Decision Transformer: Reinforcement Learning via Sequence Modeling

Lili Chen[*,1], **Kevin Lu**[*,1], **Aravind Rajeswaran**[2], **Kimin Lee**[1],
Aditya Grover[2], **Michael Laskin**[1], **Pieter Abbeel**[1], **Aravind Srinivas**[†,1], **Igor Mordatch**[†,3]
[*]equal contribution [†]equal advising
[1]UC Berkeley [2]Facebook AI Research [3]Google Brain
{lilichen, kzl}@berkeley.edu

Abstract

We introduce a framework that abstracts Reinforcement Learning (RL) as a sequence modeling problem. This allows us to draw upon the simplicity and scalability of the Transformer architecture, and associated advances in language modeling such as GPT-x and BERT. In particular, we present Decision Transformer, an architecture that casts the problem of RL as conditional sequence modeling. Unlike prior approaches to RL that fit value functions or compute policy gradients, Decision Transformer simply outputs the optimal actions by leveraging a causally masked Transformer. By conditioning an autoregressive model on the desired return (reward), past states, and actions, our Decision Transformer model can generate future actions that achieve the desired return. Despite its simplicity, Decision Transformer matches or exceeds the performance of state-of-the-art model-free offline RL baselines on Atari, OpenAI Gym, and Key-to-Door tasks.

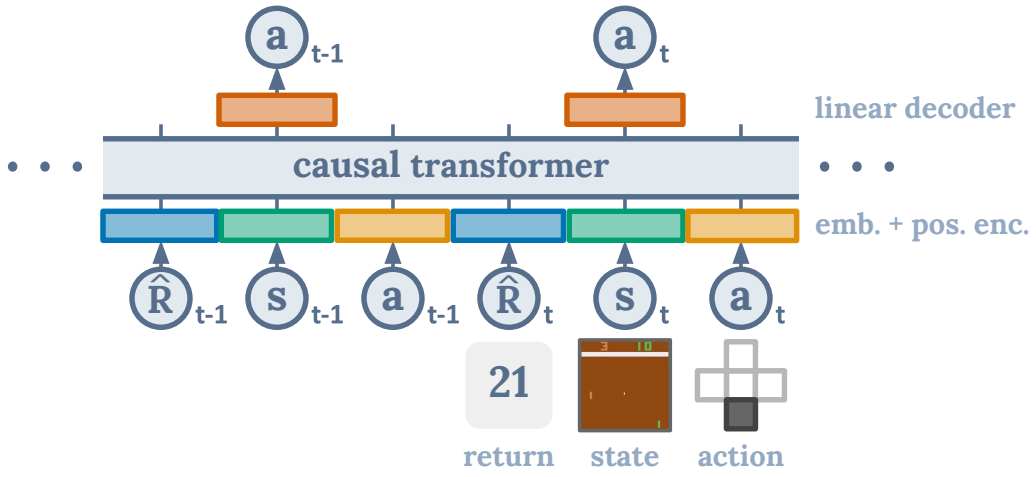

Figure 1: Decision Transformer architecture[1]. States, actions, and returns are fed into modality-specific linear embeddings and a positional episodic timestep encoding is added. Tokens are fed into a GPT architecture which predicts actions autoregressively using a causal self-attention mask.

[1]Our code is available at: https://sites.google.com/berkeley.edu/decision-transformer

Contents

1 Introduction

Recent work has shown transformers [1] can model high-dimensional distributions of semantic concepts at scale, including effective zero-shot generalization in language [2] and out-of-distribution image generation [3]. Given the diversity of successful applications of such models, we seek to examine their application to sequential decision making problems formalized as reinforcement learning (RL). In contrast to prior work using transformers as an architectural choice for components within traditional RL algorithms [4, 5], we seek to study if generative trajectory modeling – i.e. modeling the joint distribution of the sequence of states, actions, and rewards – can serve as a *replacement* for conventional RL algorithms.

We consider the following shift in paradigm: instead of training a policy through conventional RL algorithms like temporal difference (TD) learning [6], we will train transformer models on collected experience using a sequence modeling objective. This will allow us to bypass the need for bootstrapping for long term credit assignment – thereby avoiding one of the "deadly triad" [6] known to destabilize RL. It also avoids the need for discounting future rewards, as typically done in TD learning, which can induce undesirable short-sighted behaviors. Additionally, we can make use of existing transformer frameworks widely used in language and vision that are easy to scale, utilizing a large body of work studying stable training of transformer models.

In addition to their demonstrated ability to model long sequences, transformers also have other advantages. Transformers can perform credit assignment directly via self-attention, in contrast to Bellman backups which slowly propagate rewards and are prone to "distractor" signals [7]. This can enable transformers to still work effectively in the presence of sparse or distracting rewards. Finally, empirical evidence suggest that a transformer modeling approach can model a wide distribution of behaviors, enabling better generalization and transfer [3].

We explore our hypothesis by considering *offline RL*, where we will task agents with learning policies from suboptimal data – producing maximally effective behavior from fixed, limited experience. This task is traditionally challenging due to error propagation and value overestimation [8]. However, it is a natural task when training with a sequence modeling objective. By training an autoregressive model on sequences of states, actions, and returns, we reduce policy sampling to autoregressive generative modeling. We can specify the expertise of the policy – which "skill" to query – by selecting the desired return tokens, acting as a prompt for generation.

Illustrative example. To get an intuition for our proposal, consider the task of finding the shortest path on a directed graph, which can be posed as an RL problem. The reward is 0 when the agent is at the goal node and -1 otherwise. We train a GPT [9] model to predict next token in a sequence of returns-to-go (sum of future rewards), states, and actions. Training only on random walk data – with no expert demonstrations – we can generate *optimal* trajectories at test time by adding a prior to generate highest possible returns (see more details and empirical results in the Appendix) and subsequently generate the corresponding sequence of actions via conditioning. Thus, by combining the tools of sequence modeling with hindsight return information, we achieve policy improvement without the need for dynamic programming.

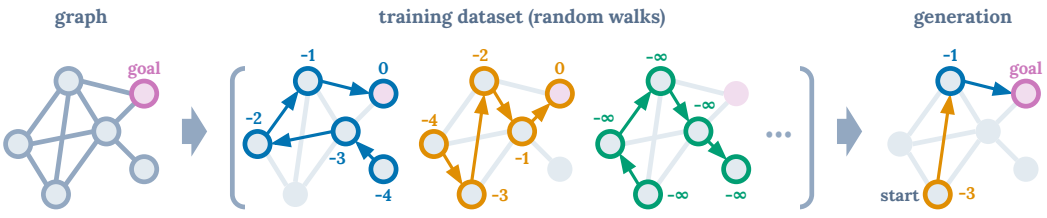

Figure 2: Illustrative example of finding shortest path for a fixed graph (left) posed as reinforcement learning. Training dataset consists of random walk trajectories and their per-node returns-to-go (middle). Conditioned on a starting state and generating largest possible return at each node, Decision Transformer sequences optimal paths.

