Predictive Coding for Decision Transformer

Tung M. Luu†, Donghoon Lee†, and Chang D. Yoo*

Abstract— **Recent work in offline reinforcement learning (RL) has demonstrated the effectiveness of formulating decision-making as return-conditioned supervised learning. Notably, the decision transformer (DT) architecture has shown promise across various domains. However, despite its initial success, DTs have underperformed on several challenging datasets in goal-conditioned RL. This limitation stems from the inefficiency of return conditioning for guiding policy learning, particularly in unstructured and suboptimal datasets, resulting in DTs failing to effectively learn temporal compositionality. Moreover, this problem might be further exacerbated in long-horizon sparse-reward tasks. To address this challenge, we propose the Predictive Coding for Decision Transformer (PCDT) framework, which leverages generalized future conditioning to enhance DT methods. PCDT utilizes an architecture that extends the DT framework, conditioned on predictive codings, enabling decision-making based on both past and future factors, thereby improving generalization. Through extensive experiments on eight datasets from the AntMaze and FrankaKitchen environments, our proposed method achieves performance on par with or surpassing existing popular value-based and transformer-based methods in offline goal-conditioned RL. Furthermore, we also evaluate our method on a goal-reaching task with a physical robot.**

I. INTRODUCTION

Reinforcement learning (RL) systems have demonstrated remarkable success across a wide array of domains, ranging from games [42] to autonomous driving [52] and robotics [25], [35]. Yet, the inherent sample inefficiency of online RL algorithms presents a formidable challenge, demanding extensive environmental interactions for agent training. This necessity for voluminous data imposes substantial requirements for human supervision, safety checks, and resets [1], thereby limiting their practical deployment in real-world scenarios. Consequently, offline RL has recently garnered increasing attention as an alternative training paradigm, wherein policies are exclusively trained from static datasets of reward-labeled demonstrations. Offline RL algorithms commonly integrate learning objectives that encourage pessimism alongside value-based methods [16], [31], [34] to achieve commendable performance. Despite their promise, these methods encounter challenges during training, often necessitating meticulous hyperparameter tuning and various implementation tricks to ensure stability and optimal performance across various tasks.

In an effort to streamline offline RL training, recent research has explored transforming offline RL into a sequence modeling problem [8], [22], employing the powerful Transformer architecture [45] for decision-making. At the core of these transformer-based approaches lies the concept of conditioning policies on a desired outcome. For instance, Decision Transformer (DT) [8] learns a model to predict actions based on historical context, encompassing states and actions, and conditioned on a target future return. Through conditioning on future returns, DT effectively conducts credit assignment across time horizons, demonstrating competitive performance across various offline RL tasks. Importantly, these approaches eliminate the necessity for temporal-difference learning methods, such as fitted value or action-value functions. This results in a simpler algorithmic framework that relies on supervised learning, facilitating the advancement of offline RL by leveraging existing progress in supervised learning.

Despite their potential, relying on reward-labeled datasets to train return-conditioned policies makes scaling DT challenging for large-scale training. Indeed, task-specific reward functions often require careful engineering for design [37], [39], making them costly to access in practice. Additionally, in sparse reward environments where rewards are often constant (*i.e.*, success or failure), DT struggles to learn tasks, especially in long-horizon tasks [10]. Furthermore, DT also faces difficulty in unstructured datasets that require the agent to learn temporal compositionality (*i.e.*, stitching) by combining sub-trajectories of different demonstrations [2], [29]. For instance, in the AntMaze maze navigation environment, where an 8-DoF quadruped robot is required to reach the target, DT typically performs worse than other value-based offline RL methods such as IQL [27].

In this study, we delve into the offline goal-conditioned RL setting [12], [19], [14]. This setting allows the agent to solve multiple tasks by learning from reward-free data. We adopt the DT as an expressive policy network to facilitate goal-conditioned RL. However, the aforementioned challenges of DT in this setting still remain. To tackle these challenges, we introduce a framework that enables DT to condition on predictive codings for action predictions instead of returns. Specifically, we first learn a goal-conditioned representation that encodes the target goal, past, and future trajectory information using a bidirectional transformer. Next, we employ a causal transformer that takes as input the sequence of state-

*Corresponding author: Chang D. Yoo

† The authors are equally contributed.

All authors are with the School of Electrical Engineering, KAIST (Korea Advanced Institute of Science and Technology), Daejeon, Republic of Korea. {tungluu2203,dh_lee99,cd_yoo}@kaist.ac.kr

This work was partly supported by Institute for Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. 2021-0-01381, Development of Causal AI through Video Understanding and Reinforcement Learning, and Its Applications to Real Environments) and partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) [RS-2021-II212068, Artificial Intelligence Innovation Hub (Seoul National University)].

action pairs and conditions on the predictive codings from the previous stage to produce actions. The predictive codings serve as behavioral guidance toward the desired goal during policy learning. Our approach bypasses the requirement of return sequences, allowing the use of easy-to-collect offline data devoid of rewards for greater efficiency. By conditioning on these predictive codings, our transformer-based policy is equipped with the ability to reason about the future, providing crucial guidance for learning in unstructured and suboptimal datasets, leading to competitive performance across goal-conditioning benchmarks, particularly for long-horizon tasks. To summarize, our main contributions include:

1) Introducing Predictive Coding for Decision Transformer (PCDT), a framework enabling transformer-based agents to learn from a large set of diverse, unstructured, and suboptimal demonstrations by conditioning on predictive codings, thereby bypassing the requirement of rewards from the dataset.
2) Proposing an effective method to learn predictive codings from datasets without requiring rewards and actions. Our results demonstrate that predictive coding effectively guides transformer-based agents to solve goal-conditioned tasks without the need for returns and enables stitching capability for DT.
3) Evaluating PCDT on eight datasets from two complex long-horizon goal-conditioning environments, AntMaze and FrankaKitchen, and validating it on the physical 7-DOF Sawyer robot. The code can be found at `https://github.com/tunglm2203/pcdt`.

