QT-TDM: Planning With Transformer Dynamics Model and Autoregressive Q-Learning

Mostafa Kotb, Cornelius Weber, Muhammad Burhan Hafez, and Stefan Wermter, *Member, IEEE*

***Abstract*—Inspired by the success of the Transformer architecture in natural language processing and computer vision, we investigate the use of Transformers in Reinforcement Learning (RL), specifically in modeling the environment's dynamics using Transformer Dynamics Models (TDMs). We evaluate the capabilities of TDMs for continuous control in real-time planning scenarios with Model Predictive Control (MPC). While Transformers excel in long-horizon prediction, their tokenization mechanism and autoregressive nature lead to costly planning over long horizons, especially as the environment's dimensionality increases. To alleviate this issue, we use a TDM for short-term planning, and learn an autoregressive discrete Q-function using a separate Q-Transformer (QT) model to estimate a long-term return beyond the short-horizon planning. Our proposed method, QT-TDM, integrates the robust predictive capabilities of Transformers as dynamics models with the efficacy of a model-free Q-Transformer to mitigate the computational burden associated with real-time planning. Experiments in diverse state-based continuous control tasks show that QT-TDM is superior in performance and sample efficiency compared to existing Transformer-based RL models while achieving fast and computationally efficient inference.**

***Index Terms*—Model learning for control, machine learning for robot control, deep learning methods.**

I. Introduction

LEARNING an accurate predictive model of environment dynamics [1] is a challenging yet promising technique in Deep RL to enhance sample efficiency [2], [3], [4] and achieve generalization [5], [6], [7]. The Transformer architecture [8] is a strong candidate for dynamics modeling, as it proves to be an excellent sequence modeler and shows outstanding performance across various domains, including Natural Language Processing [9], Computer Vision [10], and Reinforcement Learning [11].

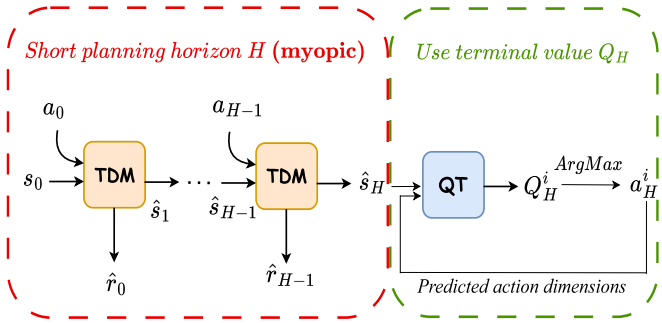


Fig. 1. QT-TDM Inference: The learned TDM model plans for short planning horizon H, while the learned QT model estimates an autoregressive terminal value Q_H^i for each action dimension a_H^i which guides the planning beyond the myopic horizon.

Transformer dynamics models (TDMs) [12], [13] have proven effective in *background planning* [14] scenarios, where an actor-critic model is trained on the imagined trajectories generated by the learned dynamics model. During inference, the learned actor-critic model selects the suitable actions. TDMs show an outstanding performance in discrete action spaces [13], [15] and in long-term memory tasks [16].

In *real-time planning* scenarios, where the learned dynamics model plans ahead by being unrolled forward from the current state to select the best action, TDMs encounter hurdles. Specifically, inference is slow and computationally inefficient [12], [17] due to the autoregressive token prediction and the per-dimension tokenization scheme, which increases sequence length as the environment's dimensionality increases. This makes planning for long horizons impractical, especially in the robotics domain, where fast inference is essential. Therefore, TDMs require more optimization on the architecture level, and more sample-efficient planning algorithms are needed to achieve faster real-time inference.

To this end, we introduce QT-TDM, a model-based algorithm that combines the strengths of a TDM and a model-free Q-Transformer (QT) [18]. Inspired by the TD-MPC algorithm [19], our proposed model achieves fast inference (as shown in Fig. 1) by combining a short planning horizon with a terminal value that is estimated by the Q-Transformer model which provides an estimate of a long-term return beyond the myopic planning horizon. Additionally, the sequence length is reduced by tokenizing the

Received 21 July 2024; accepted 4 November 2024. Date of publication 21 November 2024; date of current version 28 November 2024. This article was recommended for publication by Associate Editor R. Qi and Editor J. P. Desai upon evaluation of the reviewers' comments. This work was supported by the German Research Foundation DFG under Project CML (TRR 169). The work of Mostafa Kotb was supported by the Scholarship From the Ministry of Higher Education of the Arab Republic of Egypt. *(Corresponding author: Mostafa Kotb.)*

Mostafa Kotb is with the Knowledge Technology Group, Department of Informatics, Universität Hamburg, 22527 Hamburg, Germany, and also with the Mathematics Department, Faculty of Science, Aswan University, Aswan 81528, Egypt (e-mail: m.kotb@sci.aswu.edu.eg).

Cornelius Weber and Stefan Wermter are with the Knowledge Technology Group, Department of Informatics, Universität Hamburg, 22527 Hamburg, Germany (e-mail: cornelius.weber@uni-hamburg.de; stefan.wermter@uni-hamburg.de).

Muhammad Burhan Hafez is with the School of Electronics and Computer Science, University of Southampton, SO17 1BJ Southampton, U.K. (e-mail: burhan.hafez@soton.ac.uk).

The code is available at URL: https://github.com/2M-kotb/QT-TDM.

Digital Object Identifier 10.1109/LRA.2024.3504341

high-dimensional state space into a single token using a learned linear layer [11], as opposed to the conventional per-dimension tokenization method [12], [17].

The advantages of QT-TDM are twofold. First, the modular architecture, consisting of two components (TDM and QT) that can be trained and used individually, facilitates the replacement and testing of its components. Second, the Transformer-based architecture, which incorporates GPT-like Transformers [8], allows for scalability through training with diverse offline datasets, thereby enhancing generalization.

In this letter, we evaluate the proposed QT-TDM for real-time continuous planning with Model Predictive Control (MPC) using diverse state-based continuous control tasks from two domains: DeepMind Control Suite [20] and MetaWorld [21]. The results demonstrate the superior performance and sample efficiency of the QT-TDM model compared to baselines, while also achieving fast and computationally efficient inference. Our contributions can be summarised as follows:

- We propose QT-TDM, a Transformer-based model-based algorithm consisting of two modules (QT and TDM) in a modular architecture.
- QT-TDM addresses the slow and computationally inefficient inference associated with TDMs, while maintaining superior performance compared to baselines.

