COMP541 Tech Report
Reproducing Efficient Planning in a Compact Latent Action Space

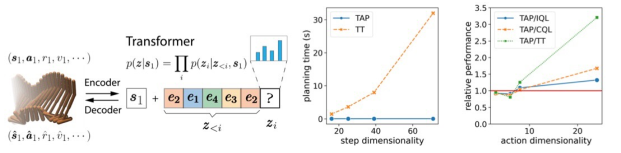
Enes M. Erciyes
Koc University
Istanbul, Turkey
merciyes18@ku.edu.tr

Abstract

The use of planning-based reinforcement learning is promising to solve tasks that require long-term reasoning. It has been successful in completing tasks that involve actions in discrete and low-dimensional continuous spaces. However, planning typically requires a lot of computational power for decision-making, and it's difficult to apply these methods to tasks with high-dimensional action spaces. To address this issue and improve the efficiency of planning for high-dimensional continuous control tasks, Trajectory Autoencoding Planner (TAP) [40] learns low-dimensional latent action codes by using a state-condidional VQ-VAE. In this report, we reimplement TAP using Knet.jl [39] for reproducibility and report results on a locomotion task in D4RL benchmark [10]. The code has been released at `https://github.com/eneserciyes/LatentPlan.jl`.

1. Introduction

Planning based methods showed success in board games [30, 29] and low dimensional continuous control [14]. Traditionally, planning is performed in the action space of the task, where future trajectories are simulated using a dynamics model of the environment, which can be either learned or predefined. Although this is the first thing to come to mind, planning in original action space is inefficient and inflexible. Firstly, the raw action space is usually huge and it is difficult to find the optimal plan in this high-dimensional space. Even if the optimizer is strong enough to find such a plan, it is difficult to ensure the dynamics model is always accurate. In such cases, optimizer can exploit the limitations of the model to make overly optimistic plans that fail in practice. Secondly, this type of planning is tied to the temporal structure in the original MDP. If actions always move forward in time with constant step size, the planning must also be like that. However, humans can use abstrac-


$(\boldsymbol{s}_1, \boldsymbol{a}_1, r_1, v_1, \cdots)$
Transformer
$p(z|\boldsymbol{s}_1) = \prod_i p(z_i|\boldsymbol{z}_{<i}, \boldsymbol{s}_1)$
Encoder
Decoder
s_1 + e_2 e_1 e_4 e_3 e_2 ?
$z_{<i}$
z_i
$(\hat{\boldsymbol{s}}_1, \hat{\boldsymbol{a}}_1, \hat{r}_1, \hat{v}_1, \cdots)$
TAP
TT
planning time (s)
step dimensionality
TAP/IQL
TAP/CQL
TAP/TT
relative performance
action dimensionality

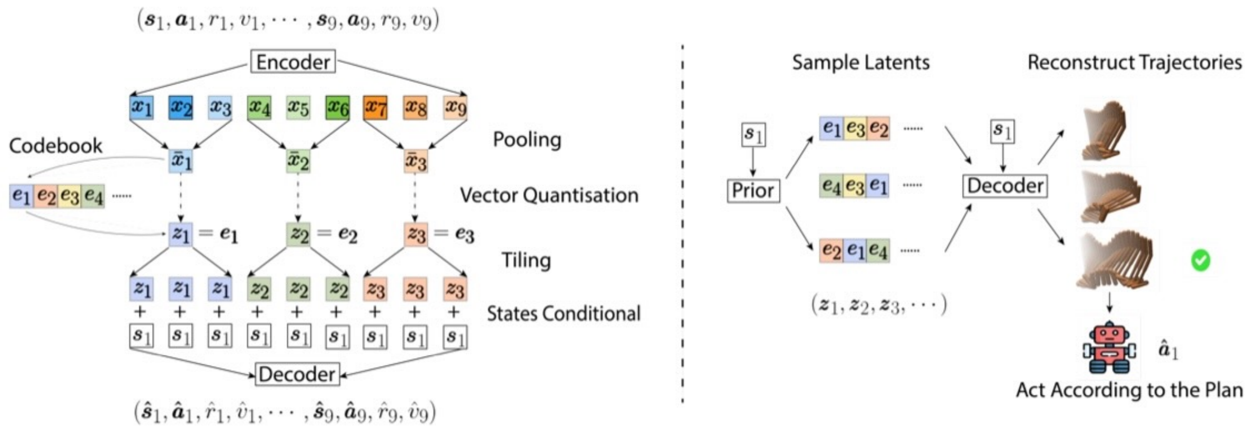
(a) TAP Modelling (b) Decision Latency (c) Relative Performance

