Hierarchical Decision Transformer

André Correia and Luís A. Alexandre

***Abstract*— Sequence models in reinforcement learning require task knowledge to estimate the task policy. This paper presents the hierarchical decision transformer (HDT). HDT is a hierarchical behavior cloning algorithm that improves the performance of transformer methods in imitation learning, improving their robustness to tasks with longer episodes and/or sparse rewards, without requiring task knowledge or user interaction currently present in the state-of-the-art. The high-level mechanism guides the low-level controller through the task by selecting sub-goals for the latter to reach. This sequence replaces the returns-to-go of previous methods, improving its performance overall, especially in tasks with longer episodes and scarcer rewards. We validate our method in multiple tasks of OpenAI Gym, D4RL, and RoboMimic benchmarks. Our method outperforms the baselines in twenty three out of thirty one settings of varied horizons and reward frequencies without prior task knowledge, showing the advantages of the hierarchical model approach for learning from demonstrations using a sequence model. We also evaluate the method on a reaching task on a physical robot.**

I. INTRODUCTION

Reinforcement learning (RL) [1] has been successfully applied to a diverse set of robotic tasks, such as pushing and grasping objects, path finding and locomotion [2], [3], [4]. However, RL algorithms require large amounts of exploration to successfully learn policies in large state and action spaces, which is costly in real-world applications. Additionally, learning a policy through RL requires specifying a reward function that is specifically designed to assist in exploration. Furthermore, RL often relies on dense rewards and struggles in environments with sparse rewards. However, it is often difficult to design such reward functions for complex real-world tasks.

We can alleviate the above-mentioned problems by leveraging prior experience collected by an expert demonstrator as an extra supervision signal using Imitation Learning (IL). Behavioral cloning (BC) formulates the problem of learning the task as a supervised learning problem to imitate an expert demonstrator. Because the demonstrator was optimizing a reward function, the data set can be used in lieu of a reward function. However, the performance of the agent is dependent on the quality of the data set which is often sub-optimal, unstructured, and diverse due to the difficulty of collecting data. One way to utilize unstructured prior experience is to identify key states that contributed to the success of the trajectory. Hierarchical models are used to solve this issue, where a high-level model identifies sub-goals that the low-level model must reach, guiding it to the final goal and consequently solving the task.

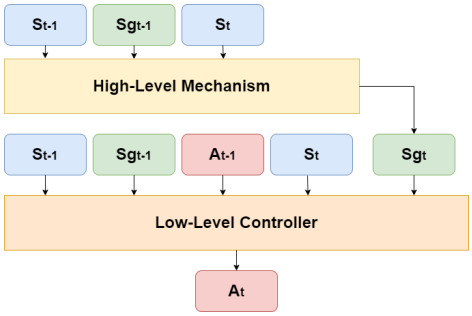

Fig. 1. HDT framework. The high-level mechanism guides the low-level controller through the task by selecting sub-goal states, based on the history of sub-goals and states. The low-level controller is conditioned on the history of past states, sub-goals, and actions to select the appropriate action. By reaching each sub-goal, the controller gets closer to completing the task.

Transformer models have revolutionized several machine learning fields and have recently been applied to RL by reformulating it as a sequence modeling problem in the form of decision transformers (DT) [5]. Instead of receiving a single observation, the agent receives a sequence of past interactions allowing it to make a more informed decision. However, the reward sequence is what guides the agent through the task. The performance of these models relies on the specification of the value of desired accumulated rewards as well as dense reward functions. However, the value of the desired accumulated rewards is non-trivial to determine and can not be arbitrarily high. The goal of this work is to train a model to identify key sub-goal states from the data set that replace the reward sequence.

