Learning to Throw with a Handful of Samples using Decision Transformers

Maxim Monastirsky, Osher Azulay and Avishai Sintov

***Abstract*—Throwing objects by a robot extends its reach and has many industrial applications. While analytical models can provide efficient performance, they require accurate estimation of system parameters. Reinforcement Learning (RL) algorithms can provide an accurate throwing policy without prior knowledge. However, they require an extensive amount of real world samples which may be time consuming and, most importantly, pose danger. Training in simulation, on the other hand, would most likely result in poor performance on the real robot. In this letter, we explore the use of Decision Transformers (DT) and their ability to transfer from a simulation-based policy into the real-world. Contrary to RL, we re-frame the problem as sequence modelling and train a DT by supervised learning. The DT is trained off-line on data collected from a far-from-reality simulation through random actions without any prior knowledge on how to throw. Then, the DT is fine-tuned on an handful (~ 5) of real throws. Results on various objects show accurate throws reaching an error of approximately 4cm. Also, the DT can extrapolate and accurately throw to goals that are out-of-distribution to the training data. We additionally show that few expert throw samples, and no pre-training in simulation, are sufficient for training an accurate policy.**

***Index Terms*—Reinforcement Learning, Transfer Learning.**

I. INTRODUCTION

THE ability of a robot to accurately throw an object to a desired target can provide better efficiency to many tasks such as packaging in warehouses, object transfer and recycling [1]. By throwing an object, the robot utilizes its dynamics for extending its reach. The robot can place objects in boxes or bins positioned in farther region without the need to physically reach them.

The throwing problem has been addressed in several analytical approaches where system models and parameter tuning are required [2], [3]. In contrary, not much work has been carried out using machine learning approaches. Nevertheless, recent work combined a physics-engine simulation with a regression network trained by real world throws [4]. While the method achieved results outperforming human throws, it requires an extensive amount of real world throws and some prior how to throw. Similarly, Reinforcement Learning (RL) applications for throwing requires a significant amount of real-world samples in order to demonstrate sufficient performance.

Manuscript received: September, 13, 2022; Revised November, 20, 2022; Accepted December, 11, 2022.

This paper was recommended for publication by Editor Jens Kober upon evaluation of the Associate Editor and Reviewers' comments. This work was partly supported by the Israel Science Foundation (grant No. 1565/20).

M. Monastirsky, O. Azulay and A. Sintov are with the School of Mechanical Engineering, Tel-Aviv University, Israel. e-mail: {maximm, osherazulay}@mail.tau.ac.il, sintov1@tauex.tau.ac.il.

Digital Object Identifier (DOI): see top of this page.

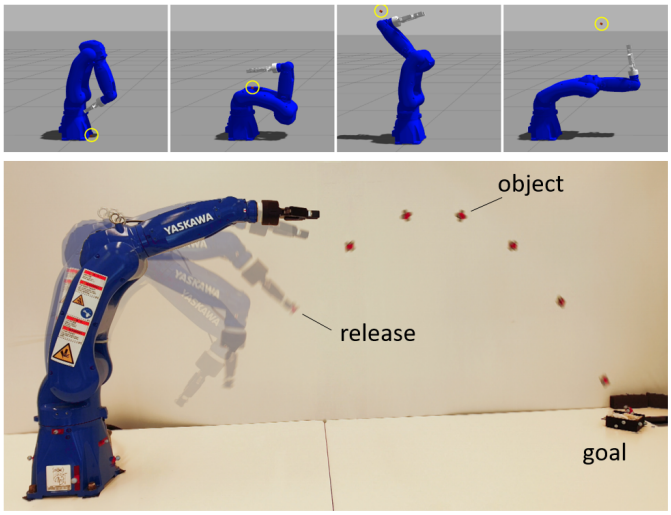

Fig. 1. (Top row) A simulated robotic arm is used to collect throw trajectories acquired by random motions and arbitrary object release time. Four random throws are shown where the object is marked with a yellow circle for better visibility. (Bottom) A Decision Transformer (DT), trained off-line with the simulated data, is shown to be able to extrapolate and generalize to out-of-distribution goals such that a real robot is able to accurately throw to desired ones.

Consequently, only few demonstrated such capabilities in limited scenarios [5].

RL has been successful in many complex simulated tasks including Atari video games [6] and physics-engine environments [7], [8] where the data is acquired at a lower cost [9]. However, training RL policies on real robots is a tedious and time consuming task [10]. Hence, the lack of extensive RL work on the object-throwing problem can be explained by the logistic requirement for a large amount of real throws and a reset mechanism to facilitate the collection. The robot may be required to work for a very long time. More important, the robot has no prior on how to throw and random actions may pose danger and cause damage. Simulation-based learning, on the other hand, provides a safe and cost-effective way to collect data through interactions with the environment. However, simulations rarely capture reality and the trained policies are usually poorly transferred [11].

Classic online RL algorithms require to continuously apply and update the current policy on the robot and collect on-policy data. This online setting is usually time consuming and very sample inefficient. Hence, it may be practical in a simulation environment, but many real robotic applications do not have the privilege of applying such online training procedures. On the other hand, offline RL algorithms, such as Deep Q-Learning and Deep Deterministic Policy Gradient

(DDPG) [7], require data that is correlated to the distribution of the current policy and are unable to extrapolate to new out-of-distribution scenarios [12]. In recent years, significant advancements in natural language processing and vision have been credited to the development of the Transformer architecture [13], [14]. In particular, an autoregressive model termed Generative Pre-trained Transformer (GPT) [15] is responsible for significant breakthrough in text-to-image [16] and language models [17]. The Transformers were recently taken to the world of RL in the form of *Decision Transformers* (DT) [18]. In DT, RL is considered as sequence modeling problem while completely eliminating the need for bootstrapping for long term credit assignment typically done in temporal-difference learning. Hence, a policy can be learned from offline and off-policy examples. Experiments in simulated environments have shown some capability for out-of-distribution learning [16]. Yet and to the best of the authors' knowledge, DT has not been evaluated on real world robots and sim-to-real transfer.