II. RELATED WORK

Offline reinforcement learning (RL) aims to learn effective policies from pre-collected datasets. Previous works have addressed the distribution shift problem [16] of offline RL through various strategies such as constraining the policy action space [16], [28], incorporating value pessimism [31], [34], or leveraging dynamics models [26], [53]. In this paper, we focus on offline RL methods based on conditional behavior cloning, avoiding the need for value estimations. Typically, RvS [14] has investigated different conditional variables for MLP policy and establishes strong baseline for conditional behavior cloning. Many prior works are constructed in different styles of RvS, where the conditional variables can be the reward/return [30], [43], [8], the target goal [36], [14], [19], other task information [9], [20], or trajectory-level aggregates [17], [47].

Recent works [8], [22] have explored another approach to perform conditional behavior cloning via sequence modeling by leveraging the Transformer architecture [45], which has achieved widespread success in various machine learning fields, including natural language processing (NLP) [41], [11], speech [44], [32], and computer vision (CV) [13], [6]. Specifically, Decision Transformer (DT) [8] utilizes a causal transformer architecture for the policy network to perform return-conditioned behavioral cloning, where instead of taking a single state as input, it takes a sequence of states, actions, and returns. Similarly, a model-based offline RL method is introduced in [22], namely Trajectory Transformer (TT), which also leverages the Transformer architecture to formulate the forward dynamics model and uses beam search for planning. Trajectory Autoencoding Planner [23] further enhances TT to achieve greater efficiency and scalability in high-dimensional action spaces.

Building upon the achievements of DT, numerous studies have expanded its application to various contexts, such as online DT for online RL [54], Q-learning DT for offline RL [50], or Prompt DT for few-shot learning [49]. In the view of RvS, these works adhere to the reward conditioning style. However, recent works [2], [4], [10] have explored the limitations of using reward as a conditional variable. Notably, DT has been identified as lacking in stitching ability [2], a crucial property for offline RL algorithms to succeed on unstructured and suboptimal datasets, which often involve combining sub-trajectories from different demonstrations. Waypoint Transformer (WT) [2] addressed this issue by conditioning the DT on predicted K-step ahead goals instead of return-to-go, with these intermediate goals serving as future guidance for policy in goal-conditioned RL. In contrast to WT, we propose conditioning on predictive coding, which offers a more generalized and effective encoding of future information for guiding policy learning.

Masked autoencoding (MAE) has been demonstrated as an effective approach for acquiring representations in both NLP [11], [5] and CV [21], [3]. In MAE, the input sequence undergoes random masking before being processed by a transformer encoder, resulting in a compressed representation that is subsequently fed into the transformer decoder to reconstruct the input. This straightforward yet effective approach has sparked recent advancements in RL [7], [46], [33]. By leveraging masked autoencoder trained on randomly masked sequences to reconstruct the original data, these works have developed versatile models capable of performing a range of decision-making tasks, such as forward dynamics, inverse dynamics, behavior cloning, and state representations, by simply changing the masking pattern at inference time. In contrast to these approaches, we leverage the masked autoencoder to compress both the state-only trajectory and the desired goal, using the obtained compressed representation to guide policy learning. While our pretraining scheme draws inspiration from MaskDP [33] and MTM [46], our focus is on addressing the challenges of DT in the goal-conditioned RL setting, particularly its stitching ability, rather than solely concentrating on representation learning.

III. PRELIMINARIES

A. Goal-conditioned Reinforcment Learning

A reinforcement learning (RL) environment is typically modeled by a Markov decision process (MDP), which can be described as a tuple of $\mathcal{M} = (\mathcal{S}, \mathcal{A}, R, P, \gamma)$. The tuple consists of states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, a reward function $r = R(s, a)$, transition dynamics distribution $P(s'|s, a)$, and a discount factor $\gamma \in [0, 1)$. A trajectory is composed of a sequence of state-action pairs: $\tau = \{(s_t, a_t)\}_{t=1}^{H}$, where H is the horizon, and s_t, a_t refer to the state and action at timestep

t, respectively. Additionally, we define $\tau_{i:j} = \{(s_t, a_t)\}_{t=i}^{j}$ as a sub-trajectory of τ. An RL agent takes actions based on a policy $\pi(a|s)$. The objective is to maximize the expected return over trajectories $\mathbb{E}_\pi \left[\sum_{t=1}^{H} \gamma^{t-1} R(s_t, a_t)\right]$.

To extend RL to multiple tasks, a goal-conditioned formulation can be used [24], where the original MDP $\mathcal{M}$ is augmented with the goal space $\mathcal{G}$. We assume that the goal space $\mathcal{G}$ is the same as the state space (*i.e.*, $\mathcal{G} \equiv \mathcal{S}$). Our goal is to learn an optimal goal-conditioned policy $\pi(a|s, g)$ that maximizes $\mathbb{E}_{g\sim p(g),\pi} \left[\sum_{t=1}^{H} \gamma^{t-1} R(s_t, a_t, g)\right]$, where the goal g is sampled from a goal distribution $p(g)$. In this setting, a sparse goal-conditioned reward function is often used, where $R(s, a, g) = 1$ when $s = g$, and 0 otherwise.