We propose the Hierarchical Decision Transformer (HDT), a dual decision transformer architecture that scales transformers in imitation learning to tasks with long episodes and removes the need for the specification of desired rewards by a user. We propose a method for selecting sub-goal states from the demonstration data set. The high-level transformer predicts key sub-goal states, identified from the demonstration data set, based on the sequence of prior states. The low-level transformer is conditioned on the sequence of sub-goals to perform the task. By successfully reaching each sub-goal, the agent completes the task. Unlike the state-of-the-art DT [5], HDT does not require any prior task knowledge and user interaction. We conduct experiments on several tasks from D4RL, OpenAI Gym, and RoboMimic benchmarks. Our results show that the performance of the

*This work was supported by NOVA LINCS (UIDB/04516/2020) with the financial support of 'FCT - Fundação para a Ciência e Tecnologia' and also through the research grant '2022.14197.BD'.

André Correia and Luís A. Alexandre are with NOVA LINCS, Universidade da Beira Interior, R. Marquês de Ávila e Bolama 6201-001, Covilhã, Portugal andre.correia@ubi.pt, luis.alexandre@ubi.pt

DT is dependent on the careful specification of desired rewards. We show that HDT successfully replaces the need for specifying desired rewards with the added benefit of an increase in performance, particularly in long tasks or environments with sparse rewards, when compared with the state-of-the-art.

Summary of Contributions:

1) We present the HDT, a dual transformer framework that enables offline learning from a large set of diverse and sub-optimal demonstrations by selectively selecting sub-goal states from the data set.
2) We present a method for sampling sub-goal states from a trajectory. Experiments show that the sequence of sub-goals can guide the agent through the task in place of the sequence of returns-to-go.
3) We evaluate the HDT on ten environments and data sets from the D4RL [6], OpenAI Gym [7] and RoboMimic [8] benchmarks. We also validate HDT on a reaching task using a physical robot. Our method outperforms the original DT baseline, especially in tasks with longer episodes, proving that the sub-goal sequence can replace the reward function. The code can be found at https://github.com/meowatthemoon/HDT.

II. Related Work

Imitation learning (IL) is a machine learning paradigm that allows robots to learn tasks from demonstrations performed by human experts and has been applied to multiple domains such as playing games, driving autonomously and robotics [9]. In this section, we provide a summary of some works related to ours. BC treats imitation learning as a supervised learning problem, where the problem maps observations into actions, and the training signal is given by how similar the actions are to the demonstrator's [10].

Hierarchical approaches learn a high-level planner and a low-level controller [11], [12]. The high-level planner finds a path built with sub-goal states that drive the agent towards the main task goal by conditioning the low-level controller to try to achieve each sub-goal. This extra guidance helps the agent learn in sparse reward environments and long tasks. Conditioning reinforcement learning and imitation learning approaches on goal observations improves sample efficiency.

Another approach is to learn an embedding space of skills from unstructured demonstrations [13] and then train the policy on the embedding space. Alternatively, complex tasks can be divided into smaller sub-tasks [14], [12]. Each of these new sub-tasks is learned from the demonstration data set and constitutes sub-goals for the agent to reach. [4] estimates a state distribution, to ensure the sub-goals are part of reachable states. Other approaches try to generalize new trajectories from the demonstration data using the insight that the trajectories intersect at certain states [15]. A particularly promising approach was proposed, using goal-conditioned policies at multiple layers of hierarchy for RL [16]. Alternatively, other methods generate sequences of sub-goals with a divide-and-conquer approach [17].

Sequence modeling with deep networks has evolved from LSTMs to Transformer architectures with self-attention [18]. The latter have revolutionized many natural language processing tasks. Recently, they have been applied to RL by re-formulating it as a sequence modeling problem [5], [19]. These treat reinforcement learning as a supervised learning paradigm that predicts action sequences from trajectories and task specification (e.g., target goal or returns), instead of traditionally learning Q-functions or policy gradients. DT [5] is a model-free context-conditioned policy, while the trajectory transformer [19] is used both as a policy and model.

