Reinforcement Learning as One Big Sequence Modeling Problem

Michael Janner **Qiyang Li** **Sergey Levine**

Abstract

Reinforcement learning (RL) is typically concerned with estimating single-step policies or single-step models, leveraging the Markov property to factorize the problem in time. However, we can also view RL as a sequence modeling problem, with the goal being to predict a sequence of actions that leads to a sequence of high rewards. Viewed in this way, it is tempting to consider whether powerful, high-capacity sequence prediction models that work well in other domains, such as natural-language processing, can also provide simple and effective solutions to the RL problem. To this end, we explore how RL can be reframed as "one big sequence modeling" problem, using state-of-the-art Transformer architectures to model distributions over sequences of states, actions, and rewards. Addressing RL as a sequence modeling problem significantly simplifies a range of design decisions: we no longer require separate behavior policy constraints, as is common in prior work on offline model-free RL, and we no longer require ensembles or other epistemic uncertainty estimators, as is common in prior work on model-based RL. All of these roles are filled by the same Transformer sequence model. In our experiments, we demonstrate the flexibility of this approach across long-horizon dynamics prediction, imitation learning, goal-conditioned RL, and offline RL.

1. Introduction

The standard treatment of reinforcement learning relies on decomposing a long-horizon problem into smaller, more local subproblems. In model-free algorithms, this takes the form of the principle of optimality (Bellman, 1957), an elegant recursion that leads naturally to the class of dynamic programming methods like Q-learning. In model-based algorithms, this decomposition takes the form of single-step predictive models, which reduce the problem of predicting high-dimensional, policy-dependent state trajectories to that of estimating a comparatively simpler, policy-agnostic transition distribution.

However, we can also view reinforcement learning as analogous to a sequence generation problem, with the goal being to produce a sequence of actions that, when enacted in an environment, will yield a sequence of high rewards. In this paper, we consider the logical extreme of this analogy: does the toolbox of contemporary sequence modeling itself provide a viable reinforcement learning algorithm? We investigate this question by treating trajectories as unstructured sequences of states, actions, and rewards. We model the distribution of these trajectories using a Transformer architecture (Vaswani et al., 2017), the current tool of choice for capturing long-horizon dependencies. In place of the trajectory optimizers common in model-based control, we use beam search (Reddy, 1997), a heuristic decoding scheme ubiquitous in natural language processing, as a planning algorithm.

Posing reinforcement learning, and more broadly data-driven control, as a sequence modeling problem handles many of the considerations that typically require distinct solutions: actor-critic algorithms require separate actors and critics, model-based algorithms require predictive dynamics models, and offline RL methods often require estimation of the behavior policy (Fujimoto et al., 2019). These components estimate different densities or probability distributions, such as that over actions in the case of actors and behavior policies, or that over states in the case of dynamics models. Even value functions can be viewed as performing inference in a graphical model with auxiliary optimality variables, amounting to estimation of the distribution over future rewards (Levine, 2018). All of these problems can be unified under a single sequence model, which treats states, actions, and rewards as simply a stream of data. The advantage of this perspective is that high-capacity sequence model architectures can be brought to bear on the problem, resulting in a more streamlined approach that could benefit from the same scalability underlying large-scale unsupervised learning results (Brown et al., 2020).

We refer to our model and approach as a Trajectory Transformer. We show that the Trajectory Transformer is a substantially more reliable long-horizon predictor than conventional dynamics models, even in Markovian environments for which the standard model parameterization is in principle sufficient. When combined with a modified beam search procedure that decodes trajectories with high reward, rather

UC Berkeley. Correspondence to: janner@berkeley.edu.

than just high likelihood, Trajectory Transformers can attain results on offline reinforcement learning benchmarks that are competitive with state-of-the-art prior methods designed specifically for that setting. Additionally, we describe how variations on the same decoding procedure can produce a model-based imitation learning method and, with a form of anti-casual conditioning, a goal-reaching method. Our results suggest that the algorithms and architectural motifs that have been widely applicable in unsupervised learning carry similar benefits in reinforcement learning.

2. Related Work

Recent advances in sequence modeling with deep networks have led to rapid improvement in the effectiveness of such models, from LSTMs and sequence-to-sequence models (Hochreiter & Schmidhuber, 1997; Sutskever et al., 2014) to Transformer architectures with self-attention (Vaswani et al., 2017). In light of this, it is tempting to consider how such sequence models can lead to improved performance in RL, which is also concerned with sequential processes (Sutton, 1988). Indeed, a number of prior works have studied applying sequence models of various types to represent components in *standard* RL algorithms, such as policies, value functions, and models (Bakker, 2002; Heess et al., 2015a; Chiappa et al., 2017; Parisotto et al., 2020; Parisotto & Salakhutdinov, 2021; Kumar et al., 2020b). While such works demonstrate the importance of such models for representing memory (Oh et al., 2016), they still rely on standard RL algorithmic advances to improve performance. The goal in our work is different: we specifically aim to *replace* as much of the RL pipeline as possible with sequence modeling, so as to produce a simpler method whose effectiveness is determined by the representational capacity of the sequence model rather than algorithmic sophistication.

Estimation of probability distributions and densities arises in many places in learning-based control. The most obvious is model-based RL, where it is used to train predictive models that can then be used for planning or policy learning (Sutton, 1990; Silver et al., 2008; Fairbank, 2008; Deisenroth & Rasmussen, 2011; Lampe & Riedmiller, 2014; Heess et al., 2015b; Chua et al., 2018; Wang & Ba, 2020; Amos et al., 2020). However, it also figures heavily in offline RL, where it is used to estimate conditional distributions over *actions* that serve to constrain the learned policy to avoid out-of-distribution behavior that is not supported under the dataset (Fujimoto et al., 2019; Kumar et al., 2019a; Ghasemipour et al., 2020); imitation learning, where it is used to fit an expert's actions to obtain a policy (Ross & Bagnell, 2010; Ross et al., 2011); and other areas such as hierarchical RL (Peng et al., 2017; Co-Reyes et al., 2018; Jiang et al., 2019). In our method, we train a single high-capacity sequence model to represent the joint distribution over sequences of states, actions, and rewards. This serves as *both* a predictive model *and* a behavior policy (for imitation) or behavior constraint (for offline RL). Our model treats states, actions, and rewards interchangeably, and does not require separate components for policies or models.