Figure 1: Trajectory Autoencoding Planner. (a) is an overview of the method. Blocks denoted by $\mathbf{z}_i$ are latent actions. (b) shows how decision time grows with dimensionality for TT [14] but remains constant for TAP. (c) TAP shows better performance with growing dimensionality compared to model-free methods like IQL [18] and CQL [19], and TT.

tions to plan over multiple time scales, or plan backward from the desired result. Planning with raw actions does not have any of this flexibility. Furthermore, due to this inflexibility, raw action planning takes time and blocks real-time deployment.

Trajectory Autoencoding Planner (TAP) learns a latent action space and latent-action model from offline data. A latent-action model takes state s_1 and latent actions z as input and predicts a finite-horizon future trajectory $\tau = (a_1, r_1, R_1, s_2, a_2, r_2, R_2, ...)$. Here a refers to action, s refers to state, r refers to immediate reward and R is the reward-to-go. This latent-action space can be much smaller than the raw action space as it does not model states/actions that are not in the training state, *i.e.* out-of-distribution, and considers only a small subset of the original space. Furthermore, the latent action allows for flexible planning methods, for example, predicting multiple steps with a single latent action.

The unsupervised learning methods are used to train the model and the action space. The encoder maps the trajectories to a sequence of *discrete* latent codes based on the current state, and the decoder reconstructs the original data using a Vector Quantised Variational AutoEncoder (VQ-VAE) that is conditioned on the state. Then, as shown in Figure



$(s_1, a_1, r_1, v_1, \cdots, s_9, a_9, r_9, v_9)$
Encoder
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9
Codebook
Pooling
$\bar{x}_1$ $\bar{x}_2$ $\bar{x}_3$
e_1 e_2 e_3 e_4
Vector Quantisation
$z_1 = e_1$ $z_2 = e_2$ $z_3 = e_3$
Tiling
States Conditional
Decoder
$(\hat{s}_1, \hat{a}_1, \hat{r}_1, \hat{v}_1, \cdots, \hat{s}_9, \hat{a}_9, \hat{r}_9, \hat{v}_9)$
Sample Latents
Reconstruct Trajectories
s_1
Prior
e_1 e_3 e_2
e_4 e_3 e_1
e_2 e_1 e_4
Decoder
$(z_1, z_2, z_3, \cdots)$
$\hat{a}_1$
Act According to the Plan

Figure 2: Left-hand side of the figure shows the training process of TAP. The right-hand shows how planning is made with the decoder and sampling.

1-a, a Transformer module is used to model the distribution of these latent codes, *i.e.* which code is more likely to come given the past codes and the current state. During inference, TAP samples latent codes from this learned distribution and uses the decoder to reconstruct the trajectory. Then, it selects the first action in the trajectory with the highest score. This planning is done with beam search. In our experiments, we use a single latent variable to model multiple steps of transitions (L=3). This creates even a more compact planning space. This space both reduces the time to plan and makes high-dimensional planning easier. Additionally, it helps a significant problem in model-based RL: compounding errors. Since the trajectory is constructed with all the sampled latents, errors in reconstruction do not propagate to later steps.

Our contributions include:

- We reimplement TAP [15] using Knet.jl [39] by following the original implementation from the original authors while sticking to the original paper for correctness of the implementation.
- We conduct experiments to evaluate quantitative results and reproducibility of the method on a D4RL benchmark locomotion task, *hopper-medium-replay-v2*.

