How Crucial is Transformer in Decision Transformer?

Max Siebenborn
Technical University of Darmstadt
Department of Computer Science
max.siebenborn@stud.tu-darmstadt.de

Boris Belousov
Technical University of Darmstadt
German Research Center for AI (DFKI)
Systems AI for Robot Learning Group

Junning Huang
Technical University of Darmstadt
Department of Computer Science
Intelligent Autonomous Systems Group

Jan Peters
Technical University of Darmstadt
German Research Center for AI (DFKI)
Hessian.AI & Centre for Cognitive Science

Abstract

Decision Transformer (DT) is a recently proposed architecture for Reinforcement Learning that frames the decision-making process as an auto-regressive sequence modeling problem and uses a Transformer model to predict the next action in a sequence of states, actions, and rewards. In this paper, we analyze how crucial the Transformer model is in the complete DT architecture on continuous control tasks. Namely, we replace the Transformer by an LSTM model while keeping the other parts unchanged to obtain what we call a *Decision LSTM* model. We compare it to DT on continuous control tasks, including pendulum swing-up and stabilization, in simulation and on physical hardware. Our experiments show that DT struggles with continuous control problems, such as inverted pendulum and Furuta pendulum stabilization. On the other hand, the proposed Decision LSTM is able to achieve expert-level performance on these tasks, in addition to learning a swing-up controller on the real system. These results suggest that the strength of the Decision Transformer for continuous control tasks may lie in the overall sequential modeling architecture and not in the Transformer per se.

1 Introduction

Transformers [27] have shown impressive results across a number of problem domains in Natural Language Processing [5, 17, 3] and Computer Vision [6, 13]. Inspired by these results, [4, 11] framed Reinforcement Learning (RL) as a sequence modeling problem, in which Transformer predicts the next element in a sequence of states, actions and rewards. In [4], the Decision Tranformer (DT) is proposed, an offline RL algorithm that auto-regressively models trajectories using the GPT-2 architecture [18]. The Trajectory Transformer architecture from [11] is similar to DT but instead of return-to-go values it utilizes beam-search planning for sequence generation and employs state and reward prediction as well as discretization. The evaluations in [4] showed that DT is stronger than straightforward Behavior Cloning (BC) on the D4RL dataset [7], which includes discrete Atari games and continuous control tasks from OpenAI gym [2]. However, from these experiments it remains unclear whether Decision Transformer is also competitive for dynamic tasks that require stabilization of systems around an unstable equilibrium, as well as for real robot control tasks.

In this paper, we evaluate Decision Transformer on robot learning tasks, focusing on two aspects. First, we evaluate DT on *stabilization tasks*—on various pendulum swing-up and stabilization environments. The goal of an agent in these tasks is to reach an unstable equilibrium and stabilize the system around it. Second, we validate our simulation results on a *real robotic platform*. This evaluation is crucial since the gap between simulation and reality is still an open issue in robotics and RL [16, 28], and

Foundation Models for Decision Making Workshop at Neural Information Processing Systems, 2022.