II. Related Work

1) Transformer Dynamics Model: Motivated by the success of Transformers in sequence modeling tasks, there has been a lot of recent attention on using Transformers as dynamics models. One of the earliest attempts is *TransDreamer* [16] which as implied by the name is a modification of the Dreamer model [14]. TransDreamer replaces the Recurrent State-Space Model (RSSM) [22] with a Transformer State-Space Model (TSSM), improving TransDreamer's performance in long-term memory tasks. *IRIS* [13] and *TWM* [15] are two sample-efficient model-based agents that are trained inside the imagination of a Transformer-based world model. IRIS' world model consists of a discrete autoencoder [23] as an observation model and a GPT-like Transformer [8] as a dynamics model, while the world model of TWM consists of a variational autoencoder [24] and a Transformer-XL [9]. Both models work with discrete action environments and they achieve impressive results on the Atari 100K benchmark. *Generalist TDM* [12] is the first attempt to use a learned TDM for continuous real-time planning with Model Predictive Control. Generalist TDM performs well in a single environment (i.e, specialist setting) and generalizes to unseen environments (i.e., generalist setting), in a few-shot and in zero-shot scenarios. Despite of its capabilities, it has two shortcomings. First, the training data is collected by an expert agent and not by its own interactions with the environment. Second, it suffers from slow inference because of the long-horizon planning and because of the design choices that are based on the Gato Transformer model [25] which uses the per-dimension tokenization scheme.

To overcome the above shortcomings, we introduce the QT-TDM model, which explores the environment to collect training data and has faster inference speed by shortening the planning horizon and utilizing the QT model [18] to estimate a long-term return beyond the short-term planning horizon.

2) Robotics Foundation Models: Inspired by the success of Vision/Language Foundation Models [26], there remains significant potential for the development of specialized Robotics Foundation Models (RFMs). Foundation Models, primarily based on Transformer architectures, are pre-trained on large-scale datasets and exhibit remarkable zero-shot and few-shot generalization capabilities. Examples of RFMs are RT-2 [27], Q-Transformer [18], Gato [25] and PaLM-E [28]. All existing RFMs adopt a model-free (model-agnostic) approach. However, many researchers argue that a model-based approach based on *Foundation World Models (FWMs)* is a promising direction for addressing complex robotics challenges [29]. While Generalist TDM [12] shows the potential of this direction, this work builds on it and further improves real-time planning capabilities and efficiency. In the future work section, we propose strategies to further advance QT-TDM toward the realization of FWMs.

III. Background

1) Reinforcement Learning: We formulate the problem of continuous control as an infinite-horizon Markov Decision Process (MDP) that can be formalized by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the continuous action space, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is a reward function, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$ is the transition function, and $\gamma \in [0, 1]$ is a discount factor. The goal of reinforcement learning is to learn a policy $\Pi_\theta : \mathcal{S} \mapsto \mathcal{A}$ from interacting with the environment that maximizes the expected cumulative discounted reward $\mathbb{E}_{\Pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t]$. In this work, the policy Π_θ is derived from planning with a learned dynamics model.

2) Model Predictive Control: In control, learning Π_θ is formulated as a trajectory optimization problem, in which at each step t, optimal actions $a_{t:t+H}$ over a finite horizon H are estimated to maximize the discounted sum of rewards:

$$\Pi_\theta(s_t) = \arg\max_{a_{t:t+H}} \mathbb{E}\left[\sum_{i=t}^{t+H} \gamma^i r_i\right], \tag{1}$$

and the first action a_t is executed. This method is known as *Model Predictive Control (MPC)*. Eq. (1) is not predicting long-term rewards beyond H. Consequently, incorporating a value function of the terminal state s_{t+H} provides an estimate of the long-term return, a method referred to as *MPC with a terminal value* [19]. An alternative approach, known as *MPC with value summation* [4], involves summing value functions over a finite horizon rather than summing rewards. In this work, we utilize Q-Transformer to estimate a terminal Q-value in a myopic planning horizon.

3) Autoregressive Q-Learning: Applying Q-learning with Transformers is challenging since Transformers require discretizing the action space into tokens to effectively apply the attention mechanism. Therefore, the standard Q-learning needs to be reformulated in order to be applied. In the Q-Transformer model [18], an autoregressive Q-learning formulation is proposed where each action dimension is treated as a separate

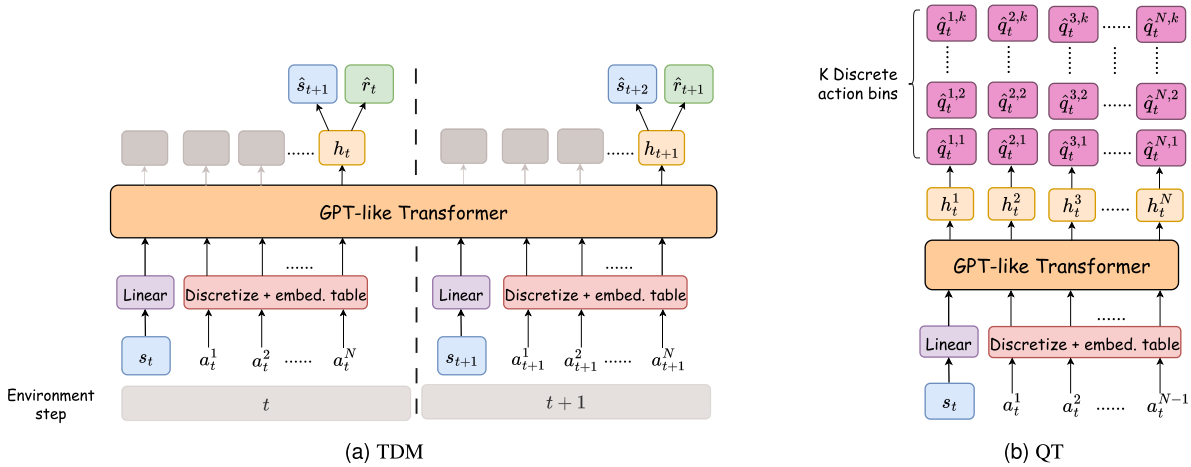


Fig. 2. QT-TDM Architecture, which consists of two modules: (a) TDM and (b) QT. Both modules have a GPT-like Transformer as a main component and share the same tokenization scheme. The state s_t is tokenized into a single token using a learned linear layer. A per-dimension tokenization is performed for the N-dimensional action by discretizing each dimension independently into K bins, then using an embedding table. The TDM module predicts the next state $\hat{s}_{t+1}$ and the reward $\hat{r}_t$ and is trained on L sampled time steps (for brevity, we only show two time steps). The QT module predicts a Q-value for each action dimension $\hat{q}_t^{i,1:K} \quad \forall i \in [1, \ldots, N]$.

