

Article

GT-TD3: A Kinematics-Aware Graph-Transformer Framework for Stable Trajectory Tracking of High-Degree-of-Freedom (DOF) Manipulators

Hanwen Miao ¹, Haoran Hou ¹, Zhaopeng Zhu ^{1,*}, Zheng Chao ¹ and Rui Zhang ²

¹ College of Mechanical and Transportation Engineering, China University of Petroleum, Beijing 102299, China; 2023010887@student.cup.edu.cn (H.M.); 2023010886@student.cup.edu.cn (H.H.); 2023010884@student.cup.edu.cn (Z.C.)

² College of Artificial Intelligence, China University of Petroleum, Beijing 102299, China; zhangr@student.cup.edu.cn

* Correspondence: zhuzp@cup.edu.cn

Abstract

Accurate trajectory tracking of redundant manipulators is difficult because the controller must simultaneously model local couplings between adjacent joints and global dependencies across the whole kinematic chain. Existing reinforcement learning methods typically employ multilayer perceptrons, which do not explicitly exploit manipulator structure and therefore show limited stability and representation ability in high-dimensional continuous control tasks. This paper proposes GT-TD3, a Graph Transformer-enhanced-Twin Delayed Deep Deterministic Policy Gradient framework, for redundant manipulator trajectory tracking. The proposed actor first converts the raw system state into joint-level node features and uses a graph neural network to extract local kinematic coupling information. A Transformer is then employed to capture long-range dependencies among joints. To strengthen the use of structural priors, topology- and distance-related bias terms are incorporated into the attention mechanism, enabling the network to encode manipulator structure during global feature learning. Experiments on a 7-DoF KUKA iiwa manipulator in PyBullet demonstrate that GT-TD3 outperforms MLP, pure GNN, and pure Transformer baselines in tracking performance. The proposed method achieves more stable training, faster convergence, and smoother and more accurate end-effector motion. The results show that the integration of local graph modeling and structure-aware global attention provides an effective solution for high-precision trajectory tracking of redundant manipulators.

Keywords: manipulators; trajectory tracking; twin delayed deep deterministic policy gradient; reinforcement learning; neural networks; kinematics; robot control



Received: 24 February 2026

Revised: 2 April 2026

Accepted: 3 April 2026

Published: 5 April 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

As robotics technology rapidly advances, the precise control of end-effectors has emerged as a primary research focus, directly determining task efficiency and safety in high-risk applications [1–4]. This requirement is particularly evident in offshore oil drilling. Here, robotic arms are increasingly deployed for pipe handling where tasks demanding stringent trajectory precision to prevent collisions, alignment errors, and equipment damage within confined, easily perturbed spaces. Similar precision constraints apply to automated assembly and the dismantling of nuclear facilities, where even marginal deviations risk structural failure [5]. Furthermore, enabling redundant and continuum robots to

navigate unstructured environments requires flexible path planning combined with dynamic obstacle avoidance [6]. Overcoming nonlinear dynamics and external disturbances to meet these strict criteria remains a substantial challenge.

Traditional control strategies, such as proportional-integral-derivative (PID) [7] and computed torque control, typically depend on exact mathematical models. Consequently, their performance degrades rapidly in the presence of modeling errors. Deep reinforcement learning (DRL) introduces a more adaptable alternative for manipulator control. Frameworks ranging from value-based approaches like DQN [8] and Double Q-learning [9] to actor–critic architectures like DDPG and TD3 [10] learn control policies directly through environmental interaction [11]. Despite their adaptability, standard DRL methods struggle with high-dimensional continuous control. While recent studies have successfully integrated neural network control [12], CNN-based predictive models [13], and variable impedance control [14] into robotic tasks, conventional multilayer perceptron (MLP) policies exhibit inherent limitations. The complex structures of high-degree-of-freedom systems generate expansive state and action spaces that easily overwhelm MLPs. More importantly, standard MLPs fail to exploit the inherent structural relationships along the robotic kinematic chain. Even though algorithms like TD3 enhance training stability via a twin-critic mechanism [10], their inability to explicitly encode joint connectivity restricts both learning efficiency and ultimate control performance.