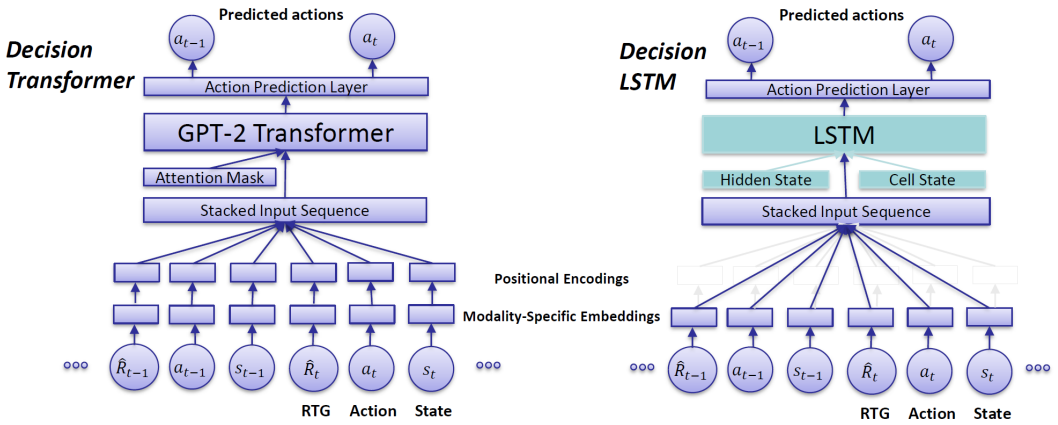


Figure 1: Comparison of the architectures of Decision Transformer (on the left) and the proposed Decision LSTM (on the right). The bottom items show the sequences of observed return-to-go (RTG) values, actions and states, which are used to predict the next action in an auto-regressive way. The key differences between the architectures are the use of an LSTM network as a replacement for the GPT-2 Transformer and the removal of positional encodings for DLSTM.

the results in simulation do not directly transfer to reality. Furthermore, for real robotic applications, the model inference time must be sufficiently small to enable real-time control.

In addition to our evaluation of Decision Transformer, we propose a related architecture, which we call Decision LSTM (DLSTM), that builds on top of DT but replaces the GPT-2 Transformer by an LSTM [10] model. We use DLSTM to test whether framing RL as a sequence modeling problem allows using other architectures apart from Transformers, and whether they can yield better results. We compare the performances of both DT and DLSTM against a straightforward Behavior Cloning (BC) model which mimics actions based on the observed states without taking rewards into account.

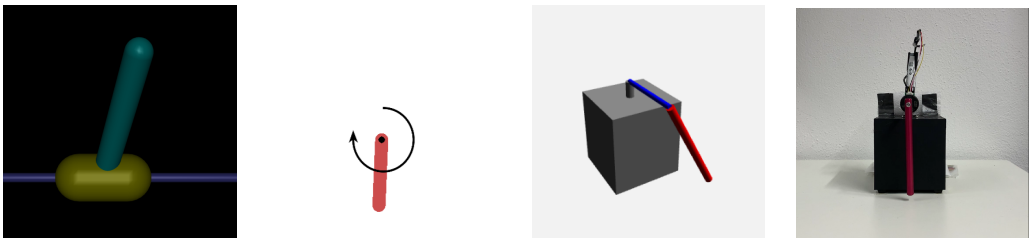
In summary, our paper provides the following three contributions.

- The introduction of Decision LSTM as a novel architecture for offline RL, which builds on top of Decision Transformer. DLSTM shows the general capabilities of framing RL as a sequence modeling problem independent of the concrete architecture.
- An evaluation of DT and DLSTM in continuous control tasks that require fine-grained stabilization. Experiments on a Furuta pendulum platform highlight the issues and trade-offs of the different architectures when deployed in the real world.
- A thorough investigation concentrated on whether the functionalities and effects of critical ingredients of the Decision Transformer, such as the return-to-go values, can be validated in the continuous control environments.

2 Background

A key motivation behind the Decision Transformer [4] architecture is to frame reinforcement learning as a sequence modeling problem [1]. Sequence modeling is predominant in natural language processing, where, e.g., a sentence can be seen as a sequence of words. Language models predict the next word in a sentence by taking the previous words as input [26]. Similarly, DT predicts the next action in a sequence of states, actions, and rewards. Recurrent Neural Networks (RNNs) [21, 12] and especially LSTMs [10] have been considered state-of-the-art sequence models thanks to their ability to process sequences of varying length and make information from previous timesteps persistent inside the network. However, such sequential processing of data precludes parallelization of RNN and LSTM computations, resulting in potentially long training times [27].

Recently Transformers [27] have become predominant in natural language processing and sequence modeling. Transformer is an architecture for auto-regressive sequence modeling that is purely based on the *attention* mechanism and does not contain any recurrent or convolutional structures. In contrast to RNNs and LSTMs, they are highly parallelizable since their attention mechanism does not require



(a) Mujoco Inverted Pendulum (Stabilization). (b) OpenAI Pendulum. (c) Furuta Pendulum (Simulation). (d) Furuta Pendulum (Real Hardware).

Figure 2: Experiment environments to test the capabilities of Decision Transformer and Decision LSTM on continuous control tasks requiring stabilization.

sequential processing of the input elements. Furthermore, models such as BERT [5], and the GPT-x architectures [17, 18] have shown the capabilities of Transformers to build large pre-trained models that can be finetuned on specific tasks.

