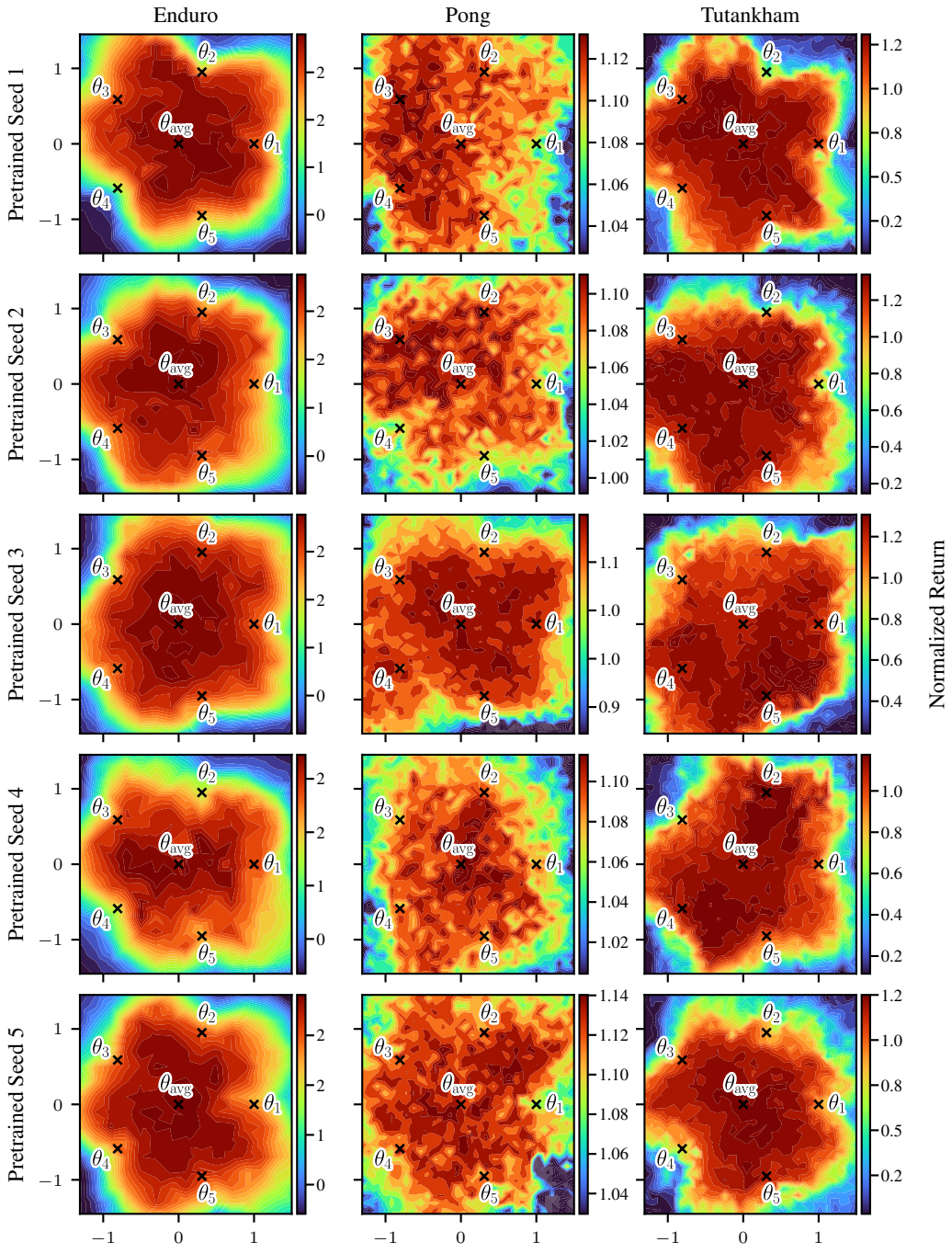

Figure 30: Performance landscapes for **DQN** in the *homogeneous* setting across *Enduro*, *Pong*, and *Tutankham*. Each column corresponds to an environment and each row to one of five pretrained seeds. For each seed, we project the five finetuned models $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_5$ and their parameter average $\boldsymbol{\theta}_{\text{avg}}$ onto a two-dimensional plane. Colors indicate human-normalized return, highlighting the connectivity of the finetuned models and the performance of their average. Each heatmap has a resolution of 32×32, with each point evaluated over 20 episodes of 27,000 steps each. For *Enduro*, we use a 16×16 resolution due to its longer evaluation time.