time step. This way, each action dimension can be discretized individually, rather than discretizing the entire action space, thus avoiding the exponential growth in the discrete action space. An autoregressive discrete Q-function is employed which predicts a separate Q-value for each action dimension. Let $a_t = (a_t^1, \ldots, a_t^N)$ be an N-dimensional action at time step t. The autoregressive Q-function predicts a Q-value for each action dimension a_t^i that is conditioned on the state s_t and the previous action dimensions $a_t^{1:i-1}$:

$$Q(s_t, a_t^{1:i-1}, a_t^i) \; \forall i \in [1, \ldots, N]. \tag{2}$$

To train the Q-function, a per-dimension Bellman update is defined as follows:

$$Q(s_t, a_t^{1:i-1}, a_t^i) \leftarrow \begin{cases} \max\limits_{a_t^{i+1}} Q(s_t, a_t^{1:i}, a_t^{i+1}) & \text{if } i < N \\ r_t + \gamma \max\limits_{a_{t+1}^1} Q(s_{t+1}, a_{t+1}^1) & \text{if } i = N. \end{cases} \tag{3}$$

The Q-targets of all action dimensions except the last one are computed by maximizing over the discretized bins of their subsequent dimension within the same time step. The Q-target of the last dimension is computed by the discounted maximization of the first dimension of the next time step plus the reward. The reward is only applied on the last dimension as it is observed after executing the whole action. In addition, the Q-values are only discounted between time steps (i.e., discount factor γ is only applied for the last dimension), to ensure the same discounting as in the original MDP. The Q-Transformer model [18] has been evaluated in an offline RL fashion with large-scale robotic sparse reward tasks. In this work, we utilize the Q-Transformer model in an online RL fashion to estimate a terminal Q-value in a short-horizon real-time planning task, in order to achieve faster planning.

IV. Methodology

To resolve the trade-off between expressiveness and speed in TDMs, we introduce QT-TDM, a model-based RL algorithm that captures the environment's dynamics by modeling trajectory data using a Transformer Dynamics Model and achieves fast inference speed by utilizing a terminal Q-value to guide a short-horizon planning (see Fig. 1). In this section, we first describe the architecture of our model, then the training procedure, and finally explain how to apply the Q-Transformer during planning.

A. Architecture

QT-TDM model shown in Fig. 2 consists of two separated modules: Transformer Dynamics Model (TDM) and Q-Transformer Model (QT) [18].

TDM is implemented as a GPT-like Transformer [8] that computes a deterministic hidden state h_t conditioned on the states and actions of past steps. We consider only the hidden state corresponding to the last action dimension, as it attends to all preceding action dimensions (see Fig. 2(a); orange boxes vs. faded gray boxes). Predictors for the next state and reward are conditioned on the hidden state which are implemented as multilayer perceptrons (MLPs). The model components are as follows:

$$\text{Hidden state:} \quad h_t = f_\theta(s \le t, a^{1:N} \le t) \tag{4a}$$

$$\text{Transition:} \quad \hat{s}_{t+1} = g_\theta(h_t) \tag{4b}$$

$$\text{Reward:} \quad \hat{r}_t \sim p_\theta(\hat{r}_t | h_t), \tag{4c}$$

the reward model outputs the mean of a normal distribution.

The Q-Transformer model consists of a GPT-like Transformer and an autoregressive discrete Q-function that predicts a Q-value for each action dimension which is implemented as MLP. The Transformer computes a deterministic hidden state h_t^i for each

Algorithm 1: QT-TDM (Training).

Require: θ: initialized TDM parameters
ϕ, $\bar{\phi}$: initialized QT parameters, EMA parameters
η_d, η_q: learning rates
$\mathcal{B}$, ζ: replay buffer, EMA coefficient
L, N: sampled time steps, action dim.

for *each training step* **do**
  // Collect episode with QT-TDM and add to buffer
  $\mathcal{B} \leftarrow \mathcal{B} \cup \{s_t, a_t, r_t, s_{t+1}\}_{t=0}^{T-1}$
  for *num updates per episode* **do**
    $\{s_t, a_t, r_t, s_{t+1}\}_{t=1}^{L} \sim \mathcal{B}$; ▷ Sample trajectory
    // Update Dynamics Model (TDM)
    for $t = 1...L$ **do**
      $h_t = f_\theta(s \le t, a^{1:N} \le t)$; ▷ Hidden state
      $\hat{s}_{t+1} = g_\theta(h_t)$; ▷ Transition
      $\hat{r}_t \sim p_\theta(\hat{r}_t|h_t)$; ▷ Reward
    $\theta \leftarrow \theta - \eta_d \nabla_\theta \mathcal{L}_\theta^{Dyn}$; ▷ Equation 6
    // Update Q-Transformer (QT)
    for $i = 1...N$ **do**
      $h_t^i = f_\phi(s_t, a_t^{1:i-1})$; ▷ Hidden state
      $\hat{q}_t^{i,1:K} = g_\phi(h_t^i)$; ▷ Q-Values
    $\phi \leftarrow \phi - \eta_q \nabla_\phi \mathcal{L}_\phi^Q$; ▷ Equation 7
    // Update Target Network
    $\bar{\phi} \leftarrow (1-\zeta)\bar{\phi} + \zeta\phi$;

Algorithm 2: QT-TDM (Planning).

