Reimplementing MR.Q: The Role of Representations

Authors. Natalia Espinosa Dice, Ava Krocheski-Meyer, Simar Parmar

Link to GitHub Repo. [https://github.com/natalia-espinosadice/COS-435-RL-MrQ]

1 Introduction

1.1 Motivation

What is the problem? Despite rapid progress in reinforcement learning, state-of-the-art performance is still scattered across highly specialized algorithms: Rainbow dominates Atari while TD3 and SAC prevail in Gym/MuJoCo tasks. Each of these methods is carefully hand-tuned for its target domain, and their hyperparameters rarely transfer beyond it. In contrast, a general model is one that maintains strong performance across very different tasks - from discrete-action pixel-based Atari games to continuous-control vector-based Gym locomotion tasks - without domain-specific hyperparameter tuning. Current generalist agents such as DreamerV3 or TD-MPC2 have made progress toward that goal, but they use enormous networks and expensive online planning, resulting in high computational costs. They therefore remain largely out of reach for practical, everyday use. Thus, the field still lacks a *lightweight*, general model that can achieve SOTA performance across the major benchmark families (Atari, Gym, DMC-visual/control). MR.Q endeavors to fill this gap: its goal is to demonstrate that with a minimalist architecture and a single set of hyperparameters, a *model-free* algorithm can deliver competitive performance across domains without the heavy compute burden of existing generalists.

Why is it interesting and important? From a practical standpoint, a general model removes the bottleneck of domain-by-domain hyperparameter sweeps, dramatically streamlining research workflows. A *lightweight* general model also dramatically reduces the required compute power. Yet, the importance of such a model extends beyond just practical efficiency. A truly general model can help us understand the essential ingredients of control across domains - whether that be representation learning, planning or exploration - without the camouflage of domain-specific tricks, giving us cleaner signals about what actually matters. Furthermore, because such an algorithm can be "dropped in" with minimal tuning, it lowers the barrier to entry for researchers with modest compute budgets or for users who want an off-the-shelf model. As a result, RL becomes far more accessible to all.

Why is it difficult? Achieving this goal is difficult precisely because generalizing is an enormously complex task. A general model must reconcile widely different observation modalities, such as Atari's use of raw pixels versus Gym's use of state vectors, and action spaces that may be discrete or continuous. In addition, different domains have different reward scales and density: Atari rewards are frequently sparser or more delayed than Gym. The domain-specific tricks that may stabilize Atari can thus have the opposite effect in continuous control, and vice versa. Even the hyperparameter choices can vary, and the settings they do share in common may still differ widely (see Appendix A). Delivering consistent, high performance with a single architecture and hyperparameter set therefore demands finding a delicate balance that works everywhere, not just somewhere.

Our Goal: Given that MR.Q addresses such an interesting, important and difficult problem, *our* goal is to better understand precisely how and to what extent it does so. We focus on the role of representations, which we consider the most crucial component of the algorithm.

1.2 Prior Work

We define three criteria that a truly general, lightweight model must satisfy: (1) SOTA performance (2) transferability across domains with minimal retuning and (3) modest computational cost (parameter count and wall-clock training time). We illustrate that no existing line of work meets all three at once. MR.Q is designed to fill that gap.

Generic Policy-Gradients: Algorithms such as PPO and TRPO make minimal assumptions about the domain and therefore can be transferred across tasks relatively easily.[12][11] For example, after minor network adjustments, the core clipped-objective update of PPO still holds for either pixels or vectors and discrete or continuous action spaces.[8] However, their on-policy nature wastes trajectories and leaves them far less sample-efficient than off-policy Q-learning methods. In practice, they trail behind SOTA performance, thus failing to fully meet criteria (1) and (2).

Domain-specific, model-free specialists: Q-learning based methods such as Rainbow on Atari and TD3 on Gym/MuJoCo achieve SOTA performance in their home domains with high sample efficiency.[4][7] To do so, each method is highly tuned to its home domain. For example, TD3 assumes a continuous action space and injects Gaussian exploration noise that is calibrated for MuJoCo's torque magnitudes and ranges. Applying it to Atari, which uses discrete actions and pixel inputs, would require dramatic overhauls to the central components of the algorithm - including Gaussian exploration noise, target smoothing and the actor head - effectively inventing a new algorithm (TD3-Discrete).[8] Precisely because these algorithms overfit to their home domains, they are more difficult to transfer across tasks. They thus fail to meet criteria (1) and (2) across domains.