Our approach to RL is most closely related to prior model-based RL methods that plan with a learned model (Chua et al., 2018; Wang & Ba, 2020), in that we also use an optimization procedure, based on the standard beam search algorithm typically used with sequence models, to select actions. However, while these prior methods typically require additional machinery to work well, such as ensembles (in the online setting) (Chua et al., 2018; Kurutach et al., 2018; Buckman et al., 2018; Malik et al., 2019) or conservatism or pessimism mechanisms (in the offline setting) (Yu et al., 2020; Kidambi et al., 2020; Argenson & Dulac-Arnold, 2020), our method does not require explicit handling of these components. Modeling the states and actions jointly already provides a bias toward generating in-distribution actions, which avoids the need for explicit pessimism (Fujimoto et al., 2019; Kumar et al., 2019a; Ghasemipour et al., 2020; Nair et al., 2020; Jin et al., 2020; Yin et al., 2021; Dadashi et al., 2021). In the context of recently proposed offline RL algorithms, our method can be interpreted as a combination of model-based RL and policy constraints (Kumar et al., 2019a; Wu et al., 2019), though, again, it does not require introducing such constraints explicitly – they emerge from our choice to jointly model trajectories and decode via beam search. In the context of model-free RL, our method also resembles recently proposed work on goal relabeling (Andrychowicz et al., 2017; Rauber et al., 2019; Ghosh et al., 2021) and reward-conditioning (Schmidhuber, 2019; Srivastava et al., 2019; Kumar et al., 2019b) to reinterpret all past experience as useful demonstrations with proper contextualization.

Concurrently with our work, Chen et al. (2021) also proposed a reinforcement learning approach centered around sequence prediction with Transformers. This work further supports the possibility that a high-capacity sequence model can be applied to reinforcement learning problems without the need for the components usually associated with reinforcement learning algorithms.

3. Reinforcement Learning and Control as Sequence Modeling

In this section, we describe the training procedure for our sequence model and discuss how it can be used for control and reinforcement learning. We refer to the model as a Trajectory Transformer for brevity, but emphasize that at the implementation level, both our model and search strategy are nearly identical to those common in natural language processing. As a result, modeling considerations are concerned less with architecture design and more with how to represent trajectory data – consisting of continuous states and actions – for processing by a discrete-token architecture.

3.1. Trajectory Transformers

At the core of our approach is the treatment of trajectory data as an unstructured sequence for modeling by a Transformer architecture. A trajectory τ consists of N-dimensional states, M-dimensional actions, and scalar rewards:

$$\tau = \{\mathbf{s}_t^0, \mathbf{s}_t^1, \ldots, \mathbf{s}_t^{N-1}, \mathbf{a}_t^0, \mathbf{a}_t^1, \ldots, \mathbf{a}_t^{M-1}, r_t\}_{t=0}^{T-1}.$$

Subscripts on all tokens denote timestep and superscripts on states and actions denote dimension (*i.e.*, $\mathbf{s}_t^i$ is the i^{th} dimension of the state at time t). In the case of continuous states and actions, we must additionally discretize each dimension; we do so using a regular grid with a fixed number of bins per dimension. Assuming $\mathbf{s}_t^i \in [\ell^i, r^i)$, the tokenization of $\mathbf{s}_t^i$ is defined as

$$\bar{\mathbf{s}}_t^i = \left\lfloor V \frac{\mathbf{s}_t^i - \ell^i}{r^i - \ell^i} \right\rfloor + Vi \tag{1}$$

in which $\lfloor \cdot \rfloor$ denotes the floor function and V is the size of the per-dimension vocabulary $\mathcal{V}$. We offset state tokens by Vi to ensure that different state dimensions are represented by disjoint sets of tokens; action tokens $\bar{\mathbf{a}}_t^j$ must analogously be offset by $V \times (N + j)$ and discretized rewards $\bar{r}_t$ must be offset by $V \times (N + M)$. Note that each step in the sequence therefore corresponds to a *dimension* of the state, action, or reward, such that a trajectory with T time steps would correspond to a sequence of length $T \times (N + M + 1)$. While this choice may seem inefficient, it allows us to model the distribution over trajectories with more expressivity, without simplifying assumptions such as Gaussian transitions.

Our model is a Transformer decoder mirroring the GPT architecture (Radford et al., 2018). We use a smaller architecture than those typically used in large-scale language modeling, consisting of four layers and six self-attention heads. A full architectural description is provided in Appendix A.

Training is performed with the standard teacher-forcing procedure (Williams & Zipser, 1989) used to train recurrent models. Denoting the parameters of the Trajectory Transformer as θ and induced conditional probabilities as P_θ, the objective maximized during training is:

$$\begin{aligned}\mathcal{L}(\bar{\tau}) = \sum_{t=0}^{T-1} \Bigg(&\sum_{i=0}^{N-1} \log P_\theta\big(\bar{\mathbf{s}}_t^i \mid \bar{\mathbf{s}}_t^{<i}, \bar{\tau}_{<t}\big) \\ &+ \sum_{j=0}^{M-1} \log P_\theta\big(\bar{\mathbf{a}}_t^j \mid \bar{\mathbf{a}}_t^{<j}, \bar{\mathbf{s}}_t, \bar{\tau}_{<t}\big) \\ &+ \log P_\theta\big(\bar{r}_t \mid \bar{\mathbf{a}}_t, \bar{\mathbf{s}}_t, \bar{\tau}_{<t}\big)\Bigg),\end{aligned}$$

in which we use $\bar{\tau}_{<t}$ as a shorthand for a tokenized trajectory from timesteps 0 through $t-1$. For brevity, probabilities are written as conditional on all preceding tokens in a trajectory, but due to the quadratic complexity of self-attention (Kitaev et al., 2020) we must limit the maximum number of conditioning tokens to 512, corresponding to a horizon of $\frac{512}{N+M+1}$ transitions. We use the Adam optimizer (Kingma & Ba, 2015) with a learning rate of 2.5×10^{-4} to train parameters θ.

3.2. Transformer Trajectory Optimization

We now describe how sequence generation with the Trajectory Transformer can be repurposed for control, focusing on three settings: imitation learning, goal-conditioned reinforcement learning, and offline reinforcement learning. These settings are listed in increasing amount of required modification on top of the sequence model decoding algorithms routinely used in natural language processing. We refer to all of the below variations collectively as Transformer trajectory optimization (TTO).