Motivated by this observation, we propose Decision Transformer, where we use the GPT architecture to autoregressively model trajectories (shown in Figure 1). We study whether sequence modeling can perform policy optimization by evaluating Decision Transformer on offline RL benchmarks in Atari [10], OpenAI Gym [11], and Key-to-Door [12] environments. We show that – *without using dynamic programming* – Decision Transformer matches or exceeds the performance of state-of-the-art model-free offline RL algorithms [13, 14]. Furthermore, in tasks where long-term credit assignment is required, Decision Transformer capably outperforms the RL baselines. With this work, we aim to bridge sequence modeling and transformers with RL, and hope that sequence modeling serves as a strong algorithmic paradigm for RL.

2 Preliminaries

2.1 Offline reinforcement learning

We consider learning in a Markov decision process (MDP) described by the tuple $(\mathcal{S}, \mathcal{A}, P, \mathcal{R})$. The MDP tuple consists of states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, transition dynamics $P(s'|s,a)$, and a reward function $r = \mathcal{R}(s,a)$. We use s_t, a_t, and $r_t = \mathcal{R}(s_t, a_t)$ to denote the state, action, and reward at timestep t, respectively. A trajectory is made up of a sequence of states, actions, and rewards: $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T)$. The return of a trajectory at timestep t, $R_t = \sum_{t'=t}^{T} r_{t'}$, is the sum of future rewards from that timestep. The goal in reinforcement learning is to learn a policy which maximizes the expected return $\mathbb{E}\left[\sum_{t=1}^{T} r_t\right]$ in an MDP. In offline reinforcement learning, instead of obtaining data via environment interactions, we only have access to some fixed limited dataset consisting of trajectory rollouts of arbitrary policies. This setting is harder as it removes the ability for agents to explore the environment and collect additional feedback.

2.2 Transformers

Transformers were proposed by Vaswani et al. [1] as an architecture to efficiently model sequential data. These models consist of stacked self-attention layers with residual connections. Each self-attention layer receives n embeddings $\{x_i\}_{i=1}^{n}$ corresponding to unique input tokens, and outputs n embeddings $\{z_i\}_{i=1}^{n}$, preserving the input dimensions. The i-th token is mapped via linear transformations to a key k_i, query q_i, and value v_i. The i-th output of the self-attention layer is given by weighting the values v_j by the normalized dot product between the query q_i and other keys k_j:

$$z_i = \sum_{j=1}^{n} \texttt{softmax}(\{\langle q_i, k_{j'} \rangle\}_{j'=1}^{n})_j \cdot v_j. \tag{1}$$

As we shall see later, this allows the layer to assign "credit" by implicitly forming state-return associations via similarity of the query and key vectors (maximizing the dot product). In this work, we use the GPT architecture [9], which modifies the transformer architecture with a causal self-attention mask to enable autoregressive generation, replacing the summation/softmax over the n tokens with only the previous tokens in the sequence ($j \in [1, i]$). We defer the other architecture details to the original papers.

3 Method

In this section, we present Decision Transformer, which models trajectories autoregressively with minimal modification to the transformer architecture, as summarized in Figure 1 and Algorithm 1.

Trajectory representation. The key desiderata in our choice of trajectory representation are that it should enable transformers to learn meaningful patterns and we should be able to conditionally generate actions at test time. It is nontrivial to model rewards since we would like the model to generate actions based on *future* desired returns, rather than past rewards. As a result, instead of feeding the rewards directly, we feed the model with the returns-to-go $\widehat{R}_t = \sum_{t'=t}^{T} r_{t'}$. This leads to the following trajectory representation which is amenable to autoregressive training and generation:

$$\tau = \left(\widehat{R}_1, s_1, a_1, \widehat{R}_2, s_2, a_2, \ldots, \widehat{R}_T, s_T, a_T\right). \tag{2}$$

At test time, we can specify the desired performance (e.g. 1 for success or 0 for failure), as well as the environment starting state, as the conditioning information to initiate generation. After executing the generated action for the current state, we decrement the target return by the achieved reward and repeat until episode termination.

Architecture. We feed the last K timesteps into Decision Transformer, for a total of $3K$ tokens (one for each modality: return-to-go, state, or action). To obtain token embeddings, we learn a linear layer for each modality, which projects raw inputs to the embedding dimension, followed by layer normalization [15]. For environments with visual inputs, the state is fed into a convolutional encoder instead of a linear layer. Additionally, an embedding for each timestep is learned and added to each token – note this is different than the standard positional embedding used by transformers, as one timestep corresponds to three tokens. The tokens are then processed by a GPT [9] model, which predicts future action tokens via autoregressive modeling.

Training. We are given a dataset of offline trajectories. We sample minibatches of sequence length K from the dataset. The prediction head corresponding to the input token s_t is trained to predict a_t – either with cross-entropy loss for discrete actions or mean-squared error for continuous actions – and the losses for each timestep are averaged. We did not find predicting the states or returns-to-go to improve performance, although it is easily permissible within our framework (as shown in Section 5.4) and would be an interesting study for future work.

Algorithm 1 Decision Transformer Pseudocode (for continuous actions)

```
# R, s, a, t: returns-to-go, states, actions, or timesteps
# transformer: transformer with causal masking (GPT)
# embed_s, embed_a, embed_R: linear embedding layers
# embed_t: learned episode positional embedding
# pred_a: linear action prediction layer

# main model
def DecisionTransformer(R, s, a, t):
    # compute embeddings for tokens
    pos_embedding = embed_t(t)  # per-timestep (note: not per-token)
    s_embedding = embed_s(s) + pos_embedding
    a_embedding = embed_a(a) + pos_embedding
    R_embedding = embed_R(R) + pos_embedding

    # interleave tokens as (R_1, s_1, a_1, ..., R_K, s_K)
    input_embeds = stack(R_embedding, s_embedding, a_embedding)

    # use transformer to get hidden states
    hidden_states = transformer(input_embeds=input_embeds)

    # select hidden states for action prediction tokens
    a_hidden = unstack(hidden_states).actions

    # predict action
    return pred_a(a_hidden)

# training loop
for (R, s, a, t) in dataloader:  # dims: (batch_size, K, dim)
    a_preds = DecisionTransformer(R, s, a, t)
    loss = mean((a_preds - a)**2)  # L2 loss for continuous actions
    optimizer.zero_grad(); loss.backward(); optimizer.step()

# evaluation loop
target_return = 1  # for instance, expert-level return
R, s, a, t, done = [target_return], [env.reset()], [], [1], False
while not done:  # autoregressive generation/sampling
    # sample next action
    action = DecisionTransformer(R, s, a, t)[-1]  # for cts actions
    new_s, r, done, _ = env.step(action)

    # append new tokens to sequence
    R = R + [R[-1] - r]  # decrement returns-to-go with reward
    s, a, t = s + [new_s], a + [action], t + [len(R)]
    R, s, a, t = R[-K:], ...  # only keep context length of K
```

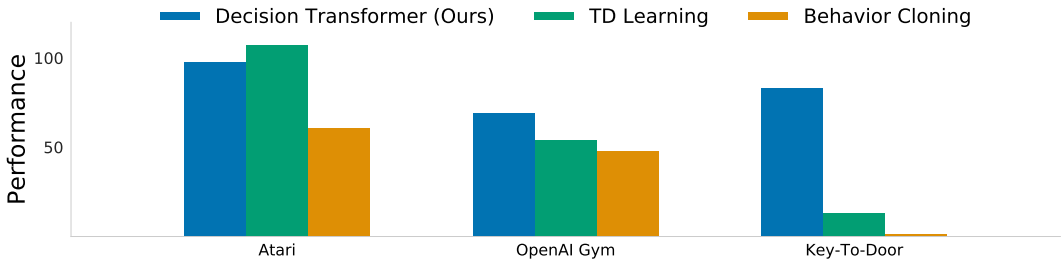

Figure 3: Results comparing Decision Transformer (ours) to TD learning (CQL) and behavior cloning across Atari, OpenAI Gym, and Minigrid. On a diverse set of tasks, Decision Transformer performs comparably or better than traditional approaches. Performance is measured by normalized episode return (see text for details).