Model-based Generalists: DreamerV3 and TD-MPC-2 narrow the gap by coupling powerful latent-dynamics models with online planning, thus enabling a single hyperparameter set to span Atari, Gym and DMC suites.[5][6] That generality, however, is bought with heavy compute: tens of millions of parameters, expensive online rollouts, and training rates of less than 20 frames per second.[3] This heavy computational expense limits their everyday use, thus failing to meet criteria (3).

Representation-only Models: Recent work such as TD7 shows that learning a rich state-action embedding - without explicit planning - can boost performance while keeping networks small.[2] TD7 was evaluated specifically on continuous control tasks, thus not fulfilling criteria (2), but we view this line of work as a crucial precursor to MR.Q.

How MR.Q Fills This Gap: Building on TD7 and in contrast to existing generalists, MR.Q decouples representation learning from planning entirely.[3] It learns representations via approximately linear state action embeddings, which serve to unify the input space, and then integrates that learned embedding into a TD3-style backbone. In contrast to TD7, MR.Q includes additional losses over reward and termination and makes a series of design choices with the goal of generality in mind, namely multi-step returns and categorical losses. Empirically, MR.Q dramatically reduces the computational cost of training, requiring only about 4 million network parameters and training at 49 frames per second, in comparison to DreamerV3's 18 and TD-MPC2's 14 frames per second training rates. It further achieves the best average performance across the 4 benchmark suites when compared to both SOTA domain-specific models and these existing generalists.[3] Thus, MR.Q meets our defined criteria: (1) SOTA performance (2) generality across domains and (3) a lightweight footprint. For more algorithmic details, see Appendix B.

Relation to Our Work: We identify the use of representations only, as initially introduced in TD7, as what sets MR.Q apart from previous generalist models. Therefore, this paper interrogates that design choice directly.

1.3 Overview of Our Approach + Limitations

We center our work around MR.Q's core assertion that the true strength of model-based learning stems primarily from the latent representation rather than explicit planning. Although MR.Q is compared against planning-based methods like DreamerV3 and TD-MPC2, the original paper does not directly contrast a representation-only agent with one that also incorporates planning atop the same learned embeddings. To fill that gap, we design two variants sharing the same MR.Q encoder: Representation-Only, which performs a standard model-free Q-update without any online rollouts, and Representation + Planning, which uses a one-step look-ahead in latent space during policy updates. If Representation + Planning offers only marginal improvements, that would bolster MR.Q's claim that representations alone drives most of the performance gains; if it substantially boosts results, it suggests that even with a strong latent embedding, an explicit planning

step still provides meaningful benefits. Finally, should planning worsen performance, it may indicate that model inaccuracies overshadow any potential gains.

Given the centrality of representations to MR.Q, we also investigate how much capacity these embeddings truly need. The original paper sets the latent state dimension (`zs_dim`), latent state-action dimension (`zsa_dim`) and latent action dimension (`za_dim`) to [512, 512, 256]. Keeping the original Representation-Only approach fixed, we sweep through three progressively smaller embedding dimensions: [256, 256, 128], [64, 64, 32] and [16, 16, 8]. We hypothesize that moderate reductions will retain most of MR.Q's benefits, but aggressively small embeddings will harm performance, especially for more complex tasks.

Ultimately, our study zeroes in on two questions that remain unanswered in the original paper: (i) Does an explicit planner still matter once you have a strong MR. Q-style representation? and (ii) How small can that representation become before performance deteriorates? We find that a single-step planning update often fails to help and even hurts performance - particularly in sparse-reward, pixel-based tasks - while scaling down the representation tends to be more forgiving in discrete or lower-dimensional settings than in complex continuous-control domains.

Our study is primarily limited by time and compute power: we select just five environments across the benchmarks suites and evaluate on three seeds. We implement only a simple one-step planning mechanism but acknowledge that there may be ways to improve it, namely by retuning certain hyperparameters or varying the look-ahead horizon. However, despite these limitations, we nevertheless offer a valuable initial exploration into what we believe is the most fundamental component of MR.Q: its learned representations.