2. Related Work

Model-based RL [33, 31, 9, 7, 20, 13, 37, 29, 2, 12] is a sub-field of RL research that tries to predict actions that maximize discounted cumulative reward by finding the transition model of the environment. TAP falls into this line of work as it tries to predict future to infer actions. Conventionally, the model does its predictions in the original state space of the MDP by taking current state and a candidate action and returning a distribution over the next state and reward. TAP is different from both of these as (1) it takes latent actions and the current state and (2) it returns the whole finite-horizon trajectory instead of just the next state. These latent variables act as a learned action space which allows faster and easier planning.

Recently, latent state-space models [12] and [26] that does its future prediction in a learned latent space emerged. However, these typically still use original action space of the problem and unrolling of the model is tied to MDP's temporal structure. [36] and [38] proposes to learn representations for actions during inference for black box optimization and path planning. Although this work seems relevant on a high level, their method assumes that the dynamics of the environment is known. TAP extends this idea to the general RL problem where environment dynamics are unknown.

TAP is one of the more recent line of work that models policy learning a sequence modelling problem. [4, 14, 40]. These works use GPT-2 style Transformers to model the whole trajectory of states, actions, rewards and values. Then, they use this prediction ability to extract a policy. Decision Transformer (DT) applies Upside Down RL [28] to learn actions conditioned on rewards. Trajectory Transformer (TT) applies planning (beam search) to find the best trajectory that gives highest return. The strong modelling power of Transformer architecture allows TT to generate long-horizon plans with high accuracy. TAP brings a layer of efficiency to TT and other discrete space planning algorithms with its latent action model to plan effectively in higher-dimension action spaces.

The idea of learning a representation of actions and doing RL in this latent action space is also explored in the realm of model-free reinforcement learning [6, 1, 41, 25, 27]. TAP wants a latent action space to enable effective planning. However, the motives in model-free cases are varied. One common ground is to provide a policy constraint. For instance, in the context of humanoid control, [25] and [27] have utilized this concept to ensure that the learned policies resemble natural human behavior seen in low-level demonstrations. [41] have used latent actions to prevent out-of-distribution (OOD) actions in an offline reinforcement learning (RL) setting. [6] have proposed a discrete latent action space to allow methods designed for discrete action spaces to be extended to continuous cases. In the teleportation literature, [21] and [16] have incorporated the robot's high-dimensional actions into low-dimensional, human-controllable latent actions.

This paper evaluates TAP in an offline reinforcement learning (RL) setting [8], where online experience cannot be used to improve the policy. One of the major challenges of offline RL is to avoid selecting out-of-distribution (OOD) actions with the learned policy to prevent exploitation of inaccuracies in the value function and model. To address this issue, conservatism has been proposed as a solution [19, 11, 22, 18, 17]. TAP can naturally prevent OOD actions by modeling trajectories generated by the behavior policy and decoding the policy, which is therefore within the distribution.

Related works in hierarchical reinforcement learning (RL) include [5, 23, 24], which all use some form of latent variable to condition policies. However, unlike TAP, these works typically have multiple modules optimizing for different goals, whereas TAP models actions, states, rewards, and values jointly with a unified VQ-VAE. Additionally, TAP's discrete latent codes are conceptually related to the options framework [34, 32, 3] as both provide a mechanism for temporal abstraction.

3. Methodology

3.1. Vector Quantised-Variational AutoEncoder

The Vector Quantised Variational Autoencoder (VQ-VAE) (van den Oord et al., 2017) is a generative model designed based on the Variational Autoencoder (VAE). It consists of three parts: (1) an encoder network mapping inputs to a collection of discrete latent codes called the codebook, (2) a decoder that takes the codebook to reconstructed input, (3) a learned prior distribution over latent variables. We can sample latent codes from the prior and take them through the decoder to get new generations.

The discrete code choice is deliberate. VAEs typically suffer from 'posterior collapse' when the prior variable becomes too powerful and the information in the encoded input latents is ignored. By discretizing this hiddent latent, we can limit the amount of information in them and avoid this issue.