B. Offline RL via Supervised Sequence Modeling

In offline RL, instead of obtaining data by interacting with the environment, we only have access to a fixed, limited dataset collected by unknown policies, such as human experts or baseline hand-engineered policies. This dataset contains a set of N trajectories, $\mathcal{D} = \{\tau^i\}_{i=1}^{N}$, associated with rewards. We follow prior work that leverages sequence modeling for offline reinforcement learning [8]. Specifically, Decision Transformer (DT) [8] considers the following trajectory representation as input:

$$\hat{\tau} = (\hat{R}_1, s_1, a_1, \ldots, \hat{R}_H, s_H, a_H),$$

where, $\hat{R}_t = \sum_{t'=t}^{H} r_{t'}$ represents the return-to-go, defined as the sum of future rewards from timestep t. Modality-specific encoders are employed to transform data (*i.e.*, state, action, and return-to-go) from the raw modality space to a common representation space, to which timestep embeddings are additionally added. After tokenization, $\hat{\tau}$ is input into a GPT-based transformer [40], serving as an expressive policy function approximator π_θ to predict the next actions. The policy is trained to maximize the likelihood of actions within the dataset:

$$\mathcal{L}_{\text{DT}} = \mathbb{E}_{\hat{\tau}\sim\mathcal{D}} \left[-\sum_t \log \pi_\theta(a_t | \hat{\tau}_{t-k+1:t-1}, s_t, \hat{R}_t) \right] \quad (1)$$

At training time, return-to-go sequences are computed from offline trajectories. During inference, this quantity guides the policy in generating actions to achieve the desired outcome. However, using return-to-go as a conditional variable presents certain disadvantages. First, at test time, the full trajectory is unknown, and its length may vary, making it difficult to calculate the sum. Consequently, users must specify the target return heuristically. Moreover, the performance of DT is shown to be sensitive to this value [10], [51]. Second, in sparse-reward tasks where rewards are almost constant (*e.g.*, $r_t = 0$) until reaching the final target, return-to-go values remain static. This challenge impedes DT's ability to learn tasks effectively, especially in long-horizon tasks [10]. Third, using a single scalar value as input may fail to capture sufficient future information. These limitations motivate the design of new, effective transformer-based algorithms.

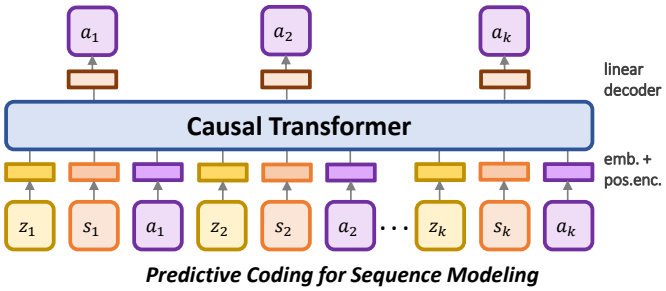

Fig. 1: The overview of the PCDT model. PCDT takes as input a length-k sub-trajectory associated with predictive latent codings to make decisions. During training, predictive codings are extracted from states within the same sub-trajectory. During inference, PCDT generates the next action in an autoregressive manner, akin to DT [8].

IV. PROPOSED APPROACH

To eliminate the reliance on reward information, inspired by future-conditioned RL [17], [14], we introduce Predictive Coding for Decision Transformer (PCDT), a framework that enables policy learning guided by future information.

A. Policy Learning based on Predictive Coding

The overview of PCDT is presented in Fig. 1. Let $f_\phi^E(\cdot)$ be a trajectory encoder, and $\pi_\theta(\cdot)$ be a policy network. In this framework, our objective is to condition the policy on predictive coding z instead of the return, resulting in an alternative representation of the trajectory input:

$$\bar{\tau} = (z_1, s_1, a_1, \ldots, z_H, s_H, a_H),$$

where, the latent variables are obtained by $z_{t-k+1:t} = f_\phi^E(s_{t-k+1:t}, g)$, as depicted in Fig. 2. These latent variables z_i are expected to capture not only the historical context and task information (*i.e.*, the desired goal) but also to be predictive, enabling reasoning over future states toward the desired goal. During inference, given past states and actions, PCDT utilizes the learned policy π_θ along with the learned trajectory encoder f_ϕ^E to autoregressively produce the action:

$$a_t \leftarrow \pi_\theta(\cdot | \bar{\tau}_{t-k+1:t-1}, s_t, z_t).$$

Note that, both the policy and trajectory encoder take the same view of the k-length input sequence, *i.e.*, from timestep $t - k + 1$ to t.

Similar to DT, we leverage the GPT-based Transformer [40] to parameterize the policy network π_θ. However, we opt for sinusoidal positional encoding over timestep encoding to mitigate overfitting [7]. Given sub-trajectories sampled from the dataset $\mathcal{D}$, we first compute the corresponding latent variables z_i, then the policy network is trained by minimizing the behavioral cloning objective as follows:

$$\mathcal{L}(\theta) = \mathbb{E}_{\bar{\tau}\sim\mathcal{D}} \left[-\sum_t \log \pi_\theta(a_t | \bar{\tau}_{t-k+1:t-1}, s_t, \lfloor z_t \rfloor) \right] \quad (2)$$

where, $\lfloor \cdot \rfloor$ denotes the stop-gradient operator. Note that during policy training, the gradient only backpropagates through the policy network. The trajectory encoder is learned by a separate learning objective, which we describe in the next

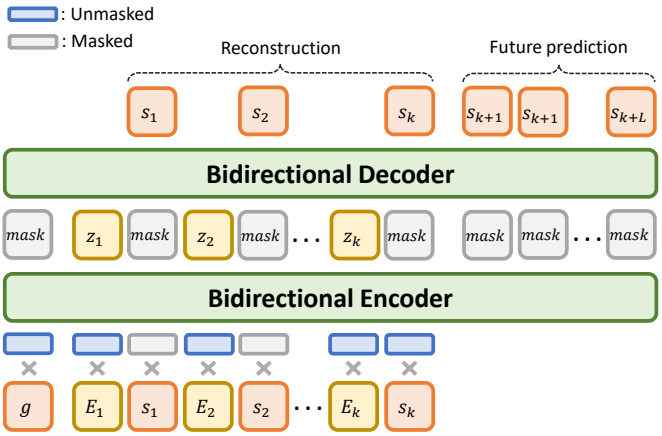

Fig. 2: We input the sequence of state-dummy token pairs concatenated with the target goal into the trajectory encoder. Within this sequence, states are randomly masked. The trajectory decoder receives stacked inputs of latent codes and masked tokens to reconstruct the input states. Additionally, in addition to reconstruction, we leverage the same latent codes for predicting future states, aiming to enhance predictiveness.

section. Utilizing predictive codings as conditioning offers several advantages. Firstly, it avoids the need for reward signals during learning, thereby enabling expansive, large-scale training opportunities. Secondly, it mitigates the issue of inconsistent behaviors stemming from return-conditioned behavioral cloning during testing, where behaviors often deviate significantly from the desired target [10], [51].