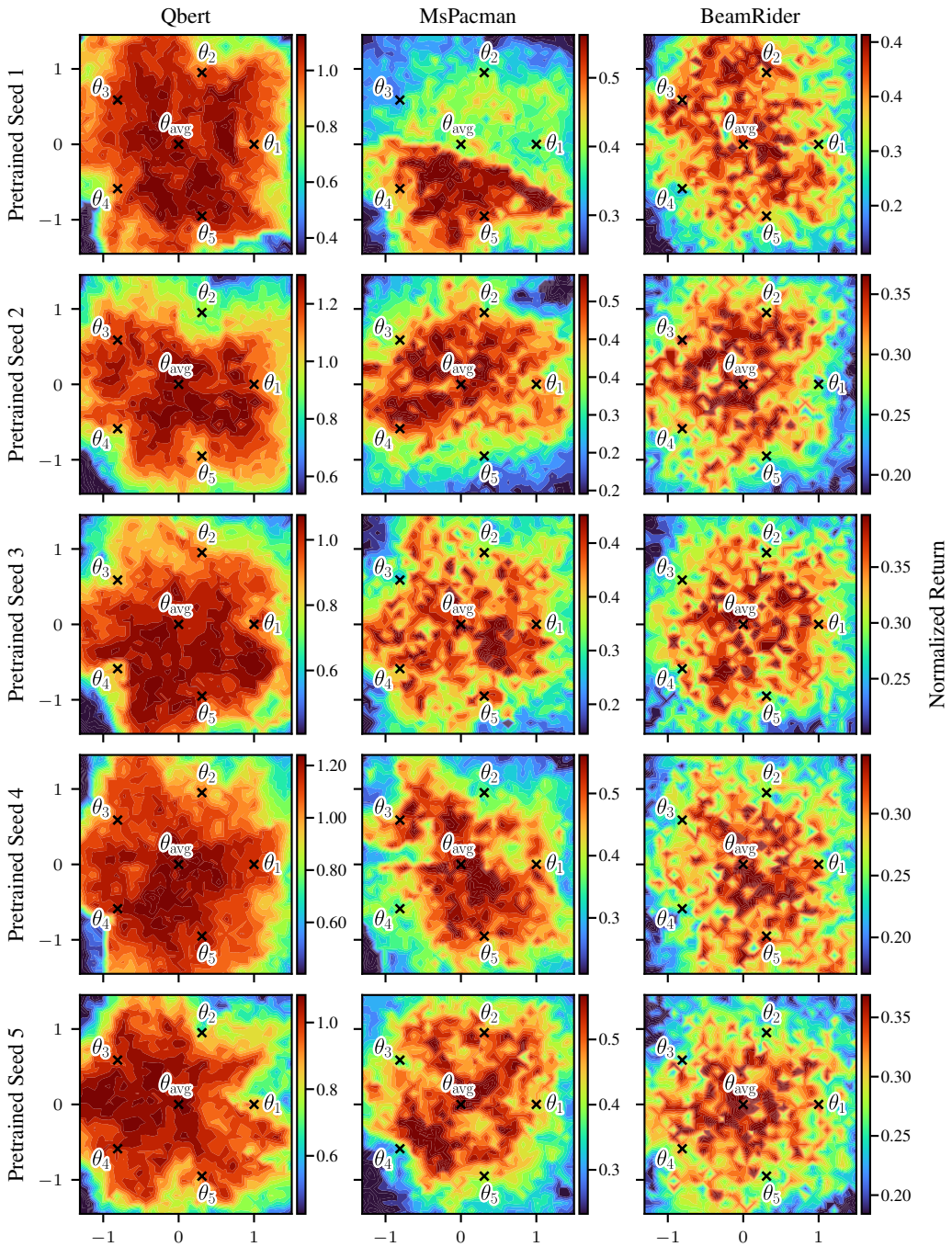

Figure 31: Performance landscapes for **DQN** in the *homogeneous* setting across *Qbert*, *MsPacman*, and *BeamRider*. Each column corresponds to an environment and each row to one of five pretrained seeds. For each seed, we project the five finetuned models $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_5$ and their parameter average $\boldsymbol{\theta}_{\text{avg}}$ onto a two-dimensional plane. Colors indicate human-normalized return, highlighting the connectivity of the finetuned models and the performance of their average. Each heatmap has a resolution of 32×32, with each point evaluated over 20 episodes of 27,000 steps each.

C GEOMETRIC ANALYSIS OF ENSEMBLE MEMBERS

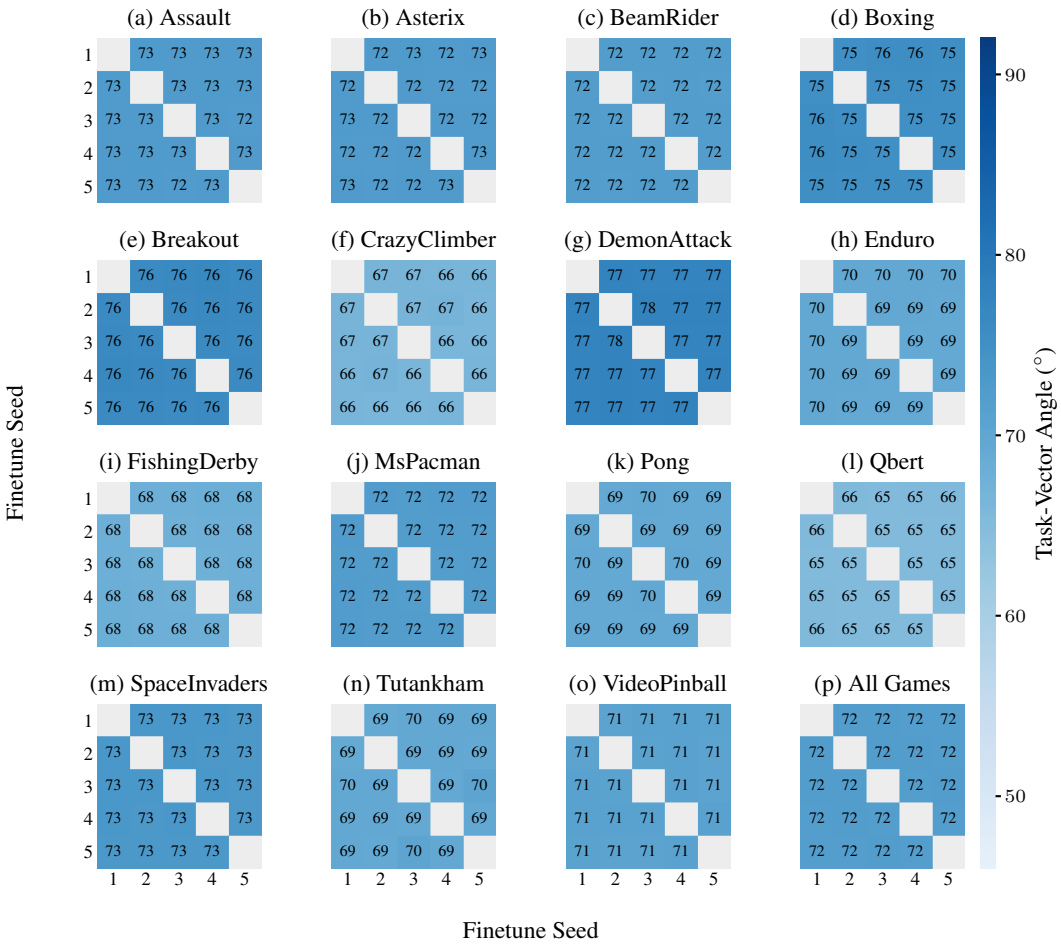

Figure 32: **Task-vector angles, homogeneous finetuning.** Pairwise angle (degrees) between the task vectors $\tau_i = \theta_i - \theta_{\text{pre}}$ of five members finetuned from the same pretrained DQN with different random seeds, at epoch 10. Panels (a)–(o) show one game each. Every cell is the median over five pretrained seeds. Panel (p) pools all games and pretrained seeds (median of 75 angles per cell). Within each game the angle is almost constant across seed pairs, so every member is about equally far from every other. The color scale is shared with fig. 33.