III. Preliminaries

A. Reinforcement Learning

We consider learning in a Markov Decision Process (MDP) described by the tuple (S, A, P, R). The tuple consists of states $s \in S$, actions $a \in A$, the state transition function $P(s' \mid s, a)$ and a reward function $r = R(s, a)$. We use s_t, a_t, and $r_t = R(s_t, a_t)$ to denote the state, action, and reward at timestep t, respectively. With sparse rewards, optimizing the expected discounted reward using RL may be difficult. Because of this, we augment this MDP with sets of absorbing goal and sub-goal states $G \subset S$ and $Sg \subset S$. Where each goal state $g \in G$ is a state of the world in which the task is considered to be solved and $sg \in Sg$ is a valuable state of the world that contributes to the success of the trajectory. A trajectory is a sequence of length N of states, actions, and rewards: $\tau = (s_1, a_1, r_1, ..., s_N, a_N, r_N)$. The accumulated rewards of a trajectory R_τ with length N are: $\sum_{t=1}^{N} r_t$. At every step t, the agent observes a state s_t and queries the policy π to choose an action $a_t = \pi(s_t)$. The agent performs the action and observes the next state $s_{t+1} \sim P(s_{t+1} \mid s_t, a_t)$ and reward $r_t = R(s_t, a_t)$. The goal is to learn a policy π which produces trajectories τ which maximize the expected return $\mathbb{E}_\pi[R_\tau]$.

B. Behavior Cloning

In BC, instead of learning from experience by interacting with the environment, the agent learns solely from the trajectories present in the demonstration data set. This setting is harder because it removes the ability for agents to explore the state space and try different trajectories. We assume there is access to a data set of trajectories $D = \{\tau_1, ..., \tau_N\}$. In BC the agent is encouraged to select the action the demonstrator took for a given state in the data set. A common approach is to maximize the likelihood of actions in the demonstration, $\max \mathbb{E}_{(s,a)\sim D}\, log\pi(a \mid s)$. Although BC remains one of the simplest forms of learning tasks from demonstrations, it comes with several limitations such as compounding errors. We tackle this particular problem by using a sequence model. By providing the agent with a history of interactions instead of a single observation, we aim to reduce the compounding error of learning from limited data.

C. Decision Transformer

Transformers are a group of architectures that model sequential data [18]. These are encoder-decoder models built with blocks of self-attention layers with residual connections. Due to their success in many natural language processing tasks, and because RL learns from sequences of trajectories, [5] created the DT. DT transforms RL into a sequence objective problem. Instead of processing a single state transition, the policy receives a sequence of 3 types of input tokens: $(rtg_1, s_1, a_1, ..., rtg_N, s_N, a_N)$, corresponding to returns to go, states and actions. Additionally, the transformer receives a mask and the sequence of timesteps. The mask is a vector with as many elements as the length of the sequence. Each element in the mask indicates if the corresponding token should be hidden from the transformer. Transformers require information about the relative position of the tokens in the sequence. In DT, this information is provided by the sequence of timesteps.

The policy is trained using the MSE loss, in a similar manner to BC, to predict the demonstrator's action. DT feeds these trajectories through a GPT 2 [20] architecture. During training, the value of the returns-to-go at timestep t is the sum of future rewards in the trajectory $rtg_t = \sum_{t'=t}^{T} r_{t'}$. However, during the test time, the full trajectory is not known to calculate the sum. Because of this, a user specifies a value for the desired accumulated returns. We performed ablation studies in Sec. V, and the results show that the DT relies on the sequence returns-to-go to learn and that the value of the desired accumulated returns impacts the performance of the agent. However, determining a good value for the desired accumulated returns is done through trial and error. The original DT sets the value to the maximum accumulated rewards found in the data set. The results in Sec. V show that this is not optimal, because in some tasks, setting a lower value resulted in increased performance. Additionally, in tasks with sparse rewards, where the environment returns a reward of zero, the value of returns-to-go is not changed for concurrent transitions. The DT struggles to learn the task when the value of returns-to-go is static.

Our goal is to replace the sequence of returns-to-go with a different sequence that does not require human interaction and knowledge beforehand and is less reliant on frequent rewards. The sequence should guide the agent through the task and result in equal or higher performance than the original DT.

IV. Proposed Approach

A. Overview

We present the HDT, represented in Fig. 1. HDT is a hierarchical behavior cloning algorithm that improves the performance of transformer methods in imitation learning, improving their robustness to tasks with longer episodes and/or sparse rewards, without requiring task knowledge or user interaction currently present in the state-of-the-art.

