Fully Online Decision Transformer for Reinforcement Learning

Austin Anhkhoi Nguyen / ngaustin
Creighton Glasscock / creiglas
@umich.edu

Abstract

Devising deep reinforcement learning (RL) algorithms with better sample efficiency, stability, and applicability is a cornerstone research problem in machine learning. While many model-free approaches have proposed learning in policy space or value space directly, others have applied the Transformer neural network architecture to model RL as a sequence modeling problem. Previous approaches have advocated for the use of Transformers in offline settings, where a dataset is provided to the Transformer to fit. However, the application of the method directly to online settings, where data has to be gathered through the policy's own exploration, has been relatively unexplored. This approach poses the problem of exploration: how can an architecture that does not directly optimize rewards derive a strong policy? As such, in this work, we adapt the Decision Transformer (DT) architecture to fully online settings, where exploration is aided by an RL policy that is trained in parallel. We show that the combination of the DT and exploratory RL policy yields performance that matches and sometimes surpasses state-of-the-art RL algorithms. Also, we show that the standard deviation of returns in intermediate evaluation episodes is lower than that of typical RL policy training. We note several key attributes of the DT that allow it to achieve such performance.

1 Introduction

Recent work in the field of Natural Language Processing has centered around the Transformer (Vaswani et al., 2017), an attention-based architecture that has shown flexibility and utility in sequence modeling problems common within the field. This success has resulted in a flurry of further work on the Transformer, with new work providing many new architectures and hardware solutions designed to exploit the power of the Transformer. Historically, Transformers have shown few useful applications within the field of Reinforcement Learning, which contains problems less readily applicable to the sequence modeling problems in other fields which easily beget the use of the Transformer. This work proposes a method that incorporates the Transformer into online settings.

In this work, we adapt the existing Decision Transformer architecture, a recent development which, for the first time, successfully applies advancements in Transformer architectures to offline Reinforcement Learning (RL) problems. We aim to build upon the work behind Decision Transformer to bring the architecture's strengths to the problem of online Reinforcement Learning, which has an increased variety of real-world use cases.

2 Related Work

Recent work has introduced the Decision Transformer architecture (Chen et al.), which models offline Reinforcement Learning as a sequence modeling problem, thereby breaking from previous work in RL and utilizing a Transformer to guide the choice of optimal action in pursuit of the desired return. This strategy allows the architecture to draw upon the wealth of Transformer-related advances in the field of Natural Language Processing, such as the BERT architecture (Devlin et al., 2018). The Decision Transformer is made possible by the flexibility of the Transformer architecture and has seen great success in the problem of offline Reinforcement Learning due to the ability of the transformer to recognize relationships between states and returns.

The drawback of the Decision Transformer is that its implementation is constrained to offline learning only, which restricts its real-world use cases. In particular, by reducing training to a sequence modeling problem, the agent only has the ability to make decisions based on a pre-existing dataset. The agent can therefore effectively exploit its initial knowledge but has no means to explore its environment to update this initial knowledge with ongoing observations. In our work, we aim to build upon the Decision Transformer by expanding it to an online setting by pairing the offline training of the Decision transformer with an approach to online exploration.

(Zheng et al., 2022) then built on this approach to allow for online fine-tuning of a Decision Transformer, and was the first to apply this architecture to the online setting in some capacity. However, despite this capability to fine-tune a Decision Transformer in an online setting, this implementation still requires offline pre-training of the Decision Transformer and thus some pre-existing information about the environment. This implementation trains on the MuJoCo benchmark (Todorov et al., 2012), a series of three-dimensional physics simulations which require online learning. For the offline pre-training of the model, this implementation sources D4RL (Fu et al., 2020) for pre-baked observations about the MuJoCo environments in question. Other approaches have utilized transformers for value decomposition (Khan et al., 2022) and hierarchical learning (Correia and Alexandre, 2022). However, to our knowledge, our implementation is the first full application of the Decision Transformer to the online setting without the need for offline pre-training.

Other work has directly applied reinforcement learning algorithms in single-agent settings. Vanilla policy gradient (Sutton et al., 1999) uses Monte Carlo Q-updates to optimize a policy through on-policy updates. More recent approaches propose actor-critic frameworks where neural networks are used to estimate both state-action values and policies. Deep Deterministic Policy Gradients (Lillicrap et al., 2015) is an example of a deterministic actor-critic method that uses Gaussian noise for exploration. Its closely related refinement to the algorithm, Twin-Delayed Deep Deterministic Policy Gradients (TD3), particularly addresses value function overestimation by minimizing between two value networks (Fujimoto et al., 2018). Other methods such as Soft Actor-Critic (Haarnoja et al., 2018) make use of entropy regularization to encourage exploration.

3 Online vs Offline Learning

Neither offline nor online learning is inherently superior to the other – these two settings for reinforcement learning simply have different starting assumptions. Offline learning assumes access to a pre-existing dataset, but cannot interact with the environment during training. On the other hand, online learning assumes the lack of a pre-existing dataset, instead opting to build one iteratively via live interaction with the environment. The goal of the fully online setting is to "solve" a game through trial and error in the environment itself. This capability proves advantageous in a broad array of real-world applications because it allows agents to solve games from scratch, figuring out the optimal actions through their own exploration without the need for prior knowledge about the environment.