B. Predictive Coding Learning

In order to condense the trajectory into a compact latent space, we utilize masked autoencoding with both the encoder and decoder parameterized by a series of bidirectional transformers and trained via masked prediction, similar to previous works [7], [33], [46]. An overview of the trajectory autoencoder is shown in Fig. 2. We use f_ϕ^E and f_ϕ^D to represent the trajectory encoder and decoder, respectively.

Specifically, the trajectory encoder takes as input the concatenation of *the target goal* g, the dummy tokens $E_{1:k}$, and the length-k sequence of states $s_{t-k+1:t}$. The dummy tokens are shared across input sequences. During training, the states are randomly masked to improve generalization, while the goal and dummy tokens remain unmasked, as illustrated in Fig. 2. The goal g is uniformly sampled from reachable states $s_{t:H}$ within the same trajectory. These input tokens are then encoded by modality-specific encoders, parameterized by a linear layer, with sinusoidal positional embeddings added. Subsequently, the embeddings are processed by the bidirectional transformer to yield latent codes z. It's worth noting that only unmasked tokens are provided to the transformer, mirroring the approach of MAE-based methods [21], [7], [33], [46]. To reconstruct the input states, the trajectory decoder takes as input the latent code tokens corresponding to the position of dummy tokens, the remaining tokens are substituted by masked tokens. To encourage predictive coding, in addition to reconstructing input states, future state prediction is concurrently performed from the latent codes.

Algorithm 1 PCDT training algorithm.

Input: Demonstration $\mathcal{D} = \{\tau_i\}_{i=1}^N$, length of historical states k, length of future states L, and learning rate α.
Initialize: trajectory autoencoder f_ϕ and policy π_θ.
// `Training the trajectory autoencoder`
while not converged **do**
  Randomly sample trajectories, $\tau \sim \mathcal{D}$
  Sample timestep for each trajectory, $t \sim [1, H]$, obtain $s_{t-k+1:t+L}$, and sample goal, $g \sim s_{t:H}$
  Calculate objective function $\mathcal{L}(\phi)$ as in Eq. (3)
  Update autoencoder parameters: $\phi \leftarrow \phi + \alpha \nabla_\phi \mathcal{L}(\phi)$
end while
// `Training the policy`
while not converged **do**
  Randomly sample trajectories, $\tau \sim \mathcal{D}$
  Sample timestep for each trajectory, $t \sim [1, H]$, obtain $s_{t-k+1:t}$ and $a_{t-k+1:t}$, and sample goal, $g \sim s_{t:H}$
  Compute predictive codings:
$$z_{t-k+1:t} = f_\phi^E(s_{t-k+1:t}, g)$$
  Calculate objective function $\mathcal{L}(\theta)$ as in Eq. (2)
  Update policy parameters: $\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}(\theta)$
end while

The encoder and decoder are jointly trained by minimizing the following learning objective:

$$\mathcal{L}(\phi) = \mathbb{E}_{s,g\sim\mathcal{D}}\left[\sum_t \left(f_\phi(s_{t-k+1:t}, g) - s_{t-k+1:t+L}\right)^2\right] \quad (3)$$

where, f_ϕ represents both the encoder and decoder. Also, we consider dummy tokens $E_{1:k}$ as part of the encoder, thus omitting them from the input of f_ϕ. The ground truth states include k historical states and L future states, *i.e.*, from timestep $t-k+1$ to $t+L$. During policy learning and inference, only the trajectory encoder is used, applied to the unmasked observed states and the target goal to acquire predictive codings. With this design, we can learn the predictive codes to perform goal-conditioned future prediction. Consequently, these predictive codes serve as guidance toward the desired goal while being aware of future behaviors during policy learning. Besides, we only leverage states from the dataset for training the predictive coding, enabling us to utilize a potentially large amount of action-free data [48], [18]. We leave this for future work. The procedure for training trajectory autoencoder and policy is summarized in Algorithm 1.

V. EXPERIMENTS

A. Experimental Setup

In this section, we evaluate PCDT on goal-conditioning benchmarks, specifically AntMaze, FrankaKitchen from D4RL [15] and a Rethinking Sawyer robot performing a goal-reaching task, as illustrated in Fig. 3. These environments pose a challenge for offline RL methods due to the scarcity of optimal trajectories and serve as rigorous benchmarks for evaluating a model's stitching ability

TABLE I: Average normalized scores of PCDT against other baselines on AntMaze and FrankaKitchen from D4RL benchmark [15]. Following [27], we bold all scores within 5 percent of the maximum per dataset ($\geq 0.95 \cdot \max$).