Here we first provide an overview of HDT and the motivation of each component. We split the decision-making process of the agent into a high-level mechanism that sets sub-goal states for a low-level controller to try and reach. We use the GPT 2 [20] transformer architecture, the same used by the DT [5], for both the high-level mechanism and the low-level controller. The high-level mechanism is conditioned on the state sequence and aims to produce sub-goal states for the low-level controller to reach, guiding it through the task. The low-level controller is conditioned on the sequence of sub-goals produced by the high-level model to predict the correct action.

Algorithm 1 HDT training algorithm.

Input: Demonstration data set $D = \{\tau_1, ..., \tau_N\}$, batch size B, token size K, training iterations I, high-level mechanism π_ϕ, low-level controller π_θ, sub-goal selection method f.
for $i = 1$ **to** I **do**
  s, **a**, **t**, **sg**, **mask** $\leftarrow$ Sample B sequences from D.
  $sg' \leftarrow \pi_\phi(\mathbf{s}, \mathbf{sg}, \mathbf{t}, \mathbf{mask})$, obtain sub-goal predictions.
  Update ϕ by minimizing $L_\phi(sg, sg')$.
  $a' \leftarrow \pi_\theta(\mathbf{s}, \mathbf{sg}, \mathbf{a}, \mathbf{t}, \mathbf{mask})$, obtain action predictions.
  Update θ by minimizing $L_\theta(a, a')$.
end for
return π_θ, π_ϕ

The algorithm learns solely from demonstration data present in a data set. The data set is first processed, where for each state, a sub-goal state is selected from the trajectory using the sub-goal selection algorithm we present. The original transition is then augmented with the selected sub-goal state. Then, both the high-level mechanism and low-level controller are trained simultaneously using the sampled batches of sequences. The high-level mechanism receives sequences of the previous states and sub-goals and tries to predict the next sub-goal state in the sequence. The low-level controller receives sequences of the previous states, sub-goals, and actions, and tries to predict the next action in the sequence. Because both models are transformers, they also receive the sequence of time steps and the mask for positional encoding and to hide the future tokens from the decoder, respectively.

The complete training loop is provided in Algorithm 1. We encourage training both models simultaneously to avoid repeating steps, such as batch sampling. However, because both models do not rely on data produced by the other to learn, they can be trained sequentially. Next, we further explain each component.

B. Sub-Goal Selection

Our method requires sequences of states, actions, time steps, and sub-goals. To obtain these sequences, it processes the demonstration data set before training. The sequences of states, actions, and time steps can be directly extracted from the data set. However, the sub-goal sequences are not explicitly present in the data set and must be inferred. We define sub-goals sg in regards to the current state s_t as later states in the trajectory, which are highly valuable for the

agent to reach. These states should mark milestones in the trajectory which when reached sequentially, would make it highly probable that the agent successfully performs the task.

Using this definition, the sub-goals guide the agent through the task, meaning they have the same purpose as the returns-to-go. Therefore, we replace the returns-to-go sequences with the sub-goal sequences. This way, the user does not need to iterate over the value of the desired returns for each task in order for the method to perform. Additionally, it replaces null rewards in tasks with sparse rewards, consequently improving the learning process. We perform experiments to show that the original DT is dependent on the returns-to-go sequences to learn and that the value of the desired returns is important.

By our definition, a sub-goal state sg has a high value for the success of the trajectory. We can model this behavior by finding a future state s_j in the trajectory with high values of accumulated rewards since the current state $W(s_j) = \sum_{k=i+1}^{j} r_k$. However, this would always prioritize the last states of the trajectory. To encourage selecting states close to the current state, we divide the accumulated rewards by the distance between the states: $W(s_j) = \sum_{k=i+1}^{j} \frac{r_k}{j-i}$.

The weighted average of the accumulated rewards in a sub-trajectory identifies sub-goal states that have correctly completed a part of the task resulting in a significant reward, without always selecting far away states, by punishing the length of the sub-trajectory. For each state-action pair, we select the state with the highest associated weight to be the sub-goal. The result is a data set of M trajectories $D = \{\tau_1, ..., \tau_M\}$, where each trajectory $\tau_j = \{(s_i, a_i, sg_i), i \in N\}$, where N is the length of the trajectory.