To address these representation limits, neural architectures incorporating structural information offer a promising alternative. Graph neural networks (GNNs), for instance, naturally align with the interconnected joint structure of robotic arms by modeling node relationships through localized message passing [15–17]. Conversely, Transformers excel at capturing long-range dependencies via self-attention, demonstrating utility in navigation and obstacle avoidance [18]. Relying solely on standard Transformers, however, is suboptimal for manipulator control. Because the vanilla self-attention mechanism operates without joint topology or kinematic distance priors, the network must relearn these physical constraints from scratch. This observation suggests the need for an attention bias mechanism rooted in structural priors, effectively forcing the global dependency modeling process to respect the physical kinematic chain.

Motivated by this insight, we propose a structure-aware control framework named the Graph Transformer-Twin Delayed Deep Deterministic Policy Gradient (GT-TD3). Our approach integrates both GNN and Transformer modules directly into the TD3 actor architecture. Specifically, the GNN extracts local kinematic coupling features [19] while the Transformer, augmented with a kinematic-aware attention bias, models global joint dependencies [20]. By embedding topology and distance priors into the attention calculation via a biased attention mechanism, GT-TD3 significantly enhances the state representation capability without altering the underlying off-policy training dynamics of standard TD3. We evaluated this framework on a 7-DoF KUKA LBR iiwa manipulator within the PyBullet physics engine. Compared to standard MLP, pure GNN, and pure Transformer baselines, experimental results demonstrate that GT-TD3 achieves substantially higher algorithmic stability, faster convergence rates, and executes smoother, higher-precision end-effector trajectories.

2. Related Work

2.1. From Traditional Control to Deep Reinforcement Learning

Traditional methods, ranging from proportional-integral-derivative (PID) and computed torque control [21] to sampling-based planners such as A* [7] and rapidly exploring random trees (RRT) [22], provide a solid foundation for robotic motion generation and control. However, their practical performance often depends on accurate analytical models

and careful parameter tuning. In real-world applications, modeling errors, actuator uncertainties, sensor noise, and external disturbances can substantially reduce their effectiveness, especially in unstructured or dynamically changing environments. These challenges become more pronounced in high-dimensional robotic systems, where joint coupling and control complexity make conventional design increasingly difficult.

To address these limitations, deep reinforcement learning (DRL) has emerged as a promising alternative for continuous robotic control. By learning policies directly from interaction data, DRL offers stronger adaptability in nonlinear and high-dimensional settings. In particular, end-to-end actor–critic methods such as DDPG, TD3 [10], and soft actor–critic (SAC) have achieved strong results in dexterous manipulation and navigation tasks [11].

2.2. Structured Neural Architectures in Robotic Control

To overcome the representation bottlenecks of standard multilayer perceptrons (MLPs), researchers have increasingly integrated structural priors into neural architectures. Graph neural networks (GNNs) exemplify this shift, naturally accommodating the chain-like topology of robotic manipulators by explicitly embedding joint relationships [16,23]. Frameworks such as NerveNet [24] and graph policy gradients [3], alongside various graph-based motion planners [17,25,26], successfully leverage localized message passing for modular robot control.

Nevertheless, GNNs exhibit inherent limitations when scaled to high-degree-of-freedom systems. Because they depend exclusively on local message passing, transferring information between distant nodes—such as from the robot base to the end-effector—requires numerous propagation steps. As theoretically analyzed by Alon and Yahav [19], this prolonged routing often induces over-smoothing and significantly impedes global information exchange.

Conversely, Transformers excel at modeling long-range dependencies, utilizing self-attention mechanisms to establish direct connections between distant states across various robotic applications [18,26]. Yet, applying vanilla Transformers to robotic control introduces a different set of challenges. Standard implementations treat joint states as unconstrained sequences, essentially discarding the robot’s physical topology. As highlighted in the recent literature [20], stripping away these inherent constraints forces the network to redundantly relearn fundamental kinematics from scratch, thereby degrading learning efficiency. Furthermore, the absence of explicit structural priors compromises local feature perception, frequently triggering training instability during continuous control [27]. To reconcile these conflicting paradigms, our framework equips the Transformer with a kinematic-aware attention bias module. By encoding joint topology, relative configurations, and physical distances directly into the attention matrix, the network is compelled to respect the manipulator’s kinematic realities during global dependency modeling, thereby maximizing structural utilization and mitigating inefficient exploration.

2.3. Position of Our Work

While the application of neural networks to robotic inverse kinematics (IK) is well-documented, ranging from early foundational explorations by Tejomurtula and Kak [28] to more recent validations of approximation accuracy and training stability by Gao [29] and Cagigas-Muñiz [30], our research diverges fundamentally from simply reaffirming these capabilities. Instead, the specific novelty and improvements of this paper manifest across three distinct dimensions.