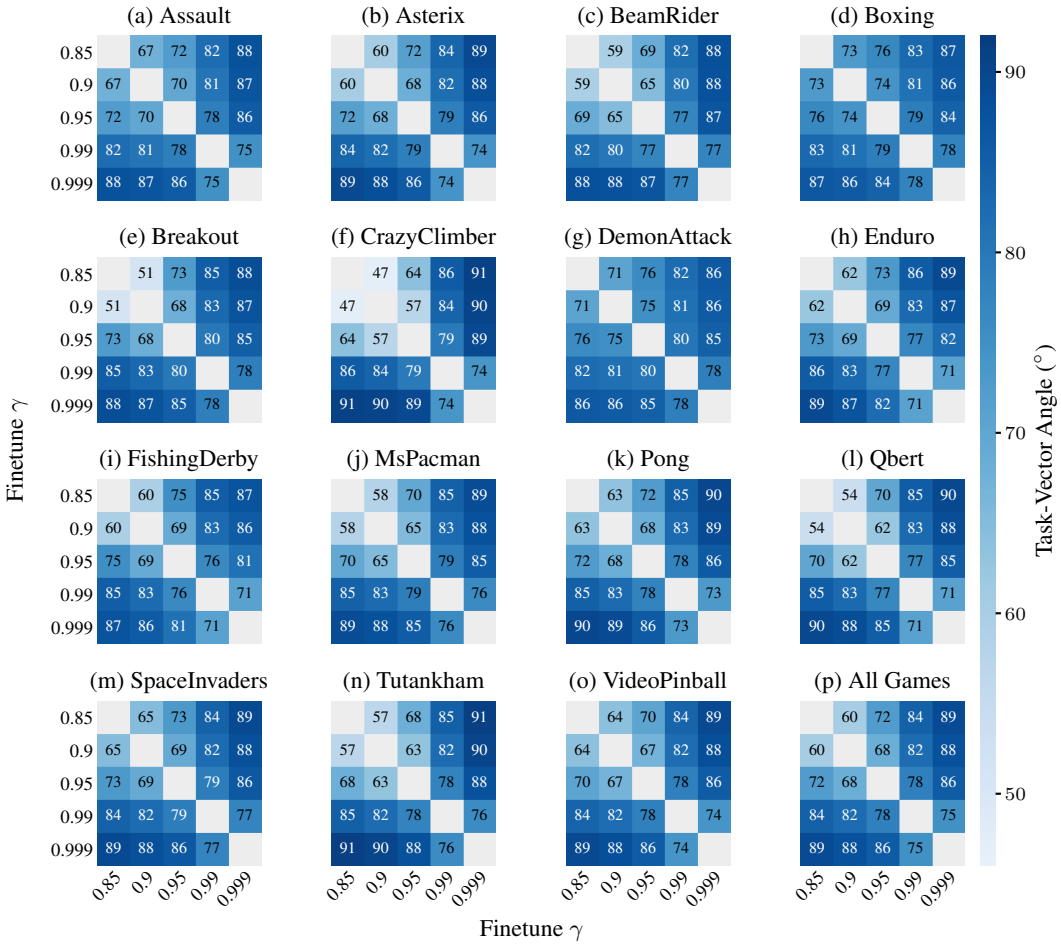

Figure 33: **Task-vector angles, heterogeneous discount factors.** Same as fig. 32, but each member is finetuned with a different discount factor $\gamma \in \{0.85, 0.9, 0.95, 0.99, 0.999\}$. The angle grows with the gap between discount factors: neighboring values give the most aligned updates, and the extremes ($\gamma = 0.85$ vs. 0.999) are close to orthogonal ($\approx 89°$). The structure is consistent across all 15 games. The color scale is shared with fig. 32.

D ADDITIONAL OBSERVATIONS

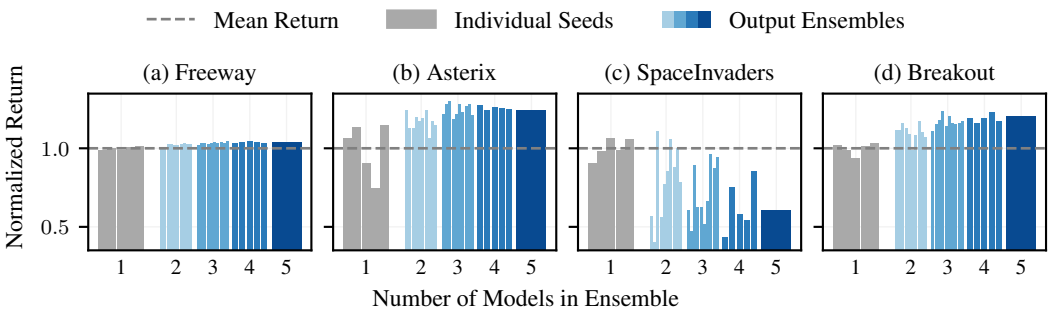

Figure 34: Empirical demonstration of how ensembling in output space can increase return across four MinAtar environments. We show the normalized returns of five independently trained PQN agents (grey) and all their output-space ensemble combinations (blue shades). Returns are normalized by the mean return of the five individual seeds.

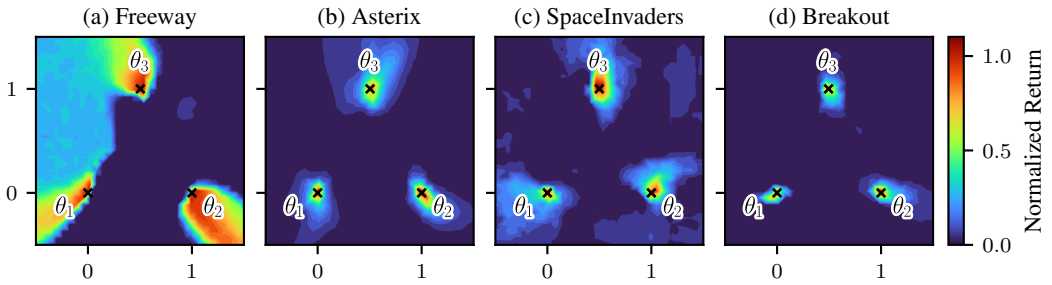

Figure 35: Empirical demonstration of the parameter landscape around three independently trained PQN agents (×) across four MinAtar environments. We construct a two-dimensional plane defined by the parameters of the three agents and show the normalized returns of the corresponding policies across this plane. Returns are normalized by the mean return of the three individual agents.

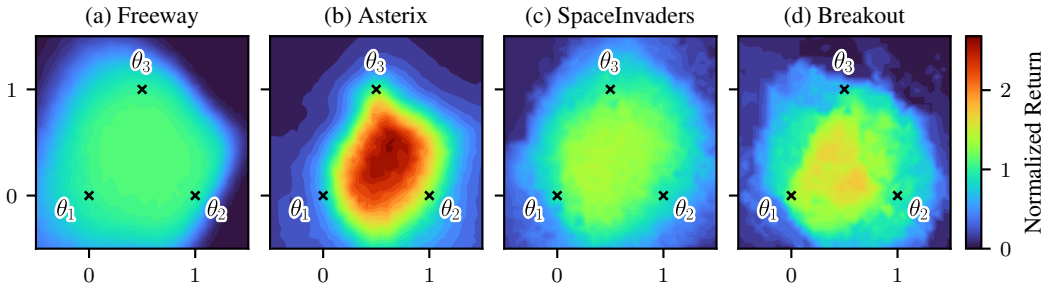

Figure 36: Empirical demonstration of the flat regions induced by simplex subspaces learned with SubQ (*SubPQN*) across four MinAtar environments. We show subspaces defined by three trained agents (×), over which the heatmaps show the normalized return of the corresponding policies. Returns are normalized by the mean return of the three individual agents.