Imitation learning. When the goal is to reproduce the distribution of trajectories in the training data, we can optimize directly for the probability of a trajectory τ beginning from a starting state $\mathbf{s}_0$. This situation matches the goal of sequence modeling exactly, and as such we may use beam search without modification. We describe this procedure in Algorithm 1.

The result of this procedure is a tokenized trajectory $\bar{\tau}$, beginning from a current state $\mathbf{s}_t$, that has high probability under the data distribution. If the first action $\bar{\mathbf{a}}_t$ in the sequence is enacted and the process is repeated, we have a receding horizon-controller. This approach is a model-based variant of behavior cloning, in which both actions and states are selected in order to produce a probable trajectory from the reference behavior instead of the usual strategy of selecting only a probable action given a current state or state history. If we set the predicted sequence length to be the action dimension, our approach corresponds exactly to the simplest form of behavior cloning with an autoregressive policy.

Algorithm 1 Beam search

Require State $\mathbf{s}$, vocabulary $\mathcal{V}$
Require Sequence length L, beam width B
Discretize $\mathbf{s}$ to $\bar{\mathbf{s}}$ (Equation 1)
Initialize $\mathcal{T}_0 = \{([\bar{\mathbf{s}}], 0)\}$ and $\mathcal{T}_{1:L} = \emptyset$
for $l \in \{1, \cdots, L\}$ **do**
  for $(\bar{\tau}_{l-1}, q_{l-1}) \in \mathcal{T}_{l-1}, v \in \mathcal{V}$ **do**
    $\bar{\tau}_l \leftarrow \bar{\tau}_{l-1} + [v]$
    $q_l \leftarrow q_{l-1} + \log P_\theta(v \mid \bar{\tau}_{l-1})$
    $\mathcal{T}_l \leftarrow \mathcal{T}_l \cup (\bar{\tau}_l,\ q_l)$
  end for
`// Select` B `most probable sequences`
  $\mathcal{T}_l \leftarrow \arg\max_{\mathcal{T} \subseteq \mathcal{T}_l, |\mathcal{T}|=B} \sum_{(\bar{\tau}, q) \in \mathcal{T}} \{q\}$
end for
Return $\arg\max_{\bar{\tau} \mid (\bar{\tau}, q) \in \mathcal{T}_L} \{q\}$

Goal-conditioned reinforcement learning. Transformer architectures feature a "causal" attention mask to ensure that predictions only depend on previous tokens in a sequence. In the context of natural language, this design corresponds to generating sentences in the linear order in which they are spoken as opposed to an ordering reflecting their hierarchical syntactic structure (see, however, Gu et al. (2019) for a discussion of non-left-to-right sentence generation with autoregressive models). In the context of trajectory prediction, this choice instead reflects physical causality, disallowing future events to affect the past. However, the conditional probabilities of the past given the future are still well-defined, allowing us to condition samples not only on the preceding states, actions, and rewards that have already been observed, but also any future context that we wish to occur. If the future context is a state at the end of a trajectory, we decode trajectories with probabilities of the form:

$$P(\bar{\mathbf{s}}_t^i \mid \bar{\mathbf{s}}_t^{<i}, \bar{\tau}_{<t}, \bar{\mathbf{s}}_{T-1})$$

We can use this directly as a goal-reaching method by conditioning on a desired final state. If we always condition sequences on a final goal state, we can leave the lower-diagonal attention mask intact and simply permute the input trajectory to $\{\bar{\mathbf{s}}_{T-1}, \bar{\mathbf{s}}_0, \bar{\mathbf{s}}_1, \ldots, \bar{\mathbf{s}}_{T-2}\}$. By prepending the goal state to the beginning of a sequence, we ensure that all other predictions may attend to it without modifying the standard attention implementation. This procedure for goal-conditioning resembles prior methods that use supervised learning to train goal-conditioned policies (Ghosh et al., 2021) and is also related to relabeling techniques in model-free RL (Andrychowicz et al., 2017). In our framework, it is identical to the standard subroutine in sequence modeling: inferring the most likely sequence given available evidence.

Offline reinforcement learning. The beam search method described in Algorithm 1 optimizes sequences for their probability under the data distribution. By replacing the log-probabilities of token predictions with the predicted reward signal, we can use the same Trajectory Transformer and search strategy for reward-maximizing behavior. Appealing to the control as inference graphical model (Levine, 2018), we are in effect replacing a transition's log-probability in beam search with its log-probability *of optimality*, which corresponds to the sum of rewards.

Using beam-search as a reward-maximizing procedure has the risk of leading to myopic behavior. To address this issue, we augment each transition in the training trajectories with reward-to-go:

$$R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$$

and include it as an additional quantity, discretized identically to the others, to be predicted alongside immediate rewards. During planning, we then have access to value estimates from our model to add to cumulative rewards. While acting greedily with respect to such Monte Carlo value estimates is known to suffer from poor sample complexity and convergence to suboptimal behavior when online data collection is not allowed, we only use this reward-to-go estimate as a heuristic to guide beam search, and hence our method does not require the estimated values to be particularly accurate. Note also that, in the offline RL case, these reward-to-go quantities estimate the value of the *behavior policy* and will not, in general, match the values achieved by TTO. Of course, it is much simpler to learn the value function of the behavior policy than that of the optimal policy, since we can simply use Monte Carlo estimates without relying on Bellman updates. A proper value estimator for the TTO policy could plausibly give us an even better search heuristic, though it would require invoking the tools of dynamic programming. In contrast, augmenting trajectories with reward-to-go and predicting with a discretized model is as simple as training a classifier with full supervision.

Because our Transformer predicts reward and reward-to-go only every $N + M + 1$ tokens, we sample all intermediate tokens using log-probabilities, as in the imitation learning and goal-reaching settings. More specifically, we sample full transitions $(\bar{\mathbf{s}}_t, \bar{\mathbf{a}}_t, \bar{r}_t, \bar{R}_t)$ using likelihood-maximizing beam search, treat these transitions as our vocabulary, and filter sampled trajectories by those with the highest cumulative reward plus reward-to-go estimate.