In terms of computational complexity, Transformer layers are linear in the input dimensionality and quadratic in the input length [27]. This is advantageous for short input sequences with high-dimensional latent representations (common in NLP) but can be problematic for long input sequences (e.g., sequences of states, actions, and rewards in RL problems). Recurrent networks, however, can have computational advantages for long input sequences with small input dimensionality because they scale linearly in the input length and quadratically in the input dimensionality.

Decision Transformer [4] brings the benefits of sequence modeling to model-free offline reinforcement learning. It frames RL as a prediction problem, with the goal of predicting the next action given the history of past transitions in a trajectory

$$\tau = (\hat{R}_1, s_1, a_1, \hat{R}_2, s_2, a_2, ..., \hat{R}_T, s_T, a_T)\,.$$

Here, $\hat{R}_t$ is the return-to-go (RTG) value, i.e., the sum of the future rewards in the trajectory, s_t is the state, and a_t is the action at time step t, respectively. DT aims to solve the RL problem without making use of conventional value-based methods such as Dynamic Programming or TD-Learning [4] by conditioning outputs on RTG values, similar to related return-conditioned approaches [14, 9, 22].

3 Methods

The main question addressed in this paper is whether Transformer is crucial for the Decision Transformer architecture. We hypothesize that the overall framing of RL as a sequence modeling problem is more responsible for the strong performance of DT than the use of the Transformer. To prove this point, we introduce Decision LSTM (DLSTM), a novel architecture which builds on the Decision Transformer but replaces the GPT-2 model by an LSTM network. The DT and DLSTM architectures are shown in Figure 1. DLSTM introduces the following architectural adjustments:

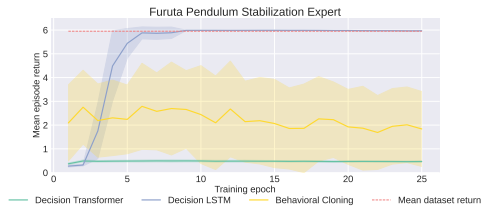
- the GPT-2 Transformer is replaced by an LSTM;
- the attention mask is removed because LSTM does not utilize it;
- positional embeddings are removed because LSTM processes inputs sequentially;
- LSTM's hidden states and cell states are initialized with zero vectors.

The LSTM architecture has proven to be a successful tool for sequence modeling [25]. It provides computational benefits especially on long sequences, because it scales linearly with the input length whereas Transformer scales quadratically. Therefore, gains in performance and real-time capability are expected from DLSTM in RL, where input lengths may range between 100's to 1000's timesteps.

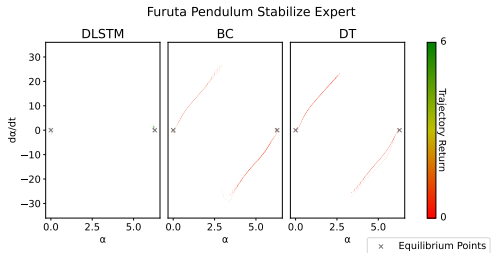
We evaluate DT and DLSTM on several continuous control tasks, which are shown in Figure 2. Additionally, we report the results of a straightforward Behavior Cloning (BC) baseline which predicts the next action using a feedforward neural network trained on the dataset of past trajectories. Our experimental methodology consists of the following steps. First, a dataset of trajectories is collected using a behavior policy. All three approaches—DT, DLSTM, and BC—operate in the offline

Table 1: Simulation results on *expert* data. Fully trained DT, DLSTM, and BC models are evaluated in 4 simulated environments. Mean episode return over the dataset $\overline{G_{\text{Data}}}$ and the mean $\pm$ standard deviation of episode returns over 30 evaluation episodes are reported. DLSTM outperforms DT in all cases, and it outperforms BC in 3/4 environments, performing on par in OpenAI pendulum swing-up.