First, whereas the traditional literature predominantly formulates IK as a static mapping problem, we reframe it as a continuous, dynamic control challenge. By continuously

outputting joint velocity commands based on instantaneous states and goal vectors, our approach is inherently optimized for the fluid trajectory tracking demands of a 7-DoF redundant manipulator.

Second, rather than relying on unstructured dense layers to approximate kinematic relationships, we synthesized a hybrid GNN–Transformer architecture. By natively incorporating link distances, joint bounds, and physical symmetries into the network’s computational graph, the policy actively leverages the exact structural fingerprint of the robotic arm itself.

Finally, instead of evaluating success purely via isolated inverse solution errors, our framework simultaneously addresses end-effector precision, aggregate trajectory deviations, and action smoothness constraints. Consequently, our proposed reinforcement learning paradigm moves beyond computing mere reachable waypoints, delivering execution-ready, structure-integrated continuous control that is significantly more stable for real-world deployment.

3. Methods

3.1. Overall Framework of GT-TD3

Our study addresses the challenge of target-conditioned trajectory tracking control by employing a 7-degree-of-freedom KUKA iiwa robotic arm within the PyBullet simulation environment. At the initialization of each episode, the environment randomly samples a reachable Cartesian target point, prompting the system to generate a straight reference trajectory that spans from the end-effector’s initial position directly to this goal. To standardize the execution phase, the temporal length of this trajectory is strictly fixed at 200 control steps. Consequently, the fundamental learning objective of the policy is to formulate continuous joint velocity commands. These control signals must not only propel the end-effector accurately toward the target point, but they must also closely track the reference path while ensuring overall kinematic smoothness.

In this work, we developed GT-TD3 on top of the standard TD3 framework specifically to tackle high-degree-of-freedom robotic arm control. To keep training stable, we exclusively enhanced the actor network’s architecture, leaving the critic branch with its traditional twin Q-network setup. The core intuition behind GT-TD3 is as follows: instead of feeding the network a raw, flat state vector, we first reconstruct it into joint-level nodes. A GNN then extracts the localized coupling relationships between adjacent joints. After that, a Transformer equipped with a kinematic-aware structural bias steps in to model the long-range global dependencies. Finally, a gated fusion module seamlessly blends these local and global features to output the continuous control action.

Rather than directly combining a GNN and a Transformer into TD3, the proposed actor is constructed through four sequential steps. First, we encode the raw environmental state into seven individual joint nodes. Second, we drive local message passing across these nodes based strictly on the robot’s serial kinematic chain. Third, we map these graph-processed features into the Transformer space, injecting a kinematic-aware attention bias right on top of the standard sinusoidal positional encoding. Fourth, a gated fusion and readout module computes the actor’s final action. This architecture enables the policy network to jointly exploit local topological structure and long-range inter-joint dependencies. The overall architecture of the proposed GT-TD3 framework is illustrated in Figure 1. Figure 1 summarizes the main components of the proposed framework, which are described in detail in the following sections.