We have taken a sequence-modeling route to what could be described as a fairly simple-looking model-based planning algorithm, in that we sample candidate action sequences, evaluate their effects using a predictive model, and select the reward-maximizing trajectory. This conclusion is in part due to the close relation between sequence modeling and trajectory optimization. There is one dissimilarity, however,

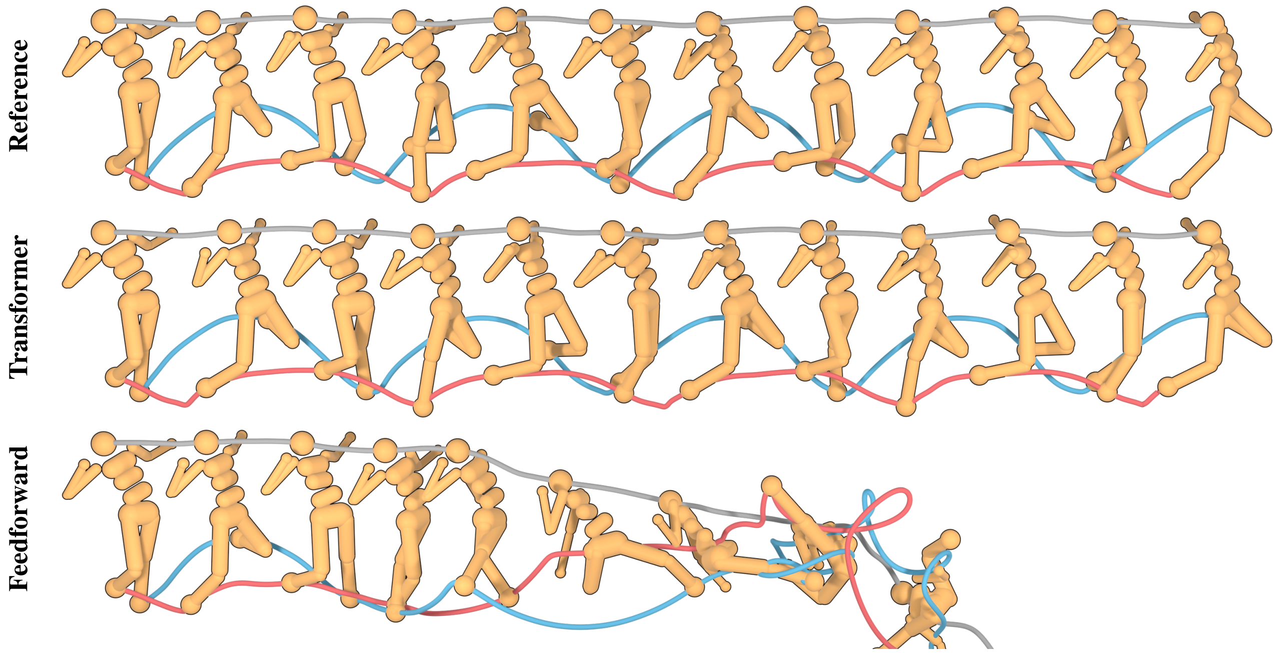

Figure 1. **(Prediction visualization)** A qualitative comparison of length-100 trajectories generated by the Trajectory Transformer and a feedforward Gaussian dynamics model from PETS, a state-of-the-art planning algorithm (Chua et al., 2018). Both models were trained on trajectories collected by a single policy, for which a true trajectory is shown for reference. Compounding errors in the single-step model lead to physically implausible predictions, whereas the Transformer-generated trajectory is visually indistinguishable from those produced by the policy acting in the actual environment. The paths of the feet and head are traced through space for depiction of the movement between rendered frames.

that is worth highlighting: by modeling actions jointly with states and sampling them using the same procedure, we can prevent the model from being queried on out-of-distribution actions. The alternative, of treating action sequences as unconstrained optimization variables that do not depend on state (Nagabandi et al., 2018), can more readily lead to model exploitation, as the problem of maximizing reward under a learned model closely resembles that of finding adversarial examples for a classifier (Goodfellow et al., 2014).

4. Experiments

Our experimental evaluation focuses on (1) the accuracy of the Trajectory Transformer as a long-horizon predictor compared to standard dynamics model parameterizations and (2) the utility of sequence modeling tools – namely beam search – as a control algorithm in the context of offline reinforcement learning, imitation learning, and goal-reaching.

4.1. Model Analysis

We begin by evaluating the Trajectory Transformer as a long-horizon policy-conditioned predictive model. The usual strategy for predicting trajectories given a policy is to rollout with a single-step model, with actions supplied by the policy. Our protocol differs from the standard approach not only in that the model is not Markovian, but also in that it does not require access to a policy to make predictions – the outputs of the policy are modeled alongside the states encountered by that policy. Here, we focus only on the quality of the model's predictions; we use actions predicted by the model for an imitation learning method in the next subsection.

Trajectory predictions. Figure 1 depicts a visualization of predicted 100-timestep trajectories from our model after having trained on a dataset collected by a trained humanoid policy. Though model-based methods have been applied to the humanoid task, prior works tend to keep the horizon intentionally short to prevent the accumulation of model errors (Janner et al., 2019; Amos et al., 2020). The reference model is the probabilistic ensemble implementation of PETS (Chua et al., 2018); we tuned the number of models within the ensemble, the number of layers, and layer sizes, but were unable to produce a model that predicted accurate sequences for more than a few dozen steps. In contrast, we see that the Trajectory Transformer's long-horizon predictions are substantially more accurate, remaining visually indistinguishable from the ground-truth trajectories even after 100 predicted steps. To our knowledge, no prior model-based RL algorithm has demonstrated predicted rollouts of such accuracy and length on tasks of comparable dimensionality.