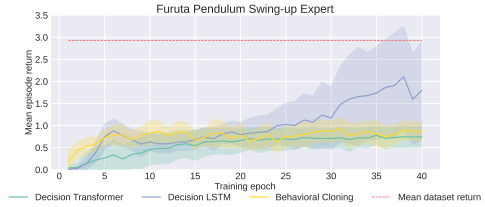
			Evaluation Episode Returns		
Environment	**Dataset**	$\overline{G_{\text{Data}}}$	**DT**	**DLSTM**	**BC**
Mujoco Pendulum Stabilization	Expert	1000.00 $\pm$ 0.00	454.72 $\pm$ 360.12	**985.31** $\pm$ **71.96**	61.61 $\pm$ 170.16
OpenAI Pendulum Swing-up	Expert	-207.53 $\pm$ 167.75	-761.44 $\pm$ 375.71	**-252.86** $\pm$ **233.21**	**-235.78** $\pm$ **204.45**
Furuta Pendulum Stabilization	Expert	5.95 $\pm$ 0.02	0.46 $\pm$ 0.03	**5.93** $\pm$ **0.01**	1.82 $\pm$ 1.60
Furuta Pendulum Swing-up	Expert	2.93 $\pm$ 0.63	0.74 $\pm$ 0.24	**1.79** $\pm$ **1.12**	0.87 $\pm$ 0.21



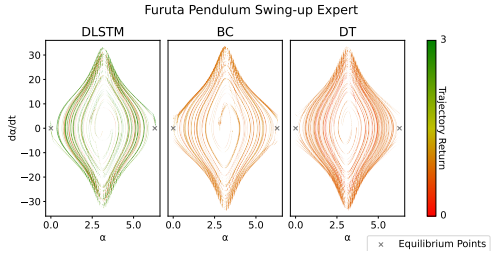
(a) Learning curves / expert data, Furuta stabilization



(b) Phase portraits / expert data, Furuta stabilization



(c) Learning curves / expert data, Furuta swing-up



(d) Phase portraits / expert data, Furuta swing-up

Figure 3: Learning curves (left) and phase portraits (right) for DT, DLSTM, and BC trained on *expert* data for stabilization (top) and swing-up (bottom) tasks. Especially stabilization (a) appears to pose a challenge to DT and BC, whereas DLSTM quickly achieves the mean dataset return and is able to keep the pendulum at equilibrium, as seen in (b). The swing-up task (c) is significantly harder, and again only DLSTM manages to reach sufficiently high return, albeit not in all runs. Phase portraits (d) show that DLSTM is the only model which is able to stabilize the pendulum.

mode. Following the D4RL [7] protocol, we collect separate datasets of *expert* quality (behavior policy solves the task at expert level) and of *replay* quality (data from early epochs of training of an online model-free RL algorithm is mixed with data from late epochs). On the replay data, DT and DLSTM are expected to perform better than BC, because they weigh experiences by the reward, whereas BC does not take the reward into account. Second, all models are trained until convergence on the collected data. Third, fully optimized models are evaluated over 30 runs in the respective environments. Our implementation is based on the original DT codebase with default parameters.

4 Experiments

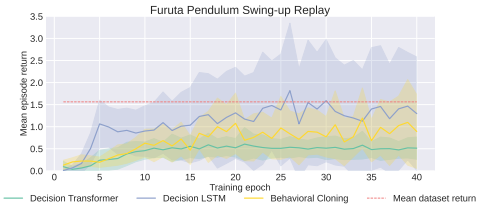
In this section, we present and discuss the results of the experiments described in Section 3.

Simulation results on expert data. Table 1 shows the performance of DT, DLSTM, and BC on simulated swing-up and stabilization tasks. Decision LSTM outperforms both Decision Transformer and Behavioral Cloning in most experiments, and performs on par with BC on OpenAI pendulum swing-up. The superior performance of DLSTM is especially apparent in the more challenging Furuta pendulum environment, both on the swing-up and stabilization tasks, on which DT and BC fail to reach the expert performance.

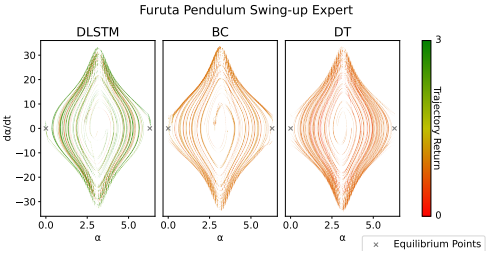
Figures 3a and 3c show the learning curves on stabilization and swing-up tasks in the Furuta pendulum environment. DLSTM is the only model that manages to solve both tasks, albeit the swing-up is

Table 2: Simulation results on *replay* data. DLSTM is the only model which achieves better mean episode return than the average return $\overline{G_{\text{Data}}}$ of the training dataset.