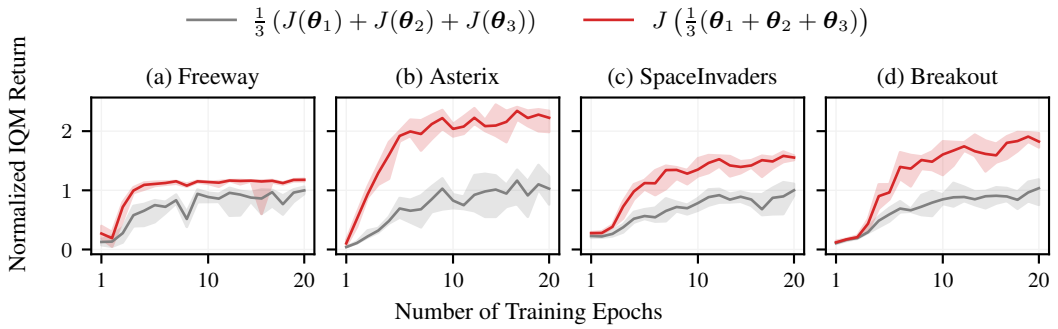

Figure 37: Empirical demonstration of the flat region observation across four MinAtar environments. We show the normalized return of the weight-averaged model (red) and the average return of the three individual agents (grey) over SubQ (SubPQN) training, reported as the IQM with a 95% confidence interval across five seeds. Returns are normalized by the mean return of the three individual agents.

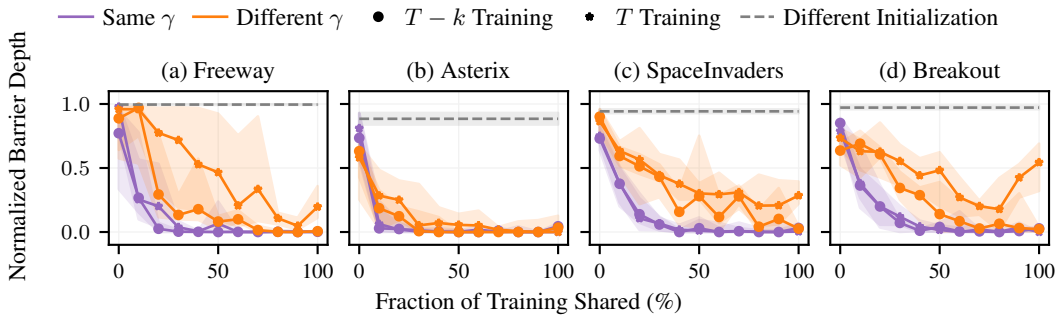

Figure 38: Normalized barrier depth as a function of the fraction of training shared for PQN across four MinAtar environments, reported as the IQM with a 95 % confidence interval across five seeds. After sharing k training steps, two models are trained independently either for the remaining $T - k$ steps or for an additional T steps. We compare runs using the same or different discount factors γ after splitting, with independently initialized models shown as a baseline.

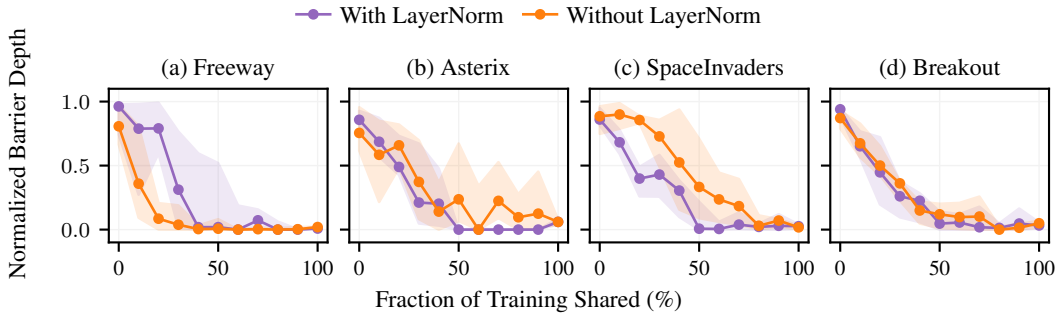

Figure 39: Normalized barrier depth as a function of the fraction of training shared for DQN across four MinAtar environments, reported as the IQM with a 95 % confidence interval across five seeds. After sharing k training steps, two models are trained independently for the remaining $T - k$ steps. We study the effect of layer normalization on linear connectivity, with independently initialized models shown as a baseline.

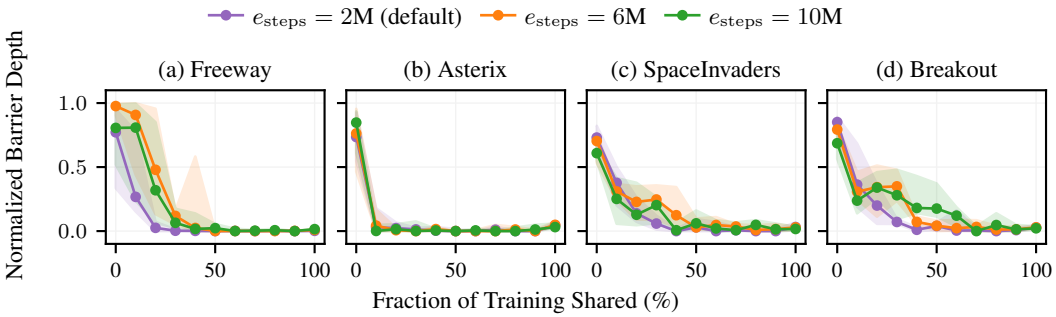

Figure 40: Normalized barrier depth as a function of the fraction of training shared for DQN across four MinAtar environments, reported as the IQM with a 95 % confidence interval across five seeds. After sharing k training steps, two models are trained independently for the remaining $T - k$ steps. We study the effect of different exploration intervals on linear connectivity, with independently initialized models shown as a baseline.

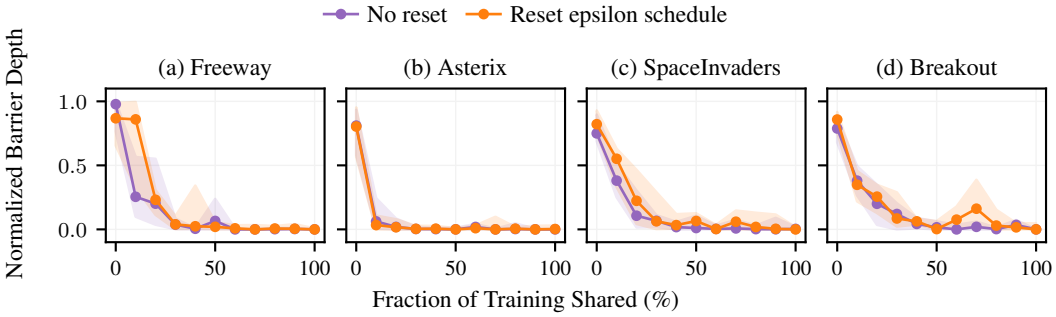

Figure 41: Normalized barrier depth as a function of the fraction of training shared for DQN across four MinAtar environments, reported as the IQM with a 95 % confidence interval across five seeds. After sharing k training steps, two models are trained independently for T steps. We study the effect of resetting ϵ used in the ϵ-greedy policy on linear connectivity, with independently initialized models shown as a baseline.