4 Evaluations on Offline RL Benchmarks

In this section, we investigate the performance of Decision Transformer relative to dedicated offline RL and imitation learning algorithms. In particular, our primary points of comparison are model-free offline RL algorithms based on TD-learning, since our Decision Transformer architecture is fundamentally model-free in nature as well. Furthermore, TD-learning is the dominant paradigm in RL for sample efficiency, and also features prominently as a sub-routine in many model-based RL algorithms [16, 17]. We also compare with behavior cloning and variants, since it also involves a likelihood based policy learning formulation similar to ours. The exact algorithms depend on the environment but our motivations are as follows:

- **TD learning**: most of these methods use an action-space constraint or value pessimism, and will be the most faithful comparison to Decision Transformer, representing standard RL methods. A state-of-the-art model-free method is Conservative Q-Learning (CQL) [14] which serves as our primary comparison. In addition, we also compare against other prior model-free RL algorithms like BEAR [18] and BRAC [19].
- **Imitation learning**: this regime similarly uses supervised losses for training, rather than Bellman backups. We use behavior cloning here, and include a more detailed discussion in Section 5.1.

We evaluate on both discrete (Atari [10]) and continuous (OpenAI Gym [11]) control tasks. The former involves high-dimensional observation spaces and requires long-term credit assignment, while the latter requires fine-grained continuous control, representing a diverse set of tasks. Our main results are summarized in Figure 3, where we show averaged normalized performance for each domain.

4.1 Atari

The Atari benchmark [10] is challenging due to its high-dimensional visual inputs and difficulty of credit assignment arising from the delay between actions and resulting rewards. We evaluate our method on 1% of all samples in the DQN-replay dataset as per Agarwal et al. [13], representing 500 thousand of the 50 million transitions observed by an online DQN agent [20] during training; we report the mean and standard deviation of 3 seeds. We normalize scores based on a professional gamer, following the protocol of Hafner et al. [21], where 100 represents the professional gamer score and 0 represents a random policy.

We compare to CQL [14], REM [13], and QR-DQN [22] on four Atari tasks (Breakout, Qbert, Pong, and Seaquest) that are evaluated in Agarwal et al. [13]. We use context lengths of $K = 30$ for Decision Transformer (except $K = 50$ for Pong). We also report the performance of behavior cloning (BC), which utilizes the same network architecture and hyperparameters as Decision Transformer but does not have return-to-go conditioning[2]. For CQL, REM, and QR-DQN baselines, we report numbers directly from the CQL and REM papers. We show results in Table 1. Our method is competitive with CQL in 3 out of 4 games and outperforms or matches REM, QR-DQN, and BC on all 4 games.

[2]We also tried using an MLP with $K = 1$ as in prior work, but found this was worse than the transformer.

Game	DT (Ours)	CQL	QR-DQN	REM	BC
Breakout	$\mathbf{267.5 \pm 97.5}$	211.1	17.1	8.9	138.9 ± 61.7
Qbert	15.4 ± 11.4	$\mathbf{104.2}$	0.0	0.0	17.3 ± 14.7
Pong	106.1 ± 8.1	$\mathbf{111.9}$	18.0	0.5	85.2 ± 20.0
Seaquest	$\mathbf{2.5 \pm 0.4}$	1.7	0.4	0.7	2.1 ± 0.3

Table 1: Gamer-normalized scores for the 1% DQN-replay Atari dataset. We report the mean and variance across 3 seeds. Best mean scores are highlighted in bold. Decision Transformer (DT) performs comparably to CQL on 3 out of 4 games, and outperforms other baselines in most games.

4.2 OpenAI Gym

In this section, we consider the continuous control tasks from the D4RL benchmark [23]. We also consider a 2D reacher environment that is not part of the benchmark, and generate the datasets using a similar methodology to the D4RL benchmark. Reacher is a goal-conditioned task and has sparse rewards, so it represents a different setting than the standard locomotion environments (HalfCheetah, Hopper, and Walker). The different dataset settings are described below.

1. Medium: 1 million timesteps generated by a "medium" policy that achieves approximately one-third the score of an expert policy.

2. Medium-Replay: the replay buffer of an agent trained to the performance of a medium policy (approximately 25k-400k timesteps in our environments).

3. Medium-Expert: 1 million timesteps generated by the medium policy concatenated with 1 million timesteps generated by an expert policy.

We compare to CQL [14], BEAR [18], BRAC [19], and AWR [24]. CQL represents the state-of-the-art in model-free offline RL, an instantiation of TD learning with value pessimism. Score are normalized so that 100 represents an expert policy, as per Fu et al. [23]. CQL numbers are reported from the original paper; BC numbers are run by us; and the other methods are reported from the D4RL paper. Our results are shown in Table 2. Decision Transformer achieves the highest scores in a majority of the tasks and is competitive with the state of the art in the remaining tasks.

Dataset	Environment	DT (Ours)	CQL	BEAR	BRAC-v	AWR	BC
Medium-Expert	HalfCheetah	$\mathbf{86.8 \pm 1.3}$	62.4	53.4	41.9	52.7	59.9
Medium-Expert	Hopper	107.6 ± 1.8	$\mathbf{111.0}$	96.3	0.8	27.1	79.6
Medium-Expert	Walker	$\mathbf{108.1 \pm 0.2}$	98.7	40.1	81.6	53.8	36.6
Medium-Expert	Reacher	$\mathbf{89.1 \pm 1.3}$	30.6	-	-	-	73.3
Medium	HalfCheetah	42.6 ± 0.1	44.4	41.7	**46.3**	37.4	43.1
Medium	Hopper	$\mathbf{67.6 \pm 1.0}$	58.0	52.1	31.1	35.9	63.9
Medium	Walker	74.0 ± 1.4	79.2	59.1	**81.1**	17.4	77.3
Medium	Reacher	$\mathbf{51.2 \pm 3.4}$	26.0	-	-	-	**48.9**
Medium-Replay	HalfCheetah	36.6 ± 0.8	46.2	38.6	**47.7**	40.3	4.3
Medium-Replay	Hopper	$\mathbf{82.7 \pm 7.0}$	48.6	33.7	0.6	28.4	27.6
Medium-Replay	Walker	$\mathbf{66.6 \pm 3.0}$	26.7	19.2	0.9	15.5	36.9
Medium-Replay	Reacher	$\mathbf{18.0 \pm 2.4}$	**19.0**	-	-	-	5.4
Average (Without Reacher)		**74.7**	63.9	48.2	36.9	34.3	46.4
Average (All Settings)		**69.2**	54.2	-	-	-	47.7

Table 2: Results for D4RL datasets[3]. We report the mean and variance for three seeds. Decision Transformer (DT) outperforms conventional RL algorithms on almost all tasks.