In this work, we investigate the use of DT for multi-goal object throwing with a robotic arm and its ability to reduce the required number of real-world samples. In particular, we are interested in investigating the sim-to-real ability of DT to generalize from a simulation-based model to the real world. Unlike prior work, no prior knowledge on how to throw nor a particular object release time are given to the simulation. Consequently, the sequence of actions for a throw motion is fully learned and is not of constant length. In addition, the simulation is not required to be tuned to the dynamic parameters of the real system and arbitrary values can be used. By using a simulation, the model is able to explore various motions without any risk. We also observe the augmentation of the simulated trajectories for data efficiency by using the Hindsight Experience Replay (HER) [19]. Then, only a handful of off-line recorded real throws is required in order to fine-tune the model yielding accurate real-world performance. The training data recorded off-line from simulation via random actions is shown to be of a short range distribution. However, the model exhibits ability, both in simulation and real world, to accurately throw to out-of distribution goals.

To summarize, this work shows that a real robotic arm can successfully learn a dynamically complex task by adopting the DT architecture while dramatically improving sample efficiency. Also, no visual perception is used in the process and control is based solely on the joint state of the robotic arm. We also introduce the first integration of HER with DT to exploit arbitrary throws. The results expose the unique abilities of the DT to transfer from an arbitrary simulation and extrapolate using out-of-distribution training data. This has implications to other systems beyond the throwing problem. To the best of the author's knowledge, this is the first implementation and experimental analysis of sim-to-real with DT and using DT for the throwing problem.

II. Related Work

Throwing with robots. The throwing problem with pure analytical models has been widely addressed [20]. In [21], torque control was proposed in order to throw a ball with an elastic manipulator. A different work provided a method to optimize the shape of an end-effector along with a test-case of planar object throwing [2]. Another work, on the other hand, focused on the parametrization and motion planning of a throwing motion [3]. However, these require knowledge of the dynamic properties of the system which are usually difficult to estimate. As a result, the approaches are not suitable for unstructured environments with various uncertainties and may exhibit low accuracy.

Not much work has studied the use of modern machine-learning approaches for throwing. Early work used motor primitives and meta-parameters learning with RL [5]. Nevertheless, the method requires some prior understanding of a throwing model and dynamic parameters. Later work used a deep neural-network to map an image state observation into a sequence of motor activations [22]. The approach was demonstrated over a ball throwing scenario. In both of these implementations, no actual throwing performance evaluation was provided. In a more recent study, a robot has learned to rapidly pick-up and throw objects based on image observations [4]. The method consists of predicting the release velocity using a physics-based controller of an ideal ballistic motion. To compensate for the shortcomings of the physics-based controller, the throwing module includes a regression neural-network that predicts a residual on top of the estimated release velocity. Hence, the method is based on a know-how throwing prior and requires a significant amount real-world samples. In contrast, our proposed method does not require any prior on how to throw and only a handful of real throws are needed.

Sim-to-real. Learning a policy solely from a simulation and deploying it to the real world is considered a hard challenge. Such problem is commonly referred as the *Reality Gap* or *sim-to-real* (simulation to reality). Many approaches have been proposed for bridging the reality gap. Early approach suggested adding noise to the simulation [23]. More recently, domain randomization was proposed where various properties in an existing simulation are constantly changed [9]. Similarly, dynamics randomization was proposed to randomly sample dynamic properties (e.g., robot link mass, damping and friction) in the simulator during training [11]. Consequently, the policy is able to adapt to uncertainties that may emerge when transferring to the real system. Such approach, however, requires full knowledge of the different dynamic parameters in the model, and can be time-consuming since the policy must experience a large variance of dynamic possibilities. All of these approaches require the formation of a physics-engine based simulation that is sufficiently close to the real system and environment. Closing the reality gap is not easy and collecting real world data is almost always inevitable. In this work, we show that DT can provide an efficient sim-to-real transfer from a simulation with arbitrary dynamic parameters.

Transformers and attention in Reinforcement Learning. Although Transformers have shown great advancements in language and vision in recent years, yet their impact on RL is relatively small. Some work has been done with combining transformers and attention mechanisms in RL [24]–[26]. However, such combination acts only as an additional mechanisms to the existing actor-critic framework. As mentioned before,

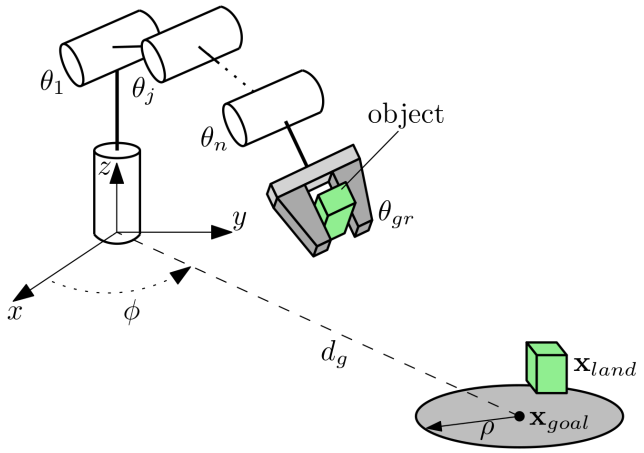

Fig. 2. Illustration of the robot with the object and goal.

a recent study re-framed the RL problem as a time sequence problem and used the DT architecture alone to predict actions [18]. In further work, the transformer was used to model distributions over trajectories followed by beam search as a planning algorithm to find the optimal trajectory [27]. Empirical evidence suggests that a transformer can model a wide distribution of behaviors, enabling better generalization and transfer [16]. This allows the DT to work well in an offline setting, a task that is traditionally challenging due to error propagation and value overestimation [28]. The DT framework is adopted in this work for further study in the context of real-world object throwing and sim-to-real.

III. Method Overview

A. Problem Formulation

State. An n degrees-of-freedom robot is given as illustrated in Figure 2. Let $\mathbf{s} \in \mathcal{S}$ be the state of the robot where $\mathcal{S} \subset \mathbb{R}^{n+1}$. As such, a state $\mathbf{s} = (\theta_1, \ldots, \theta_n, \theta_{gr})$ is comprised of robot actuator angles $\theta_1, \ldots, \theta_n$ and the binary state of the gripper $\theta_{gr} \in \{0, 1\}$ indicating closed (1) or open (0). In this form, a state $\mathbf{s}_t$ only includes current positions while missing velocity and acceleration information. Hence, determining the next state $\mathbf{s}_{t+1}$ may require a sequence of past states.