Require: θ, ϕ: TDM parameters, QT parameters
μ^0, σ^0: initial parameters of $\mathcal{N}$
J, J_{QT}: num. of samples, num. of QT samples
s_t, H, $\mathcal{I}$: current state, len. of horizon, iterations

for $n = 1...\mathcal{I}$ **do**
  Sample J action seq. from $\mathcal{N}(\mu^{n-1}, (\sigma^{n-1})^2\mathbf{I})$
  Sample J_{QT} action seq. using QT and TDM
  // Rollout trajectories and estimate total return $\mathcal{F}_\Gamma$
  for *all* $J + J_{QT}$ *action sequences* **do**
    for $t = 0...H-1$ **do**
      $h_t = f_\theta(s \le t, a^{1:N} \le t)$; ▷ Hidden state
      $\hat{s}_{t+1} = g_\theta(h_t)$; ▷ Transition
      $\mathcal{F}_\Gamma = \mathcal{F}_\Gamma + \gamma^t p_\theta(\hat{r}_t|h_t)$; ▷ Reward
    // Estimate the terminal Q-value using QT and add the value of last action dim. to $\mathcal{F}_\Gamma$
    $\mathcal{F}_\Gamma = \mathcal{F}_\Gamma + \gamma^H \max_{a_H^N} Q_\phi(\hat{s}_H, a_H^N)$
  Update μ^n and σ^n; ▷ Equation 9
return $a_t \sim \mathcal{N}(\mu_t^{\mathcal{I}}, (\sigma_t^{\mathcal{I}})^2\mathbf{I})$; ▷ First action is executed

action dimension a_t^i conditioned on the state s_t and previous action dimensions $a_t^{1:i-1}$. The model components are as follows:

$$\text{Hidden state:} \quad h_t^i = f_\phi(s_t, a_t^{1:i-1}) \;\; \forall i \in [1, \ldots, N] \tag{5a}$$

$$\text{Q-Value:} \quad \hat{q}_t^{i,1:K} = g_\phi(h_t^i) \;\; \forall i \in [1, \ldots, N], \tag{5b}$$

where K is the number of discretized action bins.

Both models, TDM and QT, tokenize the input sequences in the same way. Let $s \in \mathcal{S}$ be an M-dimensional state and $a \in \mathcal{A}$ is an N-dimensional continuous action. We follow [11] in tokenizing the state s into a single token obtained with a learned linear layer, rather than the conventional per-dimension tokenization [12], [17] which increases the input sequence length. We perform a per-dimension tokenization for the N-dimensional continuous action $a = (a^1, a^2, \ldots, a^N)$ by discretizing each dimension independently into K uniformally-spaced bins, then invoking the token embedding from a learned embedding table. TDM takes as input a sequence of $L \times (N+1)$ tokens, where L is time steps. QT takes as input a sequence of N tokens as it ignores the last action dimension.

B. Training

The dynamics model is trained in a self-supervised manner on segments of L time steps sampled from the replay buffer $\mathcal{B}$. We minimize the sum of a mean-squared error transition loss and a negative log-likelihood reward loss:

$$\mathcal{L}_\theta^{Dyn} = \sum_{t=1}^{L} \Big[\beta_1 \| g_\theta(h_t) - s_{t+1} \|_2^2 - \beta_2 \ln p_\theta(r_t|h_t) \Big], \tag{6}$$

where β_1 and β_2 are coefficients of the transition loss and the reward loss respectively.

The Q-Transformer model is trained by minimizing the Temporal Difference (TD) error loss defined by the per-dimension Bellman update [18] in Eq. (3)

$$\mathcal{L}_\phi^Q = Q_\phi(s_t, a_t) - Q_{\bar{\phi}}^*(s_t, a_t), \tag{7}$$

where $Q_\phi(s_t, a_t) = \{\hat{q}_t^i\}_{i=1}^N$ consists of the predicted Q-values of all action dimensions, and $Q_{\bar{\phi}}^*$ are the target Q-values predicted by a Q-target network whose parameters are an exponential moving average (EMA) of the Q-network. We use smooth L1 loss [30] as the TD-error which stabilizes training by avoiding exploding gradients. We follow [18] in employing n-step return [31] over action dimensions, and utilizing Monte Carlo return [32] only with sparse reward tasks (e.g., Reacher Easy), which helps accelerate learning. See Algorithm 1 for training pseudo code.

C. Planning

We evaluate the proposed QT-TDM model on real-time planning with MPC, where inference speed needs to be taken into consideration. The inference time grows with the planning horizon H, the number of planning samples J, and the dimensionality of the environment $\mathcal{D}$. While Transformers serve as large, expressive, and robust dynamics models, they are not optimized for fast inference [12]. The per-dimension tokenization and the autoregressive token prediction lead to a slow inference over long horizons. To solve this issue and achieve faster inference, we use a short planning horizon and employ the Q-Transformer model to estimate a terminal Q-value [19] that provides a long-term return beyond the short-term horizon. During planning with MPC, we sample J action sequences of length H from a time-dependent multivariate diagonal Gaussian distribution initialized by $(\mu^0, \sigma^0)_{t:t+H}$. Then, trajectories are generated using rollouts from the learned dynamics model (TDM), and

the total return $\mathcal{F}_\Gamma$ of a trajectory Γ is computed as follows:

$$\mathcal{F}_\Gamma = \mathbb{E}_\Gamma \left[\gamma^H \max_{a_H^N} Q_\phi(\hat{s}_H, a_H^N) + \sum_{t=0}^{H-1} \gamma^t p_\theta(\hat{r}_t | h_t) \right], \quad (8)$$

where $Q_\phi(\hat{s}_H, a_H^N)$ is the terminal Q-value of the last action dimension a_H^N. The distribution μ^n and σ^n at iteration n are updated to the top-k trajectories with the highest total returns $\mathcal{F}_\Gamma^\star$ as follows:

$$\mu^n = \frac{\sum_{i=1}^k \Omega_i \Gamma_i^\star}{\sum_{i=1}^k \Omega_i}, \quad \sigma^n = \sqrt{\frac{\sum_{i=1}^k \Omega_i \left(\Gamma_i^\star - \mu^n\right)^2}{\sum_{i=1}^k \Omega_i}}, \quad (9)$$

where $\Omega_i = e^{\tau(\mathcal{F}_{\Gamma,i}^\star)}$, τ is a temperature parameter controlling the sharpness of the weighting and $\Gamma_i^\star$ is the ith top-k trajectory. After a fixed number of iterations $\mathcal{I}$, the planning procedure terminates and a trajectory is sampled from the final updated distribution. The first action is only executed as we plan at each decision step t. In addition to sampling from the Gaussian distribution, we also sample J_{QT} action sequences from the learned Q-Transformer model. The planning procedure is summarized in Algorithm 2 and shown in Fig. 1