[3]Given that CQL is generally the strongest TD learning method, for Reacher we only run the CQL baseline.

5 Discussion

5.1 Does Decision Transformer perform behavior cloning on a subset of the data?

In this section, we seek to gain insight into whether Decision Transformer can be thought of as performing imitation learning on a subset of the data with a certain return. To investigate this, we propose a new method, Percentile Behavior Cloning (%BC), where we run behavior cloning on only the top $X\%$ of timesteps in the dataset, ordered by episode returns. The percentile $X\%$ interpolates between standard BC ($X = 100\%$) that trains on the entire dataset and only cloning the best observed trajectory ($X \to 0\%$), trading off between better generalization by training on more data with training a specialized model that focuses on a desirable subset of the data.

We show full results comparing %BC to Decision Transformer and CQL in Table 3, sweeping over $X \in [10\%, 25\%, 40\%, 100\%]$. Note that the only way to choose the optimal subset for cloning is to evaluate using rollouts from the environment, so %BC is not a realistic approach; rather, it serves to provide insight into the behavior of Decision Transformer. When data is plentiful – as in the D4RL regime – we find %BC can match or beat other offline RL methods. On most environments, Decision Transformer is competitive with the performance of the best %BC, indicating it can hone in on a particular subset after training on the entire dataset distribution.

Dataset	Environment	DT (Ours)	10%BC	25%BC	40%BC	100%BC	CQL
Medium	HalfCheetah	42.6 ± 0.1	42.9	43.0	43.1	43.1	**44.4**
Medium	Hopper	$\mathbf{67.6 \pm 1.0}$	65.9	65.2	65.3	63.9	58.0
Medium	Walker	74.0 ± 1.4	78.8	**80.9**	78.8	77.3	79.2
Medium	Reacher	51.2 ± 3.4	51.0	48.9	58.2	**58.4**	26.0
Medium-Replay	HalfCheetah	36.6 ± 0.8	40.8	40.9	41.1	4.3	**46.2**
Medium-Replay	Hopper	$\mathbf{82.7 \pm 7.0}$	70.6	58.6	31.0	27.6	48.6
Medium-Replay	Walker	66.6 ± 3.0	**70.4**	67.8	67.2	36.9	26.7
Medium-Replay	Reacher	18.0 ± 2.4	**33.1**	16.2	10.7	5.4	19.0
Average		56.1	**56.7**	52.7	49.4	39.5	43.5

Table 3: Comparison between Decision Transformer (DT) and Percentile Behavior Cloning (%BC).

In contrast, when we study low data regimes – such as Atari, where we use 1% of a replay buffer as the dataset – %BC is weak (shown in Table 4). This suggests that in scenarios with relatively low amounts of data, Decision Transformer can outperform %BC by using all trajectories in the dataset to improve generalization, even if those trajectories are dissimilar from the return conditioning target. Our results indicate that Decision Transformer can be more effective than simply performing imitation learning on a subset of the dataset. On the tasks we considered, Decision Transformer either outperforms or is competitive to %BC, without the confound of having to select the optimal subset.

Game	DT (Ours)	10%BC	25%BC	40%BC	100%BC
Breakout	$\mathbf{267.5 \pm 97.5}$	28.5 ± 8.2	73.5 ± 6.4	108.2 ± 67.5	138.9 ± 61.7
Qbert	15.4 ± 11.4	6.6 ± 1.7	16.0 ± 13.8	11.8 ± 5.8	$\mathbf{17.3 \pm 14.7}$
Pong	$\mathbf{106.1 \pm 8.1}$	2.5 ± 0.2	13.3 ± 2.7	72.7 ± 13.3	85.2 ± 20.0
Seaquest	$\mathbf{2.5 \pm 0.4}$	1.1 ± 0.2	1.1 ± 0.2	1.6 ± 0.4	2.1 ± 0.3

Table 4: %BC scores for Atari. We report the mean and variance across 3 seeds. Decision Transformer (DT) outperforms all versions of %BC in most games.

5.2 How well does Decision Transformer model the distribution of returns?

We evaluate the ability of Decision Transformer to understand return-to-go tokens by varying the desired target return over a wide range – evaluating the multi-task distribution modeling capability of transformers. Figure 4 shows the average sampled return accumulated by the agent over the course of the evaluation episode for varying values of target return. On every task, the desired target returns and the true observed returns are highly correlated. On some tasks like Pong, HalfCheetah and Walker, Decision Transformer generates trajectories that almost perfectly match the desired returns

(as indicated by the overlap with the oracle line). Furthermore, on some Atari tasks like Seaquest, we can prompt the Decision Transformer with higher returns than the maximum episode return available in the dataset, demonstrating that Decision Transformer is sometimes capable of extrapolation.

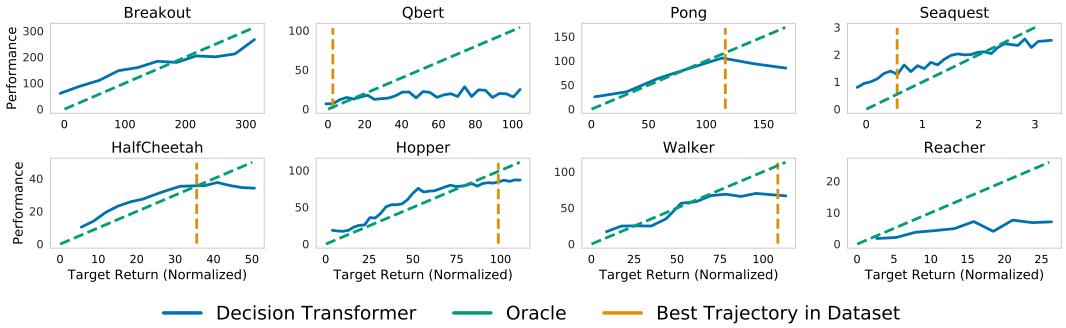

Figure 4: Sampled (evaluation) returns accumulated by Decision Transformer when conditioned on the specified target (desired) returns. **Top:** Atari. **Bottom:** D4RL medium-replay datasets.

5.3 What is the benefit of using a longer context length?

To assess the importance of access to previous states, actions, and returns, we ablate on the context length K. This is interesting since it is generally considered that the previous state (i.e. $K = 1$) is enough for reinforcement learning algorithms when frame stacking is used, as we do. Table 5 shows that performance of Decision Transformer is significantly worse when $K = 1$, indicating that past information is useful for Atari games. One hypothesis is that when we are representing a distribution of policies – like with sequence modeling – the context allows the transformer to identify which policy generated the actions, enabling better learning and/or improving the training dynamics.

Game	DT (Ours)	DT with no context ($K = 1$)
Breakout	$\mathbf{267.5 \pm 97.5}$	73.9 ± 10
Qbert	$\mathbf{15.1 \pm 11.4}$	13.6 ± 11.3
Pong	$\mathbf{106.1 \pm 8.1}$	2.5 ± 0.2
Seaquest	$\mathbf{2.5 \pm 0.4}$	0.6 ± 0.1

Table 5: Ablation on context length. Decision Transformer (DT) performs better when using a longer context length ($K = 50$ for Pong, $K = 30$ for others).