To discretize, we use a technique called Vector Quantization (VQ). The encoder first outputs continuous vectors $\mathbf{x}_i$. However, they are not input directly to the decoder but instead used as queries to a codebook of size KxD with K being the number of latent codes and D being the dimension of each code vector. The query is simply a nearest neighbor search.

$$\mathbf{z}_i = \mathbf{e}_k, \text{where k} = argmin_j ||\mathbf{x}_i - \mathbf{e}_j||_2 \tag{1}$$

are input to the decoder. The autoencoder is trained by minimizing both the reconstruction error and the distances $||\mathbf{x}_i - \mathbf{e}_k||_2$ and $||\mathbf{z}_i - \mathbf{e}_k||_2$. The gradients are calculated through straight-through gradients through this operation during backpropagation. In the original implementation, the prior is taken as a uniform distribution. However, to get better sample quality, a PixelCNN-lik [35] model is separately trained autoregressively as the prior model.

3.2. Latent-Action Model and Planning

Let the following be a trajectory τ of length T sampled from an MDP with a fixed exploration policy:

$$\tau = (\mathbf{s}_1, \mathbf{a}_1, r_1, R_1, ..., \mathbf{s}_T, \mathbf{a}_T, r_T, R_T) \tag{2}$$

TAP models the conditional distribution of the trajectory $p(\tau|\mathbf{s}, z)$ with a sequence of latent variables $z = (\mathbf{z}_1, ..., \mathbf{z}_M)$. We can assume the state and latent variables can be deterministically mapped to a single trajectory. This implies $p(\tau|\mathbf{s}, z) = \mathbb{1}(\tau = h(\mathbf{s}, z))p(z|s)$ which tells that the probability of trajectory τ is zero if $\mathbf{s}, z$ does not map to it, and if it does it is equal to $p(z|s)$. We refer to z as latent actions and $p(z|s)$ as a latent policy. Then, $h(s, z)$ is a latent-action world model. If $p(z|\mathbf{s}) > 0$, then $h(z, \mathbf{s})$ trajectory will be an executable plan and in a **deterministic** MDP, the trajectory can be recovered by following the actions in this plan, starting from state $\mathbf{s}$. Therefore, one can optimize the latent actions to find the optimal plan.

3.3. Learning a Latent-Action Model with VQ-VAEs

To learn a compact discrete latent action space, Trajectory Autoencoding Planner (TAP) is proposed. It models the trajectories with a state-conditioned VQ-VAE. $x_t := (s_t, a_t, r_t, R_t)$ is treated as a single token for the Transformer model.

This sequence of tokens is processed by a causal Transformer. The output of the transformer is T (planning horizon) feature vectors. Then a max pool with a stride of L (latent action temporal size) is applied. There, we get our trajectory features that we apply vector quantization on top of. After quantization, the codes we get are repeated to match

Method	Hopper Medium Replay
Random Action	0.012 ± 0.02
MoREL [17]	93.6
TT [14]	91.5
TAP (Paper) [15]	87.3 ± 2.3
TAP (Implementation)	**93.83 +/- 1.83**
TAP.jl (Ours)	90.89 +/- 3.95

Table 1: Normalized score results from the hopper-medium-replay-v2 task in D4RL benchmark.)

the number of features. The state condition is added to each token and passed to decoder. Decoder adds positional embeddings to each token and reconstructs the input trajectory.

During inference, we use a separately trained prior policy $p(\mathbf{z}|\mathbf{s_1})$ to guide sampling and planning. This autoregessive prior policy is also state-conditional and can be denoted as $p(z_t|z_{<t}, s_1)$.

The training and test time process of TAP is visualized in Figure 2.

3.4. Planning in the Discrete Latent Action Space

In the following section, we will explain the process of utilizing the TAP model that we have acquired, which involves evaluating the trajectory and optimizing latent actions.

The evaluation criterion is a bit complex. Given the initial state and latent action sequence $\mathbf{s_1}, z$ we obtain a sample trajectory $\hat{\tau} = (\hat{x}_1, ..., \hat{x}_T)$ from the decoder h. Here, each x is a token defined earlier. The scoring function for the trajectory is

$$g(s_1, z_1, ..., z_M) = \sum_t \gamma^t \hat{r}_t + \gamma^T \hat{R}_T + \alpha ln(min(p(z_1, ..., z_M|s_1), \beta^M))$$

The first part of the evaluation is the predicted return-to-go by summing the predicted rewards up to the planning horizon and adding the predicted return-to-go at the last step. The rest of the evaluation criterion encourages plans that are in-distribution. If the probability of the latent sequence is smaller than a threshold β^M , then, this trajectory is penalized.