Goal. The aim of the robotic arm is to throw a grasped object to a desired goal $\mathbf{x}_{goal} \in \mathbb{R}^2$. Goal $\mathbf{x}_{goal} = (x_g, y_g)$ is a position on some horizontal plane in the vicinity of the robot with respect to its base. Consequently, the throwing distance is given by $d_g = \|\mathbf{x}_{goal}\|$ and is an input to the DT. For safety reasons, the throwing direction is determined analytically by $\phi = \arctan 2(x_g, y_g)$.

Action. Let $\mathbf{a}_t \in \mathcal{A}$ be an action of the system at time t where $\mathcal{A} \subset \mathbb{R}^{n+1}$. Hence, an action is composed of actuator velocities $\omega_1, \ldots, \omega_n$ for the arm, and opening or closing command $a_{gr} \in [0, 1]$ to the gripper. The gripper is initialized while closed on the object, i.e., $a_{gr} = 1$. Once condition $a_{gr} \leq \tau$ is satisfied for some pre-defined threshold $\tau > 0$, the gripper opens.

Reward. A sparse reward function is defined for a robot throw in the form:

$$\mathcal{R}(\mathbf{s}_t, \mathbf{a}_t) = \begin{cases} 1, & \theta_{gr} = 0 \wedge \|\mathbf{x}_{land} - \mathbf{x}_{goal}\| \leq \rho, \\ -1, & \theta_{gr} = 0 \wedge \|\mathbf{x}_{land} - \mathbf{x}_{goal}\| > \rho, \\ 0, & \text{otherwise}, \end{cases} \quad (1)$$

where $\rho > 0$ and $\mathbf{x}_{land}$ is the actual landing position of the object. A throw is considered successful if the first landing point $\mathbf{x}_{land}$ is inside a circle of radius ρ around the goal position. The robot is penalized if the object did not land in the circle. We note that non-sparse rewards did not provide sufficient results in preliminary work and as indicated in [19].

The system can be described by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where $\mathcal{P} = P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t-1}, \mathbf{a}_{t-1}, \ldots)$ is the transition probability function of the system. A traditional RL algorithm requires to acquire a trajectory $\{\mathbf{s}_0, \mathbf{a}_0, r_0, \ldots, \mathbf{s}_T, \mathbf{a}_T, r_T\}$, for $r_t = \mathcal{R}(\mathbf{s}_t, \mathbf{a}_t)$, that maximizes the expected reward $\mathbb{E}\left[\sum_{t=0}^{T} r_t\right]$. In DT, on the other hand, pre-recorded roll-outs are used for off-line training. Hence, one can only find a trajectory that produces a desired reward as discussed next.

B. Decision Transformers

The Transformer was introduced by Vaswani et al. [13] to efficiently model sequential data. It consists of an encoder and decoder pair, containing stacked self-attention and cross-attention layers, respectively, with residual connections. Sequential data is the input to the transformer in which each element within it is termed a *token*. Each token is embedded through a linear layer. Furthermore, positional encoding is then added to the embedded token. The Transformer inputs m embeddings and outputs m embeddings preserving input dimensions. A GPT architecture later introduced some changes to the original Transformer by utilizing only the decoder part of the transformer and applying causal masking [15]. Consequently, the Transformer is forced to take into account only previous tokens in the sequence instead of the whole sequence. Thus, enabling autoregressive generation.

The DT, based on the GPT architecture, is illustrated in Figure 3. It is inputted with w last time-steps yielding a total of $3w$ tokens $\{\hat{R}_0, \mathbf{s}_0, \mathbf{a}_0, \ldots, \hat{R}_w, \mathbf{s}_w, \mathbf{a}_w\}$ where $\hat{R}_i$ is the *reward-to-go* [18]. Token embedding is performed with a linear layer for each modality. After the embedding and similar to positional encoding, a time-embedding is added to the embedded tokens and further fed into the DT. During training, a full recorded trajectory τ is inputted into the DT. The DT predicts action $\mathbf{a}_t$ for each timestep t in the trajectory based solely on tokens in the same or previous timesteps. For each predicted action $\tilde{\mathbf{a}}_t$, a loss is calculated comparing between the predicted action $\tilde{\mathbf{a}}_t$ and the actual action $\mathbf{a}_t$ from the recorded trajectory. The auxiliary loss is the sum of the temporal losses and is given by

$$\mathcal{L}_{DT} = \sum_{t=1}^{T} \text{loss}(\tilde{\mathbf{a}}_t, \mathbf{a}_t). \quad (2)$$

The DT network is trained by back-propagation with a set of pre-recorded trajectories to minimize $\mathcal{L}_{DT}$.

During inference and at time t, the DT predicts the next action $\tilde{\mathbf{a}}_{t+1}$ based on all previous tokens $\{\hat{R}_t, \mathbf{s}_t, \mathbf{a}_t, \hat{R}_{t-1}, \mathbf{s}_{t-1}, \mathbf{a}_{t-1}, \ldots\}$. Action $\mathbf{a}_{t+1}$ is then exerted