5.4 Does Decision Transformer perform effective long-term credit assignment?

To evaluate long-term credit assignment capabilities of our model, we consider a variant of the Key-to-Door environment proposed in Mesnard et al. [12]. This is a grid-based environment with a sequence of three phases: (1) in the first phase, the agent is placed in a room with a key; (2) then, the agent is placed in an empty room; (3) and finally, the agent is placed in a room with a door. The agent receives a binary reward when reaching the door in the third phase, but **only** if it picked up the key in the first phase. This problem is difficult for credit assignment because credit must be propagated from the beginning to the end of the episode, skipping over actions taken in the middle.

We train on datasets of trajectories generated by applying random actions and report success rates in Table 6. Furthermore, for the Key-to-Door environment we use the entire episode length as the context, rather than having a fixed content window as in the other environments. Methods that use highsight return information: our Decision Transformer model and %BC (trained only on successful episodes) are able to learn effective policies – producing near-optimal paths, despite only training on random walks. TD learning (CQL) cannot effectively propagate Q-values over the long horizons involved and gets poor performance.

Dataset	DT (Ours)	CQL	BC	%BC	Random
1K Random Trajectories	**71.8**%	13.1%	1.4%	69.9%	3.1%
10K Random Trajectories	94.6%	13.3%	1.6%	**95.1**%	3.1%

Table 6: Success rate for Key-to-Door environment. Methods using hindsight (Decision Transformer, %BC) can learn successful policies, while TD learning struggles to perform credit assignment.

5.5 Can transformers be accurate critics in sparse reward settings?

In previous sections, we established that decision transformer can produce effective policies (actors). We now evaluate whether transformer models can also be effective critics. We modify Decision Transformer to output return tokens in addition to action tokens on the Key-to-Door environment. Additionally, the first return token is not given, but it is predicted instead (i.e. the model learns the initial distribution $p(\hat{R}_1)$), similar to standard autoregressive generative models. We find that the transformer continuously updates reward probability based on events during the episode, shown in Figure 5 (Left). Furthermore, we find the transformer attends to critical events in the episode (picking up the key or reaching the door), shown in Figure 5 (Right), indicating formation of state-reward associations as discussed in Raposo et al. [25] and enabling accurate value prediction.

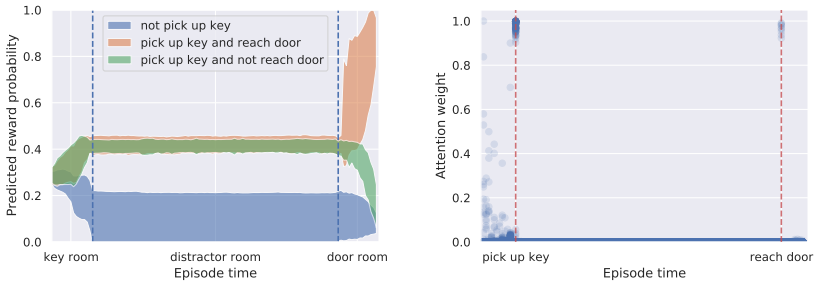

Figure 5: **Left:** Averages of running return probabilities predicted by the transformer model for three types of episode outcomes. **Right:** Transformer attention weights from all timesteps superimposed for a particular successful episode. The model attends to steps near pivotal events in the episode, such as picking up the key and reaching the door.

5.6 Does Decision Transformer perform well in sparse reward settings?

A known weakness of TD learning algorithms is that they require densely populated rewards in order to perform well, which can be unrealistic and/or expensive. In contrast, Decision Transformer can improve robustness in these settings since it makes minimal assumptions on the density of the reward. To evaluate this, we consider a delayed return version of the D4RL benchmarks where the agent does not receive any rewards along the trajectory, and instead receives the cumulative reward of the trajectory in the final timestep. Our results for delayed returns are shown in Table 7. Delayed returns minimally affect Decision Transformer; and due to the nature of the training process, while imitation learning methods are reward agnostic. While TD learning collapses, Decision Transformer and %BC still perform well, indicating that Decision Transformer can be more robust to delayed rewards.

		Delayed (Sparse)		Agnostic		Original (Dense)	
Dataset	**Environment**	**DT (Ours)**	**CQL**	**BC**	**%BC**	**DT (Ours)**	**CQL**
Medium-Expert	Hopper	$\mathbf{107.3 \pm 3.5}$	9.0	59.9	102.6	107.6	111.0
Medium	Hopper	60.7 ± 4.5	5.2	63.9	**65.9**	67.6	58.0
Medium-Replay	Hopper	$\mathbf{78.5 \pm 3.7}$	2.0	27.6	70.6	82.7	48.6

Table 7: Results for D4RL datasets with delayed (sparse) reward. Decision Transformer (DT) and imitation learning are minimally affected by the removal of dense rewards, while CQL fails.

5.7 Why does Decision Transformer avoid the need for value pessimism or behavior regularization?

One key difference between Decision Transformer and prior offline RL algorithms is that we do not require policy regularization or conservatism to achieve good performance. Our conjecture is that TD-learning based algorithms learn an approximate value function and improve the policy by optimizing this value function. This act of optimizing a learned function can exacerbate and exploit any inaccuracies in the value function approximation, causing failures in policy improvement. Since Decision Transformer does not require explicit optimization using learned functions as objectives, it avoids the need for regularization or conservatism.

5.8 How can Decision Transformer benefit online RL regimes?

Offline RL and the ability to model behaviors has the potential to enable sample-efficient online RL for downstream tasks. Works studying the transition from offline to online generally find that likelihood-based approaches, like our sequence modeling objective, are more successful [26, 27]. As a result, although we studied offline RL in this work, we believe Decision Transformer can meaningfully improve online RL methods by serving as a strong model for behavior generation. For instance, Decision Transformer can serve as a powerful "memorization engine" and in conjunction with powerful exploration algorithms like Go-Explore [28], has the potential to simultaneously model and generative a diverse set of behaviors.

6 Related Work

6.1 Offline reinforcement learning

To mitigate the impact of distribution shift in offline RL, prior algorithms either (a) constrain the policy action space [29, 30, 31] or (b) incorporate value pessimism [29, 14], or (c) incorporate pessimism into learned dynamics models [32, 33]. Since we do not use Decision Transformers to explicitly learn the dynamics model, we primarily compare against model-free algorithms in our work; in particular, adding a dynamics model tends to improve the performance of model-free algorithms. Another line of work explores learning wide behavior distribution from an offline dataset by learning a task-agnostic set of skills, either with likelihood-based approaches [34, 35, 36, 37] or by maximizing mutual information [38, 39, 40]. Our work is similar to the likelihood-based approaches, which do not use iterative Bellman updates – although we use a simpler sequence modeling objective instead of a variational method, and use rewards for conditional generation of behaviors.