C. Low-Level Controller

The low-level controller is a goal-conditioned transformer π_θ. Similar to DT [5], we model the demonstration trajectories as sequences of tokens for the transformer to learn from. The low-level controller receives sequences of states, actions, and sub-goals and tries to predict the next action. During training, sequences of size K are randomly sampled in a batch from the enhanced data set, where K indicates the number of tokens in a sequence. The low-level transformer is trained to minimize the Behavioral Cloning loss:

$$\mathbb{L}_\theta(s_{t:t+K}, a_{t:t+K}, sg_{t:t+K}) = \|a_{t:t+K} - \pi_\theta(s_{t:t+K}, a_{t:t+K}, sg_{t:t+K})\| \quad (1)$$

In practice, this corresponds to calculating the mean squared error between the generated action and the action in the data set. For each sequence, the model also receives the respective token time steps for positional encoding and a mask to hide the future tokens from the decoder. We do not include them in the formulas for simplicity. At test time, the sequences are built by keeping a history of past states, actions, and time steps from the environment and past sub-goals selected from the high-level mechanism. Hence, the new sequence model does not require any task knowledge to be provided by a user.

TABLE I

MAXIMUM ACCUMULATED RETURNS OF THE ORIGINAL DT AND OF A DT VARIANT WITHOUT THE DESIRED RETURNS INPUT SEQUENCE TRAINED FOR 100 THOUSAND ITERATIONS.

Task	Data Set	Desired Return	Original DT	DT w/o Desired Returns
Half-Cheetah	medium	2655	**5095**	47
		5309	**5092**	47
		10000	**4953**	47
Hopper	medium	1611	**2303**	300
		3222	**2557**	300
		10000	**2036**	300
Kitchen	complete	2	**2.4**	1.0
		4	**2.5**	1.0
		10000	**2.1**	1.0
Walker-2D	medium	2113	**3619**	149
		4227	**3711**	149
		10000	**3100**	149

D. High-Level Mechanism

The high-level mechanism chooses sub-goal states for the low-level controller to try and reach. It is a state-conditioned transformer π_ϕ. The high-level mechanism receives the sequence of states, past sub-goals, time steps, and a mask. The mechanism aims to predict the next sub-goal state which will help the low-level controller succeed in the task. During training, the high-level mechanism is trained to minimize its own Behavioral Cloning loss:

$$\mathbb{L}_\phi(s_{t:t+K}, sg_{t:t+K}) = \|sg_{t:t+K} - \pi_\phi(s_{t:t+K}, sg_{t:t+K})\| \quad (2)$$

In practice, this corresponds to calculating the mean squared error between the generated sub-goal state and the ground-truth sub-goal state in the data set.

V. EXPERIMENTS

In this section, we evaluate the performance of the HDT and compare it with the state-of-the-art method, DT [5], and BC baseline. We evaluate the models on ten tasks from OpenAI Gym [7], D4RL [6] and RoboMimic [8] benchmarks. The tasks range from short episodes to long episodes and from frequent rewards to sparse reward settings. We train the algorithms on the different data sets provided by the benchmarks for each task. For DT, we first find the maximum accumulated returns collected by a trajectory in the demonstration data set. We then set the desired returns to this value and also to half, following the original tests of DT. We train each model for 100 thousand epochs, using batch sizes of 64, a learning rate of $1e^{-4}$, and a sequence length of 20 tokens. To reduce the impact of outliers and because the episodes are seed-dependent, for every one thousand epochs we validate the model on 100 episodes and calculate the average accumulated rewards. In the tables, we present the highest average accumulated rewards seen throughout the 100 thousand epochs.

A. Does DT rely on desired returns?

One of the motivations of our work is to remove the requirement of specifying the value of the desired returns

TABLE II

MAXIMUM ACCUMULATED RETURNS OF THE HDT AND OF A HDT VARIANT INCLUDING THE DESIRED RETURNS INPUT SEQUENCE TRAINED FOR 100 THOUSAND ITERATIONS.