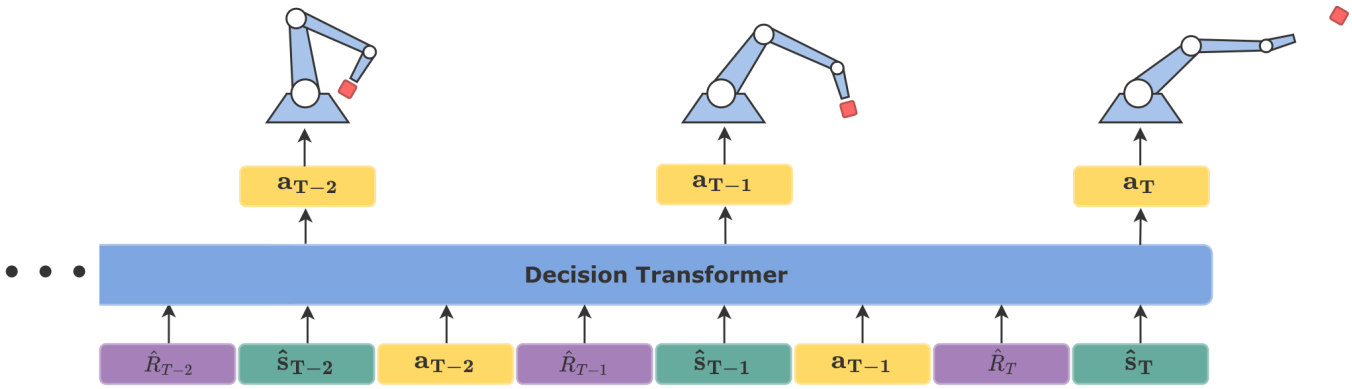

Fig. 3. Illustration of the Decision Transformer pipeline. In a given timestep, the DT is inputted with previous rewards-to-go, states, actions and current reward-to-go and state. It then predicts the next action to be taken in order to achieve the desired reward. After the predicted action is executed by the robot, we observe the next state and calculate the new reward-to-go based on the received reward.

on the robot followed by observing the next state $\mathbf{s}_{t+1}$ and updating the reward-to-go $\hat{R}_{t+1}$. This process is repeated until reaching the goal.

C. Data Collection and Augmentation

Training data is collected from a simulated environment where the kinematics of the robotic arm are modeled. However, dynamic parameters, such as link inertia and mass, are chosen arbitrarily and may be very different from the values of the real robot. We assume that no prior motion of how to throw exists. Hence, temporally correlated noise is injected into the actuators of the simulated arm resulting in efficient exploration through random motion. To generate such noise, the Ornstein-Uhlenbeck process [29] is used in the form

$$x_{k+1} = (\mu - x_k)\theta\Delta t + \sigma\varepsilon_k\sqrt{\Delta t} \tag{3}$$

where x_k is the noise at time-step k, Δt is the sample time, ε_k is a normal noise $\varepsilon_k \sim \mathcal{N}(0,1)$ and, θ, μ and σ are process parameters. At the beginning of each throw trajectory, a random goal is selected. In addition, a random time-step is chosen for when the gripper will open and release the object.

In this work, we explore the benefits of data augmentation based on the Hindsight Experience Replay (HER) [19]. Given a trajectory where the object landed at position $\mathbf{x}_{land}$, K_{her} samples are generated with the same trajectory while their corresponding goals are randomly picked inside a circle of radius 2ρ around the landing spot $\mathbf{x}_{land}$. For $K_{her} = 0$, we always pick the landing spot as a goal. For $K_{her} > 0$, goals with distance $[0, \rho]$ and $(\rho, 2\rho]$ from $\mathbf{x}_{land}$ are considered success and miss, respectively. This is done in order to create a balanced dataset of successful and unsuccessful throws, and to diversify the variance of rewards in the training.

With the above, a recorded trajectory will be of the form

$$\tau_i = \{\mathbf{s}_0, \mathbf{a}_0, r_0 \ldots, \mathbf{s}_{T_i}, \mathbf{a}_{T_i}, r_{T_i}\} \tag{4}$$

where T_i is the length of the motion and $\mathbf{s}_{T_i}$ is the state in which the object was released. Trajectory τ_i is accompanied with the goal $\mathbf{x}_{goal}$ determined in the augmentation. The rewards included in the trajectory are updated based on (1) and indicate whether the goal was successfully reached. Finally, a dataset of M trajectories $\mathcal{P} = \{\tau_1, \ldots, \tau_M\}$ is obtained for generating DT trajectories and training as discussed next.

D. DT Trajectories

As mentioned above, DT does not maximize expected reward but instead produces a sequence of actions that should yield a specifically desired reward. The desired reward is inputted into the DT, and must be updated on the fly once any reward is awarded. Therefore, we define the *reward-to-go* token at time t as

$$\hat{R}_t = \sum_{t'=t}^{T} r_{t'}. \tag{5}$$

Furthermore and since this is a multi-goal problem, the DT must receive the desired goal for which to generate a sequence of actions. Hence, the desired goal distance d_g is concatenated to the state, creating a state-goal token $\hat{\mathbf{s}}_t$ defined as

$$\hat{\mathbf{s}}_t = (\mathbf{s}_t \| d_g), \tag{6}$$

where $\|$ denotes concatenation. The last token will be the action as defined before. With the above said, each trajectory $\tau_i \in \mathcal{P}$ is reformulated to a DT trajectory consisting of rewards-to-go, state-goals and actions in the form

$$\tau_{DT_i} = (\hat{R}_0, \hat{\mathbf{s}}_0, \mathbf{a}_0, ..., \hat{R}_{T_i}, \hat{\mathbf{s}}_{T_i}, \mathbf{a}_{T_i}). \tag{7}$$

The DT training set is now $\mathcal{P}_{DT} = \{\tau_{DT_1}, \ldots, \tau_{DT_M}\}$. In this form, the DT can autoregressively train and generate new actions on the fly.

E. Sim2Real Adaptation

The strength of training the DT in simulation is the ability to learn how to throw without any prior while avoiding physical risks. Such process over a real robot is very dangerous and cannot be done. Nevertheless, the robot learns the general motions in simulations while the reality gap prevents direct transfer to the real robot. In order to fully transfer our model to the real robot, the simulation-trained DT is fine-tuned with a small number of throws collected from the real robot. To do so, we exert the simulation-trained DT on the real robot in order to conduct real throws to random goals. For each throw, we multiply the actions $\{\mathbf{a}_0, \ldots, \mathbf{a}_T\}$ with a random gain $\alpha \sim \mathcal{U}(1, \alpha_{max})$. Multiplying the actions by α assists in exploring the real robot domain and in bridging the reality gap. We further augment the collected data as described in Section III-C. The new throw samples are used to re-train the

DT model and refine its weights for it to adapt to the new domain.