V. Experiments

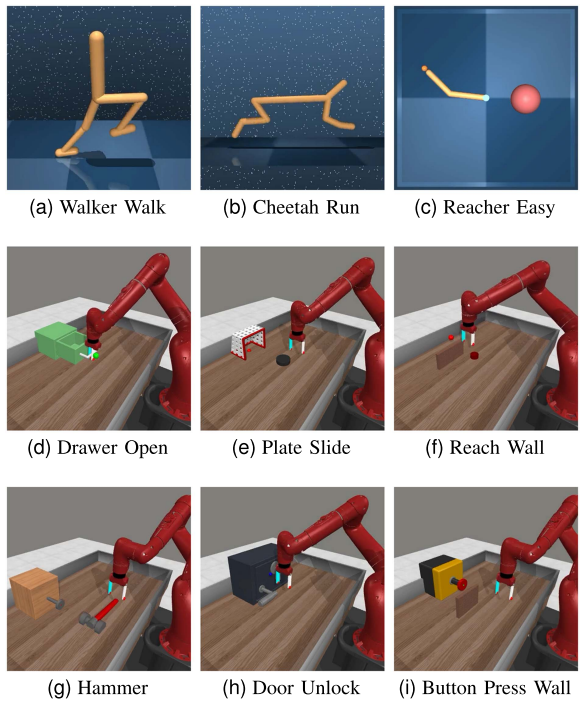
A. Description and Details

1) Benchmarks: We evaluate the performance of QT-TDM model on diverse state-based continuous control tasks from two benchmarks: DeepMind Control Suite (DMC) [20] and MetaWorld [21]. From DMC, we choose two high-dimensional locomotion tasks (*Walker Walk* and *Cheetah Run*) and a sparse reward task (*Reacher Easy*). MetaWorld contains 50 different robotic manipulation tasks, and because of time and computational constraints, we choose six tasks with various challenges. All tasks are shown in Fig. 3.

2) Baselines: Since the *Generalist TDM* [12] is the first Transformer-based model to perform continuous real-time planning, it serves as an eligible baseline. However, a comparison with it was not possible because its implementation is not publicly accessible. We compare the performance of QT-TDM against *PlaNet* [22], *DreamerV3* [6] and its two individual modules *(QT and TDM)* to serve as an ablation study as well. Both, PlaNet and DreamerV3 are model-based algorithms that use Recurrent State-Space Model (RSSM) as dynamics model. While PlaNet performs real-time planning with MPC, DreamerV3 performs background planning. Q-Transformer [18] is a Transformer-based model-free algorithm that uses an autoregressive Q-Learning. We provide an extensive evaluation of QT on diverse tasks in an online RL scenarios. TDM is a model-based algorithm that performs real-time planning but without the guidance from a terminal value function.

3) Experimental Setup: We list all the environment details for the tasks from the two benchmarks in Table I. For a fair comparison between the two Transformer-based model-based algorithms (QT-TDM and TDM), we use the same planning parameters shown in Table II. For the Recurrent-based model-based algorithm (PlaNet), we use its default planning parameters. All the compared models are evaluated after every 10K environment steps averaging over 10 episodes, except for DreamerV3, for which we use the final performance after convergence that we obtained from [33].

4) Computational Resources: For each task, we trained our method and the baselines with 3 different random seeds. We ran



(a) Walker Walk (b) Cheetah Run (c) Reacher Easy

(d) Drawer Open (e) Plate Slide (f) Reach Wall

(g) Hammer (h) Door Unlock (i) Button Press Wall

Fig. 3. Continuous Control Tasks. Two locomotion tasks with high-dimensional action space (Walker and Cheetah) and one sparse reward task (Reacher) from DMC [20]. Six robotic manipulation tasks (d)-(i) with various challenges from MetaWorld [21].

TABLE I
Environment Details Used Across All Methods for the Two Domains. We Use Action Repeat of 4 for DMC Tasks Except for Walker, Where Action Repeat of 2 is Used.

	DMC	**MetaWorld**
Episode length	1000	200
Action repeat	2 / 4	2
Effective length	500 / 250	100
Environment steps	500K	1M
Performance metric	Reward	Success
Observation dim. (M)	6 *(Reacher)* 17 *(Cheetah)* 24 *(Walker)*	39 *(all tasks)*
Action dim. (N)	6 *(Walker, Cheetah)* 2 *(Reacher)*	4 *(all tasks)*

TABLE II
MPC Planning Parameters Used for All Tasks

Parameter	**QT-TDM (ours) / TDM**	**PlaNet [22]**
Initial parameters (μ^0, σ^0)	$(0, 2)$	$(0, 1)$
Planning horizon H	3	12
Num. of samples J	512	1000
Num. of iterations $\mathcal{I}$	6	10
Num. of top-k trajectories	64	100