In regards to limitations, the current Decision Transformer architecture, one that models the environment as a sequence modeling problem, does not reason about rewards in the environment. As a result, the Decision Transformer architecture is ineffective in deciding how to explore an environment intelligently. Instead, at a high level, it simply imitates state transition data to achieve a given target return. As long as it has trained on data that has achieved that return, it is effective in replicating that return. In a sequence modeling framework, the Decision Transformer is well-equipped to find patterns in data but does not optimize a function that finds high rewards. Our approach aims to provide this capability.

4 Approach

In this section, we outline the proposed approach to adapt Decision Transformers (DT) to online settings. Given the lack of any assumption about a pre-existing dataset, such an approach must find a way to gather training data for the model to fit. We propose various methods in which this dataset can be incrementally built over time while concurrently improving the DT policy.

4.1 DT-Based Exploration

First, we will test a direct application of the DT algorithm to the online setting. We will make the decision transformer stochastic in its action outputs, sampling from a multivariate Gaussian with a diagonal covariance matrix, as shown in (Zheng et al., 2022). The decision transformer policy will be used to explore the environment and gather transition data, using a sampled return target as input from a univariate Gaussian. After every N episodes, we will fit the DT to the trajectory data and fit the return distribution to the dataset. This cycle of data collection and fitting is shown in Figure 1, and the fully enumerated algorithm is shown in Algorithm 1.

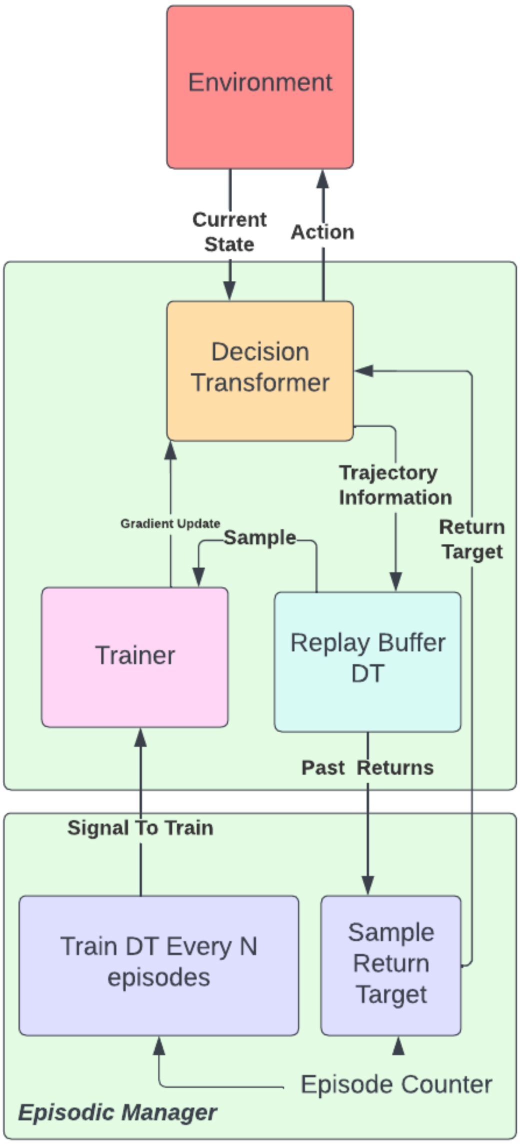

Figure 1: Full architecture of the fully online decision transformer implementation. The decision transformer will stochastically choose actions to explore the environment and store trajectories in a replay buffer. The DT will fit to sampled batches from the replay buffer every N episodes. Target returns are determined by fitting a univariate Gaussian to a subset of returns from past trajectories, and sampling from the distribution to determine future episode target returns.

The intention in keeping our transformer stochastic is to provide a natural way for the Decision Transformer to explore the environment and, through random chance, achieve higher return trajectories that it has not previously seen. In effect,

Algorithm 1: Fully Online Decision Transformer Naive

Input: Decision Transformer π_θ, Number of Iterations T, Proportion of Trajectories for Return Target β, Train Every N, Train Iterations N_{train}, Batch Size B, Learning Rate λ, Replay Buffer R, Univariate Gaussian Sampler ρ

for *each iteration i of T* **do**
- Sample $R_{tar} \sim \rho$. Return 0 if ρ has not been fit to data yet
- Use π to gather trajectory information τ
- Store τ in R
- **if** $i\%N == 0$ **then**
  - **for** N_{train} *steps* **do**
    - Sample batch $B \sim R$
    - Update Decision Transformer parameters θ with B according to Algorithm 2
- Fit ρ to proportion β of most recent trajectory returns in R

Algorithm 2: Decision Transformer Update Function

Input: Decision Transformer π_θ, Learning Rate λ, Batch B of (Returns-to-go, States, Actions, Time step)