IV. EXPERIMENTS

A. Setup

Robotic Hardware. The proposed approach is experimented with a six-degrees-of-freedom Yaskawa Motoman GP8 industrial arm equipped with a Robotiq 2f-85 parallel gripper (Figure 1, bottom). Needless to say, training a policy entirely on such powerful robot without any prior of how to throw is extremely dangerous. Without loss of generality and for safety during testing, the throw motion is bounded to a plane perpendicular to the horizontal floor and, thus, the DT learns only to move the second, third and fifth joints. The direction of throw is, therefore, determined analytically according to (x_g, y_g) by ϕ and set by the first joint which is perpendicular to the floor. Similarly, the remaining joints are set constant to zero. Due to the dynamic specifications of the robotic arm, the throwing distance is bounded to $d_g \in [50, 200]$ centimeters. Furthermore, the gripper opens and releases the object when value a_{gr} of the current action is below a threshold τ. The value for τ is chosen to be the mean of all the gripper actions in the training dataset acquired in simulation. The communication frequency with the arm is 10Hz while the maximum trajectory length is set to 1 second yielding an upper bound of $T_i \leq 10$ timesteps for a trajectory. Furthermore, we allow the DT to access all previous steps at any given time along the trajectory. The system is controlled using the Robot Operating System (ROS). Videos of the experiments can be seen in the supplementary material.

Throwing Evaluation. Fine-tuning and initial evaluation is done by throwing a cube of size $1.5 \times 1.5 \times 1.5\ cm$ with mass of $15\ g$. Only for evaluation of the throwing performance in the real world, a motion capture system with eight OptiTrack Prime 41 cameras was used. Markers were attached on the base of the robot, on a target plate and on the thrown object. In this way, the robotic arm can automatically detect the goal $\mathbf{x}_{goal}$ relative to itself and throw the object to the target. The object landing position $\mathbf{x}_{land}$ is also detected by the cameras. A throw error is calculated by $e = \|\mathbf{x}_{goal} - \mathbf{x}_{land}\|$. We define a test procedure in which an object is thrown to 30 uniformly distributed pre-defined goals. The test accuracy is the mean $\frac{1}{30}\sum_{i=1}^{30} e_i$ of all throws.

Simulation. The same robot was modeled in the ROS-Gazebo physics engine. The dynamic properties of the simulated arm were chosen arbitrary while in the same scale of the real one. Throw data was collected with an object of the same size and mass as for the real system. Noise parameters were set to $\theta = 0.02$, $\mu = 0$ and $\sigma = 0.2$. In addition, data was collected with a success radius of $\rho = 0.5\ cm$. With these conditions, random trajectories were generated and recorded for training the DT. Since no prior of how to throw was given to the simulation, the recorded trajectories are out-of-distribution to feasible throws. Examples of these random non-feasible throws used for training are seen in Figure 1 (top row).

B. Model Training

The acquired simulation data is used to train a DT model as described in Section III. The model architecture and hyper-parameters were optimized to yield the best simulation scores. In particular, the DT architecture, based on GPT, was chosen to be with embedded dimension of 128, 1 hidden layer, 1 attention head and a ReLU activation function. Actions are predicted by including an additional linear layer at the DT output. A Tanh and Sigmoid activation functions are applied to the actuator velocities $\omega_1, \ldots, \omega_n$ and gripper command a_{gr}, respectively, from the outputted action vector $\mathbf{a}_t$. This DT model yields a total of 210,058 trainable parameters. For the DT model to predict actions, the loss function to be minimized in training was defined to be the sum of the Mean Square Error (MSE) on actuator velocities and Binary Cross Entropy (BCE) on the gripper action. The model was trained using the AdamW optimizer with a learning rate of 10^{-4}, weight decay of 10^{-4}, dropout of 0.1 and a linear warm-up for the first 10^4 gradient steps. The model was trained for 100 epochs.

C. Simulation Results

We first analyze the performance of the DT in the simulation environment.

Data quantity and augmentation. We begin by studying the performance of the DT with regards to the amount of simulated data and augmentation parameter K_{her}. A set of 1,000 random trajectories was collected in simulation as described above. The DT was trained multiple times with an increasing number M of trajectories. The baseline approach is training the DT directly with the raw data without including HER and augmentation. Hence, the DT is trained with trajectories labeled by randomly generated goals and rewards given accordingly for success or miss of these goals (0.005 probability of a success). Furthermore, for a specific number of trajectories, training was performed over four different HER augmentation parameters $K_{her} = \{0, 1, 3, 5\}$. For $K_{her} = 0$, the actual landing position $\mathbf{x}_{land}$ was set as the goal $\mathbf{x}_{goal}$ of the corresponding trajectory. With data augmentation and when $K_{her} > 0$, K_{her} additional trajectories were generated with sampled goals in the vicinity of $\mathbf{x}_{land}$ as described in Section III-C.

Figure 4 presents throw accuracy results with regards to the number of training trajectories and K_{her}. First, the model reaches saturation with above 500 throws. In addition, HER augmentation is shown to be significant in increasing the accuracy. Nevertheless, augmentation with $K_{her} > 0$ (i.e. adding more trajectories beyond the modified one) exhibits no significant improvement over accuracy. Moreover, various values for the goal radius in the range $\rho \in [0.5, 20]\ cm$ were tested while not providing significant change in performance.

Data Sparsity. We next analyze the ability of the DT to generalize and extrapolate to out-of-distribution goals. Figure 5 shows the distribution of 500 collected random trajectories (in black) used for training with respect to the throwing distance d_g and when $K_{her} = 0$. Examples of such throws are seen in Figure 1 (top row). Only 14% of the random throws resulted in the object landing within our desired working range

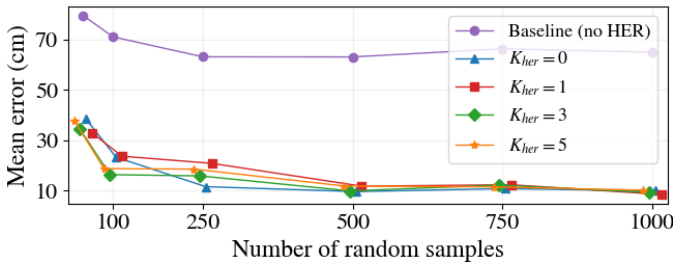

Fig. 4. Throw accuracy with regards to the number of training trajectories and HER augmentation parameter K_{her}.