Dataset	CQL	IQL	GC-IQL	BC	GCBC	DT_R	DT_G	WT	PCDT (Ours)
antmaze-umaze	74.0	$\mathbf{87.5 \pm 2.6}$	-	54.6	65.6 ± 9.9	53.6 ± 7.3	61.2 ± 5.8	64.9 ± 6.1	70.8 ± 3.9
antmaze-umaze-diverse	**84.0**	62.2 ± 13.8	-	45.6	60.9 ± 11.2	42.2 ± 5.4	66.0 ± 5.3	71.5 ± 7.6	71.5 ± 3.0
antmaze-medium-play	61.2	71.2 ± 7.3	70.9 ± 11.2	0.0	$\mathbf{71.9 \pm 16.2}$	0.0 ± 0.0	56.0 ± 9.4	62.8 ± 5.8	$\mathbf{75.2 \pm 5.7}$
antmaze-medium-diverse	53.7	70.0 ± 10.9	63.5 ± 14.6	0.0	67.3 ± 10.1	0.0 ± 0.0	59.5 ± 7.6	66.7 ± 3.9	$\mathbf{83.2 \pm 4.8}$
antmaze-large-play	15.8	39.6 ± 5.8	56.5 ± 14.4	0.0	23.1 ± 15.6	0.0 ± 0.0	22.0 ± 7.5	$\mathbf{72.5 \pm 2.8}$	$\mathbf{74.8 \pm 6.2}$
antmaze-large-diverse	14.9	47.5 ± 9.5	50.7 ± 18.8	0.0	20.2 ± 9.1	0.0 ± 0.0	21.0 ± 6.6	$\mathbf{72.0 \pm 3.4}$	$\mathbf{73.6 \pm 5.7}$
antmaze average	50.6	63.0	60.4	16.7	51.5	16.0	47.6	68.4	**74.9**
kitchen-partial	49.8	46.3	39.2 ± 13.5	38.0	38.5 ± 11.8	31.4 ± 19.5	65.2 ± 1.9	63.8 ± 3.5	$\mathbf{74.5 \pm 5.0}$
kitchen-mixed	51.0	51.0	51.3 ± 12.8	51.5	46.7 ± 20.1	25.8 ± 5.0	55.4 ± 3.8	70.9 ± 2.1	$\mathbf{75.6 \pm 4.7}$
kitchen average	50.4	48.7	45.3	44.8	42.6	28.6	60.3	67.4	**75.1**
average	50.6	59.4	54.6	23.7	49.3	19.1	50.8	68.1	**74.9**

[15]. AntMaze comprises long-horizon navigation tasks with sparse rewards, involving the control of an 8-DoF Ant robot to navigate from its initial position to a specified goal location. Meanwhile, FrankaKitchen presents long-horizon manipulation tasks, where the objective is to complete four subtasks (*e.g.*, closing the microwave or sliding open the cabinet) using a 9-DoF Franka robot. We train our algorithms on diverse datasets provided by these environments. Each dataset comprises suboptimal and undirected data, where the desired targets in demonstrations are unrelated to the evaluation tasks. Additionally, we also validate the effectiveness of our method on the real-world environment by conducting goal-reaching tasks with a 7-DOF Sawyer robot arm. By this, we provide concrete evidence of its viability and robustness in navigating real environments.

For the trajectory autoencoder, we employ a two-layer bidirectional transformer for the encoder and a one-layer bidirectional transformer for the decoder, each with four self-attention heads. The length of future states (L) in training the trajectory autoencoder is set to 100 for AntMaze and 40 for FrankaKitchen. For PCDT, we utilize a causal transformer with the number of layers searched within $\{3, 4, 5\}$, each with one self-attention head. All transformer use hidden size of 256. Both training phases use the Adam optimizer with a learning rate of $1e-4$, a batch size of 1024, and 80 epochs. The input sequence length is set to 10 for AntMaze and 5 for FrankaKitchen. Following [27], we average over 100 episodes for AntMaze and 50 episodes for FrankaKitchen for each evaluation run. Reported scores include the mean and standard deviation over five seeds for each experiment.

B. Comparison with Previous Approaches

We compare the performance of PCDT with both value-based offline RL and behavior cloning methods. Among the value-based techniques, we include CQL [31], IQL [27], and its goal-conditioned variant, GC-IQL [38]. For the MLP-based behavior cloning approach, we evaluate BC and its goal-conditioned counterpart, GCBC [12], [14], both of which utilize MLPs to parameterize the policy network. For transformer-based policies, we consider DT_R [8], which conditions on return-to-go, DT_G, which conditions on the target goal, and WT [2], similar to our approach but conditioned

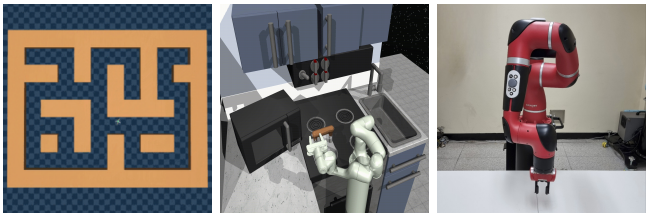

Fig. 3: We evaluate the performance of PCDT on three sparse-reward goal-conditioned tasks. From left to right: AntMaze and FrankaKitchen, taken from the D4RL benchmark [15], and a physical Rethink Sawyer robot performing goal-reaching task.

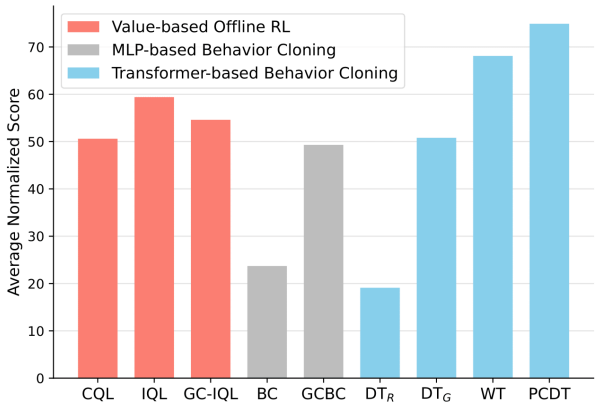

Fig. 4: A plot illustrating the average performance of each algorithm listed in Table I. The bars are color-coded based on a simplified algorithm categorization.

on predicted K-step ahead goals. For all methods except DT_G, we use reported results from [2] and [38]. For DT_G, we modify DT by replacing return-to-go with the target goal using the official implementation provided by [8].

Table I and Fig. 4 present the results across eight offline datasets. When compared to value-based methods, PCDT outperforms six out of eight datasets. Across all datasets, PCDT achieves a score of 74.9, representing a substantial improvement over the top-performing value-based method, IQL (59.4), with a relative percentage increase of 26.1%. Notably, in the most challenging datasets requiring stitching like AntMaze Large, our method exhibits remarkable enhancements, with relative improvements of 88.9% ("play") and

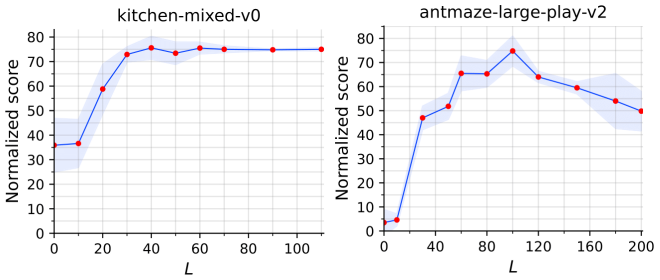

Fig. 5: The effect of different lengths of future states (L) during the learning of predictive coding on agent performance.