2 Methods

2.1 Reimplementation Overview

Our reimplementation is based on the existing codebase for MR.Q, but we significantly refactor the code from scratch to better suit our needs.[9] Key differences include:

- **models.py:** We retain the same network architectures as in the original implementation to match baseline performance. However, we refactor the `Encoder` to improve modularity and include a helper function for one-step planning.
- **mrq_agent.py:** We rewrite the code to more explicitly follow a TD3 backbone. Our central `train` function thus follows the pseudocode from the paper explicitly (see Appendix B). We rewrite the losses from scratch in a separate **losses.py** file to make them explicit and easily accessible. We also add a branch for planning logic.
- **main.py:** We introduce a minimal Typer interface enabling straightforward experimentation via flags for `use_planning` and embedding dimensions.[10] We replace the original Logger with Weights and Biases logging, through which we track training and evaluation rewards, losses and wall-clock time.[13] We omit checkpointing, loading and saving logic, which was unnecessary for our purposes.

We use the original reward two-hot encoding, replay buffer and environment preprocessing logic. Other than our ablations, we use the hyperparameters from the original paper, located in **hyperparameters.py.** Details are thoroughly documented in our repository.

2.2 Experiments

Environment Selection: As illustrated in Table 1, we select five environments to provide a representative yet manageable test suite spanning different action types (discrete vs. continuous), input modalities (pixel vs. vector) and task complexities. In the Atari domain, *Alien* is a classic maze-shooting game, while *Frostbite* demands precise hops across shifting ice blocks to build an igloo, reflecting sparser rewards and higher difficulty. Among the Gym tasks, *Ant* and *Humanoid* both involve high degree of freedom continuous control tasks, with Humanoid being notably more complex. Finally, *DMC-Visual Ball in Cup Catch* presents a pixel-based continuous control manipulation challenge, in contrast to the vector-based locomotion of the Gym tasks. We use three random seeds (0, 1, 2) and align our training durations with those recommended in the original MR.Q repository.[9]

Environment	Total Timesteps	Raw Observation Space	Raw Action Space
Atari Alien-v5	2 M	Pixel (210, 160, 3)	Discrete (18)
Atari Frostbite-v5	2 M	Pixel (210, 160, 3)	Discrete (18)
DMC-Visual ball_in_cup-catch	500 K	Pixel (84, 84, 3)	Continuous $[-1, 1]^2$
Gym Ant-v4	1 M	State Vector (27,)	Continuous $[-1, 1]^8$
Gym Humanoid-v4	1 M	State Vector (376,)	Continuous $[-0.4, 0.4]^{17}$

Note that MR.Q standardizes pixel inputs to the shape (84, 84, 3).

Table 1: Five benchmark environments used in this study[1]

Representation Only vs. Representation + Planning Experiment: Our first experiment compares two variants of training the policy: one that directly uses the learned Q-value for the current state and action ("Representation Only") and another that performs a one-step look-ahead in latent space ("Representation + Planning"). Both approaches share the same underlying policy and encoder networks but differ in how they compute the policy update. Put concretely, we test:

$$\tilde{Q}_i \;=\; \begin{cases} Q_{\theta,i}(\mathbf{z}_{sa_\pi}), & \texttt{use\_planning} = \text{False} \\ \hat{r} + \gamma_{\text{plan}}(1-\hat{d})\, Q_{\theta,i}(\mathbf{z}'_s, a'), & \texttt{use\_planning} = \text{True} \end{cases} \tag{1}$$

where $\gamma_{\text{plan}} = 0.99$, z'_s is the predicted next latent state, $\hat{r}$ is the predicted reward (scaled and decoded from two-hot representation) and $\hat{d}$ is the predicted termination signal. The action a' is computed by the current policy evaluated at $z'_{s'}$ but is held fixed (no gradient flows through a'). The policy then minimizes the loss:

$$\mathcal{L}_{\text{policy}}(a_\pi) \;=\; -0.5 \sum_{i=\{1,2\}} \tilde{Q}_i(\mathbf{z}_{sa_\pi}) \;+\; \lambda_{\text{pre-activ}}\, \mathbf{z}^2_\pi, \qquad \text{where } a_\pi = \text{activ}(z_\pi) \tag{2}$$

A simple `use_planning` flag as a Typer argument and a corresponding branch in the `train_policy` function within the agent file allows us to easily toggle between these two strategies without altering the rest of the MR.Q pipeline.