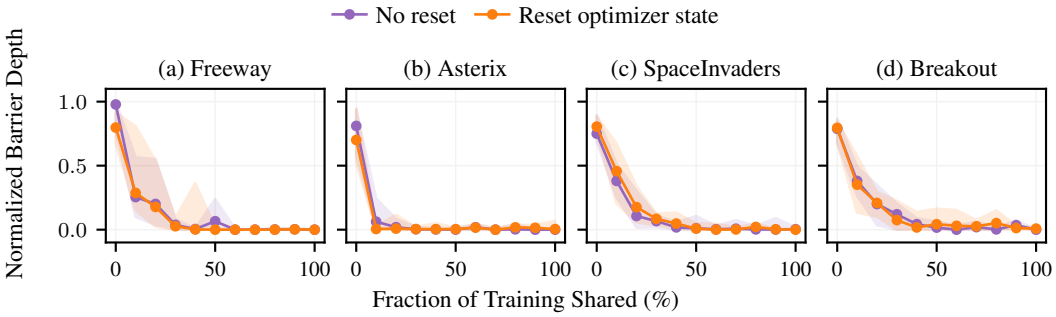

Figure 42: Normalized barrier depth as a function of the fraction of training shared for DQN across four MinAtar environments, reported as the IQM with a 95 % confidence interval across five seeds. After sharing k training steps, two models are trained independently for T steps. We study the effect of resetting optimizer state on linear connectivity, with independently initialized models shown as a baseline. The outcome matches what is observed in supervised learning.

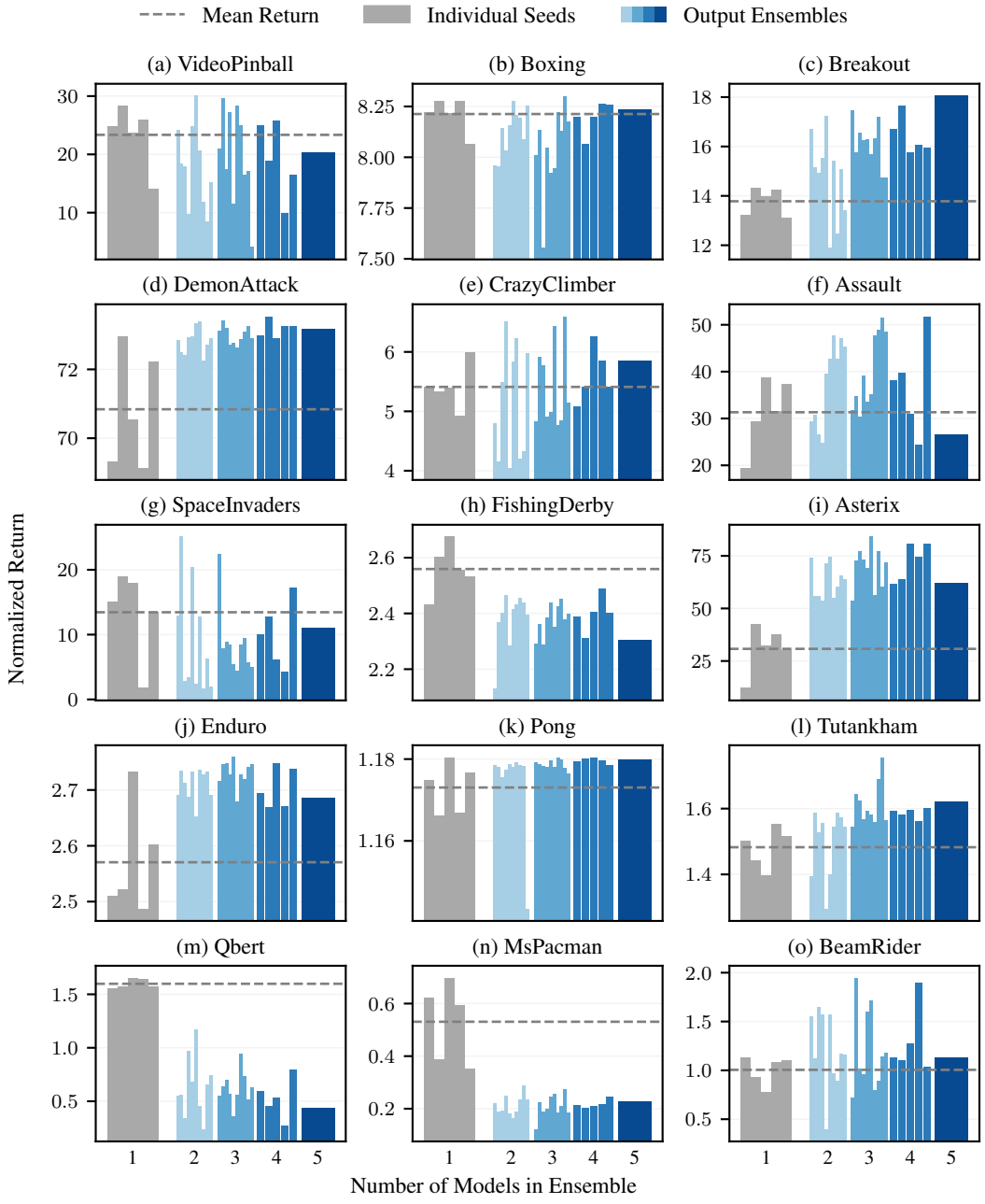

Figure 43: Empirical demonstration of how ensembling in output space can increase return across 15 Atari environments. We show the normalized returns of five independently trained PQN agents (grey) and all their output-space ensemble combinations (blue shades). The normalization of the return is carried out by human-score normalization of the raw scores.