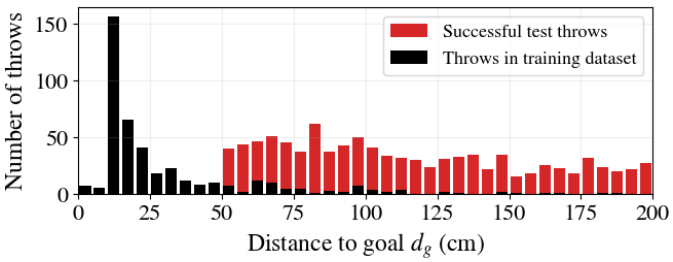

Fig. 5. Distribution of throws in the training dataset (black) compared to successful test throws (red). Despite the sparsity of throws for $d_g > 50cm$ in the training dataset, DT manages to make accurate throws to out-of-distribution goals.

$d_g \in [50, 200]$. On the other hand, Figure 5 also shows (in red) the distribution of 1,000 successful throws (i.e., hit within 1cm radius) to the desired range $d_g \in [50, 200]$ while using the trained DT policy. Hence and during test time, the DT manages to hit out-of-distribution goals within this range with high success rate. Figure 6 shows an example of a successful simulated throw where the goal is out-of-distribution to the training trajectories.

Baseline Comparison. We compare our results to DDPG [7], Behavioral Cloning (BC) [30] and Residual Physics (RP) based on [4]. DDPG is a common Temporal-Difference learning algorithm for continuous control. Similar to DT, BC is a supervised learning algorithm. The objective of the comparison is to analyze whether common RL approaches, i.e., DDPG and BC, can match the throw accuracy and data efficiency of the DT. Contrary to DT, RP is based on prior throw know-how based on a simple ballistic motion. A residual network is used to compensate for model uncertainties. A similar setup and training process were performed for all methods. All methods were trained and tested in simulation.

The hyper-parameters of the models were optimized to converge and reach minimal loss. For DDPG, the actor was

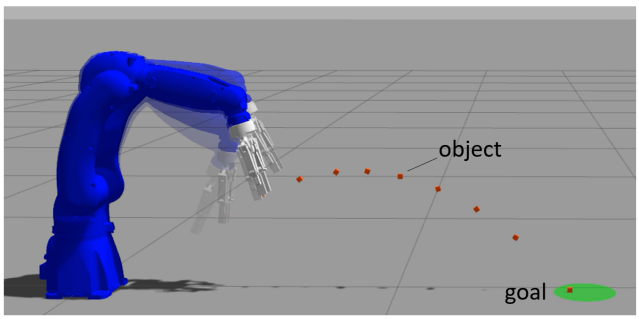

Fig. 6. Deployment example of the trained DT on the simulated robot for throwing the object to a desired target.

TABLE I
BASELINE COMPARISON

Method	Number of throws	Mean error (cm)	Stability (cm^2)
Random throws	500	121±136	-
BC	500 Random	39.3±32.8	6
DDPG	6,360 On-policy	17.8±14.0	3,020
RP [4]	500 Model-based	22.1±20.3	96
DT	500 Random	8.2±6.4	67

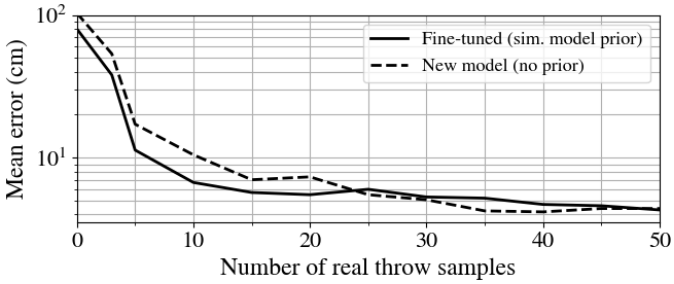

Fig. 7. Number of real throws required for fine-tuning the DT model.

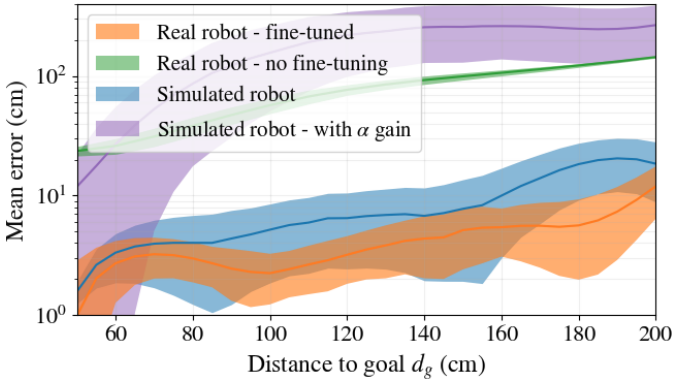

Fig. 8. Throw accuracy with regards to the distance of the goal d_g for (blue) simulated robot, (green) real robot with non fine-tuned DT and (orange) real robot with fine-tuned DT.

formed by a Multi-Layer Perceptron (MLP) with two layers and 64 neurons, each. Similarly, the critic is formed of two layers and 256 neurons, each. A buffer of 10,000 trajectories was used and trained for 100 gradient steps for every 40 trajectories executed. The learning parameters are as described in Section IV-B. Similarly, data augmentation is as described in Section III-C. Different K_{her} values were tested while the best convergence was achieved with $K_{her} = 7$. Furthermore, the optimal radius value for the reward (1) is $\rho = 2cm$. BC is implemented similarly to DT with only successful trajectories by relabeling goals with true landing positions and with $K_{her} = 0$. Failed trajectories, on the other hand, were not included as the policy would perform poorly. BC is implemented with an MLP of three layers comprising $\{64, 128, 256\}$ neurons. The current and ten past states are inputted to the MLP which, in turn, outputs the next action. RP is implemented by fixing the release position at a fixed angle of $45°$ as in [4]. Only the magnitude of the release velocity is controlled and is a function of the desired goal distance according to ballistic equations. After collecting 500 throws to random goals (with accuracy of $34.6cm$), a residual model based on an MLP is trained with the throw data to compensate

Fig. 9. Seven test objects including (a) cuboid (with reflective markers, (b) squeeze ball, (c) cylinder, (d) box (e) sand ball, (f) copper coil and (g) pencil.