There are multiple sampling methods discussed in the original paper and implemented in the code. In this work, we only implemented the most performant solution: Beam Search in the Latent Space.

Beam Search in the the Latent Space The decoder of our Transformer does not depend on the future tokens as we are using a causal mask. This allows us to decode partial trajectories. Therefore, we can use the beam search planning algorithm. Beam search works by only expanding the most promising branches of the search tree. It always keeps B (branch count) best branches and samples E (expansion factor) codes. This approach is both computationally more efficient and more performant.

4. Experimental Setup and Analysis

The experiments are performed on one of the Gym locomotion tasks in D4RL benchmark [10] : hopper-medium-replay-v2. The hopper is a one-legged figure with four body parts. The aim is to control it by applying torques on the hinges to make it move forward in hops. The environment emphasizes independent state and control variables.

A ”medium” dataset in D4RL benchmark is created by training a policy with Soft Actor-Critic, stopping early, and collecting 1M samples. ”Medium-replay” consists of samples recorded in the replay buffer until the policy reaches medium performance. In hopper case, there are 200920 samples that consists of an observation, action and a state. The dataset is first preprocessed to properly batch according to episode termination points and the return-to-go values are calculated by summing the rewards in the next H steps after the relevant step. All of the samples are used for training. The model architecture and training details can be respectively found in Section F and Section G. The overall model implementation, training setup, and hyperparameters are chosen the same as the original implementation.

As baselines, a random action model, MOReL [17] and Trajectory Transformer [14] are used. MOReL and Trajectory Transformer results are taken from the original TAP paper [40].

The original paper reports a lot of results on about 15 different environments. Even though the training is fast ($\sim$ 10 hours including prior training and evaluation), due to the project timeline and resource constraints, only the results on a single environment is reproduced. With the same reasons, ablations studies are not reproduced.

4.1. Results

TAP results are on par with other model-based solutions as it can be seen in Table 1. The score reported in the table is the normalized environment score from D4RL benchmark. This score is calculated as following: $100 * \frac{score - randomscore}{expertscore - randomscore}$. This is why random baseline is shown to almost get 0 score in the experiments.

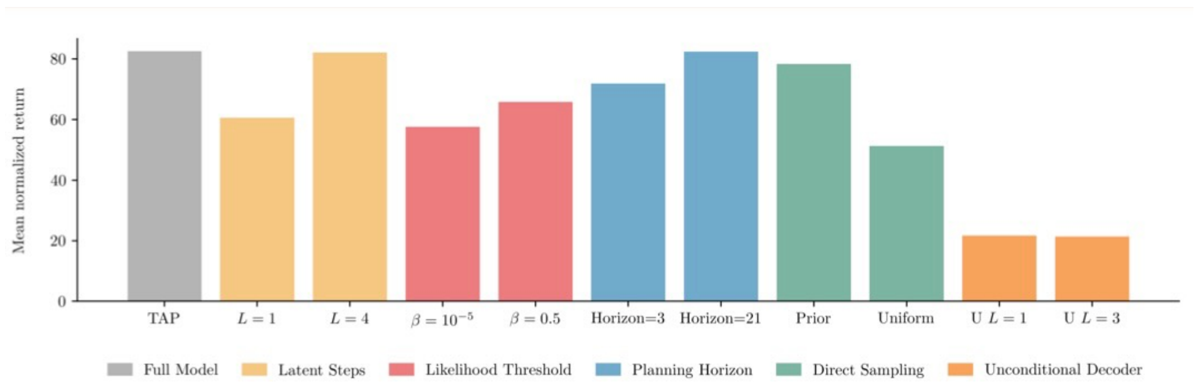


Figure 3: The analysis results from the original paper, scores are mean normalized scores from all Gym locomotion tasks.