Embedding Dimension Experiment: For our second experiment, we compare the original embedding dimensions (`zs_dim`, `zsa_dim`, `za_dim`) of [512, 512, 256] to three progressively smaller dimensions: [256, 256, 128], [64, 64, 32] and [16, 16, 8]. These new embedding dimensions are specified via Typer flags and fed directly into the `Encoder`. To isolate the impact of the embedding dimensions, the policy and value networks sizes remain fixed.

Evaluation Metrics and Logging: Using Weights and Biases, we log training rewards, episode lengths and losses (encoder, policy and value).[13] We also log the wall-clock training time per *run* to see whether our ablations have a significant impact on training time. We realize, however, that this metric only proves useful when rewards are comparable; otherwise, differences in training times may be due to poorer performance (resulting in shorter episode lengths) rather than increased computational efficiency. Finally, we log evaluation rewards, evaluating over 10 episodes every 100,000 steps for Atari and every 5000 steps for Gym/DMC-Visual, matching the original paper.[9]

Limitations: Our study is restricted to five environments and three seeds due to computational and time constraints. Future work could investigate a broader set of tasks, though we endeavor to make our set as representative as possible. With respect to the first experiment, future work could examine multi-step latent planning and consider whether retuning certain hyperparameters might help improve performance. With respect to the second experiment, future work could explore different state to action dimension ratios.

3 Results

3.1 Reimplementation vs. Original MR.Q Performance

Before presenting our main experiments, we first verify that our reimplementation achieves comparable performance to the original MR.Q algorithm. We plot the original repository's

reported results against our reimplementation and provide the figures in Appendix C. Although training trajectories diverge slightly, the final performance converges closely across all five environments. We attribute small discrepancies to routine run-to-run variance and our subtle implementation changes, but overall, these results confirm a faithful reimplementation of MR.Q.