1. Let $\{x_i\}_{i=1}^n$ be the sequence of $\{R_i, s_i, a_i, t_i\}_{i=1}^n$
2. Let key k_i, query q_i, and value v_i be mapped via linear transformations from x_i
3. Let $\pi_\theta(\{x_i\}_{i=1}^n) = \sum_{j=1}^n softmax(\{\langle q_i, k_{j'}\rangle\}_{j'=1}^n)_j \cdot v_j$
4. $a_{preds} = \pi_\theta(\{x_i\}_{i=1}^n)$
5. $L = mean((a_{preds} - a)^2)$
6. $\theta = \theta - \nabla_\theta L$

the replay buffer will provide higher return trajectories for the transformer to imitate in the future. The addition of a higher return trajectory also shifts the distribution of past returns to a higher mean. This implies that the algorithm will more likely attain higher-return targets to explore later on.

4.2 RL-Based Exploration

It is important to note that the DT policy itself may not be effective in exploration. Because training occurs at the trajectory level, the DT may have trouble exploiting information in state transitions to maximize rewards. As such, the DT does not reason about reward optimization. Instead, it looks at past trajectory information and, at a high level, imitates behavior that yields certain return targets. Furthermore, in the offline case, the DT assumes trajectories in the dataset are favorable or have some semblance of optimality. However, this is not the case in online learning, where trajectories are far more likely to be suboptimal, or even useless. Because the DT's performance is only as strong as that of its dataset, we propose a method to aid exploration.

The exploration method should be able to reason directly about rewards and optimize actions to yield high returns. A natural selection for this exploration method is to train a reinforcement learning policy in parallel with the DT. The intuition behind this method is to gather useful, high-return trajectory information using the reinforcement learning algorithm to jump-start the training of the DT. Then, the DT will be more informed in its decisions when stochastically exploring the environment.

The choice of reinforcement learning algorithm is discussed in the following section. To weave in the exploratory reinforcement learning algorithm, we stochastically choose between using the exploration (RL) policy and the DT policy to gather data every episode. We opt for the algorithm to choose between the policies in every episode as opposed to in every action in order to ensure that each trajectory is generated by a single policy. This allows for on-policy training if the chosen exploration policy requires it, assuming only trajectories generated by the exploration policy are used to train the exploration policy. Also, this provides consistency. It is noted in previous papers that the DT is able to draw its strength in offline training because its use of attention allows it to differentiate which policy generated various trajectories in the dataset. Keeping trajectories generated by a single policy is meant to take advantage of this.

We denote the probability in which the exploration policy is chosen over the DT policy as α. This value is decreased linearly over training iterations (per episode). By doing this, the responsibility of exploration and policy optimization is shifted over to the DT policy.

4.3 Choice of Reinforcement Learning Algorithm

We apply various algorithms to help gather better trajectories for the DT to train on. First, for its simplicity, we first use an approach akin to the Vanilla Policy Gradient (VPG) algorithm as the exploration policy. This approach is outlined in Algorithm 3. It is known that Monte Carlo sampled Q-values for training policy gradients typically have high variance and do not typically perform as well in complex environments compared to more state-of-the-art approaches. However, the implementation serves as a good baseline to observe whether the DT can reason about the environment with relatively suboptimal trajectories.

We also implement a more state-of-the-art reinforcement learning algorithm as the exploration policy: Twin-Delayed Deep Deterministic Policy Gradients (TD3). TD3 is an actor-critic method that outputs deterministic actions, where Gaussian noise is added to allow for additional exploration. There are two strengths to TD3: a delayed policy update and minimization between two target Q-networks when making gradient updates. The former allows for additional training stability so that value networks are sufficiently trained before being used to update the policy. The latter helps reduce Q value overestimation, an issue commonly present in deep Q-networks. This algorithm was chosen for its powerful learning ability and the omission of other optimizations such as entropy loss, allowing for easier, quicker finetuning. The algorithm for TD3 is outlined in Algorithm 4

Next, we outline the full online decision transformer architecture and algorithm. These are shown in Figure 2 and Algorithm 5.

Algorithm 3: Policy Gradient Update Function

Input: RL Policy $\pi'_{\theta'}$, Batch Size B, Learning Rate λ, Discount Factor γ, Replay R

for *each episode* **do**
- **for** *each iteration in episode* **do**
  - Observe state s and select action $a \sim \pi'_{\theta'}(s)$
  - Execute a in the environment
  - Observe next state s', reward r, and done signal d to indicate if s' is terminal
  - Store (s, a, r, s', d) in R
  - If s' terminal, reset the environment
- **if** $|R| \geq B$ **then**
  - Gather a batch size B of $\{x_i\}_n^{i=1} = (s_i, a_i, r_i, s'_i d_i)_{i=1}^n$
  - Calculate baseline $b = mean(r_{i\,i=1}^{\;n})$
  - Calculate scale $s = std((r_{i\,i=1}^{\;n})$
  - Calculate returns to go $R_i = \sum_{j=i}^{n} (r_j - b)/s$
  - $L = \frac{1}{n} \sum_{i=1}^{n} log(\pi'_{\theta'}(a_i|s_i)) * R_i$
  - $\theta' = \theta' - \nabla_{\theta'} L$
  - Clear Buffer R