The results I obtained, as we can see in Table 1, are slightly better than the original reported results but it is worse than the results obtained when original implementation is run. The model and the training process is replicated completely thanks to a clear written code and no points left unanswered in the paper. The difference in the results can be attributed to the stochastic environment and the different seeds used in model training. When we run the original implementation, we used 5 different seeds to train 5 models and report the average score of these models. For our Knet.jl implementation, a single model is trained and evaluated.

4.2. Ablation Study

5 main ideas were combined in the method that gave the reported performance, which were multiple latent steps, likelihood threshold for in-distribution actions, long planning horizon, prior sampling and state-conditional decoder.

Latent Steps TAP allows temporal abstraction by letting one latent action represent L steps in the original MDP. This increases the efficiency of planning as longer horizons are reachable with the same computational budget. However, it also increases the performance of the model. As shown in yellow bars in Figure 3, reducing L to 1 from 3 results in a significant drop in performance. The authors of the original paper hypothesize that this is due to the regularization effect of L on the VQ-VAE as they see the model overfits and gives higher return estimates and higher prediction errors with $L = 1$.

OOD Penalty The threshold in OOD penalty is varied to understand its effect on performance. When $\beta = 1e - 5$, OOD penalty is practically useless and this causes a 30.2% drop. When $\beta = 0.5$, the OOD penalty is always active and the model tends to ignore the return estimates and tries to behaviour clone the policy in data. This results in a 20.2% drop.

Planning Horizon TAP is told to be robust to planning horizon as simply doing 1 step planning results in a minor score drop (-12.9%). However, this is task-dependent. Gym locomotion tasks do not need long-horizon planning as much as more complex tasks. Therefore, we believe planning horizon is very critical for tasks in general.

Beam Search The first green bar shown as 'Prior' is actually a version where beam search is replaced with random shooting. Beam search performs slightly better than this version when you compare with the complete model performance.

Uniform Sampling The next green bar is comparing uniform sampling vs. using a learned prior. Using uniform sampling greatly reduces the performance (-37.9%).

State-Conditional Decoder By conditioning the decoder on the first state, TAP allows to represent the distribution of possible trajectories with a small number of latent variables as the latents do not need to encode the entire information in the environment but only how it evolves over time. Therefore, if decoder is designed to not take state as input, the performance greatly diminishes. Even when latent variables are increased 3x by setting $L = 1$, the drop cannot be recovered.

5. Conclusion

In this paper, we reimplemented and evaluated the Trajectory Autoencoding Planner to observe its reproducibility. TAP proposes a novel method to reduce the complexity of using Transformer to solve reinforcement learning problem through sequence prediction. By auto-encoding trajectories to a sequence of low-dimensional latent actions, the attention mechanism works on small, constant-sized tokens. This allows efficient and flexible planning that results in better performance in especially complex and long-horizon tasks. We found that the reported performance is almost the same as the results we obtained in our Knet.jl reimplementation. The differences in performance can be attributed to the stochastic evaluation environment and different seeds used in model training. It remains as future work to try all of the experiments in many environments in the original work and do ablation studies to fully validate the reproducibility of the original results.

F. Module Architecture

In the model, the encoder module ($\mathbf{E}$) consists of a GPT-2 style Transformer. Encoder has 4 self-attention blocks. The vector quantization module ($\mathbf{VQ}$) queries a codebook finds the nearest neighbor of encoded features. The decoder model, architecturally, is the same as the encoder module, except it has a latent mixing MLP inside that conditions the model on the first state of the trajectory. Each block in the Transformer consists of two layer normalization layers, a self-attention module and an MLP. The prior model is also a Transformer that consists of 4 self-attention blocks. Different from the main model, the state was also passed through an MLP and its embeddings were added to the rest before passing through the causal attention module. For all linear layers, ReLU is used as the activation function except the MLP in the Transformer block where GeLU is used instead. The model is implemented in Knet.jl exactly like the original implementation.