3.2 Representation Only vs. Representation + Planning Results

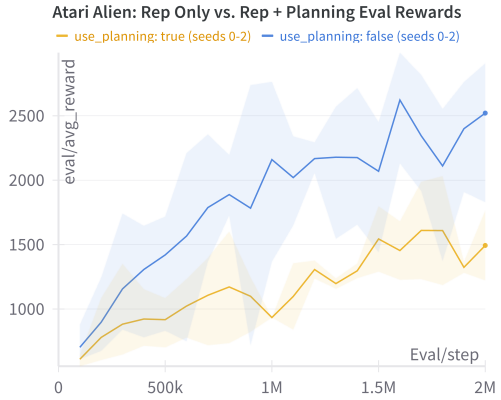

Figure 1: Atari Alien: Rep Only vs. Rep + Planning Eval Rewards

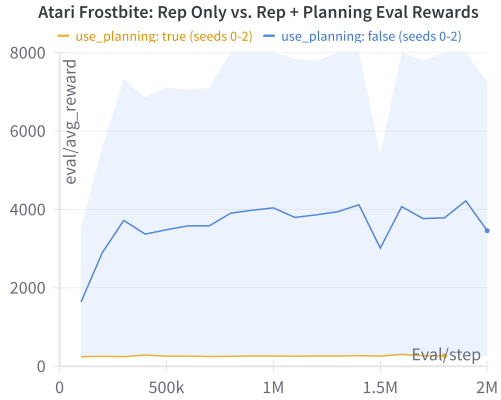

Figure 2: Atari Frostbite: Rep Only vs. Rep + Planning Eval Rewards

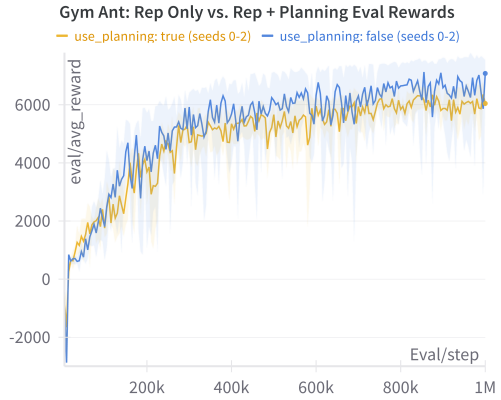

Figure 3: Gym Ant: Rep Only vs. Rep + Planning Eval Rewards

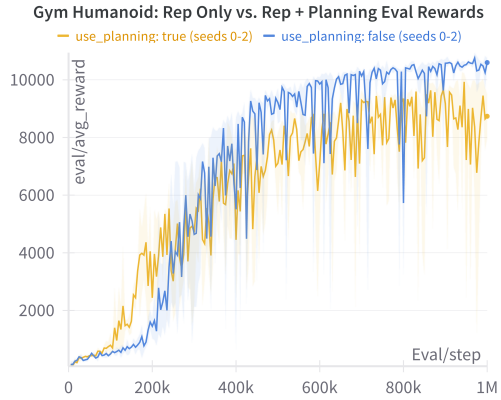

Figure 4: Gym Humanoid: Rep Only vs. Rep + Planning Eval Rewards

Figures 1 - 5 shows the results of our first experiment across our five environments. Overall, we see no universal advantage from adding a single planning step.

Atari Alien and Frostbite: Representation Only attains higher evaluation rewards throughout training, with the discrepancy especially pronounced for Frostbite, where the agent hardly learns at all. One likely explanation is that single-step model prediction is error-prone for high-dimensional pixel-based states, and these errors can mislead the planning updates. Frostbite poses a particularly sparse-reward challenge, making accurate next-state and reward predictions especially critical.

Gym Ant and Humanoid: Both methods track each other more closely, although with greater variance in the more complex Humanoid task. The smoother dynamics and denser reward structure likely make single-step errors less common and harmful than in Atari.

DMC-Visual Ball in Cup Catch: While the raw input space is pixel-based, the environment is simpler than Atari in terms of task and reward structure (grasp and lift the ball), resulting

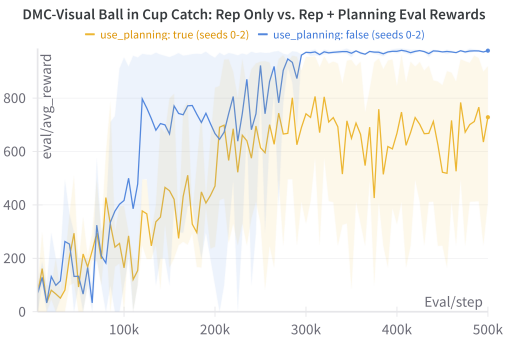

Figure 5: DMC-Visual Ball in Cup Catch: Rep Only vs. Rep + Planning Eval Rewards

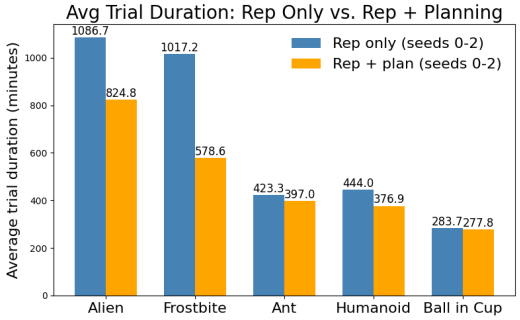

Figure 6: Rep Only vs. Rep + Planning Average Run Duration (minutes) Across 5 Environments

in smaller performance gaps. The two methods converge to similar ranges of final reward, though the Representation Only setting is ultimately higher.

Wall-clock Training Time: We were initially curious to see whether this experiment might produce differences in wall-clock training time, which are illustrated in Figure 6. However, we refrain from drawing any conclusions based on the existing results, as it is very likely that the poorer performance of the planning runs led to faster training time, as it reduced the length of episodes, rather than actually improving computational efficiency.

Trends Across Environments: Overall, adding a single planning step seems to harm, rather than help, performance. This is particularly true for the Atari environments and especially Frostbite, suggesting that discrete-control settings with sparser rewards are particularly hard to predict in latent space. Even small inaccuracies can compound and steer learning away from good solutions. Meanwhile, the continuous-control tasks with denser rewards and smoother transitions (Gym, DMC-Visual) do not exhibit as strong a discrepancy, suggesting that single-step planning is more sensitive to domain complexity and reward sparsity than was initially expected. Additional graphs - including episode lengths and losses over training - can be found in Appendix D.