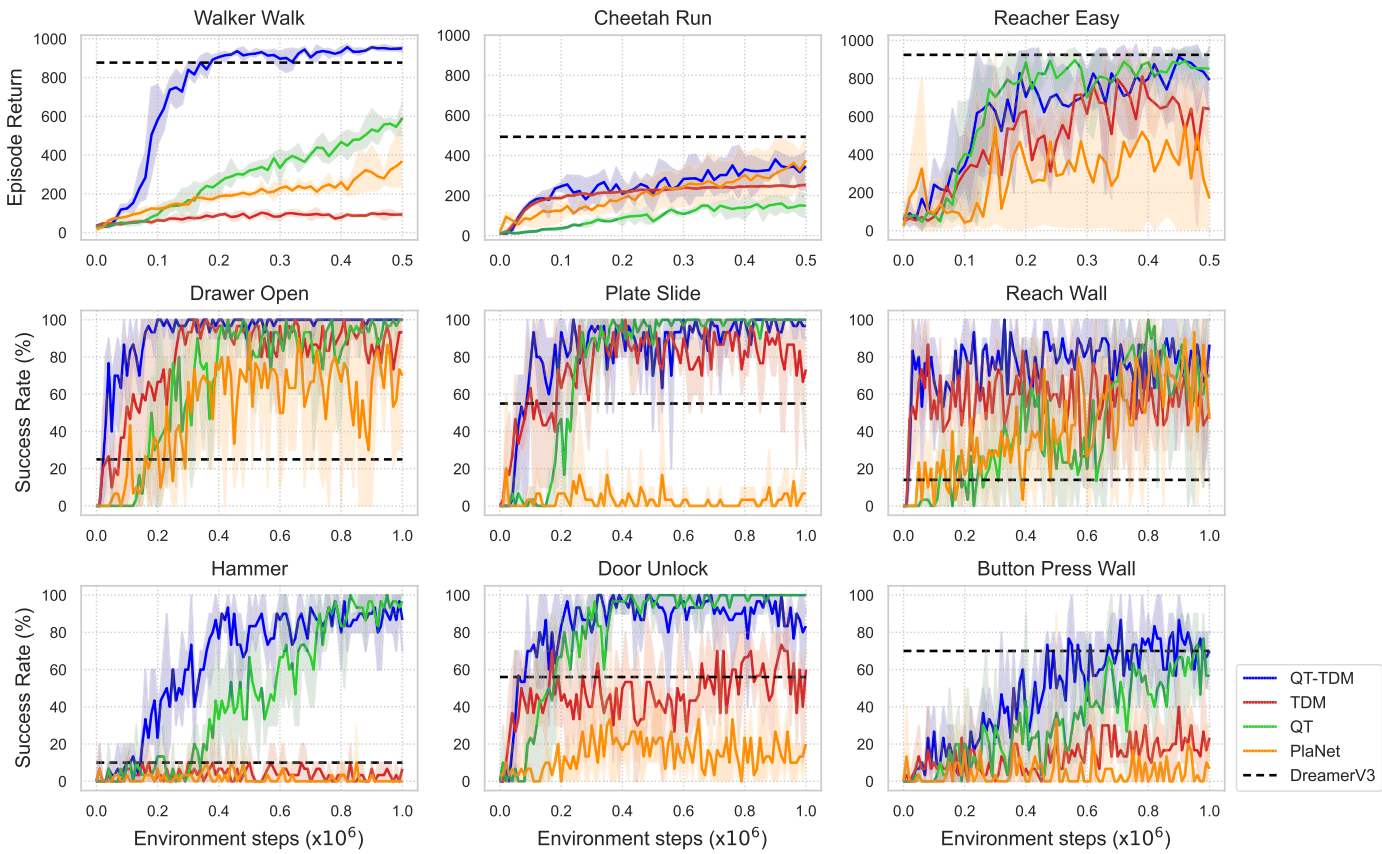


Fig. 4. Learning curves. Three tasks from DMC *(top row)*, episode return as performance metric. Six tasks from MetaWorld *(middle and bottom rows)*, success rate (%) as performance metric. Mean over 3 seeds; shaded areas are standard deviations. For DreamerV3, we report the final performance from [33].

our experiments with 6 Nvidia Quadro 6000 GPUs (24GB) using one GPU for one seed. For one DMC task, the total training and evaluation of our method takes on average 2 days while TDM takes 1.5 days. For one MetaWorld task, our method takes on average 4 days while TDM takes 3.5 days. The model-free QT takes 2 and 6 hours for training one DMC task and one MetaWorld task respectively.

B. Results

Results for all 9 tasks from the two benchmarks are shown in Fig. 4. We summarize our findings as follows:

1) Planning Efficiency: The two compared Transformer-based model-based algorithms (*QT-TDM* and *TDM*) perform real-time planning with a myopic planning horizon ($H = 3$). However, QT-TDM relies on a learned terminal Q-value to guide the short-horizon planning. *In DMC tasks*, TDM fails to solve the *Walker* task, its learning stagnates at approximately 200 returns after 100 K environment steps for the *Cheetah* task, and it relatively solves the sparse reward *Reacher* task at approximately 600 returns. In contrast, our proposed QT-TDM model successfully solves all tasks, except for the *Cheetah* task where it struggles a bit achieving approximately 400 returns. We achieved improved results with planning horizons $H = 5$ and $H = 9$ as shown in Fig. 5 but with the cost of higher inference time. *In MetaWorld tasks*, while TDM struggles to solve hard tasks such as *Hammer*, *Door Unlock*, and *Button Press Wall*, QT-TDM successfully solves all six tasks. QT-TDM outperforms TDM with only a $1.3\times$ increase in running time (e.g., from 1.5 days to 2 days for DMC tasks). This is more efficient than the over $2\times$ increase in running time required when extending the planning horizon ($H \geq 6$). This demonstrates that our proposed QT-TDM achieves efficient real-time planning in terms of both performance and computational demands.

2) Transformer Vs. Recurrent: Comparing our Transformer-based model against two Recurrent-based models highlights the superiority of TDMs in modeling dynamics. QT-TDM consistently outperforms PlaNet across all tasks, even though PlaNet utilizes a longer planning horizon (see table II). When compared to the state-of-the-art DreamerV3 which performs background planning, QT-TDM surpasses it in all MetaWorld tasks, while DreamerV3 achieves better performance in two DMC tasks (in Reacher Easy and Cheetah Run).