G. Implementation Details

Similar to the original implementation, a batch size of 512 is used and both the model and the prior is trained for 125 iterations. The raw data from D4RL is preprocessed to index and batch data from different episode lengths properly. No data augmentation is used. The training time is around 8 hours on a single NVIDIA A100 GPU with our implementation in Knet.jl [39] for the main model. It takes 3 hours to train the smaller prior model. During implementation, AdamW optimizer is implemented in Julia. Also, dropdims and repeat functions on KnetArray are re-implemented in a way that allows backpropagation. The result of layers and the corresponding gradients is tested during development by comparing the values to the ones taken from the original implementation. In the codebook, 512 discrete codes of size 512 is used. The learning rate for the whole model is 0.0002. The codebook used exponential moving average to decay the value of frequently used codes. Differently, from the original implementation, during training exponential average of rarely used embeddings went to zero that caused division by zero instabilities. Therefore, we added an epsilon term of $1e-10$ in this operation. During optimization, some parameters of the model are subject to weight decay while others are not. Per-parameter optimization technique detailed in the Knet documentation is used for this.

References

[1] Arthur Allshire, Roberto Martín-Martín, Charles Lin, Shawn Manuel, Silvio Savarese, and Animesh Garg. LASER: learning a latent action space for efficient reinforcement learning. In *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*, pages 6650–6656. IEEE, 2021. 3

[2] Brandon Amos, Samuel Stanton, Denis Yarats, and Andrew Gordon Wilson. On the model-based stochastic value gradient for continuous reinforcement learning. In Ali Jadbabaie, John Lygeros, George J. Pappas, Pablo A. Parrilo, Benjamin Recht, Claire J. Tomlin, and Melanie N. Zeilinger, editors, *Proceedings of the 3rd Annual Conference on Learning for Dynamics and Control, L4DC 2021, 7-8 June 2021, Virtual Event, Switzerland*, volume 144 of *Proceedings of Machine Learning Research*, pages 6–20. PMLR, 2021. 2

[3] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In Satinder Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 1726–1734. AAAI Press, 2017. 3

[4] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 15084–15097, 2021. 2

[5] John D. Co-Reyes, Yuxuan Liu, Abhishek Gupta, Benjamin Eysenbach, Pieter Abbeel, and Sergey Levine. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1008–1017. PMLR, 2018. 3

[6] Robert Dadashi, Léonard Hussenot, Damien Vincent, Sertan Girgin, Anton Raichuk, Matthieu Geist, and Olivier Pietquin. Continuous control with action quantization from demonstrations. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 4537–4557. PMLR, 2022. 3

[7] Marc Peter Deisenroth and Carl Edward Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 465–472. Omnipress, 2011. 2

[8] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *J. Mach. Learn. Res.*, 6:503–556, 2005. 3

[9] Michael Fairbank. Reinforcement learning by value gradients. *CoRR*, abs/0803.3539, 2008. 2

[10] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. Apr. 2020. 1, 4

[11] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 20132–20145, 2021. 3

[12] Danijar Hafner, Timothy P. Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. 2

[13] Nicolas Heess, Gregory Wayne, David Silver, Timothy P. Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2944–2952, 2015. 2

[14] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. June 2021. 1, 2, 4

[15] Zhengyao Jiang, Tianjun Zhang, Michael Janner, Yueying Li, Tim Rocktäschel, Edward Grefenstette, and Yuandong Tian. Efficient planning in a compact latent action space. *CoRR*, abs/2208.10291, 2022. 2, 4

[16] Siddharth Karamcheti, Albert J. Zhai, Dylan P. Losey, and Dorsa Sadigh. Learning visually guided latent actions for assistive teleoperation. In Ali Jadbabaie, John Lygeros, George J. Pappas, Pablo A. Parrilo, Benjamin Recht, Claire J. Tomlin, and Melanie N. Zeilinger, editors, *Proceedings of the 3rd Annual Conference on Learning for Dynamics and Control, L4DC 2021, 7-8 June 2021, Virtual Event, Switzerland*, volume 144 of *Proceedings of Machine Learning Research*, pages 1230–1241. PMLR, 2021. 3

[17] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. 3, 4

[18] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. 1, 3

[19] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. 1, 3

[20] Thomas Lampe and Martin A. Riedmiller. Approximate model-assisted neural fitted q-iteration. In *2014 International Joint Conference on Neural Networks, IJCNN 2014, Beijing, China, July 6-11, 2014*, pages 2698–2704. IEEE, 2014. 2