3.3 Embedding Dimension Ablations Results

In Figures 7-11, we show the results of our embedding dimension ablation study. We note that due to time constraints, we only ran the [16, 16, 8] setting on seeds 0-1 for the DMC-Visual environment and seed 2 for Atari Frostbite (indicated in figures); those results may not be statistically significant. We omit wall clock times, as we doubled up some runs on a single GPU and therefore cannot attribute differences to embedding dimension changes alone. Loss figures can be found in Appendix E.

Gym Ant and Humanoid: Figure 7 demonstrates that embedding dimensions of [512, 512, 256] perform best for Ant, while [256, 256, 128] and [64, 64, 32] perform comparably worse and [16, 16, 8] hardly learns at all. Figure 8 demonstrates a similar trend for Humanoid, though rewards decrease with embedding size in a more proportional manner: the gap between 512 and 256, for example, is smaller than between 512 and 64.

Atari Alien: In Figure 9, [512, 512, 256], [256, 256, 128] and [64, 64, 32] all perform more comparably for Alien than in the Gym tasks. Interestingly, [64, 64, 32] actually seems to outperform [256, 256, 128] and roughly matches [512, 512, 256]. In Figure 10, we see a similar trend in Frostbite, where the three largest embedding sizes perform rather comparably, though [256, 256, 128] does slightly outdo [64, 64, 32]. In both cases, an embedding size of [16, 16, 8] performs noticeably worse, especially for Frostbite. Both environments show more sizable variance across seeds, particularly in the largest embedding for Frostbite, making it more difficult to declare a definitive "winner."

DMC-Visual Ball in Cup Catch: In Figure 11, all four embedding sizes manage to maximize reward, though interestingly, it takes slightly longer for the [512, 512, 256] size to learn.