54.9% ("diverse") over IQL. Similarly, in the FrankaKitchen Mixed and Partial, we observe substantial enhancements of 60.9% and 48.2%, respectively. These results demonstrate the efficacy of our predictive codings in generalizing behaviors from suboptimal trajectories and effectively addressing the stitching problem encountered by DT.

For the behavior cloning baselines, we find that our method performs on par with or better than all the previous methods. In particular, PCDT exhibits significant improvements over DT_R by a factor of 3.x in average performance. This notable improvement underscores the efficacy of leveraging predictive codings, which offer a stronger signal for policy learning compared to the return sequence. Moreover, by naively conditioning on target goals, DT_G demonstrates some learning capability across AntMaze datasets, which again emphasizes the limitations of using rewards as conditional variables in sparse-reward goal-conditioned RL. Despite its efforts, the performance of DT_G remains considerably distant from PCDT, with ours showcasing a substantial 47.4% improvement in average performance over DT_G. In comparison with the prior state-of-the-art, our method improves over WT by 9.5% and 11.4% on AntMaze and FrankaKitchen, respectively. These results underscore the ability of predictive codings to encode more comprehensive future information compared to intermediate future goals, thereby more effectively guiding the policy toward the desired target.

C. Ablation Studies

Effectiveness of Future State Prediction. We first investigate how future prediction impacts the quality of learned predictive coding. For this analysis, we consider *antmaze-large-play-v2* and *kitchen-mixed-v0*, two tasks crucial for assessing the stitching ability of offline RL algorithms. To elucidate the influence of predictiveness on policy learning, we vary the L-length future states when training trajectory encoders (see Section IV-B) and evaluate the agent's performance based on the learned predictive codings. Here, $L = 0$ indicates that the trajectory autoencoder solely reconstructs the input sequence without considering future prediction. The agent's normalized score is plotted against L in Fig. 5. In AntMaze Large, an optimal L is around 100, while in FrankaKitchen, $L \geq 30$ yields good performance. Importantly, when L is too small, *e.g.*, ≤ 10 in both cases, the agent's performance drastically drops compared to other L values; the score decreases by a factor of 2.1× and 16.3× in FrankaKitchen and AntMaze Large, respectively, compared to the optimal L. This analysis reveals that incorporating future information through predictive coding can markedly enhance task-solving capabilities, empowering the agent to anticipate future behaviors.

TABLE II: We ablate transformer configurations during policy learning, including dropout probability (p_{drop}) and number of transformer layers (N). The highest score is highlighted in bold and used in the main results.

p_{drop}	antmaze-medium-diverse	antmaze-large-play	kitchen-mixed
0.00	74.0 ± 8.8	68.0 ± 10	72.0 ± 2.3
0.05	73.2 ± 5.2	70.7 ± 6.4	$\mathbf{75.6 \pm 4.7}$
0.10	$\mathbf{83.2 \pm 4.8}$	$\mathbf{74.8 \pm 6.2}$	70.0 ± 5.2
0.15	80.4 ± 7.1	62.5 ± 3.4	71.0 ± 2.9
0.20	76.0 ± 5.2	59.8 ± 4.9	69.8 ± 2.2
0.25	70.0 ± 5.2	48.9 ± 8.4	66.0 ± 7.1
N	antmaze-medium-diverse	antmaze-large-play	kitchen-mixed
1	70.8 ± 3.9	24.0 ± 1.8	52.8 ± 7.9
2	74.0 ± 5.8	49.5 ± 5.7	62.3 ± 3.9
3	$\mathbf{83.2 \pm 4.8}$	61.5 ± 6.3	52.3 ± 6.0
4	72.4 ± 6.5	$\mathbf{74.8 \pm 6.2}$	70.1 ± 5.1
5	72.0 ± 3.1	63.3 ± 8.1	$\mathbf{75.6 \pm 4.7}$
6	75.2 ± 4.3	58.5 ± 8.4	70.3 ± 4.9
7	64.0 ± 5.8	58.0 ± 3.9	73.8 ± 1.4

Capacity and Regularization. Following the work in [14], we balance between capacity and regularization to maximize policy performance. We explore transformer configurations during policy learning, specifically, ablation of the dropout probability p_{drop} and the number of transformer layers N. For this investigation, we consider the *antmaze-medium-diverse-v2*, *antmaze-large-play-v2*, and *kitchen-mixed-v0* datasets. Based on the results in Table II, we observe that sensitivity to the various ablated hyperparameters is relatively low in terms of performance on *antmaze-medium-diverse-v2* and *kitchen-mixed-v0*, with a decrease by a factor of 1.1-1.4 compared to the best hyperparameter. However, in *antmaze-large-play-v2*, we observe that choosing the appropriate number of layers matters for the agent's performance, where $N = 1$ decreases performance by a factor of 3.x compared to $N = 4$.