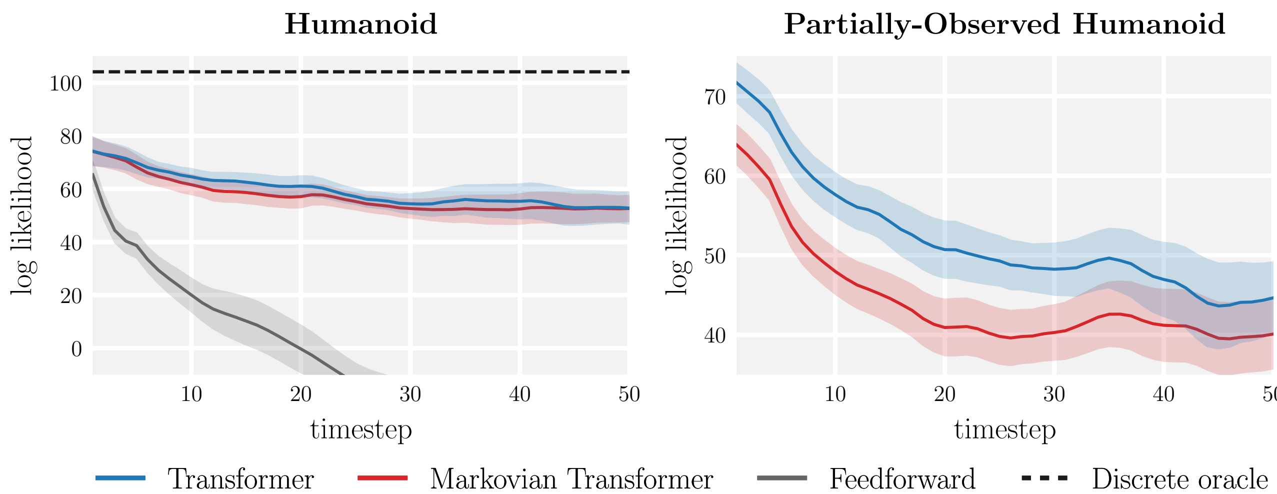

Figure 2. **(Compounding model errors)** We compare the accuracy of the Trajectory Transformer to that of the probabilistic feedforward model ensemble (Chua et al., 2018) over the course of a planning horizon in the humanoid environment, corresponding to the trajectories visualized in Figure 1. We find that the trajectory Transformer has substantially better error compounding with respect to prediction horizon than the feedforward model. The discrete oracle is the maximum log likelihood attainable given the discretization size; see Appendix B for a discussion.

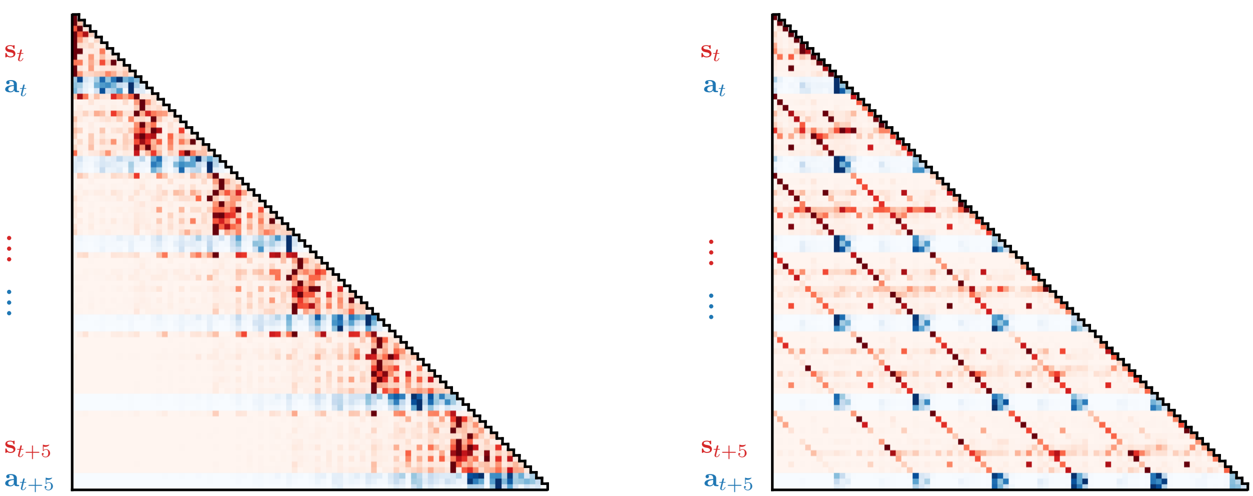

Figure 3. **(Attention patterns)** We observe two distinct types of attention masks during trajectory prediction. In the first, both states and actions are dependent primarily on the immediately preceding transition, corresponding to a model that has learned the Markov property. The second strategy has a striated appearance, with state dimensions depending most strongly on the same dimension of multiple previous timesteps. Surprisingly, actions depend more on past actions than they do on past states, reminiscent of the action smoothing used in some trajectory optimization algorithms (Nagabandi et al., 2019). Masks are produced by a first- and third-layer attention head during sequence prediction on the hopper benchmark; reward dimensions are omitted for this visualization.

Error accumulation. A quantitative account of the same finding is provided in Figure 2, in which we evaluate the model's accumulated error versus prediction horizon. Standard predictive models tend to have excellent single-step errors but poor long-horizon accuracy, so instead of evaluating a test-set single-step likelihood, we sample 1000 trajectories from a fixed starting point to estimate the per-timestep state marginal predicted by each model. We then report the likelihood of the states visited by the reference policy on a held-out set of trajectories under these predicted marginals. To evaluate the likelihood under our discretized model, we treat each bin as a uniform distribution over its specified range; by construction, the model assigns zero probability outside of this range.

To better isolate the source of the Transformer's improved accuracy over standard single-step models, we also evaluate