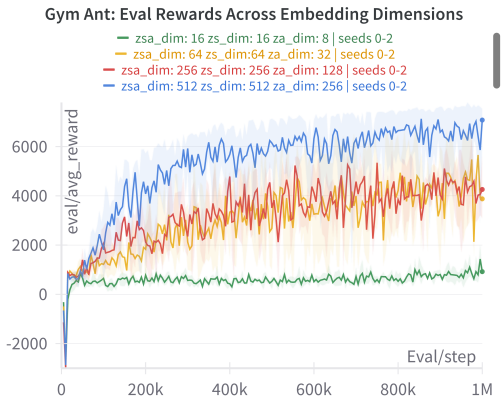

Figure 7: Gym Ant: Eval Rewards Across Embedding Dimension Ablations

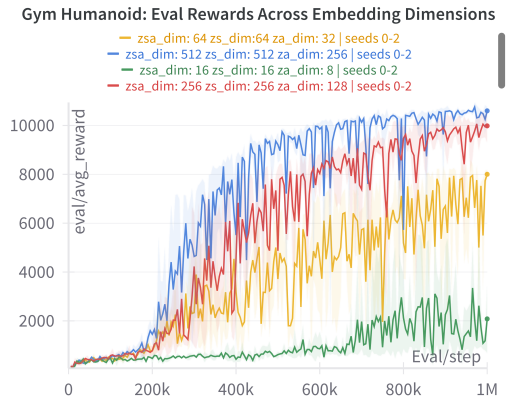

Figure 8: Gym Humanoid: Eval Rewards Across Embedding Dimension Ablations

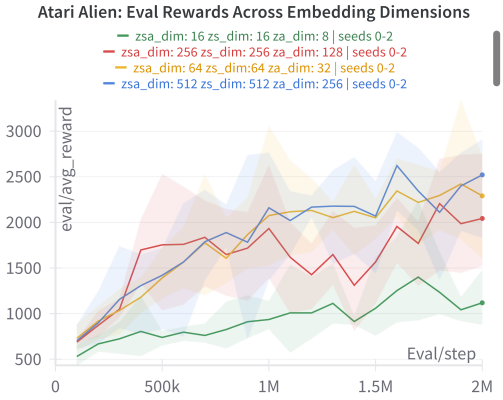

Figure 9: Atari Alien: Eval Rewards Across Embedding Dimension Ablations

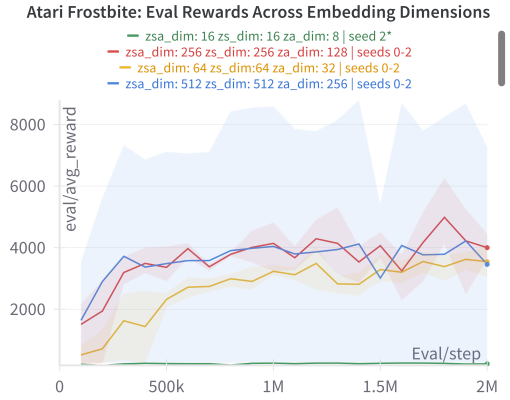

Figure 10: Atari Frostbite: Eval Rewards Across Embedding Dimension Ablations

Trends Across Environments: We consider these results in the context of the environments' observation and action spaces, as initially laid out in Table 1. The Gym environments have relatively large, continuous control observation and action spaces. Larger embeddings may allow the latent state to more flexibly encode the salient features and manage the higher dimensional control problem - hence the top performance with [512,512,256]. When the embedding size is reduced, we see a decrease in performance, especially for the more complex Humanoid task. For the Atari environments, even though the standardized pixel input shape is (84x84x3), the action space is discrete and smaller, and the CNN used for the encoder likely already helps compress the input. Therefore, it seems that beyond a certain capacity, the embeddings can capture most of the important features, meaning that the larger embeddings are closer in performance. For both the Gym and Atari environments however, an embedding dimension of [16, 16, 8] is simply too small to capture enough information. Finally, DMC-Visual's results further illustrate that with a pixel-based input but simpler action space, smaller embeddings can learn the key dynamics even *more quickly* than the largest embedding.

4 Discussion and Limitations

We investigated two complementary questions regarding MR.Q's learned representations: (1) whether incorporating a single-step latent-space planning update improves learning and (2) how different embedding dimensionalities affect performance. Our results across five benchmark environments suggest that single-step planning does not provide a uniform advantage over a purely representation-based approach, *especially* in sparse-reward, pixel-

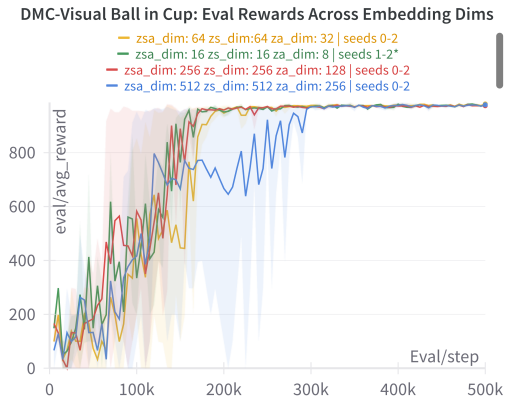

Figure 11: DMC-Visual Ball in Cup Catch: Eval Rewards Across Embedding Dimension Ablations

based settings like Atari Frostbite. Meanwhile, embedding size has a clear effect in high-dimensional continuous control tasks such as Gym Ant and Humanoid, where larger embeddings consistently outperform smaller ones. On simpler or discrete domains such as Atari, differences between moderate and large embeddings are less pronounced, and on DMC-Visual Ball in Cup Catch, smaller embeddings even learn faster initially.

A critical assumption in the one-step planning approach is that the learned latent model can predict future states and rewards accurately enough for the additional Q-value update to help, rather than hinder. Our experiments suggest that when prediction errors are high (especially in pixel-based, sparse-reward Atari tasks), these inaccuracies can distort the Q-values and degrade performance. Our work was limited to a single planning step, but it is possible that searching over the planning horizon (and perhaps retuning other hyperparameters) might yield more consistent benefits in future work.

With respect to our embedding ablations, we limited our investigation to a constant state to action dimension ratio of (1:2), but future work could explore other ratios. We also limited each environment to three seeds, which illustrated major trends but left smaller effects susceptible to run-to-run variance, particularly for the Atari environments, in which it was more difficult to decisively declare any single embedding size "best."