Object	Mass (g)	Dimensions (cm)	CM grasp Success Rate for dist. 80cm	CM grasp Success Rate for dist. 130cm	CM grasp Success Rate for dist. 180cm	Non-CM grasp Success Rate for dist. 80cm	Non-CM grasp Success Rate for dist. 130cm	Non-CM grasp Success Rate for dist. 180cm
(a) Cuboid	15	$1.5 \times 1.5 \times 1.5$	100%	100%	100%	-	-	-
(b) Squeeze ball	25	radius 3	100%	100%	100%	100%	50%	50%
(c) Cylinder	50	radi. 2, heig. 8	100%	100%	100%	100%	60%	60%
(d) Box	120	$3 \times 6 \times 9$	100%	100%	100%	100%	40%	40%
(e) Sand ball	60	radius 2.5	100%	100%	100%	100%	0%	0%
(f) Copper coil	386	radi. 2.2, heig. 6.5	100%	100%	90%	90%	40%	0%
(g) Pencil	6	radi. 0.4, len. 18	100%	100%	100%	90%	50%	40%

Fig. 10. Throwing success rate out of 10 throws for seven test objects.

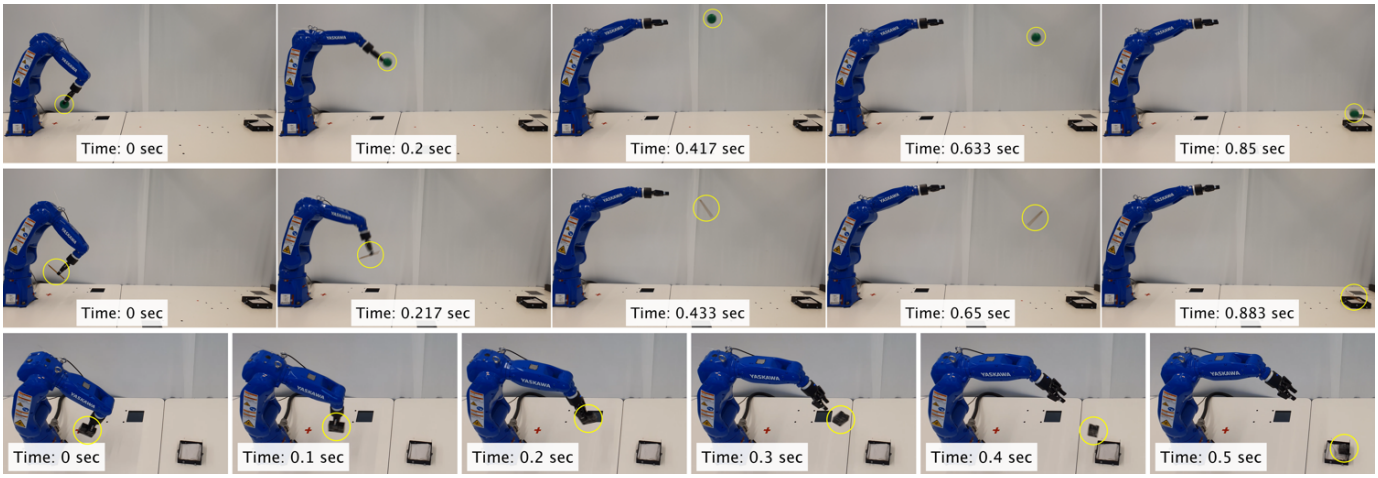

Fig. 11. Snapshots of four throws (from top to bottom): squeeze ball ($d_g = 180cm$), pencil ($d_g = 180cm$)and box ($d_g = 90cm$). Objects are marked with a yellow circle for better visualization.

for uncertainties in the ballistic model. It is composed of two layers and 32 neurons each. The model receives the desired goal distance and outputs correction residual to add to the release velocity.

Table I presents the comparative results between the three methods along with another baseline of sole random throws to arbitrary goals without any policy. We note that in the test, goals are set in out-of-distribution distances. Hence and in terms of accuracy, DT outperforms with the best accuracy and exhibits the ability to extrapolate to farther distances. On the other hand, DDPG, BC and RP provide poor results while better than random throws. While DDPG has better results than BC, it requires a large amount of training data. Similarly, RP requires much more data in order to compensate for uncertainties not included in the ballistic model. The learning stability was also evaluated by measuring the variance of the evaluation score across the epochs. Results show that DT stability is lower by an order of magnitude than DDPG.

D. Real Robot Results

Fine-tuning with Real Throws. When transferring the pre-trained DT model to the real robot, the yielded mean error for test throws is 80cm. As described in Section III-E, the simulation model acts as a prior and real throws are required in order to fine-tune it. Hence, a dataset of real throws was collected by generating a set of random goals and attempting to throw to them using the pre-trained model and when $\alpha_{max} = 3.0$. After applying HER, the prior DT model is refined. We next analyze the required number of real throws to reach accurate performance. Figure 7 presents the mean throw error with regards to the number of real throws. For comparison, we also train a new DT model without prior training in simulation and with only the real throws. Real throws, in this case, are not random throws but expert demonstrations that were recorded while exerting the simulation-trained policy. These could not be acquired with random actions as few hundreds would be needed (see Figure 4) while posing danger. Results show significant accuracy improvement with only a handful of real throws. For instance, fine-tuning with only 5 and 10 throws reduces error to less than 11cm or 6cm, respectively. With more throws, accuracy keeps improving while improvement rate declines. The error with 50 throws is 4.3cm. Results with only expert throws are similar and show ability to train a new model with only few good samples. We note that deploying and fine-tuning the simulation-trained BC model on the real robot failed due to disordered and dangerous motions.