6.2 Supervised learning in reinforcement learning settings

Some prior methods for reinforcement learning bear more resemblance to static supervised learning, such as Q-learning [41, 42], which still uses iterative backups, or likelihood-based methods such as behavior cloning, which do not (discussed in previous section). Recent work [43, 44, 45] studies "upside-down" reinforcement learning (UDRL), which are similar to our method in seeking to model behaviors with a supervised loss conditioned on the target return. A key difference in our work is the shift of motivation to sequence modeling rather than supervised learning: while the practical methods differ primarily in the context length and architecture, sequence modeling enables behavior modeling even without access to the reward, in a similar style to language [9] or images [46], and is known to scale well [2]. The method proposed by Kumar et al. [44] is most similar to our method with $K = 1$, which we find sequence modeling/long contexts to outperform (see Section 5.3). Ghosh et al. [47] extends prior UDRL methods to use state goal conditioning, rather than rewards, and Paster et al. [48] further use an LSTM with state goal conditioning for goal-conditoned online RL settings.

Concurrent to our work, Janner et al. [49] propose Trajectory Transformer, which is similar to Decision Transformer but additionally uses state and return prediction, as well as discretization, which incorporates model-based components. We believe that their experiments, in addition to our results, highlight the potential for sequence modeling to be a generally applicable idea for reinforcement learning.

6.3 Credit assignment

Many works have studied better credit assignment via state-association, learning an architecture which decomposes the reward function such that certain "important" states comprise most of the credit [50, 51, 12]. They use the learned reward function to change the reward of an actor-critic algorithm to help propagate signal over long horizons. In particular, similar to our long-term setting, some works have specifically shown such state-associative architectures can perform better in delayed reward settings [52, 7, 53, 25]. In contrast, we allow these properties to naturally emerge in a transformer architecture, without having to explicitly learn a reward function or a critic.

6.4 Conditional language generation

Various works have studied guided generation for images [54] and language [55, 56]. Several works [57, 58, 59, 60, 61, 62] have explored training or fine-tuning of models for controllable text generation. Class-conditional language models can also be used to learn disciminators to guide generation [63, 55, 64, 65]. However, these approaches mostly assume constant "classes", while in reinforcement learning the reward signal is time-varying. Furthermore, it is more natural to prompt the model desired target return and continuously decrease it by the observed rewards over time, since the transformer model and environment jointly generate the trajectory.

6.5 Attention and transformer models

Transformers [1] have been applied successfully to many tasks in natural language processing [66, 9] and computer vision [67, 68]. However, transformers are relatively unstudied in RL, mostly due to differing nature of the problem, such as higher variance in training. Zambaldi et al. [5] showed that augmenting transformers with relational reasoning improve performance in combinatorial environments and Ritter et al. [69] showed iterative self-attention allowed for RL agents to better utilize episodic memories. Parisotto et al. [4] discussed design decisions for more stable training of transformers in the high-variance RL setting. Unlike our work, these still use actor-critic algorithms for optimization, focusing on novelty in architecture. Additionally, in imitation learning, some works have studied transformers as a replacement for LSTMs: Dasari and Gupta [70] study one-shot imitation learning, and Abramson et al. [71] combine language and image modalities for text-conditioned behavior generation.

7 Conclusion

We proposed Decision Transformer, seeking to unify ideas in language/sequence modeling and reinforcement learning. On standard offline RL benchmarks, we showed Decision Transformer can match or outperform strong algorithms designed explicitly for offline RL with minimal modifications from standard language modeling architectures.

We hope this work inspires more investigation into using large transformer models for RL. We used a simple supervised loss that was effective in our experiments, but applications to large-scale datasets could benefit from self-supervised pretraining tasks. In addition, one could consider more sophisticated embeddings for returns, states, and actions – for instance, conditioning on return distributions to model stochastic settings instead of deterministic returns. Transformer models can also be used to model the state evolution of trajectory, potentially serving as an alternative to model-based RL, and we hope to explore this in future work.

For real-world applications, it is important to understand the types of errors transformers make in MDP settings and possible negative consequences, which are underexplored. It will also be important to consider the datasets we train models on, which can potentially add destructive biases, particularly as we consider studying augmenting RL agents with more data which may come from questionable sources. For instance, reward design by nefarious actors can potentially generate unintended behaviors as our model generates behaviors by conditioning on desired returns.

Acknowledgements

This research was supported by Berkeley Deep Drive, Open Philanthropy, and the National Science Foundation under NSF:NRI #2024675. Part of this work was completed when Aravind Rajeswaran was a PhD student at the University of Washington, where he was supported by the J.P. Morgan PhD Fellowship in AI (2020-21). We also thank Luke Metz and Daniel Freeman for valuable feedback and discussions, as well as Justin Fu for assistance in setting up D4RL benchmarks, and Aviral Kumar for assistance with the CQL baselines and hyperparameters.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[2] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[3] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. *arXiv preprint arXiv:2102.12092*, 2021.

[4] Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. In *International Conference on Machine Learning*, 2020.

[5] Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, et al. Deep reinforcement learning with relational inductive biases. In *International Conference on Learning Representations*, 2018.

[6] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT Press, 2018.

[7] Chia-Chun Hung, Timothy Lillicrap, Josh Abramson, Yan Wu, Mehdi Mirza, Federico Carnevale, Arun Ahuja, and Greg Wayne. Optimizing agent behavior over long time scales by transporting value. *Nature communications*, 10(1):1–12, 2019.

[8] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[9] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[10] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[11] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[12] Thomas Mesnard, Théophane Weber, Fabio Viola, Shantanu Thakoor, Alaa Saade, Anna Harutyunyan, Will Dabney, Tom Stepleton, Nicolas Heess, Arthur Guez, et al. Counterfactual credit assignment in model-free reinforcement learning. *arXiv preprint arXiv:2011.09464*, 2020.

[13] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, 2020.

[14] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

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

[16] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *ICML*, 1990.

[17] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, pages 12498–12509, 2019.

[18] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *arXiv preprint arXiv:1906.00949*, 2019.

[19] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

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

[21] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[22] Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Conference on Artificial Intelligence*, 2018.

[23] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[24] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

[25] David Raposo, Sam Ritter, Adam Santoro, Greg Wayne, Theophane Weber, Matt Botvinick, Hado van Hasselt, and Francis Song. Synthetic returns for long-term credit assignment. *arXiv preprint arXiv:2102.12425*, 2021.

[26] Yang Gao, Huazhe Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell. Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313*, 2018.

[27] Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[28] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

[29] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, 2019.

[30] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, 2019.

[31] Noah Y Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, and Martin Riedmiller. Keep doing what worked: Behavioral modelling priors for offline reinforcement learning. In *International Conference on Learning Representations*, 2020.

[32] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

[33] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. In *Advances in Neural Information Processing Systems*, 2020.

[34] Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. Opal: Offline primitive discovery for accelerating offline reinforcement learning. *arXiv preprint arXiv:2010.13611*, 2020.

[35] Víctor Campos, Alexander Trott, Caiming Xiong, Richard Socher, Xavier Giro-i Nieto, and Jordi Torres. Explore, discover and learn: Unsupervised discovery of state-covering skills. In *International Conference on Machine Learning*, 2020.

[36] Karl Pertsch, Youngwoon Lee, and Joseph J Lim. Accelerating reinforcement learning with learned skill priors. *arXiv preprint arXiv:2010.11944*, 2020.