[21] Dylan P. Losey, Hong Jun Jeon, Mengxi Li, Krishnan Srinivasan, Ajay Mandlekar, Animesh Garg, Jeannette Bohg, and Dorsa Sadigh. Learning latent actions to control assistive robots. *Auton. Robots*, 46(1):115–147, 2022. 3

[22] Cong Lu, Philip J. Ball, Jack Parker-Holder, Michael A. Osborne, and Stephen J. Roberts. Revisiting design choices in offline model based reinforcement learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. 3

[23] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura, editors, *3rd Annual Conference on Robot Learning, CoRL 2019, Osaka, Japan, October 30 - November 1, 2019, Proceedings*, volume 100 of *Proceedings of Machine Learning Research*, pages 1113–1132. PMLR, 2019. 3

[24] Ajay Mandlekar, Fabio Ramos, Byron Boots, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Dieter Fox. IRIS: implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. In *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*, pages 4414–4420. IEEE, 2020. 3

[25] Josh Merel, Leonard Hasenclever, Alexandre Galashov, Arun Ahuja, Vu Pham, Greg Wayne, Yee Whye Teh, and Nicolas Heess. Neural probabilistic motor primitives for humanoid control. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. 3

[26] Sherjil Ozair, Yazhe Li, Ali Razavi, Ioannis Antonoglou, Aäron van den Oord, and Oriol Vinyals. Vector quantized models for planning. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8302–8313. PMLR, 2021. 2

[27] Xue Bin Peng, Yunrong Guo, Lina Halper, Sergey Levine, and Sanja Fidler. ASE: large-scale reusable adversarial skill embeddings for physically simulated characters. *ACM Trans. Graph.*, 41(4):94:1–94:17, 2022. 3

[28] Jürgen Schmidhuber. Reinforcement learning upside down: Don't predict rewards - just map them to actions. *CoRR*, abs/1912.02875, 2019. 2

[29] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, dec 2019. 1, 2

[30] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. Dec. 2017. 1

[31] David Silver, Richard S. Sutton, and Martin Müller. Sample-based learning and search with permanent and transient memories. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, volume 307 of *ACM International Conference Proceeding Series*, pages 968–975. ACM, 2008. 2

[32] Martin Stolle and Doina Precup. Learning options in reinforcement learning. In Sven Koenig and Robert C. Holte, editors, *Abstraction, Reformulation and Approximation, 5th International Symposium, SARA 2002, Kananaskis, Alberta, Canada, August 2-4, 2002, Proceedings*, volume 2371 of *Lecture Notes in Computer Science*, pages 212–223. Springer, 2002. 3

[33] R. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. 1990. 2

[34] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, 1999. 3

[35] Aäron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Koray Kavukcuoglu, Oriol Vinyals, and Alex Graves. Conditional image generation with pixelcnn decoders. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4790–4798, 2016. 3

[36] Linnan Wang, Rodrigo Fonseca, and Yuandong Tian. Learning search space partition for black-box optimization using monte carlo tree search. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. 2

[37] Tingwu Wang and Jimmy Ba. Exploring model-based planning with policy networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. 2

[38] Kevin Yang, Tianjun Zhang, Chris Cummins, Brandon Cui, Benoit Steiner, Linnan Wang, Joseph E. Gonzalez, Dan Klein, and Yuandong Tian. Learning space partitions for

path planning. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 378–391, 2021. 2

[39] Deniz Yuret. Knet: beginning deep learning with 100 lines of julia. In *Machine Learning Systems Workshop at NIPS 2016*, 2016. 1, 2, 7

[40] Qinqing Zheng, Amy Zhang, and Aditya Grover. Online decision transformer. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 27042–27059. PMLR, 2022. 1, 2, 4

[41] Wenxuan Zhou, Sujay Bajracharya, and David Held. PLAS: latent action space for offline reinforcement learning. In Jens Kober, Fabio Ramos, and Claire J. Tomlin, editors, *4th Conference on Robot Learning, CoRL 2020, 16-18 November 2020, Virtual Event / Cambridge, MA, USA*, volume 155 of *Proceedings of Machine Learning Research*, pages 1719–1735. PMLR, 2020. 3