Algorithm 4: Twin-Delayed Deep Deterministic Policy Gradients Update Function

Input: RL Policy $\pi'_{\theta'}$, Q functions Q_{ψ_1}, Q_{ψ_2}, Replay R

Set target parameters equal to main parameters $\theta_{target} \leftarrow \theta$, $\psi_{target_1} \leftarrow \psi_1$, $\psi_{target_2} \leftarrow \psi_2$

for *each episode* **do**
- **for** *each iteration in episode* **do**
  - Observe state s and select action $a = clip(\pi'_{\theta'}(s) + \epsilon, a_{low}, a_{high})$ where $\epsilon \sim \mathcal{N}$
  - Execute a in the environment
  - Observe next state s', reward r, and done signal d to indicate if s' is terminal
  - Store (s, a, r, s', d) in R
  - If s' terminal, reset the environment
- **if** *time to update* **then**
  - **for** *j in range number of updates* **do**
    - Sample batch size $B = (s, a, r, s', d)$ from replay R
    - Compute target actions $a'(s') = clip(\pi'_{\theta'}(s') + clip(\epsilon, -c, c), a_{low}, a_{high})$ where $\epsilon \sim \mathcal{N}(0, \sigma)$
    - Compute targets $y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\psi_{target_i}}(s', a'(s'))$
    - Update Q functions for i=1,2 $\nabla_{\psi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in R} (Q_{\psi_i} - y(r, s', d))^2$
    - **if** *j mod policy delay == 0* **then**
      - Update policy $\nabla_\theta \frac{1}{|B|} \sum_{s \in R} Q_{\psi 1(s, \pi'_{\theta'}(s))}$
      - Update target networks for i = 1,2 $\psi_{target_i} \leftarrow \rho \psi_{target_i} + (1 - \rho)\psi_i$
      - $\theta_{target} \leftarrow \rho \theta_{target} + (1 - \rho)\theta$

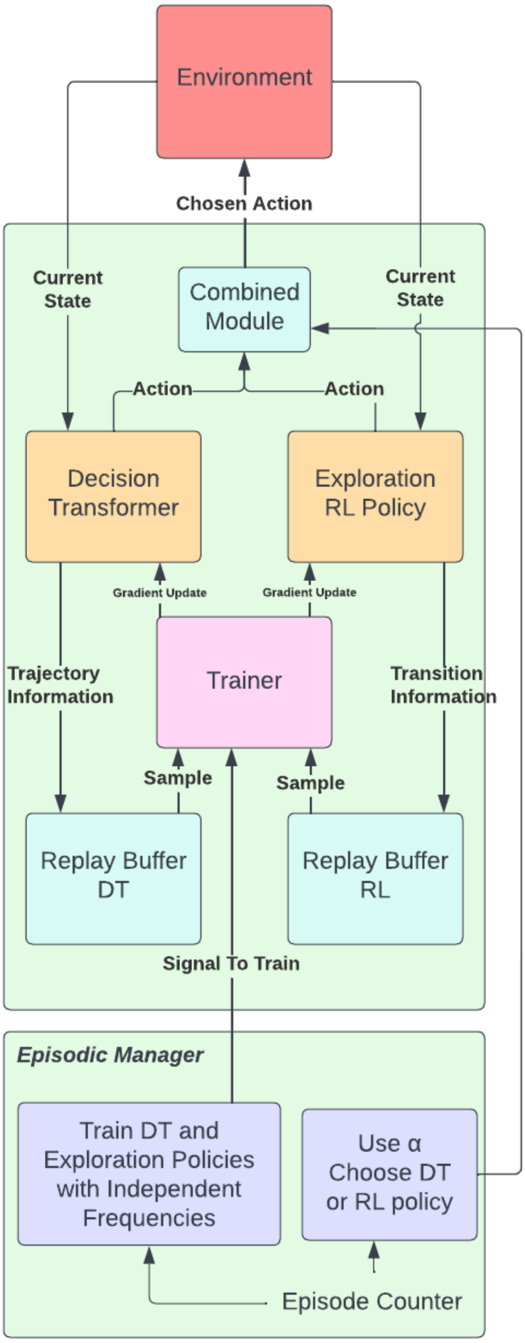

Figure 2: Full training architecture when the reinforcement learning exploration policy is included. The two policies are trained in parallel and independently, taking batch updates at different frequencies. Trajectory information is stored in separate buffers and the policy choice is decided every episode. Actions from both policies are filtered through a combined module, which determines which action gets executed in the environment.

4.4 Implementation

We adopt the Online Decision Transformer code from Daniel Lawson. We also adapt code from

Algorithm 5: Fully Online Decision Transformer

Input: Decision Transformer π_{dt}, RL Exploration Policy π_{rl}, Number of Iterations N, Replay Buffer R_{dt}, Replay Buffer for RL Policy R_{rl}, Univariate Gaussian Sampler ρ, Initial Alpha α_{init}, Final Alpha $\alpha final$, All other Hyperparameters are specified in the DT and RL policy specific algorithms.