With respect to MR.Q's core claim that the true strength of model-based learning lies in the learned representations rather than model itself, we conclude that *single step* planning indeed does not offer any significant performance gains, though again more sophisticated planning algorithms might perform better. Our experiments also reveal that MR.Q's ability to learn effective latent representations is strongly dependent on the environment. This outcome underscores a broader limitation: the learned world-model may be significantly noisier for discrete, pixel-based Atari than for continuous, vector-based Gym. Consequently, (simple) planning in such settings can cause large compounding errors in Q-value estimation. Furthermore, we find that it may be possible to reduce the embedding size for certain domains while largely maintaining performance. This comes with the tradeoff of including an additional hyperparameter to specify, but it could help reduce computational costs and make MR.Q even more lightweight.

Ultimately, our work reiterates the broader challenge of developing a truly general model: finding an approach that works *across* domains is extremely challenging. Nevertheless, by examining how our experiments affect performance across different environments, we move closer to understanding how general models might allocate representational and planning resources in the future.

References

[1] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym. https://www.gymlibrary.dev/.

[2] Fujimoto, S., Chang, W.-D., Smith, E. J., Gu, S. S., Precup, D., and Meger, D. (2023). For SALE: State-Action Representation Learning for Deep Reinforcement Learning. Available at https://arxiv.org/abs/2306.02451.

[3] Fujimoto, S., D'Oro, P., Zhang, A., Tian, Y., and Rabbat, M. (2025). Towards General-Purpose Model-Free Reinforcement Learning. International Conference on Learning Representations (ICLR), 2025.

[4] Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing Function Approximation Error in Actor-Critic Methods. Available at https://arxiv.org/abs/1802.09477.

[5] Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. (2024). Mastering Diverse Domains through World Models. Available at https://arxiv.org/abs/2301.04104.

[6] Hansen, N., Su, H., and Wang, X. (2024). TD-MPC2: Scalable, Robust World Models for Continuous Control. Available at https://arxiv.org/abs/2310.16828.

[7] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2017). Rainbow: Combining Improvements in Deep Reinforcement Learning. Available at https://arxiv.org/abs/1710.02298.

[8] Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., and Araújo, J. G. (2023). CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms. https://github.com/vwxyzjn/cleanrl.

[9] Meta AI Research and Fujimoto, S. (2024). MR.Q: Towards General-Purpose Model-Free Reinforcement Learning (Source code). https://github.com/facebookresearch/MRQ.

[10] Ramírez, S. (2022). Typer. https://typer.tiangolo.com.

[11] Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2017a). Trust Region Policy Optimization. Available at https://arxiv.org/abs/1502.05477.

[12] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017b). Proximal Policy Optimization Algorithms. Available at https://arxiv.org/abs/1707.06347.

[13] Weights & Biases (2025). Weights and Biases. https://wandb.ai.

Appendix A Further Illustrating the Challenge of Generality

Figure 12, taken from the original paper, helps illustrate the difficulty of developing a general model. Rainbow and TD3 - two SOTA models on their home domain - use different hyperparameter settings, and even those they share in common differ widely.[3] This helps demonstrate both why domain-specific SOTA models fail to transfer easily across domains and why the problem of generality is so challenging: there is no immediate one-set fits all solution to hyperparameter tuning.

Appendix B Detailed Overview of Original MR.Q Algorithm

Figure 13, taken from the original paper, provides psuedocode illustrating the key aspects of the MR.Q algorithm.[3] MR.Q learns a latent-space representation alongside a standard Q-value function, effectively combining model-based and model-free RL in a single framework. Given a transition (s, a, r, d, s') from the replay buffer (utilizing Loss-Adjusted Prioritization), MR.Q first encodes the state s via a state encoder $z_s = f_w(s)$. It then merges this latent state z_s with the action a to form a state-action embedding. A linear MDP predictor maps this embedding to predictions of the next latent state, reward $\hat{r}$ and done signal $\hat{d}$. Meanwhile, a value network Q_θ uses z_{sa} to estimate the Q-value, and a policy network π_ϕ outputs actions based on z_s. MR.Q updates each component periodically: target networks are synced, rewards are rescaled and the encoder is trained. The value and policy networks are updated each step. Ultimately, by learning both a latent-space forward model and a standard Q-function, MR.Q can leverage representation learning benefits while still using an off-policy approach to optimize the policy.