[37] Avi Singh, Huihan Liu, Gaoyue Zhou, Albert Yu, Nicholas Rhinehart, and Sergey Levine. Parrot: Data-driven behavioral priors for reinforcement learning. In *International Conference on Learning Representations*, 2021.

[38] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019.

[39] Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. Reset-free lifelong learning with skill-space planning. *arXiv preprint arXiv:2012.03548*, 2020.

[40] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations*, 2020.

[41] Christopher Watkins. Learning from delayed rewards. 01 1989.

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

[43] Rupesh Kumar Srivastava, Pranav Shyam, Filipe Mutz, Wojciech Jaśkowski, and Jürgen Schmidhuber. Training agents using upside-down reinforcement learning. *arXiv preprint arXiv:1912.02877*, 2019.

[44] Aviral Kumar, Xue Bin Peng, and Sergey Levine. Reward-conditioned policies. *arXiv preprint arXiv:1912.13465*, 2019.

[45] Acting without rewards. 2019. URL `https://ogma.ai/2019/08/acting-without-rewards/`.

[46] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *International Conference on Machine Learning*, pages 1691–1703. PMLR, 2020.

[47] Dibya Ghosh, Abhishek Gupta, Justin Fu, Ashwin Reddy, Coline Devin, Benjamin Eysenbach, and Sergey Levine. Learning to reach goals without reinforcement learning. *arXiv preprint arXiv:1912.06088*, 2019.

[48] Keiran Paster, Sheila A McIlraith, and Jimmy Ba. Planning from pixels using inverse dynamics models. *arXiv preprint arXiv:2012.02419*, 2020.

[49] Michael Janner, Qiyang Li, and Sergey Levine. Reinforcement learning as one big sequence modeling problem. *arXiv preprint arXiv:2106.02039*, 2021.

[50] Johan Ferret, Raphaël Marinier, Matthieu Geist, and Olivier Pietquin. Self-attentional credit assignment for transfer in reinforcement learning. *arXiv preprint arXiv:1907.08027*, 2019.

[51] Anna Harutyunyan, Will Dabney, Thomas Mesnard, Mohammad Azar, Bilal Piot, Nicolas Heess, Hado van Hasselt, Greg Wayne, Satinder Singh, Doina Precup, et al. Hindsight credit assignment. *arXiv preprint arXiv:1912.02503*, 2019.

[52] Jose A Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. Rudder: Return decomposition for delayed rewards. *arXiv preprint arXiv:1806.07857*, 2018.

[53] Yang Liu, Yunan Luo, Yuanyi Zhong, Xi Chen, Qiang Liu, and Jian Peng. Sequence modeling of temporal credit assignment for episodic reinforcement learning. *arXiv preprint arXiv:1905.13420*, 2019.

[54] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Conference on Computer Vision and Pattern Recognition*, 2019.

[55] Marjan Ghazvininejad, Xing Shi, Jay Priyadarshi, and Kevin Knight. Hafez: an interactive poetry generation system. In *Proceedings of ACL, System Demonstrations*, 2017.

[56] Lilian Weng. Controllable neural text generation. *lilianweng.github.io/lil-log*, 2021. URL `https://lilianweng.github.io/lil-log/2021/01/02/controllable-neural-text-generation.html`.

[57] Jessica Ficler and Yoav Goldberg. Controlling linguistic style aspects in neural language generation. *arXiv preprint arXiv:1707.02633*, 2017.

[58] Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P Xing. Toward controlled generation of text. In *International Conference on Machine Learning*, 2017.

[59] Nazneen Fatema Rajani, Bryan McCann, Caiming Xiong, and Richard Socher. Explain yourself! leveraging language models for commonsense reasoning. *arXiv preprint arXiv:1906.02361*, 2019.

[60] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI conference on artificial intelligence*, 2017.

[61] Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

[62] Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*, 2019.

[63] Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. Plug and play language models: A simple approach to controlled text generation. *arXiv preprint arXiv:1912.02164*, 2019.

[64] Ari Holtzman, Jan Buys, Maxwell Forbes, Antoine Bosselut, David Golub, and Yejin Choi. Learning to write with cooperative discriminators. *arXiv preprint arXiv:1805.06087*, 2018.

[65] Ben Krause, Akhilesh Deepak Gotmare, Bryan McCann, Nitish Shirish Keskar, Shafiq Joty, Richard Socher, and Nazneen Fatema Rajani. Gedi: Generative discriminator guided sequence generation. *arXiv preprint arXiv:2009.06367*, 2020.

[66] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[67] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, 2020.

[68] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[69] Sam Ritter, Ryan Faulkner, Laurent Sartran, Adam Santoro, Matt Botvinick, and David Raposo. Rapid task-solving in novel environments. *arXiv preprint arXiv:2006.03662*, 2020.

[70] Sudeep Dasari and Abhinav Gupta. Transformers for one-shot visual imitation. *arXiv preprint arXiv:2011.05970*, 2020.

[71] Josh Abramson, Arun Ahuja, Iain Barr, Arthur Brussee, Federico Carnevale, Mary Cassin, Rachita Chhaparia, Stephen Clark, Bogdan Damoc, Andrew Dudzik, et al. Imitating interactive intelligence. *arXiv preprint arXiv:2012.05672*, 2020.

[72] Thomas Wolf, Julien Chaumond, Lysandre Debut, Victor Sanh, Clement Delangue, Anthony Moi, Pierric Cistac, Morgan Funtowicz, Joe Davison, Sam Shleifer, et al. Transformers: State-of-the-art natural language processing. In *Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2020.

[73] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

A Experimental Details

Code for experiments can be found in the supplementary material.

A.1 Atari

We build our Decision Transformer implementation for Atari games off of minGPT (`https://github.com/karpathy/minGPT`), a publicly available re-implementation of GPT. We use most of the default hyperparameters from their character-level GPT example (`https://github.com/karpathy/minGPT/blob/master/play_char.ipynb`). We reduce the batch size (except in Pong), block size, number of layers, attention heads, and embedding dimension for faster training. For processing the observations, we use the DQN encoder from Mnih et al. [20] with an additional linear layer to project to the embedding dimension.

For return-to-go conditioning, we use either $1\times$ or $5\times$ the maximum return in the dataset, but more possibilities exist for principled return-to-go conditioning. In Atari experiments, we use Tanh instead of LayerNorm (as described in Section 3) after embedding each modality, but did this does not make a significant difference in performance. The full list of hyperparameters can be found in Table 8.

Table 8: Hyperparameters of DT (and %BC) for Atari experiments.