			Evaluation Episode Returns		
Environment	**Dataset**	$\overline{G_{\text{Data}}}$	**DT**	**DLSTM**	**BC**
OpenAI Pendulum Swing-up	Replay	-837.35 $\pm$ 414.12	-1083.78 $\pm$ 346.79	**-569.89 $\pm$ 568.49**	-815.41 $\pm$ 577.83
Furuta Pendulum Swing-up	Replay	1.56 $\pm$ 1.70	0.51 $\pm$ 0.25	**1.30 $\pm$ 1.28**	0.89 $\pm$ 0.83



(a) Learning curves / replay data, Furuta swing-up



(b) Phase portraits / replay data, Furuta swing-up

Figure 4: Learning curves (left) and phase portraits (right) for DT, DLSTM, and BC trained on *replay* data for the Furuta swing-up task. Notably, only DLSTM is able to achieve higher returns than the mean dataset return (the shaded blue area in (a) goes higher than the dotted red line). The phase portraits (b) again show that DLSTM is the only model that stabilizes the pendulum.

not successful in every run. Figures 3b and 3d show the phase portraits of the trained models. In the stabilization task in Figure 3b, the pendulum starts in the upright unstable equilibrium state, indicated by the points $\alpha = 0$ and $\alpha = 2\pi$. DLSTM achieves stabilization and high returns in all evaluation episodes. Meanwhile, BC sometimes achieves stabilization (indicated by green trajectories, high return), but often fails (red trajectories, low return). DT always fails at the stabilization task: trajectories diverge from the equilibrium state in all episodes. On the swing-up task (Figure 3d), BC and DT manage to bring the pendulum to the upright position, but fail to stabilize it. DLSTM, on the other hand, is able to swing-up and stabilize the pendulum, albeit not at expert level in every run.

Simulation results on replay data. On the replay data, DLSTM again achieves better performance than the other models (Table 2). Notably, DLSTM is the only model which is able to improve upon the demonstrations, i.e., achieve a higher return than in the training dataset. Figure 4a shows the corresponding learning curves and Figure 4b the phase portraits. The results are similar to Figure 3d.

Real platform results. For a real-world evaluation, the models trained on an expert dataset recorded on the real Furuta pendulum platform are evaluated in this environment. Table 3 indicates that DLSTM significantly outperforms DT and BC. On the standard swing-up task, DLSTM achieves stabilization and high return in many but not all episodes, while the other models fail to bring the pendulum to the upright position altogether. Despite successful swing-up, DLSTM fails at stabilizing the pendulum in most cases, therefore the episode return is lower than in the expert data. Such performance gap can be explained by the sim-to-real discrepancy and the real-time requirements of the physical platform.

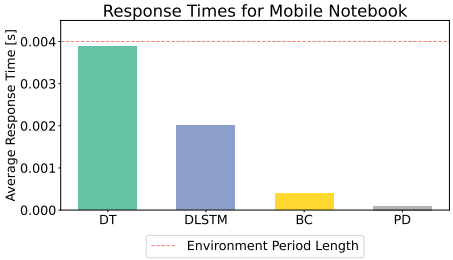
To show that DLSTM is both capable of swinging up and stabilizing the pendulum, we test its capabilities on the respective tasks independently. First, we let the DLSTM policy swing up the pendulum and, after a certain pose is reached, let a PD controller take over and stabilize the pendulum. The second row in Table 3 shows that this combination of the controllers leads to higher mean returns than before. Apparently, DLSTM has problems dealing with the high velocities which occur during the swing-up phase but which were not observed in the simulation training data.

Finally, we evaluate all models on the pure stabilization task on the real Furuta pendulum platform. We use a given expert policy to swing up and stabilize the pendulum for a short period of time, and then let the respective models (DT, DLSTM or BC) take over to continue stabilizing the pendulum. DLSTM and BC achieve expert performance, while DT fails on this task (3rd row in Table 3).