$\alpha = \alpha_{init}$
for *each iteration i of* N **do**
  Sample $R_{tar} \sim \rho$. Return 0 if ρ has not been fit to data yet
  Choose policy $\pi \sim (\pi_{dt} = (1 - \alpha), \pi_{rl} = \alpha)$
  Use π to gather trajectory information τ
  Store τ in R_{dt}
  if *(Used* π_{rl} *and* π_{rl} *is on-policy) or (*π_{rl} *is off-policy)* **then**
    Store τ in R_{rl}
  if $i\%T == 0$ **then**
    for N_{train} *steps* **do**
      Sample batch $B \sim R$
      Update Decision Transformer parameters π according to Algorithm 2
  if $i\%T_{rl} == 0$ **then**
    for $N_{trainrl}$ *steps* **do**
      Sample batch $B_{rl} \sim R$
      Update Exploration policy paramters π_{rl} according to Algorithm 3 or 4
  Fit ρ to proportion β of past trajectory returns in R
  Update α linearly towards α_{final}

Vanilla Policy Gradient and TD3 from OpenAI's SpinningUp repository. To run experiments, we use the University of Michigan's Great Lakes Computing Clusters.

5 Evaluation

We test our implementation on the MuJoCo testbed, a collection of three-dimensional physics simulations designed for reinforcement learning. MuJoCo contains multiple tasks, each of which provides a challenging online task focused on spatial reasoning. For the purposes of this paper, we focus our testing on the Hopper benchmark. A screenshot of the replay of the Hopper task is shown in Figure 3, and the goal of the Hopper task is described as follows, from the Open AI website: "The hopper is a two-dimensional one-legged figure that consist of four main body parts - the torso at the top, the thigh in the middle, the leg in the bottom, and a single foot on which the entire body rests. The goal is to make hops that move in the forward (right) direction by applying torques on the three hinges connecting the four body parts." The agent receives more rewards the further right it is able to move the hopper, and this reward is used to train the policy.

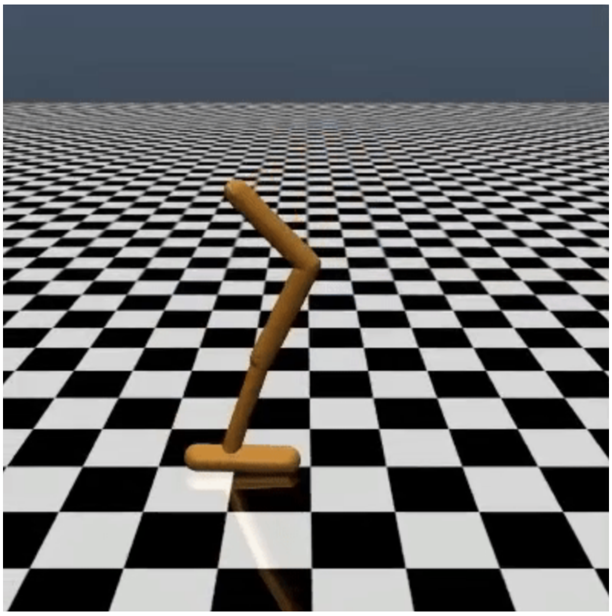

Figure 3: A screencap of the MuJoCo Hopper environment, in which the agent assumes control of the "hopper" and is tasked with navigating it to the right.

We chose MuJoCo as our testbed for several reasons. It is a widely-used benchmark for online RL and is the testbed used in the Online Decision Transformer paper, allowing us to directly replicate and compare the results of that paper. MuJoCo provides a diverse set of environments that randomize initial conditions and give agents fine-tune control, so to succeed on this benchmark, our architecture must be capable of effectively reasoning about the spatial characteristics of the environment. Figure 4 shows benchmarks by OpenAI for various RL policies on the MuJoCo Hopper benchmark. The Y-axis, performance, represents the ongoing rewards achieved by the agent while playing Hopper. Figure 5 is an excerpt from the Online Decision Transformer paper and represents the rewards achieved by that architecture during fine-tuning, in proportion to the reward expected by the decision transformer. Our goal is to replicate this performance with our architecture without the need for pre-training.

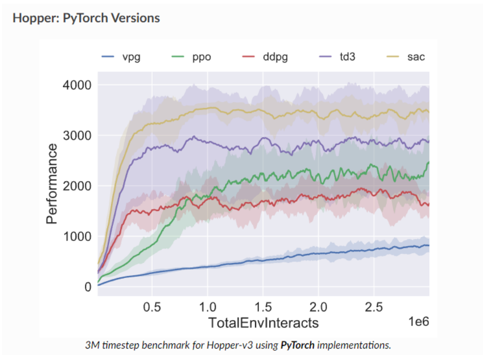

Figure 4: OpenAI SpinningUp benchmarks for various reinforcement learning algorithm implementations on the Hopper Environment.