3) Planning Vs. Policy: The compared model-free Q-Transformer selects actions with a value-based policy by maximizing Q-values over the discretized bins for all action dimensions. The QT model successfully solves all tasks from MetaWorld, but with less sample efficiency than our QT-TDM model. In DMC tasks with high-dimensional action spaces (*Walker* and *Cheetah*), QT was extremely sample-inefficient compared to QT-TDM. In the Walker task, QT achieves approximately 150 returns at 100K environment steps and 600 returns at 500K environment steps, compared with our proposed QT-TDM that achieves approximately 600 returns at 100K environment steps and 900 returns at 500K environment steps. It is expected that the model-based algorithm is more sample-efficient than its

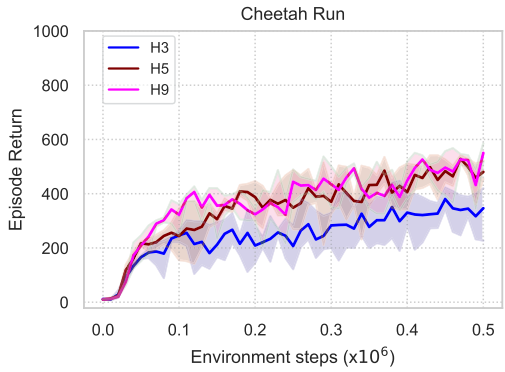


Fig. 5. QT-TDM with different planning horizon (H) on Cheetah Run task.

TABLE III
TRANSFORMER HYPERPARAMETERS

Hyperparameter	TDM	QT
Input sequence	$L \times (N+1)$ tokens	N tokens
Time steps (L)	20	1
Discretize action bins (K)	256	256
Embedding dim.	256	128
attention heads	4	8
Num. of layers	5	2
Embedding dropout	0.1	0.1
Attention dropout	0.1	0.1
Residual dropout	0.1	0.1

model-free counterpart [3], [4]. Nevertheless, the results demonstrate that QT [18] is a capable model-free algorithm that can perform effectively in both online and offline RL scenarios with sparse and dense rewards.

C. Implementation

Our GPT-like Transformer in both models (TDM and QT) is based on the implementation of *minGPT* [34]. See Table III for the Transformer hyperparameters. The reward and next state predictors in TDM are implemented as 3-layer MLPs with dimension 512, *Leaky ReLU* activation, and 0.01 dropout. We implement 2 Q-functions in QT model as 2-layer MLPs with dimension 128 and *ReLU* activation. TD-targets are computed as the minimum of these 2 Q-functions. Both models use Adam optimizer and Table IV shows the optimization hyperparameters for TDM and QT.

D. Complexity Analysis

We compare the complexity of our QT-TDM model against the Generalist TDM model [12] in terms of *model size* (i.e., number of parameters) and *inference speed*. Since the implementation of Generalist TDM is not available, we do not use quantitative measures for inference speed such as wall-time or FLOPs. Instead, we measure inference speed based on *planning horizon* H, *terminal value* Q^H, *number of planning samples* J and *number of tokens per timestep* $\mathcal{T}$ (see Table V). Due to the per-dimension tokenization, Generalist TDM requires $(M+N+1)$ tokens per timestep: M state tokens, N action tokens, and one reward token. In contrast, QT-TDM requires

TABLE IV
OPTIMIZATION HYPERPARAMETERS

Hyperparameter	Value
TDM	
Batch size	512
Learning rate (η_d)	1×10^{-4}
Weight decay	1×10^{-6}
Max gradient norm	30
Transition loss coef (β_1).	1.0
Reward loss coef. (β_2)	2.0
QT	
Batch size	512
Learning rate (η_q)	3×10^{-4} (fixed) *(DMC)* 3×10^{-4} (decay) *(MetaWorld)*
Weight decay	1×10^{-6}
Max gradient norm	20
EMA coef. (ζ)	0.005
Target $(\bar{\phi})$ update freq.	5 *(DMC)*, 10 *(MetaWorld)*
n-step return	3
Monte Carlo return	sparse reward tasks (Reacher)
Discount (γ)	0.98

TABLE V
COMPLEXITY RELATED PARAMETERS

Parameter	QT-TDM (ours)	Generalist TDM [12]
Num. of parameters	6M	77M
Planning horizon(H)	3	20 – 100
Terminal value (Q^H)	N	Not used
Num. of samples (J)	512	64 – 128
Num. of per timestep tokens $(\mathcal{T})$	$1+N$	$M+N+1$

only $(1+N)$ tokens per timestep by reducing the state tokens to a single token using a learned linear layer and by not using the reward token. Additionally, our model utilizes an 85% shorter planning horizon compared to the Generalist TDM model. However, our model leverages a terminal Q-value N times, with one value for each action dimension. Despite the additional steps, the total planning steps required by our model *$(3+N$ steps)* remain fewer than those required by the Generalist TDM *(at least 20 steps)*. Consequently, QT-TDM achieves faster inference speed with 92% fewer parameters than Generalist TDM. The computational demands of handling a high number of samples J can be mitigated by increasing parallelization (using multiple cores) [12].

VI. CONCLUSION AND FUTURE WORK

In this letter, we propose QT-TDM, a Transformer-based model-based algorithm that overcomes the slow and computationally inefficient inference associated with TDMs.

Model size: Although the QT-TDM model comprises two separate GPT-like Transformers, it has a relatively small number of parameters (6 M) compared to other Transformer-based models such as Generalist TDM (77 M). This helps mitigate the overfitting issue commonly encountered with high-capacity Transformers.

Inference speed: The proposed QT-TDM achieves fast real-time inference by reducing the number of per timestep tokens

and combining short-horizon planning with a learned terminal Q-value to guide the planning process. In addition to sampling random trajectories from a Gaussian distribution, we sample a small number of trajectories (only 24) from the learned Q-Transformer. Further improvements to inference speed could be achieved by reducing the number of random trajectories and incorporating more policy-sampled trajectories. We plan to investigate this strategy in future work. Another straightforward approach to increase inference speed is to train the Q-Transformer model using the imagined trajectories generated by TDM. The learned QT model can then be used to select actions during inference. This technique is referred to as *learning inside imagination* [14].

Limitations: First, QT-TDM relies heavily on the learned Q-values to guide the myopic planning horizon. However, learning a value function to approximate future returns is known to be unstable and prone to overestimation. We observe that the Q-Transformer model struggles to solve complex and hard-to-explore environments such as *pick Place* and *Shelf Place* from MetaWorld benchmark. As part of our future work, we plan to explore the use of an *ensemble of Q-functions* instead of just two Q-functions [33] which helps mitigate the overestimation issue. Additionally, we intend to employ a *categorical cross-entropy loss* as the TD-error rather than the traditional MSE regression loss, as it has been shown to be more effective and can accelerate the learning process [35]. Second, the use of per-dimension tokenization for the action space makes it difficult to scale to high-dimensional action spaces (e.g., humanoid robots) because it increases both the sequence length and the inference time.