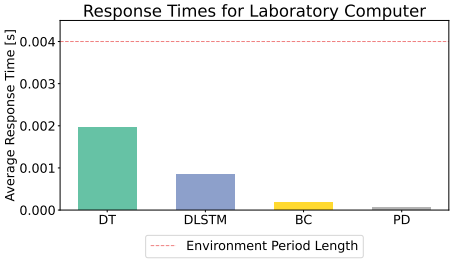
Real-time capabilities. One reason for the worse performance of the considered models on the physical platform compared to simulation may be the time delays in the real-time control loop. To investigate this issue, we compare the mean inference times, i.e., the time needed to generate an action, for the different models in the Furuta pendulum environment. Figure 5 shows response times

Table 3: Experimental results on the real Furuta pendulum (denoted FPRR) on expert data. DLSTM matches expert performance on the stabilization task and achieves high but less than expert reward on swing-up and swing-up with PD-stabilization tasks. Only DLSTM was evaluated with PD-stabilization, because DT and BC failed to swing-up the real Furuta pendulum.

			Evaluation Episode Returns		
Environment	**Dataset**	$\overline{G_{\text{Data}}}$	**DT**	**DLSTM**	**BC**
FPRR Swing-up	Expert	2.93 ± 0.63	0.38 ± 0.15	$\mathbf{1.11 \pm 0.52}$	0.22 ± 0.18
FPRR Swing-up with PD stabilization	Expert	2.93 ± 0.63	–	$\mathbf{2.17 \pm 0.60}$	–
FPRR Stabilization	Expert	5.95	0.38 ± 0.08	$\mathbf{5.98 \pm 0.00}$	$\mathbf{5.96 \pm 0.02}$



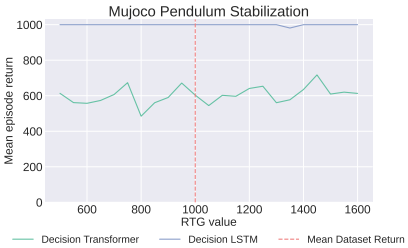
(a) Inference times on laptop with Intel(R) Core(TM) i5-7200U CPU, 2 cores @ 2.50 GH.



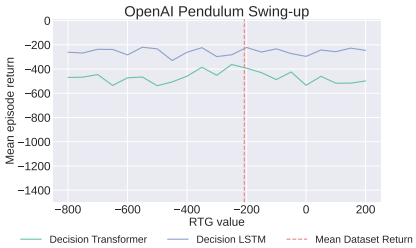
(b) Inference times on computer with Intel(R) Core(TM) i7-9700K CPU, 8 cores @ 3.60 GH.

Figure 5: Mean inference times for DT, DLSTM, BC, and a PD controller on a real Furuta pendulum commanded from a laptop (a) and a stationary PC (b). In both cases, the mean inference time is below the control interval 0.004s for all methods. However, DT takes twice the time compared to DLSTM.

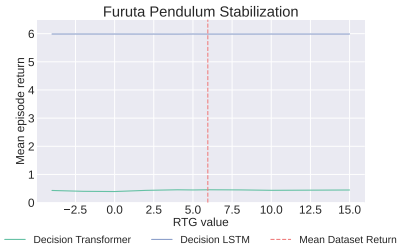
measured on a laptop and a stationary PC. The horizontal orange line shows the maximum allowed time for action generation in the real-time loop at 250Hz. This control frequency is necessary to enable stable and reliable pendulum stabilization. DT and DLSTM have higher inference times compared to BC and a PD-controller. DT on average takes twice the time compared to DLSTM.



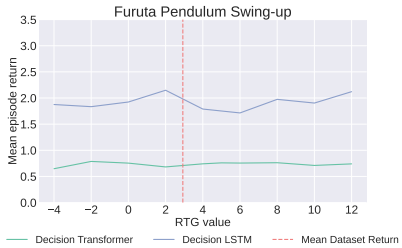
(a) RTG influence in Mujoco InvPend



(b) RTG influence in OpenAI pendulum



(c) RTG influence in Furuta stabilization



(d) RTG influence in Furuta swing-up

Figure 6: Effects of the desired return-to-go (RTG) values used for conditioning action generation by DT and DLSTM on the mean episode return in different environments. The query RTG value appears to have no influence on the episode return, which indicates that the models are not making use of the RTG value for action generation.