Task	Data Set	Desired Return	HDT	HDT with Desired Returns
Half-Cheetah	medium	2655	**5205**	5004
		5309	**5205**	5002
Hopper	medium	1611	**3071**	1213
		3222	**3071**	1198
Kitchen	complete	2	**2.6**	2.1
		4	**2.6**	1.4
Walker 2D	medium	2113	**3879**	3645
		4227	**3879**	3516

in transformer models in imitation learning, which is present in the state-of-the-art, DT. We first need to know whether the DT truly depends on the returns-to-go sequence to perform or if they can simply be removed from the model's input. Table I shows the accumulated returns obtained by the original DT model and by a DT model variant without the desired returns sequence. For the original DT, we set the value of the desired returns for the task to the maximum value of accumulated rewards present in the data set and to half of this value. We also set the value of the desired return to an arbitrarily high value, for example 10000.

The results in Table. I show that the DT model is dependent on the returns-to-go sequence to perform. Removing them prevents DT from learning the task. Additionally, it also shows that the value of the desired returns is not obvious. Because DT only reaches the desired return of half of the maximum value on the kitchen task. Then, when doubling the value of the desired returns, it only slightly increases the performance. Lastly, when using an arbitrarily high value, DT underperforms. In conclusion, the value of the desired returns is crucial for the performance of the DT model while being non-trivial to determine. This means that the state-of-the-art transformer model in imitation learning relies on task knowledge and user interaction to perform.

B. Does HDT replace the need for desired returns?

Next, we test whether the high-level mechanism, by selecting sub-goals, replaces the need for the sequence of desired returns, in regards to guiding the agent through the task. We test this hypothesis by adding the sequences of desired returns as an extra input to the low-level controller and compare the performance with our base method without this extra input. Similarly to the previous experiment, we test the performance by setting the value of the desired returns to the maximum accumulated reward present in the data set and to half of this value. Table II shows the accumulated returns obtained by our base HDT model and by the HDT model variant with the extra sequence of desired returns.

The results show that HDT performs slightly better without the desired returns sequence. This means that the high-level mechanism is able to output sub-goal states from the history of past states. Moreover, such generated sequences are capable of guiding the low-level controller through the task successfully replacing the need for the sequence returns-to-go. Consequently, HDT removes the requirement for external knowledge about the specific task present in DT.

C. Baseline Comparison

Next, we compare the performance of HDT on ten tasks against DT and BC baselines and present the results in Table III. For BC, we used an MLP with two fully connected layers. Because the three methods clone the behavior present in the data set, their performance is limited by the quality of the demonstrations. Hence, we include the average accumulated rewards collected by the demonstrator in the data set. HDT outperforms the baselines in twenty three out of thirty one settings. This means that our method successfully improved upon the original DT in raw performance while also removing its dependency on desired reward specification, making it task-independent. Particularly, the Maze 2D constitutes a task with sparse rewards where on average 90% of the transitions receive no reward. Similarly, in the kitchen task, the agent only receives a reward after completing each of the four sub-tasks. However, the data sets for the Kitchen task are augmented to include extra rewards. HDT vastly outperforms the baselines in the Maze 2D task and also outperforms the baselines in the kitchen task. This shows that the sequence of sub-goals provides a stronger guiding signal for the model than the sequence of returns-to-go. This means that HDT is more robust to tasks with sparse rewards. For tasks with longer horizons, such as half-cheetah and walker 2D, HDT also surpasses the two baselines. The discrepancy between the transformer methods and BC is more prominent in these types of tasks. We conclude that our method offers superior performance overall, and is more robust to both long and sparse-reward tasks.