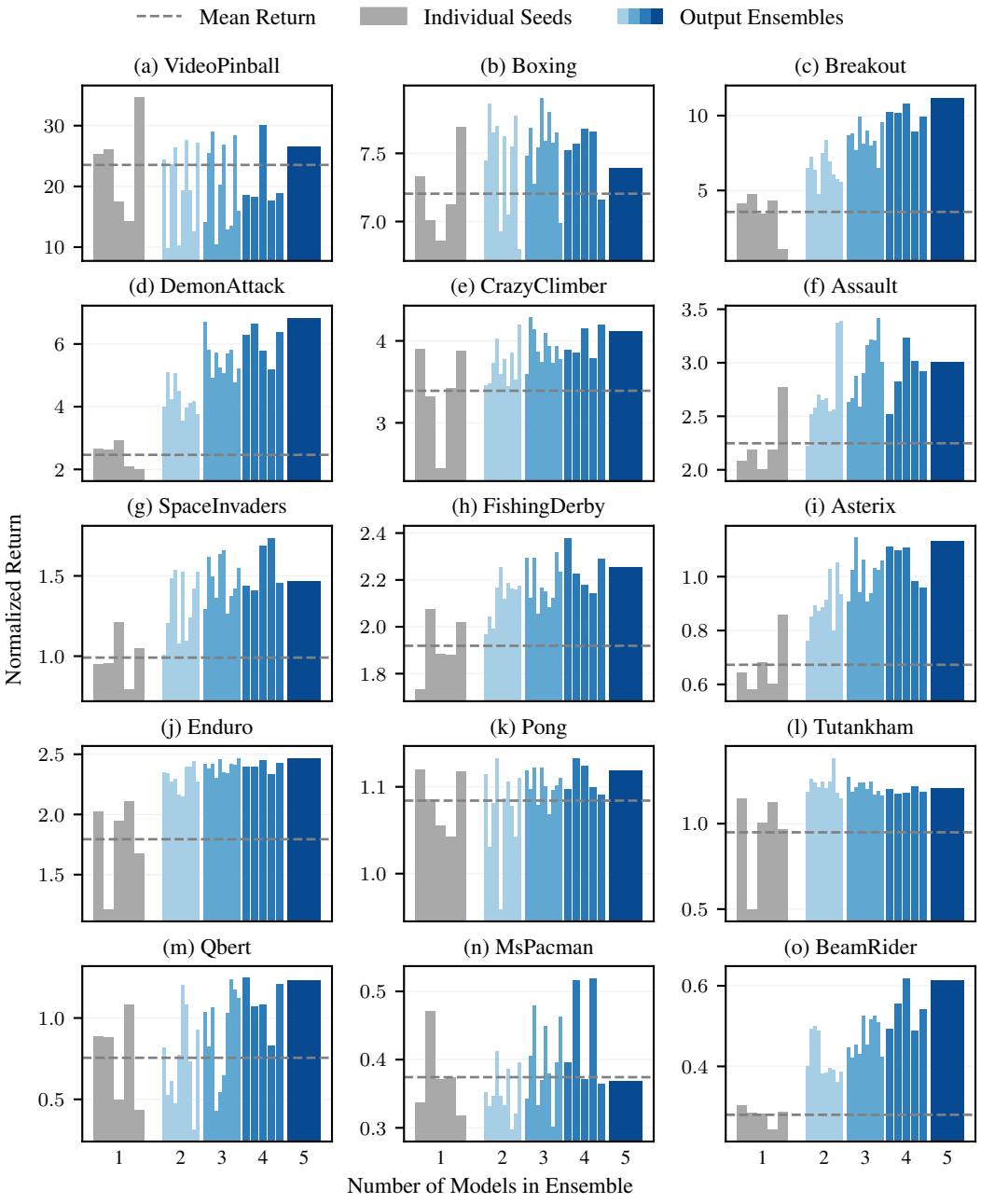

Figure 44: Empirical demonstration of how ensembling in output space can increase return across 15 Atari environments. We show the normalized returns of five independently trained DQN agents (grey) and all their output-space ensemble combinations (blue shades). The normalization of the return is carried out by human-score normalization of the raw scores.