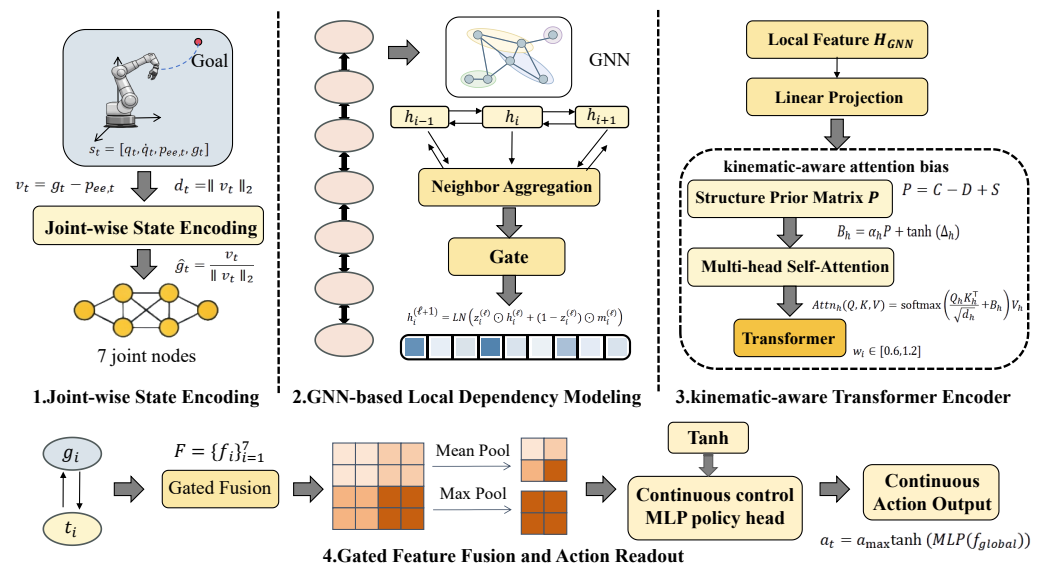


Figure 1. The overall architecture of the proposed GT-TD3 actor network, consisting of joint-wise state encoding, GNN-based local dependency modeling, kinematic-aware Transformer encoder, and gated feature fusion.

3.2. TD3-Based Learning Framework

This method is built on the TD3 framework, but the main modification is made on the actor side, where the policy function $\pi_{\theta}(s_t)$ is redesigned as a GNN–Transformer network that explicitly follows the joint structure of the manipulator. The critic branch remains unchanged.

As shown in Figure 2, the overall framework still follows the asymmetric actor–critic design of TD3. In other words, only the policy network is structurally enhanced.

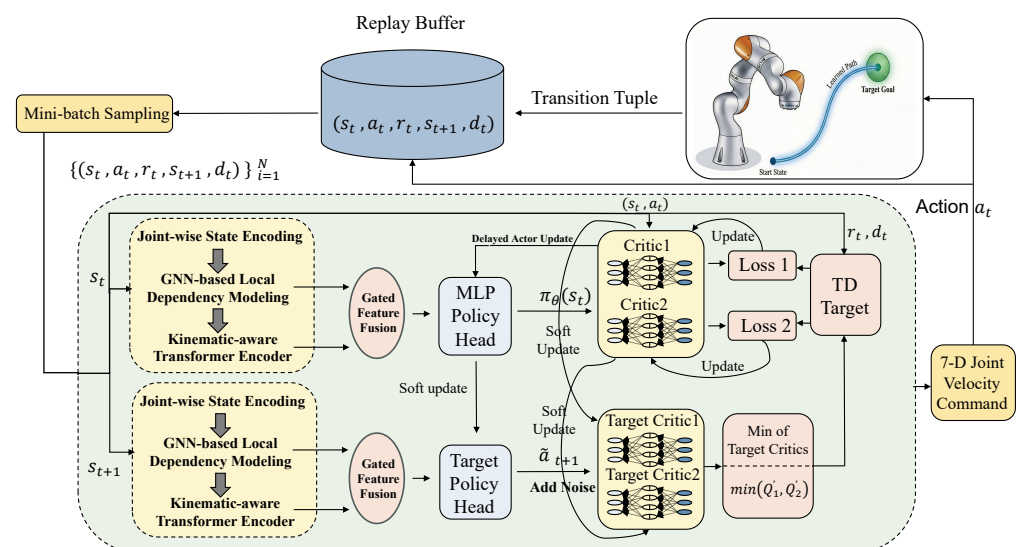


Figure 2. The overall TD3-based deep reinforcement learning framework.

More specifically, the method employs two independent critics, denoted as $Q_{\phi_1}(s_t, a_t)$ and $Q_{\phi_2}(s_t, a_t)$. Both critics use standard multilayer perceptrons to model the concatenated state-action pair. The detailed network architecture is shown in Figure 3. This keeps the value branch simple. It also helps maintain stable training and provides the actor with value gradients that are less affected by estimation bias.