D. UR3 Reaching Task

We evaluate HDT by performing an experiment on a physical UR3 robot performing a reaching task. We defined five positions that the robot must reach in any sequence. A state is defined by the six joint angles and a one hot-encoded vector representing the goal positions reached. An action is the translation of the angle of each joint, and the reward is the negative absolute error between the angles of the current joints and the nearest goal position. We generated 100 demonstrations, each starting the robot at a random position and teleoperating the robot through the sequence of goal positions. 80 of the demonstrations are used for training the methods, and 10 for validation. We also evaluate the performance of the algorithms when trained with fewer demonstrations. The remaining 10 demonstrations are used for setting the robot's initial position during the test phase. During testing, we run the algorithms for a maximum of 200 steps and measure the number of goal positions reached. The results are presented in Table IV. Results show that HDT reaches the five goal positions on all 10 initial positions when trained with 70 demonstrations. DT is never able to consistently reach the five positions. Moreover, HDT consistently outperforms DT when trained with the same

TABLE III

MAXIMUM ACCUMULATED RETURNS OF THE HDT COMPARED TO DT AND BC BASELINES. WE TEST THE MODELS ON TEN TASKS FROM D4RL [6], OPENAI GYM [7], AND ROBOMIMIC [8] BENCHMARKS AND VARYING DEMONSTRATION DATA SETS.

Task	Data set	Data Avg.	BC	DT	HDT
Ant	expert	4621	5261	5319	**5365**
	medium	3051	3952	3943	**3993**
	replay	390	3244	**3624**	3325
Door	cloned	304	-53	**758**	275
	expert	2915	-53	3053	**3056**
	human	796	363	**459**	338
Half-Cheetah	expert	10656	10549	10984	**11092**
	medium	4770	5198	5095	**5205**
	replay	3093	0	**4567**	4412
Hammer	cloned	786	-202	118	**775**
	expert	12393	16399	16492	**16507**
	human	3072	50	-68	**3115**
Hopper	expert	3511	3467	3626	**3654**
	medium	1422	1969	2557	**3071**
	replay	467	1017	**2762**	1169
Kitchen	complete	325	0.8	2.5	**2.6**
	mixed	302	0.6	2.1	**2.7**
	partial	255	0.6	**2.7**	**2.7**
Maze 2D	large	2	126	31	**145**
	medium	4	37	68	**188**
	open	6	72	57	**75**
	umaze	7	49	54	**228**
Pen	cloned	3334	1561	3716	**3961**
	expert	3297	2540	4551	**4696**
	human	6326	867	3606	**3897**
Relocate	cloned	1223	3	50	**58**
	expert	4329	3730	**4697**	4624
	human	3674	**232**	35	35
Walker 2D	expert	4921	4261	5063	**5065**
	medium	2852	3723	3711	**3879**
	replay	896	876	**3627**	3325
		Count:	1	8	23

TABLE IV

NUMBER OF GOAL POSITIONS ACHIEVED BY HDT AND DT ON UR3 REACHING TASK, VARYING THE NUMBER OF TRAIN DEMONSTRATIONS.

Train Demos	10	20	30	40	50	60	70	80
HDT	1.4 ±1.6	**2.7 ±2.3**	**2.3 ±2.3**	**3.6 ±2.2**	**4.4 ±1.3**	**4.5 ±1.5**	**5 ±0**	**5 ±0**
DT	**2.1 ±1.2**	0.3 ±0.6	1 ±2	3.2 ±2.1	2.6 ±2.4	1.5 ±2.3	3 ±2.4	4.5 ±1.5

number of demonstrations. This shows that HDT can be applicable to tasks using a physical robot and improves the performance over DT, especially in long-horizon tasks, while not requiring human intervention.

VI. CONCLUSIONS

In this work, we introduced HDT, a hierarchical transformer model. We show that the state-of-the-art DT relies on the sequence of returns-to-go to perform. This sequence requires precise specification of the value of the desired accumulated rewards, which is non-trivial to determine. By replacing the sequence of returns-to-go with a sub-goal selection method, we achieved a fully task-independent model that outperforms DT and BC baselines in various tasks. Notably, HDT is more robust to longer episodes and sparse rewards, making it suitable for real-world applications.