Influence of return-to-go values. An important feature of DT is the use of return-to-go (RTG) values, i.e., one can in principle control the optimality of the generated trajectories: at evaluation time,

the RTG value specifies the desired expected return, and therefore DT should generate trajectories that achieve this desired RTG value. We perform evaluations to verify whether the RTG value indeed has an influence on the episode return. Contrary to the findings in [4], Figure 6 indicates no influence of the desired RTG values on the actual returns both for DT and DLSTM. These results raise the question whether the RTG values are even necessary in the DT architecture and what role they play.

5 Related Work

The original Decision Transformer [4] presents a basic approach to framing RL as sequence modeling problem, allowing for extensions in multiple directions. In [8], a *Generalized Decision Transformer* is introduced, which performs hindsight information matching by generating trajectories that match any statistics of the future trajectories and not only return-to-go values. The *Online DT* [29] combines offline pre-trained DT models with an online fine-tuning procedure, thereby overcoming the distributional shift inherent in offline RL and affecting the DT. A similar online DT approach but in a *multi-agent* setting is proposed in [15]. *Transfer learning* for the DT architecture is addressed in [20], where the DT is pre-trained on data from other domains and modalities, e.g., Wikipedia articles, and subsequently fine-tuned on a given offline RL problem. On a massive scale, such cross-modal generalization capability of DT was demonstrated in *Gato* [19]. These results indicate the existence of an underlying universal structure across sequence modeling problems that enables cross-domain and multi-modal transfer learning.

6 Conclusion

Our empirical evaluations of Decision Transformer on continuous control tasks such as pendulum swing-up and stabilization have shown that DT struggles on problems that require fine-tuned actions. The model trained on offline data fails in the online setting and is not able to solve the task on a real system. On the other hand, the proposed modification of DT, which we call Decision LSTM—and which only differs from DT in that the Transformer is replaced by an LSTM—has shown strong performance in the same environments. Therefore, we conclude that the advantages of DT observed in prior works may be rather due to the sequence modeling approach than to the particular choice of the prediction module.

The paper does not consider discrete actions (e.g., discretizing continuous actions into bins, or discrete-action domains such as Atari), where transformer-style architectures may have an advantage. Moreover, performance of DT-like policy architectures can depend on many factors, including the domain/task (transition dynamics and reward function), action parameterization, discretization of RTG, network architecture, etc. In general, our results only apply to the continuous control tasks and a further investigation is necessary to evaluate our hypothesis on a broader set of domains.

Despite the good performance of DLSTM, it remains an open question whether approaches that frame RL as a sequence modeling problem provide significant advantages over standard Behavioral Cloning. Our results indicate no correlation between the return-to-go values and the model performance in the stabilization experiments. Therefore, the effectiveness of RTG values as task-defining inputs that provide hindsight information to the decision architectures in continuous control tasks is unclear.

Finally, to make Decision Transformer and Decision LSTM applicable in real-world settings, the sim-to-real gap and the inference times of the models are crucial. The inference times of the decision architectures are significantly longer compared to standard BC, which yielded problems in our real-time experiments, making it necessary to investigate the inference times of the models further. Purely relying on successful simulation runs where the actual inference times of the models are ignored may be a potential cause of problems under real-world conditions. In settings where Decision Transformer generates actions too slow for the real-time requirements, the proposed Decision LSTM architecture may be preferred due to the faster run time.

Acknowledgments and Disclosure of Funding

This project has received funding from BMWSB ZukunftBau under grant Nr. 10.08.18.7-21.34. Calculations for this research were partially conducted on the Lichtenberg high performance computer of the TU Darmstadt.

References

[1] S. Bai, J. Z. Kolter, and V. Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *CoRR*, abs/1803.01271, 2018.

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[4] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 15084–15097, 2021.

[5] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, pages 4171–4186, 2019.

[6] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

[7] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4RL: datasets for deep data-driven reinforcement learning. *CoRR*, abs/2004.07219, 2020.

[8] H. Furuta, Y. Matsuo, and S. S. Gu. Generalized decision transformer for offline hindsight information matching. In *International Conference on Learning Representations (ICLR)*, 2022.

[9] Y. Guo, J. Choi, M. Moczulski, S. Feng, S. Bengio, M. Norouzi, and H. Lee. Memory based trajectory-conditioned policies for learning from sparse rewards. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

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

[11] M. Janner, Q. Li, and S. Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, pages 1273–1286, 2021.