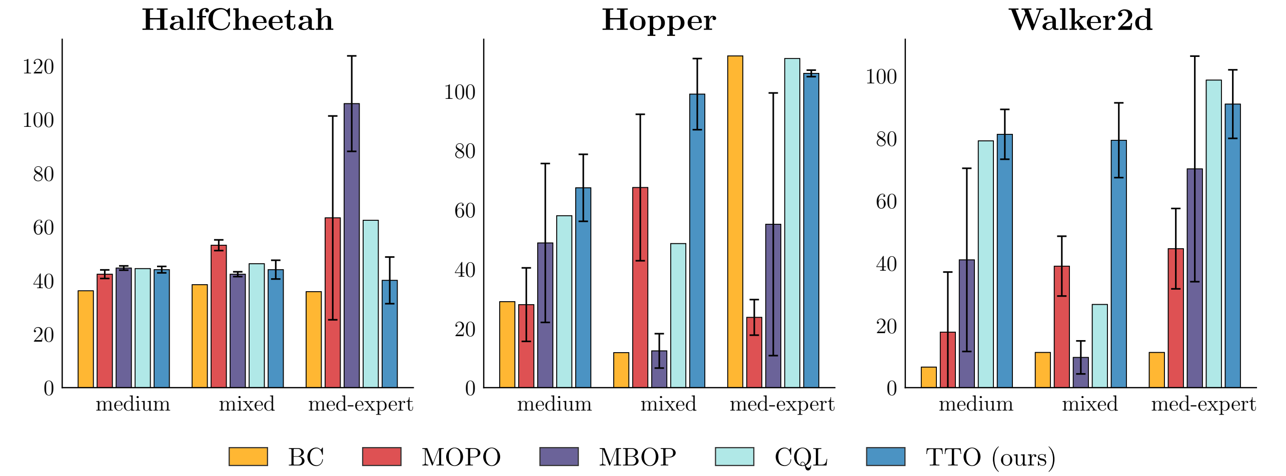

Figure 4. **(Offline reinforcement learning):** TTO performs on par with or better than the best prior offline reinforcement learning algorithms on the D4RL benchmark suite. Results for TTO correspond to the mean over 15 random seeds (5 independently trained Transformers and 3 trajectories per Transformer), with error bars depicting standard deviation between runs. We detail the sources of the performance for other methods in Appendix C. A listing of these results in tabular form is provided in Appendix E.

a Markovian variant of our same architecture. This ablation has a truncated context window that prevents it from attending to more than one timestep in the past. We find that this model performs similarly to the trajectory Transformer on fully-observed environments, suggesting that architecture differences and increased expressivity from the autoregressive state discretization play a large role in the trajectory Transformer's long-horizon accuracy. We construct a partially-observed version of the same humanoid environment, in which each dimension of every state is masked out with 50% probability (Figure 2 right), and find that, as expected, the long-horizon conditioning plays a larger role in the model's accuracy in this setting.

Attention patterns. We visualize the attention maps during model predictions in Figure 3. We find two primary attention patterns. The first is a discovered Markovian strategy, in which a state prediction attends overwhelmingly to the previous transition. The second is qualitatively striated, with the model attending to specific dimensions in multiple prior states for each state prediction. Simultaneously, the action predictions attend to prior actions more than they do prior states. This contrasts with the usual formulation of behavior cloning, in which actions are a function of only past states, but is reminiscent of the action filtering technique used in some planning algorithm to produce smoother action sequences (Nagabandi et al., 2019).

4.2. Reinforcement Learning and Control

Offline reinforcement learning. We evaluate TTO on the D4RL offline RL benchmark suite, with results shown in Figure 4. This evaluation is the most difficult of our control settings, as reward-maximizing behavior is the most qualitatively dissimilar from the types of behavior that are normally associated with unsupervised modeling – namely, imitative behavior. We compare against four other methods: (1) conservative Q-learning (CQL; (Kumar et al., 2020a)), (2) model-based offline policy optimization (MOPO; (Yu et al., 2020)), model-based offline planning (MBOP; (Argenson & Dulac-Arnold, 2020)), and behavior cloning (BC). The first two comprise the current state-of-the-art in model-free and model-based offline reinforcement learning. MBOP provides a point of comparison for a planning algorithm that uses a single-step dynamics model as opposed to a Transformer. We find that on the hopper and walker benchmarks, across all dataset types, TTO performs on par with or better than the best prior offline RL methods. On the halfcheetah environment, TTO matches the performance of prior methods except on the medium-expert dataset, possibly due to the increased range of the velocities in the expert data causing the state discretization to become too coarse.

Imitation and goal-reaching. We additionally run TTO using standard likelihood-maximizing, as opposed to return-maximizing, beam search. We find that after training the Trajectory Transformer on datasets collected by expert policies (Fu et al., 2020), using beam search as a receding-horizon controller achieves an average normalized return of 104% and 109% in the hopper and walker2d environments, respectively. While this result is perhaps unsurprising, as behavior cloning with standard feedforward architectures is already able to reproduce the behavior of the expert policies, it demonstrates that a decoding algorithm used for language

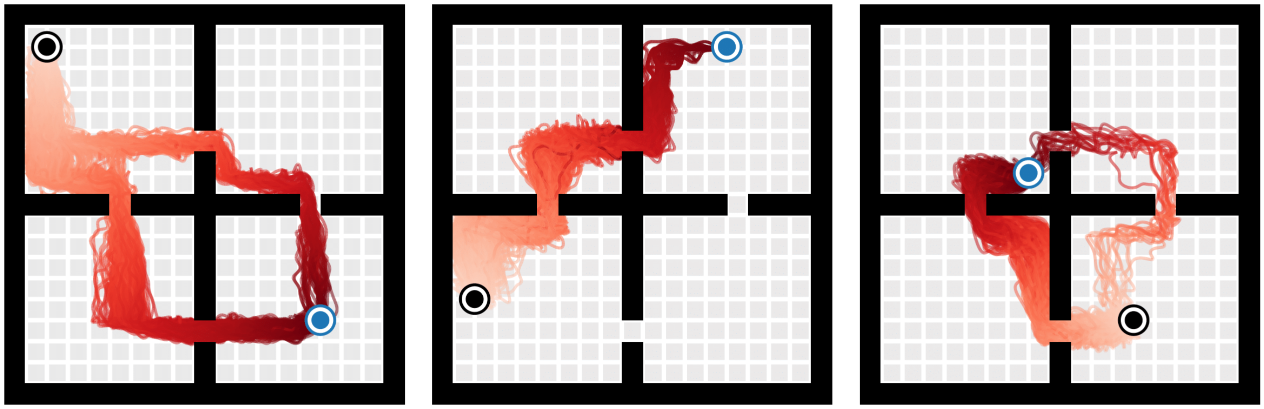

Figure 5. **(Goal-reaching)** Trajectories collected by TTO with anti-causal goal-state conditioning in a continuous variant of the four rooms environment. Trajectories are visualized as curves passing through all encountered states, with color becoming more saturated as time progresses. Note that these curves depict real trajectories collected by the controller and not sampled sequences. The starting state is depicted by ⦿ and the goal state by ⦿. Best viewed in color.

modeling can be effectively repurposed for control.

Finally, we evaluate the goal-reaching variant of likelihood-maximizing TTO, which conditions on a future desired state alongside previously encountered states. We use a continuous variant of the classic four rooms environment as a testbed (Sutton et al., 1999). Our training data consists of trajectories collected by a pretrained goal-reaching agent, with start and goal states sampled uniformly at random across the state space. Figure 5 depicts routes taken by TTO; we see that anti-causal conditioning on a future state allows for beam search to be used as a goal-reaching method. No reward shaping, or rewards of any sort, are required; the planning method relies entirely on goal relabeling.