Generalization: In this work, we evaluate QT-TDM in online RL scenarios to solve single tasks *(i.e., specialist agent)*. For future work, we plan to assess the generalization capabilities of the QT-TDM model *(i.e., generalist agent)* by training it with large, diverse offline datasets and evaluating its performance in unseen environments through few-shot and zero-shot scenarios.

Pixel observations: In this work, we experiment exclusively with state-based environments. We plan to extend our approach to pixel-based environments in future work by developing an observation model such as ViT [10] or discrete autoencoder [23].

Acknowledgment

The authors would like to thank Ozan Özdemir for his suggestions during the revision phase.

References

[1] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," in *Proc. Neural Inf. Process. Syst*, 2018, pp. 4759–4770.

[2] Y. Yao, L. Xiao, Z. An, W. Zhang, and D. Luo, "Sample efficient reinforcement learning via model-ensemble exploration and exploitation," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 4202–4208.

[3] Y. Seo et al., "Masked world models for visual control," in *Proc. Conf. Robot Learn.*, 2023, pp. 1332–1344.

[4] M. Kotb, C. Weber, and S. Wermter, "Sample-efficient real-time planning with curiosity cross-entropy method and contrastive learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2023, pp. 9456–9463.

[5] R. Sekar, O. Rybkin, K. Daniilidis, P. Abbeel, D. Hafner, and D. Pathak, "Planning to explore via self-supervised world models," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 8583–8592.

[6] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, "Mastering diverse domains through world models," 2023, *arXiv:2301.04104*.

[7] K. Lee, Y. Seo, S. Lee, H. Lee, and J. Shin, "Context-aware dynamics model for generalization in model-based reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 5757–5766.

[8] A. Vaswani et al., "Attention is all you need," in *Proc. Neural Inf. Process. Syst.*, 2017, pp. 6000–6010.

[9] Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-XL: Attentive language models beyond a fixed-length context," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 2978–2988.

[10] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Representations*, 2021.

[11] L. Chen et al., "Decision transformer: Reinforcement learning via sequence modeling," in *Proc. Neural Inf. Process. Syst.*, 2021, pp. 15084–15097.

[12] I. Schubert et al., "A generalist dynamics model for control," 2023, *arXiv:2305.10912*.

[13] V. Micheli, E. Alonso, and F. Fleuret, "Transformers are sample-efficient world models," in *Proc. Int. Conf. Learn. Representations*, 2023.

[14] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," in *Proc. Int. Conf. Learn. Representations*, 2020.

[15] J. Robine, M. Höftmann, T. Uelwer, and S. Harmeling, "Transformer-based world models are happy with 100k interactions," in *Proc. Int. Conf. Learn. Representations*, 2023.

[16] C. Chen, Y.-F. Wu, J. Yoon, and S. Ahn, "TransDreamer: Reinforcement learning with transformer world models," 2022, *arXiv:2202.09481*.

[17] M. Janner, Q. Li, and S. Levine, "Offline reinforcement learning as one big sequence modeling problem," in *Proc. Neural Inf. Process. Syst.*, 2021, pp. 1273–1286.

[18] Y. Chebotar et al., "Q-transformer: Scalable offline reinforcement learning via autoregressive Q-functions," in *Proc. Conf. Robot Learn.*, 2023, pp. 3909–3928.

[19] N. Hansen, X. Wang, and H. Su, "Temporal difference learning for model predictive control," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 8387–8406.

[20] S. Tunyasuvunakool et al., "dm_control: Software and tasks for continuous control," *Softw. Impacts*, vol. 6, 2020, Art. no. 100022.

[21] T. Yu et al., "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Proc. Conf. Robot Learn.*, 2020, pp. 1094–1100.

[22] D. Hafner et al., "Learning latent dynamics for planning from pixels," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 2555–2565.

[23] A. Van Den et al., "Neural discrete representation learning," in *Proc. Neural Inf. Process. Syst.*, 2017, pp. 6306–6315.

[24] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *Proc. Int. Conf. Learn. Representations*, 2014.

[25] S. Reed et al., "A generalist agent," *Trans. Mach. Learn. Res.*, 2022. [Online]. Available: https://openreview.net/pdf?id=1ikK0kHjvj

[26] M. Oquab et al., "Dinov2: Learning robust visual features without supervision," 2023, *arXiv:2304.07193*.

[27] B. Zitkovich et al., "RT-2: Vision-language-action models transfer web knowledge to robotic control," in *Proc. Conf. Robot Learn.*, 2023, pp. 2165–2183.

[28] D. Driess et al., "PaLM-E: An Embodied multimodal Language Model," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 8469–8488.

[29] Y. LeCun, "A path towards autonomous machine intelligence," *Open Rev.*, vol. 62, no. 1, pp. 1–62, 2022.

[30] R. Girshick, "Fast R-CNN," in *Proc. Int. Conf. Comput. Vis.*, 2015, pp. 1440–1448.

[31] R. Sutton, "Learning to predict by the method of temporal differences," *Mach. Learn.*, vol. 3, pp. 9–44, 1988.

[32] A. Wilcox, A. Balakrishna, J. Dedieu, W. Benslimane, D. Brown, and K. Goldberg, "Monte Carlo augmented actor-critic for sparse reward deep reinforcement learning from suboptimal demonstrations," in *Proc. Neural Inf. Process. Syst.*, 2022, pp. 2254–2267.

[33] N. Hansen, H. Su, and X. Wang, "TD-MPC2: Scalable, robust world models for continuous control," 2023, *arXiv:2310.16828*.

[34] A. Karpathy, "minGPT: A minimal PyTorch re-implementation of the OpenAI GPT (generative pretrained transformer) training," 2020. [Online]. Available: https://github.com/karpathy/minGPT

[35] J. Farebrother et al., "Stop regressing: Training value functions via classification for scalable deep RL," 2024, *arXiv:2403.03950*.