[12] M. I. Jordan. Serial order: A parallel distributed processing approach. In *Advances in Psychology*, volume 121, pages 471–495. Elsevier, 1997.

[13] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah. Transformers in vision: A survey. *ACM Computing Surveys*, 54(10s):1–41, jan 2022.

[14] M. Liu, M. Zhu, and W. Zhang. Goal-conditioned reinforcement learning: Problems and solutions. In *International Joint Conference on Artificial Intelligence*, pages 5502–5511, 2022.

[15] L. Meng, M. Wen, Y. Yang, C. Le, et al. Offline pre-trained multi-agent decision transformer: One big sequence model tackles all SMAC tasks. *CoRR*, abs/2112.02845, 2021.

[16] F. Muratore, F. Ramos, G. Turk, W. Yu, M. Gienger, and J. Peters. Robot learning from randomized simulations: A review. *Frontiers in Robotics and AI*, 9, 2022.

[17] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. Improving language understanding by generative pre-training. Technical report, OpenAI, 2018.

[18] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.

[19] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, et al. A generalist agent. *CoRR*, abs/2205.06175, 2022.

[20] M. Reid, Y. Yamada, and S. S. Gu. Can wikipedia help offline reinforcement learning? *CoRR*, abs/2201.12122, 2022.

[21] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge, MA, 1986.

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

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.

[24] K. Schweighofer, M. Hofmarcher, M.-C. Dinu, P. Renz, A. Bitto-Nemling, V. P. Patil, and S. Hochreiter. Understanding the effects of dataset characteristics on offline reinforcement learning. *ArXiv*, abs/2111.04714, 2021.

[25] R. C. Staudemeyer and E. R. Morris. Understanding lstm – a tutorial into long short-term memory recurrent neural networks, 2019.

[26] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3104–3112, 2014.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5998–6008, 2017.

[28] W. Zhao, J. P. Queralta, and T. Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *Symposium Series on Computational Intelligence (SSCI)*, pages 737–744. IEEE, 2020.

[29] Q. Zheng, A. Zhang, and A. Grover. Online decision transformer. In *International Conference on Machine Learning (ICML)*, volume 162, pages 27042–27059, 2022.

A Appendix

A.1 Datasets

Table 4 provides an overview of the datasets used in our experiments. The behavior policy is given by a PPO agent [23], i.e., the *replay* datasets are comprised of the experiences collected during all epochs of the PPO training, while the *expert* datasets contain demonstrations of the PPO policy in the final epoch, i.e., after it has been trained to expert-level. As a measure of the quality of demonstrations in the dataset, the expected trajectory return (TQ) values [24] are used, which quantify the relation between the average return of a trajectory to the maximal return in the dataset. A high TQ value indicates high quality (i.e., high mean returns) in the dataset, while a low TQ value indicates low-return demonstrations.

Name	**Environment**	**Num. Traj**	**Beh. Policy**	**Task**	**TQ**
mujoco-inverted-pendulum-expert	Mujoco Inv. Pendulum	500	PPO	Stabilization	1.00
openai-pendulum-expert	OpenAI Pendulum	250	PPO	Swing up	0.83
openai-pendulum-replay	OpenAI Pendulum	100	PPO	Swing up	0.32
furuta-pendulum-stabilize-expert	Furuta Pendulum	500	PPO	Stabilization	0.99
furuta-pendulum-swing-up-expert	Furuta Pendulum	500	PPO	Swing up	0.49
furuta-pendulum-swing-up-replay	Furuta Pendulum	515	PPO	Swing up	0.29

Table 4: Overview of the used training datasets for the stabilization experiments.

A.2 Hyperparameter Settings

For the experiments, the default hyperparameter settings from [4] were used, as shown in Table 5.

	DT	**DLSTM**	**BC**
Context length K	20	20	20
Number of hidden layers	3	3	3
Hidden layer size	128	128	256
Batch size	64	64	128
Number of training steps per training epoch	3000	3000	3000
Input normalization	yes	yes	yes
Dropout	0.1	0.1	0
Activation function	$\tanh$	$\tanh$	$\tanh$
Learning rate	3×10^{-5}	3×10^{-5}	3×10^{-5}
Number of attention heads	1	-	-

Table 5: Overview of the used hyperparameters for the different evaluated architectures.