Our experimental results demonstrate the effectiveness of HDT on three benchmarks and a reaching task with a real-world robot. HDT achieved higher accumulated rewards compared to DT and BC in most tasks and reached a higher number of positions in the reaching task. Future work will explore different architectures for both models to further improve their performance. Our findings suggest that using a hierarchical transformer model with a sub-goal selection method can enhance the performance of learning methods and facilitate their application to complex tasks.

REFERENCES

[1] Sutton, R. & Barto, A. Reinforcement Learning: An Introduction. A Bradford Book. (2018)

[2] Hansen, N., Su, H., Wang, X. Stabilizing deep q-learning with convnets and vision transformers under data augmentation. *Advances In Neural Information Processing Systems*. **34** (2021)

[3] Yu, T., Kumar, A., Chebotar, Y., Hausman, K., Levine, S., Finn, C. Conservative data sharing for multi-task offline reinforcement learning. *Advances In Neural Information Processing Systems*. **34** (2021)

[4] Chane-Sane, E., Schmid, C., Laptev, I. Goal-conditioned reinforcement learning with imagined subgoals. *International Conference On Machine Learning*. pp. 1430-1440 (2021)

[5] Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. *Advances In Neural Information Processing Systems*. **34** (2021)

[6] Fu, J., Kumar, A., Nachum, O., Tucker, G., Levine, S. D4RL: Datasets for Deep Data-Driven Reinforcement Learning. (2020)

[7] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W. Openai gym. *ArXiv Preprint ArXiv:1606.01540*. (2016)

[8] Mandlekar, A., Xu, D., Wong, J., Nasiriany, S., Wang, C., Kulkarni, R., Fei-Fei, L., Savarese, S., Zhu, Y., Martín-Martín, R. What matters in learning from offline human demonstrations for robot manipulation. *ArXiv Preprint ArXiv:2108.03298*. (2021)

[9] Argall, B., Chernova, S., Veloso, M., Browning, B. A survey of robot learning from demonstration. *Robotics And Autonomous Systems*. **57**, 469-483 (2009)

[10] Ross, S., Gordon, G., Bagnell, J. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. (2011)

[11] Mandlekar, A., Ramos, F., Boots, B., Savarese, S., Fei-Fei, L., Garg, A., Fox, D. : Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. *2020 IEEE International Conference On Robotics And Automation (ICRA)*. pp. 4414-4420 (2020)

[12] Krishnan, S., Garg, A., Liaw, R., Thananjeyan, B., Miller, L., Pokorny, F., Goldberg, K. SWIRL: A sequential windowed inverse reinforcement learning algorithm for robot tasks with delayed rewards. *The International Journal Of Robotics Research*. **38**, 126-145 (2019)

[13] Pertsch, K., Lee, Y., Lim, J. Accelerating reinforcement learning with learned skill priors. *ArXiv Preprint ArXiv:2010.11944*. (2020)

[14] Paul, S., Vanbaar, J., Roy-Chowdhury, A. Learning from trajectories via subgoal discovery. *Advances In Neural Information Processing Systems*. **32** (2019)

[15] Mandlekar, A., Xu, D., Martín-Martín, R., Savarese, S., Fei-Fei, L. Learning to generalize across long-horizon tasks from human demonstrations. *ArXiv Preprint ArXiv:2003.06085*. (2020)

[16] Nachum, O., Gu, S., Lee, H., Levine, S. Data-efficient hierarchical reinforcement learning. *Advances In Neural Information Processing Systems*. **31** (2018)

[17] Pertsch, K., Rybkin, O., Ebert, F., Zhou, S., Jayaraman, D., Finn, C., Levine, S. Long-horizon visual planning with goal-conditioned hierarchical predictors. *Advances In Neural Information Processing Systems*. **33** pp. 17321-17333 (2020)

[18] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, Ł., Polosukhin, I. Attention is all you need. *Advances In Neural Information Processing Systems*. **30** (2017)

[19] Janner, M., Li, Q., Levine, S. Offline Reinforcement Learning as One Big Sequence Modeling Problem. *Advances In Neural Information Processing Systems*. **34** (2021)

[20] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., Others Language models are unsupervised multitask learners. *OpenAI Blog*. **1**, 9 (2019)