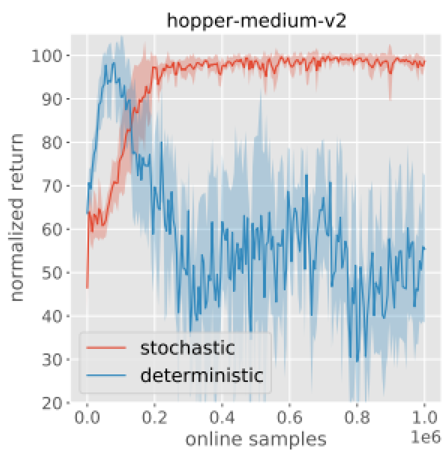

Figure 5: In red, we show the Online Decision Transformer fine-tuning of a model trained on offline data over episodic iterations from (Zheng et al., 2022)

6 Results

We show our results by applying Decision Transformers directly to the MuJoCo environment, where we rely on its stochastic outputs for exploration. We plot the training trajectory of this method in Figure 6, where the return evaluation target is set to 3600.

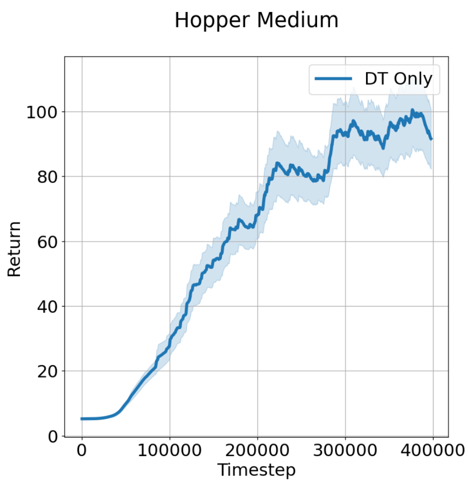

Figure 6: Evaluated return over the first 400,000 timesteps of online training for the Decision Transformer with no additional exploration policy.

As we hypothesized, the Decision Transformer when applied directly to the online setting is unable to garner higher returns over the course of training. After 400,000 iterations, the method is only able to achieve a return of approximately 100, whereas typical state-of-the-art benchmarks achieve over a return of 2500 with a similar number of iterations. This low performance is attributed to the fact that the DT is incapable reasoning about which actions are favorable over others at a state transition level. As a result, its method of exploration is almost purely uniform, randomly attempting new actions in hopes that the resulting trajectory has increased return. As such, this result implies the requirement of an exploration policy to assist exploration.

Next, we test the implementation of two different exploration policies: Vanilla Policy Gradient (VPG) and Twin-Delayed Deep Deterministic Policy Gradients (TD3). In order to determine a stronger candidate for the exploration policy, we compare the training return trajectories of the two methods. The results are shown in Figure 7.

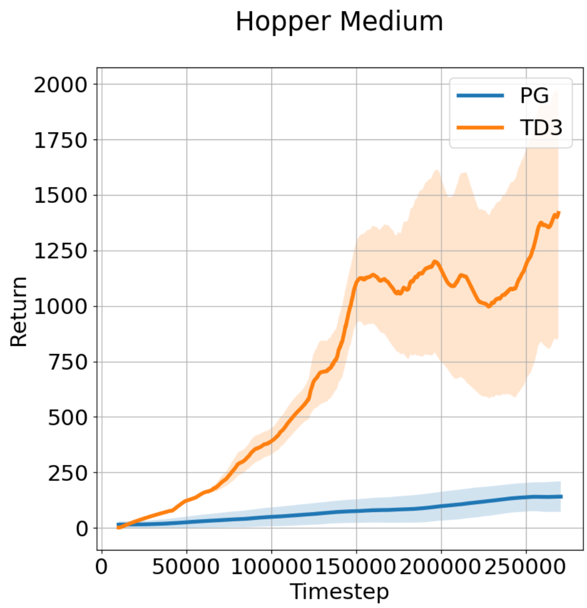

Figure 7: Evaluated return for the first 250,000 timesteps of our implementations of PG and TD3.

The results show that TD3 exhibits stronger sample efficiency and performance than those of the policy gradient method, as expected, given the OpenAI benchmarks for the two algorithms on this environment detailed in the previous section. This is likely caused by the policy gradient's high variance updates and the fact that it is constrained to on-policy learning, as opposed to off-policy with a replay buffer. It is also interesting to note the standard deviations of each of the methods. Policy gradient's training trajectory has a relatively low standard deviation compared to that of TD3. While this may be due to the fact that the policy gradient method has lower return and, therefore, less room to deviate, this can also be caused by on-policy learning's relative training stability when compared to off-policy.

Based on the previous experiment, we choose TD3 as our exploration policy method. In order to provide a strong candidate, we run a few more ablation tests on TD3 to determine what hyperparameter configurations provide the best trajectory data. We note that the largest source of training variance is in how the critics, or Q-networks, are trained. As a result, we experiment with various critic loss functions: L2 loss, L1 loss, and SmoothL1 loss. Furthermore, we test whether adding higher, scaled policy noise (for exploration) and critic target noise (for training stability) results in better performance.