5. Discussion

We have presented a sequence modeling view on reinforcement learning that enables us to derive a single algorithm for a diverse range of problem settings, unifying many of the standard components of reinforcement learning algorithms (such as policies, models, and value functions) under a single sequence model. The algorithm involves training a sequence model jointly on states, actions, and rewards and sampling from it using a minimally modified beam search. Despite drawing from the tools of large-scale language modeling instead of those normally associated with control, we find that this approach is effective in imitation learning, goal-reaching, and offline reinforcement learning.

The simplicity and flexibility of TTO do come with limitations. Prediction with Transformers is slower and more resource-intensive than prediction with the types of single-step models often used in model-based control. While real-time control with Transformers for most dynamical systems is currently out of reach, growing interest in computationally-efficient Transformer architectures (Tay et al., 2021) could cut runtimes down substantially. Further, in TTO we have chosen to discretize continuous data to fit a standard architecture instead of modifying the architecture to handle continuous inputs. While we found this design to be much more effective than conventional continuous dynamics models, it does in principle impose an upper bound on prediction precision. More sophisticated discretization approaches such as adaptive grids (Sinclair et al., 2019) or learned discretizations (Maddison et al., 2016; Jang et al., 2016; van den Oord et al., 2017) could alleviate these issues.

One of the interesting implications of our results is that reinforcement learning problems can be reframed as supervised learning tasks with an appropriate choice of model. This can allow bringing to bear high-capacity models trained with stable and reliable algorithms. While we are not the first to make this observation, our results are perhaps an especially extreme illustration of this principle: TTO dispenses with many of the standard assumptions in reinforcement learning, including the Markov property, and still attains results on a range of offline reinforcement learning benchmarks that are competitive with the best prior methods. A particularly exciting direction for future work is to investigate whether further increasing model size and devising more effective representations can further simplify learning-based control methods.

References

Amos, B., Stanton, S., Yarats, D., and Wilson, A. G. On the model-based stochastic value gradient for continuous reinforcement learning. *arXiv preprint arXiv:2008.12775*, 2020.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. Hindsight experience replay. In *Advances in Neural Information Processing Systems*. 2017.

Argenson, A. and Dulac-Arnold, G. Model-based offline planning. *arXiv preprint arXiv:2008.05556*, 2020.

Bakker, B. Reinforcement learning with long short-term memory. *Neural Information Processing Systems*, 01 2002.

Bellman, R. *Dynamic Programming*. Dover Publications, 1957.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Buckman, J., Hafner, D., Tucker, G., Brevdo, E., and Lee, H. Sample-efficient reinforcement learning with stochastic ensemble value expansion. *arXiv preprint arXiv:1807.01675*, 2018.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision Transformer: Reinforcement learning via sequence modeling. *arXiv preprint arXiv:2106.01345*, 2021.

Chiappa, S., Racaniere, S., Wierstra, D., and Mohamed, S. Recurrent environment simulators. 2017.

Chua, K., Calandra, R., McAllister, R., and Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*. 2018.

Co-Reyes, J., Liu, Y., Gupta, A., Eysenbach, B., Abbeel, P., and Levine, S. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In *International Conference on Machine Learning*, pp. 1009–1018. PMLR, 2018.

Dadashi, R., Rezaeifar, S., Vieillard, N., Hussenot, L., Pietquin, O., and Geist, M. Offline reinforcement learning with pseudometric learning. *arXiv preprint arXiv:2103.01948*, 2021.

Deisenroth, M. and Rasmussen, C. E. PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning*, 2011.

Fairbank, M. Reinforcement learning by value gradients. *arXiv preprint arXiv:0803.3539*, 2008.

Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine, S. D4RL: Datasets for deep data-driven reinforcement learning, 2020.

Fujimoto, S., Meger, D., and Precup, D. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pp. 2052–2062. PMLR, 2019.

Ghasemipour, S. K. S., Schuurmans, D., and Gu, S. S. Emaq: Expected-max q-learning operator for simple yet effective offline and online rl. *arXiv preprint arXiv:2007.11091*, 2020.

Ghosh, D., Gupta, A., Reddy, A., Fu, J., Devin, C. M., Eysenbach, B., and Levine, S. Learning to reach goals via iterated supervised learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=rALA0Xo6yNJ.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Gu, J., Liu, Q., and Cho, K. Insertion-based Decoding with Automatically Inferred Generation Order. *Transactions of the Association for Computational Linguistics*, 2019.

Heess, N., Hunt, J. J., Lillicrap, T., and Silver, D. Memory-based control with recurrent neural networks. *ArXiv*, abs/1512.04455, 2015a.

Heess, N., Wayne, G., Silver, D., Lillicrap, T., Tassa, Y., and Erez, T. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, 2015b.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Janner, M., Fu, J., Zhang, M., and Levine, S. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, 2019.

Jiang, Y., Gu, S., Murphy, K., and Finn, C. Language as an abstraction for hierarchical deep reinforcement learning. *arXiv preprint arXiv:1906.07343*, 2019.

Jin, Y., Yang, Z., and Wang, Z. Is pessimism provably efficient for offline rl? *arXiv preprint arXiv:2012.15085*, 2020.

Kidambi, R., Rajeswaran, A., Netrapalli, P., and Joachims, T. Morel: Model-based offline reinforcement learning. *arXiv preprint arXiv:2005.05951*, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Kitaev, N., Kaiser, Ł., and Levskaya, A. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Kumar, A., Fu, J., Tucker, G., and Levine, S. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, 2019a.

Kumar, A., Peng, X. B., and Levine, S. Reward-conditioned policies. *arXiv preprint arXiv:1912.13465*, 2019b.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020a.

Kumar, S., Parker, J., and Naderian, P. Adaptive transformers in RL. *arXiv preprint arXiv:2004.03761*, 2020b.

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

Lampe, T. and Riedmiller, M. Approximate model-assisted neural fitted Q-iteration. In *International Joint Conference on Neural Networks*, 2014.

Levine, S. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Malik, A., Kuleshov, V., Song, J., Nemer, D., Seymour, H., and Ermon, S. Calibrated model-based deep reinforcement learning. In *International Conference on Machine Learning*, pp. 4314–4323. PMLR, 2019.

Nagabandi, A., Kahn, G., S. Fearing, R., and Levine, S. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *International Conference on Robotics and Automation*, 2018.