Qualitative Result. To gain insights into the agent's behavior under the guidance of predictive codings, we qualitatively evaluate the performance across rollouts of trained policies in the *antmaze-large-play-v2* dataset. Specifically, we examined the performance of three transformer-based agents: DT_R, DT_G, and PCDT. The visualization of the ant's location across 100 trajectories for each agent is presented in Fig. 6. Our observations indicate that DT_R (Fig. 6a), guided solely by return-to-go, struggles to consistently reach the desired target. While DT_G (Fig. 6b), guided by target goals, occasionally reaches the target, the ant's behaviors exhibit inconsistencies, often veering off course in the middle of the map. In contrast, PCDT (Fig. 6c) demonstrates a higher level of ability and consistency in reaching the goal location. Additionally, it sometimes reaches the target via alternative routes (*e.g.*, bottom left of Fig. 6c), indicating that our method does not solely overfit to the dataset but also has the capability of generalization. As a result, PCDT

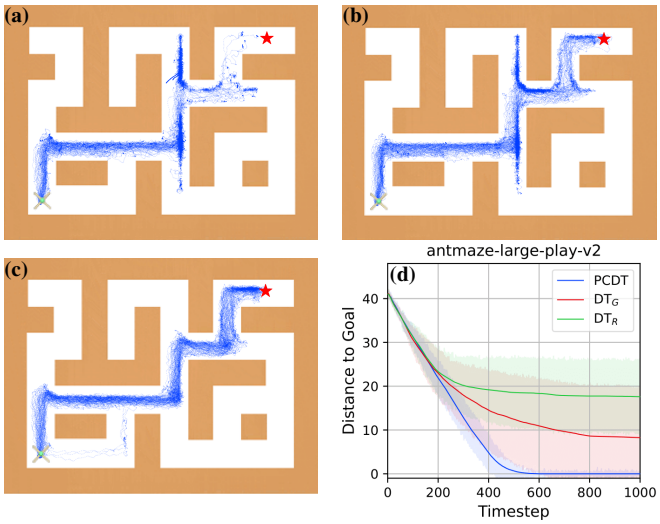

Fig. 6: Visualization of 100 trajectories from the policies trained on *antmaze-large-play-v2*: (a) DT$_R$ policy, (b) DT$_G$ policy, and (c) PCDT policy; (d) the distance from the ant's location to the target goal position at each timestep.

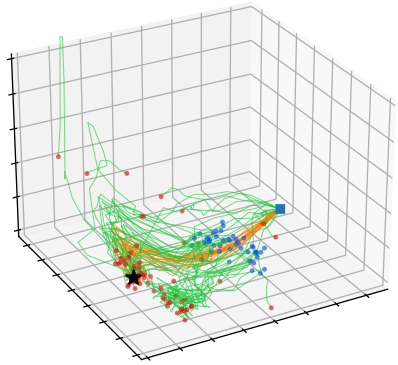

Fig. 7: The visualization of the Sawyer Reaching dataset shows blue dots for start positions and red dots for end positions of trajectories. Orange lines indicate "completed" trajectories, while green lines represent "play" trajectories. A blue square marks the initial position, and a black star marks the target region during evaluation.

achieves a significantly higher score than DT$_G$ by about 3 times and takes fewer steps to complete the task (Fig. 6d).

D. Sawyer Reaching Task

We evaluate the stitching ability of PCDT on a 7-DoF Sawyer arm (Fig. 3) for a goal-reaching task, where the objective is to reach an arbitrary goal. The state space is represented by a 6-D vector, with the first 3 dimensions representing the x, y, z position of the end effector (EE), and the last 3 dimensions representing the x, y, z velocity of the EE. The agent controls the arm by commanding positional translations $(\Delta_x, \Delta_y, \Delta_z)$ of the EE. In each episode, the desired goal is randomly sampled from a specific region, similar to AntMaze. We use a sparse reward function, where $R = 1$ when the distance between the current EE's position and the goal is less than 5cm, and $R = 0$ otherwise. The episode's length is set to 20. For the offline dataset, we follow a similar data collection method used in AntMaze Large.

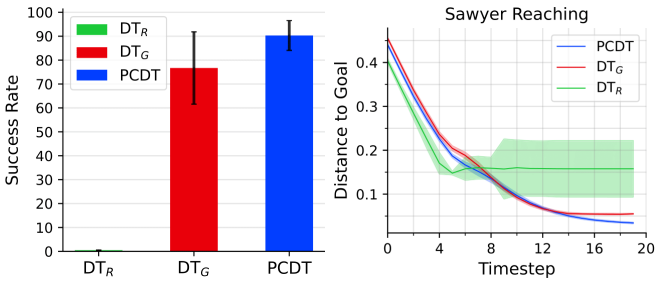

Fig. 8: Compasison between DT$_R$, DT$_G$, and PCDT in terms of the success rate and the distance to the target goal.

Specifically, we collect 110 "play" trajectories starting from hand-picked initial positions and reaching specific hand-picked goals, along with 10 "completed" trajectories starting from the robot's initial position and ending at the task's goal position, resulting in a total of 120 trajectories. During data collection, a scripted policy is used to generate actions, with random noise added to the initial position, specified goals, and generated actions to encourage diversity. It's worth noting that the goals during collection may differ from those during evaluation. A visualization of a portion of the collected dataset is shown in Fig. 7. For this experiment, we use the same hyperparameters as in FrankaKitchen, except we set $k = 3$ and $L = 8$. We compare the performance of DT$_R$, DT$_G$, and PCDT agents. The results are presented in Fig. 8. Despite being a simple task, DT$_R$ fails to complete the task when guided by return. In comparison, our method achieves a higher success rate and a smaller final distance to the goal than DT$_G$. This indicates that PCDT has the potential to enhance performance in tasks involving physical robots compared to DTs, particularly in dealing with unstructured and suboptimal datasets commonly encountered in real-world scenarios.

VI. CONCLUSIONS

In this work, we introduce Predictive Coding for Decision Transformer (PCDT), a method for offline goal-conditioned RL through supervised sequence modeling. By encoding future states into predictive codings, our transformer-based policy can reason future behaviors to reach desired targets. Furthermore, our framework removes the need for rewards from the dataset, enabling training on potentially large unlabeled datasets. Empirical results demonstrate the effectiveness of PCDT compared to various competitive baselines.

REFERENCES

[1] Christopher G Atkeson, Benzun P Wisely Babu, Nandan Banerjee, Dmitry Berenson, Christoper P Bove, Xiongyi Cui, Mathew DeDonato, Ruixiang Du, Siyuan Feng, Perry Franklin, et al. No falls, no resets: Reliable humanoid behavior in the darpa robotics challenge. In *IEEE-RAS*, 2015.