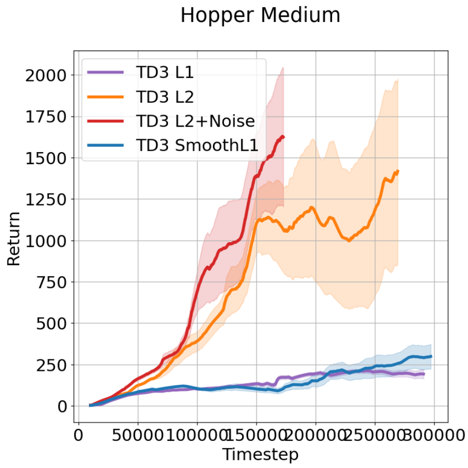

Figure 8: Evaluated return for the first 500,000 timesteps for various versions of our TD3 implementation across our ablation study, in which we vary critic loss functions and inclusion of policy noise.

It is interesting to note TD3's vastly different training return trajectories when varying its critic loss function. Whereas L1 loss and SmoothL1 loss have relatively slow training curves, the use of L2 loss provides higher sample efficiency. This is likely due to the L2 loss function's sensitivity to outliers. Q-value updates with higher temporal difference values have stronger gradients with L2 loss, while L1 loss treats all magnitudes with the same gradient. Furthermore, the method with higher policy and target noise is quicker to garner higher returns than the method without. This can be credited to additional policy exploration and more stable critic updates. The performance difference between the two methods (with and without noise) is non-negligible, but we were uncertain about how the added noise might affect Decision Transformer training when plugged into the hybrid model. As a result, we decide to test both implementations as exploration policies.

Next, we show how the results when the Decision Transformer and TD3 with the chosen hyperparameter settings are combined together in the full architecture. Note that all evaluation episodes of the DT hybrid use action outputs from the Decision Transformer itself, not the exploration policy. Furthermore, the evaluation return targets for each of the training trajectories including the DT was set to 1800. However, it is interesting that, despite being given a fixed return target, the DT policy seems to disregard it. This behavior can likely be attributed to the small buffer size set for the DT policy. This causes the DT to only look at high-return trajectories towards the latter end of training. As a result, the DT learns to ignore the return target and simply replicate recent behavior. Future work could center around an investigation of increasing the size of the buffer to properly allow the DT to output accurate trajectories for given target returns.

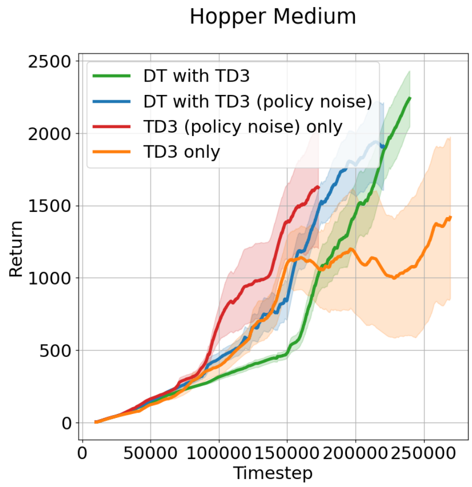

Figure 9: Evaluated return for the first 250,000 timesteps of TD3 alongside our version, the TD3/Decision Transformer hybrid. Both versions are shown with and without added noise in the TD3 policy.

As shown in Figure 9, training when combining the DT and TD3 (without policy noise) shows stronger performance than training with only TD3 (without policy noise). While TD3 exhibits training stability after 50,000 timesteps and drops in average return, the hybrid model with the DT, on average, monotonically increases and surpasses the TD3 benchmark. While MuJoCo benchmarks typically train for above one million timesteps, time and resource constraints prevented further training. Regardless, it is very promising that the hybrid model shows stronger performance.

Our results on methods with additional policy and target noise in Figure 9 are inconclusive regarding the sample efficiency of the hybrid and TD3 methods. Limited time caused our training to stop early, and we are unable to see full results comparing the two. It is notable to see that in both cases, with and without policy noise, TD3 more quickly achieves high returns when compared to the hybrid. This is likely attributed to the fact that, in the hybrid model, while the exploration TD3 policy may achieve high returns already, the Decision Transformer still needs to train on that gathered data. This delay between the exploration and DT policy is reflected in the figure.

It is also interesting to note the relative volatility of TD3's training compared to the hybrid model (with DT). We attribute this disparity by noting that TD3's training depends on deep value networks and temporal difference updates. This type of training is known to be unstable, as not only do value networks tend to have inherent estimation bias but also policy updates depend on these unstable value networks. In the case of the Decision Transformer, the DT is simply fitting to past trajectories without the need to solve a dynamic programming problem, whereas value networks need to. Thus, our approach allows for better stability.