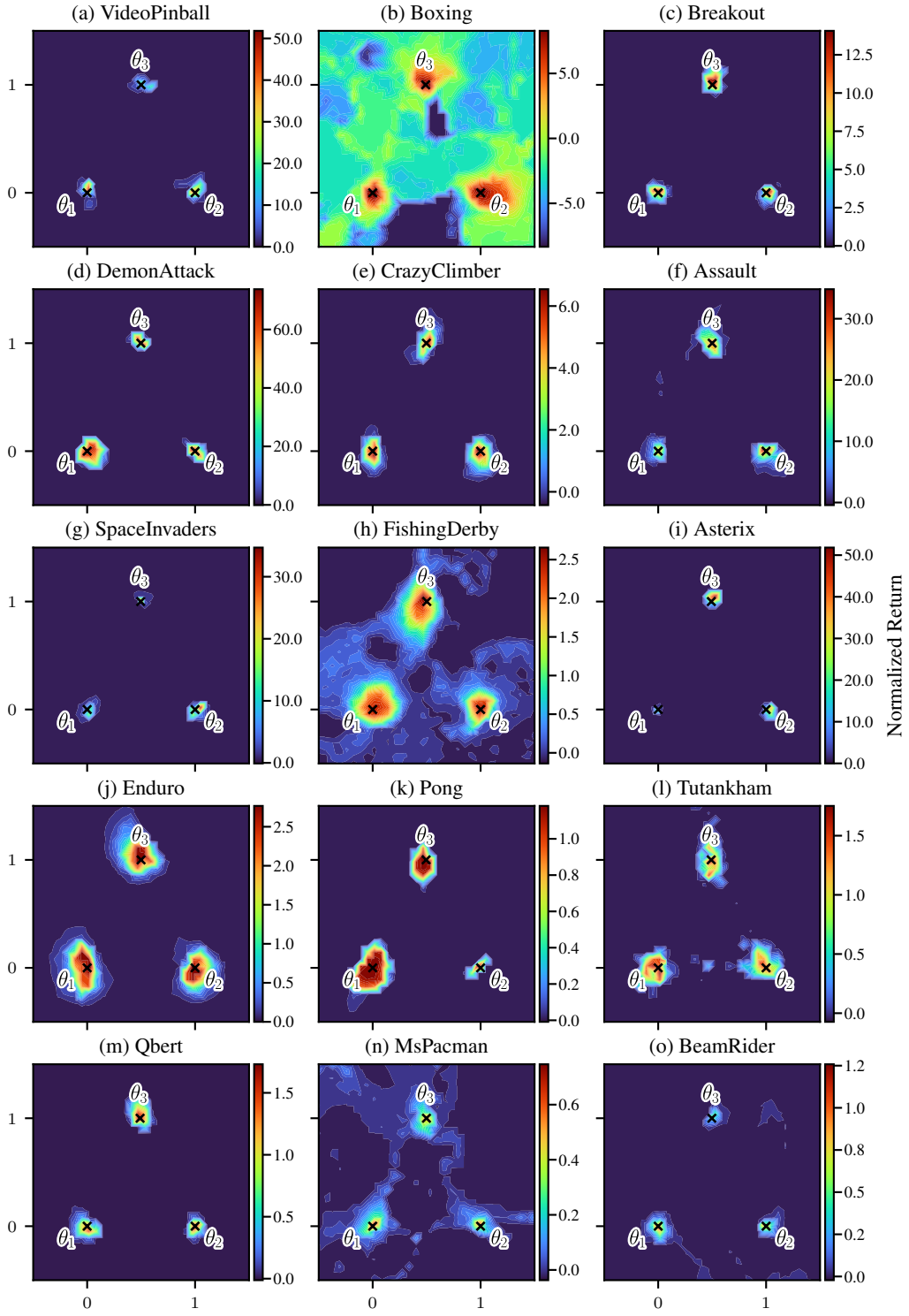

Figure 45: Empirical demonstration of the parameter landscape around three independently trained PQN agents (×) across 15 Atari environments. We construct a two-dimensional plane defined by the parameters of the three agents and show the normalized returns of the corresponding policies across this plane. Returns are normalized by the mean return of the three individual agents.

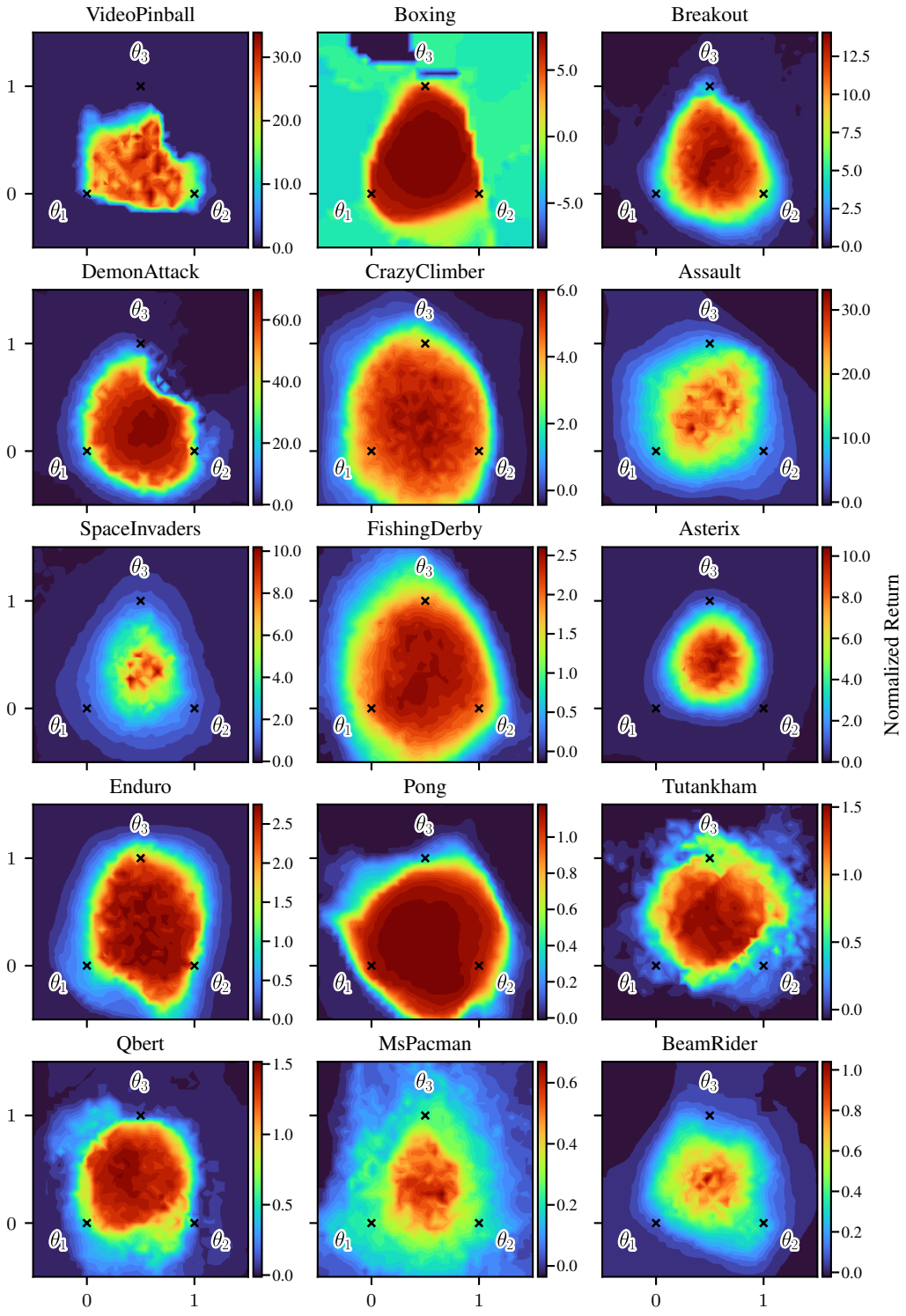

Figure 46: Empirical demonstration of the flat regions induced by simplex subspaces learned with SubQ (*SubPQN*) across 15 Atari environments. We show subspaces defined by three trained agents ($\times$), over which the heatmaps show the normalized return of the corresponding policies. Returns are normalized by the mean return of the three individual agents.