Nagabandi, A., Konoglie, K., Levine, S., and Kumar, V. Deep Dynamics Models for Learning Dexterous Manipulation. In *Conference on Robot Learning*, 2019.

Nair, A., Dalal, M., Gupta, A., and Levine, S. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

Oh, J., Chockalingam, V., Lee, H., et al. Control of memory, active perception, and action in minecraft. In *International Conference on Machine Learning*, pp. 2790–2799. PMLR, 2016.

Parisotto, E. and Salakhutdinov, R. Efficient transformers in reinforcement learning using actor-learner distillation. In *International Conference on Learning Representations*, 2021.

Parisotto, E., Song, F., Rae, J., Pascanu, R., Gulcehre, C., Jayakumar, S., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S., et al. Stabilizing transformers for reinforcement learning. In *International Conference on Machine Learning*, 2020.

Peng, X. B., Berseth, G., Yin, K., and Van De Panne, M. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):1–13, 2017.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. Improving language understanding by generative pretraining. 2018.

Rauber, P., Ummadisingu, A., Mutz, F., and Schmidhuber, J. Hindsight policy gradients. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg2viA5FQ.

Reddy, R. Speech understanding systems: Summary of results of the five-year research effort at Carnegie Mellon University, 1997.

Ross, S. and Bagnell, D. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 661–668. JMLR Workshop and Conference Proceedings, 2010.

Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Schmidhuber, J. Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*, 2019.

Silver, D., Sutton, R. S., and Müller, M. Sample-based learning and search with permanent and transient memories. In *Proceedings of the International Conference on Machine Learning*, 2008.

Sinclair, S. R., Banerjee, S., and Yu, C. L. Adaptive discretization for episodic reinforcement learning in metric spaces. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(3):1–44, 2019.

Srivastava, R. K., Shyam, P., Mutz, F., Jaśkowski, W., and Schmidhuber, J. Training agents using upside-down reinforcement learning. *arXiv preprint arXiv:1912.02877*, 2019.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf.

Sutton, R. S. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9, 1988.

Sutton, R. S. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *International Conference on Machine Learning*, 1990.

Sutton, R. S., Precup, D., and Singh, S. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1): 181 – 211, 1999.

Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., and Metzler, D. Long range arena : A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=qVyeW-grC2k.

van den Oord, A., Vinyals, O., and kavukcuoglu, k. Neural discrete representation learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/7a98af17e63a0ac09ce2e96d03992fbc-Paper.pdf.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Wang, T. and Ba, J. Exploring model-based planning with policy networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1exf64KwH.

Williams, R. J. and Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

Wu, Y., Tucker, G., and Nachum, O. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

Yin, M., Bai, Y., and Wang, Y.-X. Near-optimal offline reinforcement learning via double variance reduction. *arXiv preprint arXiv:2102.01748*, 2021.

Yu, T., Thomas, G., Yu, L., Ermon, S., Zou, J., Levine, S., Finn, C., and Ma, T. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.

A. Model and training specification

Architecture and optimization details. In all environments, we use a Transformer architecture with four layers and six self-attention heads. The total input vocabulary of the model is $V \times (N + M + 2)$ to account for states, actions, rewards, and rewards-to-go, but the output linear layer produces logits only over a vocabulary of size V; output tokens can be interpreted unambiguously because their offset is uniquely determined by that of the previous input. The dimension of each token embedding is 192. Dropout is applied at the end of each block with probability 0.1.

We follow the learning rate scheduling of (Radford et al., 2018), increasing linearly from 0 to 2.5×10^{-4} over the course of 2000 updates. We use a batch size of 64 for most experiments, but increase this up to 256 when GPU memory allows (for example, in low-dimensional environments like four rooms).

Hardware. Model training took place on NVIDIA Tesla V100 GPUs (NCv3 instances on Microsoft Azure) for 80 epochs, taking approximately 6-12 hours (varying with dataset size) per model on one GPU.

B. Discrete oracle

The discrete oracle in Figure 2 is the maximum log-likelihood attainable by a model under our discretization granularity. For a single state dimension i, this maximum is achieved by a model that places all probability mass on the correct token, corresponding to a uniform distribution over an interval of size

$$\frac{r_i - \ell_i}{V}.$$

The total log-likelihood over the entire state is then given by:

$$\sum_{i=1}^{N} \log \frac{V}{r_i - \ell_i}.$$

C. Baseline performance sources

Imitation learning The performance of the behavior cloning (BC) baseline is taken from Kumar et al. (2020a).

Offline reinforcement learning The performance of MOPO is taken from Table 1 in Yu et al. (2020). The performance of MBOP is taken from Table 1 in Argenson & Dulac-Arnold (2020). The performance of BC and CQL are taken from Table 1 in Kumar et al. (2020a).

D. Datasets

The D4RL (Fu et al., 2020) dataset that we used in our experiments is under the Creative Commons Attribution 4.0 License (CC BY). The license information can be found at

`https://github.com/rail-berkeley/d4rl/blob/master/README.md`

under the "Licenses" section.

E. Offline Reinforcement Learning Results

Environment	Dataset type	BC	TTO (ours)	CQL	MOPO	MBOP
halfcheetah	medium	36.1	44.0 ± 1.2	44.4	42.3 ± 1.6	44.6 ± 0.8
halfcheetah	mixed	38.4	44.1 ± 3.5	46.2	53.1 ± 2.0	42.3 ± 0.9
halfcheetah	med-expert	35.8	40.8 ± 8.7	62.4	63.3 ± 38.0	105.9 ± 17.8
hopper	medium	29.0	67.4 ± 11.3	58.0	28.0 ± 12.4	48.8 ± 26.8
hopper	mixed	11.8	99.4 ± 12.6	48.6	67.5 ± 24.7	12.4 ± 5.8
hopper	med-expert	111.9	106 ± 1.1	111.0	23.7 ± 6.0	55.1 ± 44.3
walker2d	medium	6.6	81.3 ± 8.0	79.2	17.8 ± 19.3	41.0 ± 29.4
walker2d	mixed	11.3	79.4 ± 12.8	26.7	39.0 ± 9.6	9.7 ± 5.3
walker2d	med-expert	11.3	91.0 ± 10.8	98.7	44.6 ± 12.9	70.2 ± 36.2

Table 1. Offline reinforcement learning results from Figure 4 in tabular form.