	μ	**Max**
Offline	1960.17 ± 495.82	3222.36
Fine-tuned	$\mathbf{3101.81} \pm 317.21$	3189.17
TD3 Only	2062.0 ± 746.12	3282.40
DT+TD3	$2969.56 \pm \mathbf{6.11}$	**3408.28**

Table 1: Caption

In Table 1, we list the ending performance metrics of the various methods, where Offline pre-training and Fine-tuned (online, following offline pre-training) are replicated from (Zheng et al., 2022), and DT+TD3 is our approach. We report the ending performance metrics from (Zheng et al., 2022), where for offline pre-training we average across all five iterations, and for online fine-tuning, we average over the last 10 out of 200 total iterations of training. For the TD3 and the DT+TD3 hybrid architectures, we instead average over the last 1000 iterations, as we perform a great number many more iterations with these methods, and seek to overall compare the evaluated returns near the end of training for each method. It is paramount to note that some of the methods were trained for a different number of iterations, and as such, it is not immediately conclusive which method is superior to the other without further testing. Nevertheless, in these results, our hybrid model outperforms offline learning and the TD3 policy, and in addition, nearly matches the online fine-tuned version's performance as introduced in (Zheng et al., 2022). This is significant, because our method needs not pre-train offline to achieve the same results as the online fine-tuned version, and in addition successfully applies the benefits of the Decision Transformer in conjunction with TD3 to outperform TD3 on its own. Hence, these results demonstrate significant promise for our method's combination of Decision Transformers with state-of-the-art RL algorithms.

Average Std. Dev.	**TD3**	**TD3 + DT**
No Policy Noise	341.26	**296.36**
With Policy Noise	281.40	**207.15**

Table 2: Average standard deviation of evaluation trajectories throughout training when comparing various methods. We notice that the standard deviation when evaluating Decision Transformer policies is lower than those of TD3 policies. This increased stability can be credited to the fact that DT's are not trained on deep value functions, which can provide unstable updates, but rather through sequence modeling.

We further elaborate on the training instability of TD3 when compared to the hybrid model by noting the standard deviations of evaluation returns throughout training. In Table 2, it is shown that the average standard deviation across all evaluation episodes is lower when using the hybrid model than when using only TD3. This further supports the notion that the addition of the DT to TD3 allows for more stable training.

7 Conclusion

This work introduces, to our knowledge, the first algorithm for a fully online Decision Transformer (DT). By using a reinforcement learning policy to aid exploration, the DT is able to yield performance that is competitive with current state-of-the-art algorithms. It is noted that the DT and RL hybrid is able to produce a policy with resulting performance matching or even surpassing common RL baselines in the tested environment. Furthermore, we find that intermediate training and ending evaluation is more stable in their return values, showing lower standard deviations from their means when compared to all other approaches. This is attributed to the fact that the DT does not require a critic function to train, but rather directly optimizes to imitate a set of trajectories.

Our primary goal with regard to future work is to complete a wider range of trials with more iterations in the Hopper environment. Due to limited computing resources, we were unable to obtain a full set of results, and as such, have yet to see our implementation trained to its full potential. We hope to perform a full range of trials for each ablation of our Decision Transformer TD3 hybrid architecture, as while we have shown that our implementation meets the performance of previous baselines without the need for offline pre-training, we have yet to see whether our implementation will exceed that baseline with further training. By completing a fuller range of trials, we will also have sufficient data to resolve our current inconclusiveness on whether the policy noise version of our hybrid architecture surpasses the performance of TD3 with policy noise. Another desired alteration to our architecture which we omitted for the sake of time is the increase of the buffer size of the Decision Transformer policy, which would allow it to more effectively remember target return values during extended training. In addition, we wish to extend our results to include other MuJoCo benchmarks, such as Cheetah, Ant, and Walker, which would provide further insight into the comparative performance between the architectures studied in this paper.

One benefit of our implementation is its modularity. Our implementation is based on that of the original Decision Transformer, which uses Transformer architecture from BERT and GPT-2. We posit that substituting this architecture with more recent or more specialized Transformer architectures may increase or alter the performance of our architecture on certain tasks. Furthermore, we may also easily replace the TD3 policy in our hybrid architecture with another exploration policy, which, depending on the effectiveness of that policy on a certain benchmark, may increase the performance of our hybrid architecture. For these reasons, we believe that the high modularity of our architecture will allow it to combine advances in performance in Transformers as well as new exploration policies.

8 Acknowledgements

Thank you to the University of Michigan for providing us with the computing resources to run these experiments. We would also like to thank Joyce Chai and the teaching staff of EECS 595 for facilitating the space to explore this research topic.

References

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097.

André Correia and Luís A Alexandre. 2022. Hierarchical decision transformer. *arXiv preprint arXiv:2209.10447*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

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

Scott Fujimoto, Herke Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR.

Muhammad Junaid Khan, Syed Hammad Ahmed, and Gita Sukthankar. 2022. Transformer-based value function decomposition for cooperative multi-agent reinforcement learning in starcraft. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 18, pages 113–119.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12.

Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Qinqing Zheng, Amy Zhang, and Aditya Grover. 2022. Online decision transformer. *arXiv preprint arXiv:2202.05607*.