[2] Anirudhan Badrinath, Yannis Flet-Berliac, Allen Nie, and Emma Brunskill. Waypoint transformer: Reinforcement learning via supervised learning with intermediate targets. *NeurIPS*, 2023.

[3] Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. Beit: Bert pre-training of image transformers. *arXiv:2106.08254*, 2021.

[4] David Brandfonbrener, Alberto Bietti, Jacob Buckman, Romain Laroche, and Joan Bruna. When does return-conditioned supervised learning work for offline reinforcement learning? *NeurIPS*, 2022.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *NeurIPS*, 2020.

[6] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020.

[7] Micah Carroll, Orr Paradise, Jessy Lin, Raluca Georgescu, Mingfei Sun, David Bignell, Stephanie Milani, Katja Hofmann, Matthew Hausknecht, Anca Dragan, et al. Uni [mask]: Unified inference in sequential decision problems. *NeurIPS*, 2022.

[8] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *NeurIPS*, 2021.

[9] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *ICRA*, 2018.

[10] André Correia and Luís A Alexandre. Hierarchical decision transformer. In *IROS*, 2023.

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

[12] Yiming Ding, Carlos Florensa, Pieter Abbeel, and Mariano Phielipp. Goal-conditioned imitation learning. *NeurIPS*, 2019.

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

[14] Scott Emmons, Benjamin Eysenbach, Ilya Kostrikov, and Sergey Levine. Rvs: What is essential for offline rl via supervised learning? *ICLR*, 2022.

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

[16] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *ICML*, 2019.

[17] Hiroki Furuta, Yutaka Matsuo, and Shixiang Shane Gu. Generalized decision transformer for offline hindsight information matching. *ICLR*, 2022.

[18] Dibya Ghosh, Chethan Anand Bhateja, and Sergey Levine. Reinforcement learning from passive data via latent intentions. In *ICML*, 2023.

[19] Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Devin, Benjamin Eysenbach, and Sergey Levine. Learning to reach goals via iterated supervised learning. *ICLR*, 2021.

[20] Jeffrey Hawke, Richard Shen, Corina Gurau, Siddharth Sharma, Daniele Reda, Nikolay Nikolov, Przemysław Mazur, Sean Micklethwaite, Nicolas Griffiths, Amar Shah, et al. Urban driving with conditional imitation learning. In *ICRA*, 2020.

[21] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *CVPR*, 2022.

[22] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *NeurIPS*, 2021.

[23] Zhengyao Jiang, Tianjun Zhang, Michael Janner, Yueying Li, Tim Rocktäschel, Edward Grefenstette, and Yuandong Tian. Efficient planning in a compact latent action space. *ICLR*, 2023.

[24] Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, 1993.

[25] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *CoRL*, 2018.

[26] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. *NeurIPS*, 2020.

[27] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *ICLR*, 2022.

[28] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *NeurIPS*, 2019.

[29] Aviral Kumar, Joey Hong, Anikait Singh, and Sergey Levine. When should we prefer offline reinforcement learning over behavioral cloning? *ICLR*, 2022.

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

[31] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *NeurIPS*, 2020.

[32] N. Li, S. Liu, Y. Liu, S. Zhao, and M. Liu. Neural speech synthesis with transformer network. In *AAAI*, 2019.

[33] Fangchen Liu, Hao Liu, Aditya Grover, and Pieter Abbeel. Masked autoencoding for scalable and generalizable decision making. *NeurIPS*, 2022.

[34] Yao Liu, Adith Swaminathan, Alekh Agarwal, and Emma Brunskill. Provably good batch off-policy reinforcement learning without great exploration. *NeurIPS*, 2020.

[35] Tung M Luu and Chang D Yoo. Hindsight goal ranking on replay buffer for sparse reward environment. *IEEE Access*, 2021.

[36] Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. *NeurIPS*, 2018.

[37] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, 1999.

[38] Seohong Park, Dibya Ghosh, Benjamin Eysenbach, and Sergey Levine. Hiql: Offline goal-conditioned rl with latent states as actions. *NeurIPS*, 2024.

[39] Ivaylo Popov, Nicolas Heess, Timothy Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv:1704.03073*, 2017.

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

[41] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[42] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

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

[44] Cem Subakan, Mirco Ravanelli, Samuele Cornell, Mirko Bronzi, and Jianyuan Zhong. Attention is all you need in speech separation. In *ICASSP*, 2021.

[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 2017.

[46] Philipp Wu, Arjun Majumdar, Kevin Stone, Yixin Lin, Igor Mordatch, Pieter Abbeel, and Aravind Rajeswaran. Masked trajectory models for prediction, representation, and control. *ICML*, 2023.

[47] Zhihui Xie, Zichuan Lin, Deheng Ye, Qiang Fu, Yang Wei, and Shuai Li. Future-conditioned unsupervised pretraining for decision transformer. In *ICML*. PMLR, 2023.

[48] Haoran Xu, Li Jiang, Li Jianxiong, and Xianyuan Zhan. A policy-guided imitation approach for offline reinforcement learning. *NeurIPS*, 2022.

[49] Mengdi Xu, Yikang Shen, Shun Zhang, Yuchen Lu, Ding Zhao, Joshua Tenenbaum, and Chuang Gan. Prompting decision transformer for few-shot policy generalization. In *ICML*. PMLR, 2022.

[50] Taku Yamagata, Ahmed Khalil, and Raul Santos-Rodriguez. Q-learning decision transformer: Leveraging dynamic programming for conditional sequence modelling in offline rl. In *ICML*. PMLR, 2023.

[51] Mengjiao Yang, Dale Schuurmans, Pieter Abbeel, and Ofir Nachum. Dichotomy of control: Separating what you can control from what you cannot. *ICLR*, 2023.

[52] Changxi You, Jianbo Lu, Dimitar Filev, and Panagiotis Tsiotras. Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning. *RAS*, 2019.

[53] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *NeurIPS*, 2020.

[54] Qinqing Zheng, Amy Zhang, and Aditya Grover. Online decision transformer. In *ICML*. PMLR, 2022.