Real Robot Performance. The best performing model from the previous section is chosen for further analysis. Figure 8 shows throw accuracy for the simulated and real robots with regards to the distance to the goal d_g. For reference, we include simulated results with scaling of the same α as in the above fine-tuning. First, the real robot performs very poorly with a non fine-tuned DT emphasizing the significant reality gap. However, with only few real throw samples for fine-tuning, the DT model achieves better accuracy than the simulation with a mean error of 4.3cm. For both domains, higher errors are apparent for targets farther than 170cm as the throw is more complex. On the other hand, throwing to a 15×15 cm target box positioned in distance of up to 180cm would yield

approximately 100% success rate.

Generalization To Other Objects. We next test the ability of the DT to generalize to other objects without additional fine-tuning. Along with the cuboid, we test an additional six objects, seen in Figure 9, including: squeeze ball, cylinder, box, sand ball, copper coil and pencil. Each object is tested for success rate in throwing to short (80 cm), medium (130 cm) and long (180 cm) distance goals. For each goal, the object is thrown 10 times. A success is defined to be hitting a rectangular plate of size $15 \times 15\ cm$. Table 10 summarizes the success rate for the throws along with physical properties of the objects. For better understanding, we make a distinction between throws where the initial grasp is on or off the Center-of-Mass (CM) of the object. Off-set to the CM is randomly placed in each throw. When grasping the object on its CM, the model generalizes well and the success rate is high for all objects. However, throws with non-CM grasps to medium and long distances have lower success rate. While having some ability to hit the goal, the model was not trained to compensate for object elongation yielding throw bias. For such compensation feature, the model must have a mean to measure the location of the CM, which is not available in this work. We leave this to future work.

V. Conclusion

In this work, we have proposed a data-efficient framework for object throwing with DT. A policy is trained off-line using data recorded in simulation through randomized actions without any prior on how to throw. In addition, the simulation consists of arbitrary physical parameters without any pre-tuning to the real robot. Then, the DT is fine-tuned with only several real throw examples. In particular, a set of 5-10 throws is sufficient to provide throw accuracy of less than 10cm. Furthermore, experiments on a set of different object yielded high success rate. However, when grasping an object off its CM, the success rate declines. Future work may consider addressing this by including visual feedback or a Force/Torque sensor that can embed grasp off-sets from object CM. Also, Promt-DT [31] can be used to ease the sim-to-real transfer by adding real trajectories as prompt to the simulated trained DT.

References

[1] F. Raptopoulos, M. Koskinopoulou, and M. Maniadakis, "Robotic pick-and-toss facilitates urban waste sorting," in *IEEE Inter. Conference on Automation Science and Engineering*, 2020, pp. 1149–1154.

[2] O. Taylor and A. Rodriguez, "Optimal shape and motion planning for dynamic planar manipulation," *Autonomous Robots*, vol. 43, no. 2, pp. 327–344, 2019.

[3] A. Sintov and A. Shapiro, "A stochastic dynamic motion planning algorithm for object-throwing," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 2475–2480.

[4] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1307–1319, 2020.

[5] J. Kober, E. Oztop, and J. Peters, "Reinforcement learning to adjust robot movements to new situations," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[8] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Inter. conference on machine learning*, 2016, pp. 1329–1338.

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

[10] V. Lim, H. Huang, L. Y. Chen, J. Wang, J. Ichnowski, D. Seita, M. Laskey, and K. Goldberg, "Planar robot casting with real2sim2real self-supervised learning," *arXiv preprint arXiv:2111.04814*, 2021.

[11] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *IEEE Inter. conf. on Robotics and Automation*, 2018, pp. 3803–3810.

[12] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *International conference on machine learning*. PMLR, 2019, pp. 2052–2062.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[14] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[15] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[16] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-shot text-to-image generation," in *International Conference on Machine Learning*, 2021, pp. 8821–8831.

[17] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[18] 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," *Advances in neural information processing systems*, vol. 34, pp. 15 084–15 097, 2021.

[19] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," *Adv. in neural info. proc. sys.*, 2017.

[20] T. Senoo, A. Namiki, and M. Ishikawa, "High-speed throwing motion based on kinetic chain approach," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 3206–3211.

[21] Y. Gai, Y. Kobayashi, Y. Hoshino, and T. Emaru, "Motion control of a ball throwing robot with a flexible robotic arm," *Inter. Journal of Computer and Information Eng.*, vol. 7, no. 7, pp. 937–945, 2013.

[22] A. Ghadirzadeh, A. Maki, D. Kragic, and M. Björkman, "Deep predictive policy training using reinforcement learning," in *IEEE/RSJ Inter. Conf. on Intelligent Robots and Systems*, 2017, pp. 2351–2358.

[23] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *European Conference on Artificial Life*, 1995, pp. 704–720.

[24] E. Parisotto, F. Song, J. Rae, R. Pascanu, C. Gulcehre, S. Jayakumar, M. Jaderberg, R. L. Kaufman, A. Clark, S. Noury *et al.*, "Stabilizing transformers for reinforcement learning," in *International conference on machine learning*. PMLR, 2020, pp. 7487–7498.

[25] V. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. Reichert, T. Lillicrap, E. Lockhart *et al.*, "Relational deep reinforcement learning," *arXiv preprint arXiv:1806.01830*, 2018.

[26] S. Ritter, R. Faulkner, L. Sartran, A. Santoro, M. Botvinick, and D. Raposo, "Rapid task-solving in novel environments," *arXiv preprint arXiv:2006.03662*, 2020.

[27] M. Janner, Q. Li, and S. Levine, "Offline reinforcement learning as one big sequence modeling problem," *Advances in neural information processing systems*, vol. 34, pp. 1273–1286, 2021.

[28] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint arXiv:2005.01643*, 2020.

[29] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Physical review*, vol. 36, no. 5, p. 823, 1930.

[30] F. Torabi, G. Warnell, and P. Stone, "Behavioral cloning from observation," in *International Joint Conf. on AI*, 2018, p. 4950–4957.

[31] M. Xu, Y. Shen, S. Zhang, Y. Lu, D. Zhao, J. Tenenbaum, and C. Gan, "Prompting decision transformer for few-shot policy generalization," in *Int. Conf. on Machine Learning*. PMLR, 2022, pp. 24 631–24 645.