Hyperparameter	**Value**
Number of layers	6
Number of attention heads	8
Embedding dimension	128
Batch size	512 Pong 128 Breakout, Qbert, Seaquest
Context length K	50 Pong 30 Breakout, Qbert, Seaquest
Return-to-go conditioning	90 Breakout ($\approx 1\times$ max in dataset) 2500 Qbert ($\approx 5\times$ max in dataset) 20 Pong ($\approx 1\times$ max in dataset) 1450 Seaquest ($\approx 5\times$ max in dataset)
Nonlinearity	ReLU, encoder GeLU, otherwise
Encoder channels	$32, 64, 64$
Encoder filter sizes	$8 \times 8, 4 \times 4, 3 \times 3$
Encoder strides	$4, 2, 1$
Max epochs	5
Dropout	0.1
Learning rate	$6 * 10^{-4}$
Adam betas	$(0.9, 0.95)$
Grad norm clip	1.0
Weight decay	0.1
Learning rate decay	Linear warmup and cosine decay (see code for details)
Warmup tokens	$512 * 20$
Final tokens	$2 * 500000 * K$

A.2 OpenAI Gym

A.2.1 Decision Transformer

Our code is based on the Huggingface Transformers library [72]. Our hyperparameters on all OpenAI Gym tasks are shown below in Table 9. Heuristically, we find using larger models helps to model the distribution of returns, compared to standard RL model sizes (which learn one policy). For reacher we use a smaller context length than the other environments, which we find to be helpful as the environment is goal-conditioned and the episodes are shorter. We choose return targets based on expert performance for each environment, except for HalfCheetah where we find 50% performance to be better due to the datasets containing lower relative returns to the other environments. Models were trained for 10^5 gradient steps using the AdamW optimizer [73] following PyTorch defaults.

Table 9: Hyperparameters of Decision Transformer for OpenAI Gym experiments.

Hyperparameter	**Value**
Number of layers	3
Number of attention heads	1
Embedding dimension	128
Nonlinearity function	ReLU
Batch size	64
Context length K	20 HalfCheetah, Hopper, Walker 5 Reacher
Return-to-go conditioning	6000 HalfCheetah 3600 Hopper 5000 Walker 50 Reacher
Dropout	0.1
Learning rate	10^{-4}
Grad norm clip	0.25
Weight decay	10^{-4}
Learning rate decay	Linear warmup for first 10^5 training steps

A.2.2 Behavior Cloning

As briefly mentioned in Section 4.2, we found previously reported behavior cloning baselines to be weak, and so run them ourselves using a similar setup as Decision Transformer. We tried using a transformer architecture, but found using an MLP (as in previous work) to be stronger. We train for 2.5×10^4 gradient steps; training more did not improve performance. Other hyperparameters are shown in Table 10. The percentile behavior cloning experiments use the same hyperparameters.

Table 10: Hyperparameters of Behavior Cloning for OpenAI Gym experiments.

Hyperparameter	**Value**
Number of layers	3
Embedding dimension	256
Nonlinearity function	ReLU
Batch size	64
Dropout	0.1
Learning rate	10^{-4}
Weight decay	10^{-4}
Learning rate decay	Linear warmup for first 10^5 training steps

A.3 Graph Shortest Path

We give details of the illustrative example discussed in the introduction. The task is to find the shortest path on a fixed directed graph, which can be formulated as an MDP where reward is 0 when the agent is at the goal node and -1 otherwise. The observation is the integer index of the graph node the agent is in. The action is the integer index of the graph node to move to next. The transition dynamics transport the agent to the action's node index if there is an edge in the graph, while the agent remains at the past node otherwise. The returns-to-go in this problem correspond to negative path lengths and maximizing them corresponds to generating shortest paths.

In this environment, we use the GPT model as described in Section 3 to generate both actions and return-to-go tokens. This makes it possible for the model it generate its own (realizable) returns-to-go $\hat{R}$. Since we require a return prompt to generate actions and we do assume knowledge of the optimal path length upfront, we use a simple prior over returns that favors shorter paths: $P_{\text{prior}}(\hat{R} = k) \propto T + 1 - k$, where T is the maximum trajectory length. Then, it is combined with the return probabilities generated by the GPT model: $P(\hat{R}_t | s_{0:t}, a_{0:t-1}, \hat{R}_{0:t-1}) =$

Figure 6: Histogram of steps to reach the goal node for random walks on the graph, shortest possible paths to the goal, and attempted shortest paths generated by the transformer model. ∞ indicates the goal was not reached during the trajectory.

$P_{\text{GPT}}(\hat{R}_t|s_{0:t}, a_{0:t-1}, \hat{R}_{0:t-1}) \times P_{\text{prior}}(\hat{R}_t)^{10}$. Note that the prior and return-to-go predictions are entirely computable by the model, and thus avoids the need for any external or oracle information like the optimal path length. Adjustment of generation by a prior has also been used for similar purposes in controllable text generation in prior work [65].

We train on a dataset of $1,000$ graph random walk trajectories of $T = 10$ steps each with a random graph of 20 nodes and edge sparsity coefficient of 0.1. We report the results in Figure 6, where we find that transformer model is able to significantly improve upon the number of steps required to reach the goal, closely matching performance of optimal paths.

There are two reasons for the favorable performance on this task. In one case, the training dataset of random walk trajectories may contain a segment that directly corresponds to the desired shortest path, in which case it will be generated by the model. In the second case, generated paths are entirely original and are not subsets of trajectories in the training dataset - they are generated from stitching sub-optimal segments. We find this case accounts for 15.8% of generated paths in the experiment.

While this is a simple example and uses a prior on generation that we do not use in other experiments for simplicity, it illustrates how hindsight return information can be used with generation priors to avoid the need for explicit dynamic programming.

B Atari Task Scores

Table 11 shows the normalized scores used for normalization used in Hafner et al. [21]. Tables 12 and 13 show the raw scores corresponding to Tables 1 and 4, respectively. For %BC scores, we use the same hyperparameters as Decision Transformer for fair comparison. For REM and QR-DQN, there is a slight discrepancy between Agarwal et al. [13] and Kumar et al. [14]; we report raw data provided to us by REM authors.

Game	Random	Gamer
Breakout	2	30
Qbert	164	13455
Pong	−21	15
Seaquest	68	42055

Table 11: Atari baseline scores used for normalization.

Game	DT (Ours)	CQL	QR-DQN	REM	BC
Breakout	$\mathbf{76.9 \pm 27.3}$	61.1	6.8	4.5	40.9 ± 17.3
Qbert	2215.8 ± 1523.7	$\mathbf{14012.0}$	156.0	160.1	2464.1 ± 1948.2
Pong	17.1 ± 2.9	$\mathbf{19.3}$	-14.5	-20.8	9.7 ± 7.2
Seaquest	$\mathbf{1129.3 \pm 189.0}$	779.4	250.1	370.5	968.6 ± 133.8

Table 12: Raw scores for the 1% DQN-replay Atari dataset. We report the mean and variance across 3 seeds. Best mean scores are highlighted in bold. Decision Transformer performs comparably to CQL on 3 out of 4 games, and usually outperforms other baselines.

Game	DT (Ours)	10%BC	25%BC	40%BC	100%BC
Breakout	$\mathbf{76.9 \pm 27.3}$	10.0 ± 2.3	22.6 ± 1.8	32.3 ± 18.9	40.9 ± 17.3
Qbert	2215.8 ± 1523.7	1045 ± 232.0	2302.5 ± 1844.1	1674.1 ± 776.0	$\mathbf{2464.1 \pm 1948.2}$
Pong	$\mathbf{17.1 \pm 2.9}$	-20.3 ± 0.1	-16.2 ± 1.0	5.2 ± 4.8	9.7 ± 7.2
Seaquest	$\mathbf{1129.3 \pm 189.0}$	521.3 ± 103.0	549.3 ± 96.2	758 ± 169.1	968.6 ± 133.8

Table 13: %BC scores for Atari. We report the mean and variance across 3 seeds. Decision Transformer usually outperforms %BC.