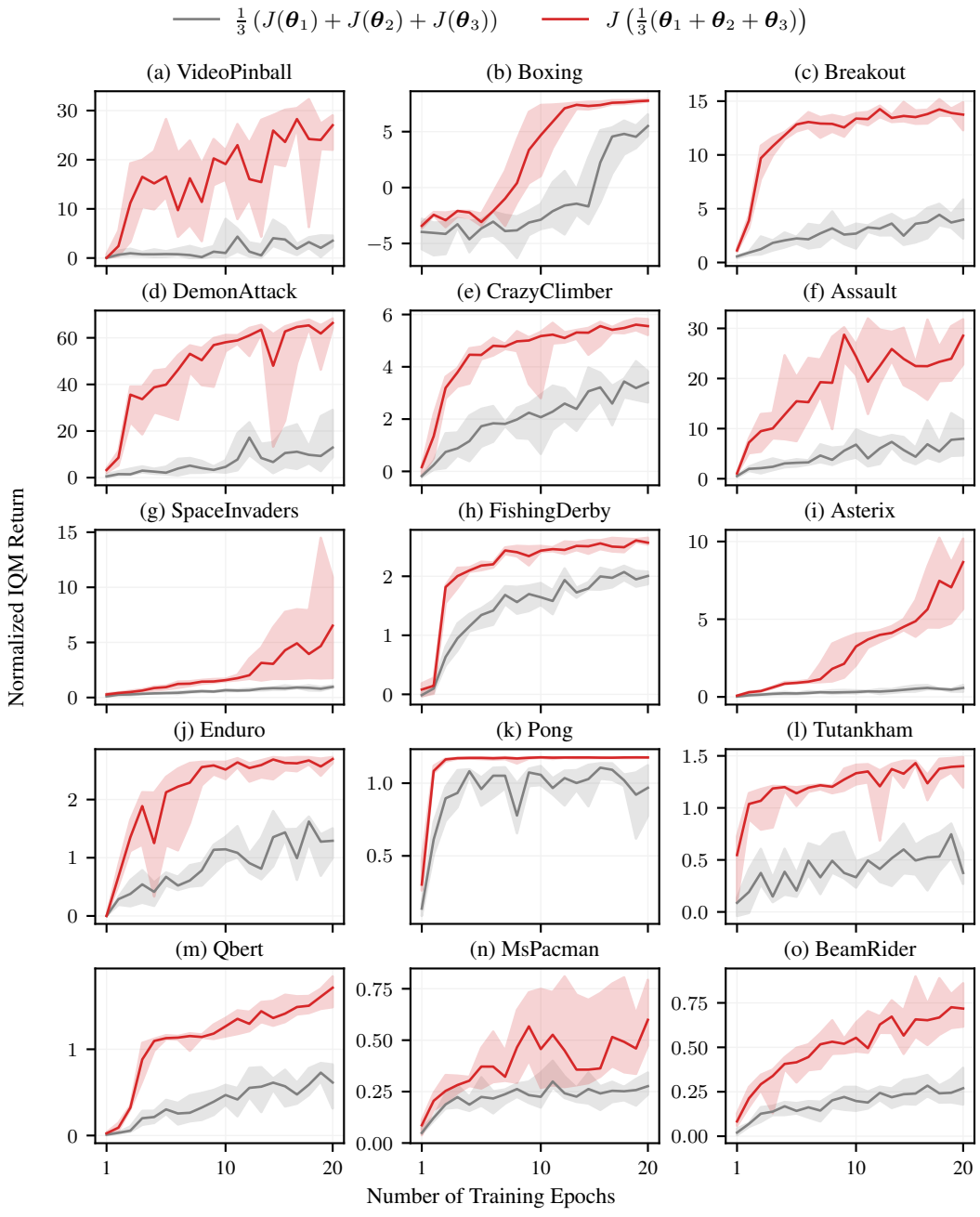

Figure 47: Empirical demonstration of the flat region observation across 15 Atari environments. We show the normalized return of the weight-averaged model (red) and the average return of the three individual agents (grey) over SubQ (SubPQN) training, reported as the IQM with a 95% confidence interval across five seeds. Returns are normalized by the mean return of the three individual agents.

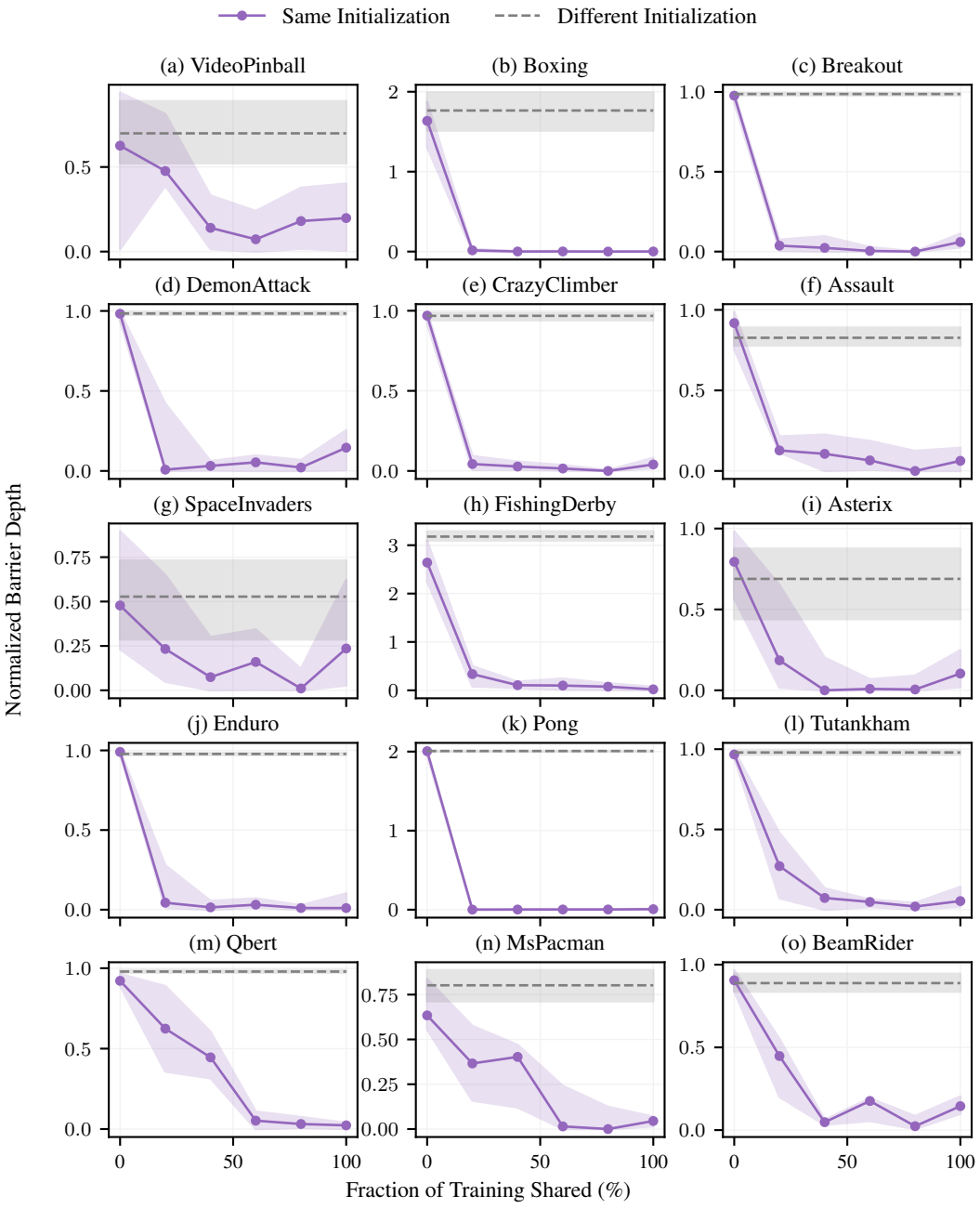

Figure 48: Normalized barrier depth as a function of the fraction of training shared for PQN across 15 Atari environments, reported as the IQM with a 95 % confidence interval across five seeds. After sharing k training steps, two models are trained independently for the remaining $T - k$ steps. We compare runs with identical configuration except SGD noise (random seed) after splitting, with independently initialized models shown as a baseline.