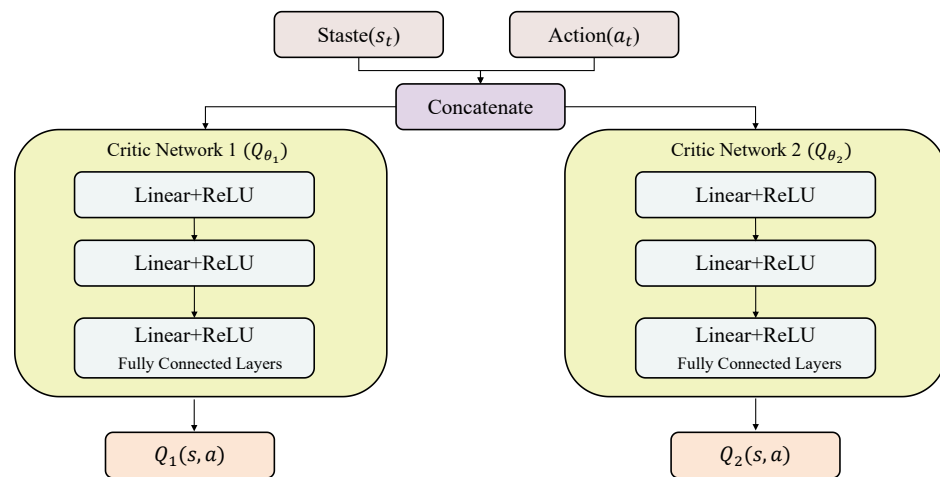


Figure 3. The architecture of the twin critic networks. Unlike the structure-aware actor, the critics utilize standard MLPs to process concatenated state-action pairs, ensuring training stability during Q-value estimation.

Given a transition sample $(s_t, a_t, r_t, s_{t+1}, d_t)$, where $d_t \in \{0, 1\}$ is the terminal flag, the non-terminal mask is defined as $m_t = 1 - d_t$. The target value in TD3 is then written as:

$$y_t = r_t + \gamma m_t \min_{k \in \{1,2\}} Q_{\phi'_k}(s_{t+1}, \tilde{a}_{t+1}) \quad (1)$$

where the next action is generated by target policy smoothing:

$$\tilde{a}_{t+1} = \text{clip}(\pi_{\theta'}(s_{t+1}) + \epsilon, -a_{\max}, a_{\max}) \quad (2)$$

$$\epsilon = \text{clip}(\mathcal{N}(0, \sigma^2 I), -c, c) \quad (3)$$

Here, θ' and ϕ'_k denote the target parameters of the actor and the two critics, respectively. The symbols σ and c represent the standard deviation of the policy noise and the clipping threshold of that noise. This formulation follows the standard TD3 setting.

Like the original TD3, both critics are trained to regress toward the same clipped double-Q target. The loss function, however, is slightly different. Instead of mean squared error, this implementation uses the Smooth L1 loss:

$$L_{\text{critic}} = \text{SmoothL1}(Q_{\phi_k}(s_t, a_t), y_t), k \in \{1, 2\} \quad (4)$$

This choice improves robustness when the TD error contains occasional large values and makes critic training less sensitive to outliers. The two critics are updated with separate optimizers, and their target networks are softly updated after each training iteration. The actor, in contrast, follows the delayed update mechanism of TD3, which means that it is updated once every d critic updates.

The actor objective in this implementation is defined as:

$$L_{\text{actor}} = -\mathbb{E}_{s_t \sim \mathcal{D}}[Q_{\phi_1}(s_t, \pi_{\theta}(s_t))] + \lambda \mathbb{E}_{s_t \sim \mathcal{D}}[\text{mean}(\pi_{\theta}(s_t)^2)] \quad (5)$$

Here, λ is 0.001. The first term is the standard deterministic policy gradient objective. The second term is an action-magnitude regularizer. This additional regularization term helps alleviate the weak-gradient problem that can appear when the $\tanh$ output approaches saturation. This, therefore, makes policy optimization more stable in practice.

After each actor update, the actor target network is also softly updated. In addition, the gradient norms of both the actor and the critics are clipped to an upper bound of 1.0. This

further improves the training stability. Therefore, the contribution of this method does not come from changing the off-policy learning rule itself. The novelty lies in the actor design. More specifically, the main contribution is the combination of structured state representation and hierarchical feature modeling on the policy side. This is where the method differs from standard TD3. By placing the structural prior only in the actor, the method keeps the critic simple, stable, and easy to optimize. At the same time, it avoids the extra instability that graph modules or attention layers may introduce into Q-value estimation. The overall framework of our improved GT-TD3 algorithm is presented in Algorithm 1.

Algorithm 1. Training Procedure of GT-TD3

Input: actor π_θ , twin critics Q_{ϕ_1}, Q_{ϕ_2} , target networks $\pi_{\theta'}, Q_{\phi'_1}, Q_{\phi'_2}$, replay buffer $\mathcal{D}$, total steps T , warm-up steps T_0 , policy delay d , discount factor γ , soft update rate τ , target noise σ , clipping threshold c
 Initialize target networks: $\theta' \leftarrow \theta, \phi'_1 \leftarrow \phi_1, \phi'_2 \leftarrow \phi_2$

for $t = 1, 2, \dots, T$ **do**

 Observe state s_t .

 Select action a_t : random exploration if $t \leq T_0$, otherwise $a_t = \pi_\theta(s_t) + \epsilon$

 Execute a_t , observe r_t, s_{t+1} , and terminal flag d_t

 Store $(s_t, a_t, r_t, s_{t+1}, d_t)$ in $\mathcal{D}$

 Sample a mini-batch $(s_t, a_t, r_t, s_{t+1}, d_t)$ from $\mathcal{D}$

Compute target action:

$$\tilde{a}_{t+1} = \text{clip}(\pi_{\theta'}(s_{t+1}) + \text{clip}(\mathcal{N}(0, \sigma^2), -c, c))$$

Compute target value:

$$y_t = r_t + (1 - d_t)\gamma \min(Q_{\phi'_1}(s_{t+1}, \tilde{a}_{t+1}), Q_{\phi'_2}(s_{t+1}, \tilde{a}_{t+1}))$$

 Update critics by minimizing

$$L_{\text{critic}}^{(i)} = \text{SmoothL1}(Q_{\phi_i}(s_t, a_t), y_t), i \in \{1, 2\}$$

If $t \bmod d = 0$, update actor by minimizing

$$L_{\text{actor}} = -\mathbb{E}[Q_{\phi_1}(s_{t+1}, \pi_\theta(s_{t+1}))] + \lambda \mathbb{E}[\text{mean}(\pi_\theta(s_{t+1})^2)]$$

Soft-update target networks

end for

3.3. Joint-Based State Encoding

To better exploit the kinematic structure of the 7-DoF serial manipulator, the raw 20-dimensional environment state is first reorganized into seven joint-level nodes. The raw state is defined as:

$$s_t = [q_t, \dot{q}_t, p_{ee,t}, g_t] \quad (6)$$

where $q_t \in \mathbb{R}^7$ denotes the joint positions, $\dot{q}_t \in \mathbb{R}^7$ denotes the joint velocities, $p_{ee,t} \in \mathbb{R}^3$ denotes the current end-effector position, and $g_t \in \mathbb{R}^3$ denotes the goal position.

The encoder first constructs several goal-related quantities. These terms provide a compact description of the current task.

$$v_t = g_t - p_{ee,t} \quad (7)$$

$$\rho_t = \|v_t\|_2 \quad (8)$$

$$\hat{g}_t = \frac{v_t}{\max(\rho_t, \varepsilon)} \quad (9)$$

Here, ρ_t is the Euclidean distance from the end effector to the goal, $\hat{g}_t$ is the unit direction vector pointing toward the goal, and ε is a small constant introduced to avoid division by zero.

The scale-dependent quantities are then normalized. This keeps the input range more stable during training. Specifically, joint velocities are normalized by the maximum velocity bound of 1.5 rad/s, while the goal distance is normalized by the workspace scale of 1.2 m. In addition, a cumulative joint-angle feature is introduced to describe the accumulated pose from the base to the current joint:

$$q_{i,t}^{\text{cum}} = \frac{\sum_{k=1}^i q_{k,t}}{7 \times 3.0} \quad (10)$$

Under the default setting, each joint node is assigned a 6-dimensional hand-crafted feature vector:

$$\tilde{x}_{i,t} = \left[q_{i,t}, \frac{\dot{q}_{i,t}}{1.5}, q_{i,t}^{\text{cum}}, \frac{\rho_t}{1.2}, \hat{g}_{x,t}, \hat{g}_{y,t} \right] \quad (11)$$

One detail should be clarified here. The default implementation does not use the full three-dimensional direction vector $(\hat{g}_x, \hat{g}_y, \hat{g}_z)$. Instead, only $\hat{g}_x$ and $\hat{g}_y$ are retained in the node feature. This choice mainly serves to control the node dimension, and it is also reasonable because $\hat{g}_t$ is a unit vector, so part of the information in $\hat{g}_z$ can still be reflected indirectly through the other components.

To help the network distinguish different physical joints, a 1-dimensional learnable joint identity embedding e_i is further added to each node. It is concatenated with the hand-crafted feature as:

$$x_{i,t} = [\tilde{x}_{i,t}, e_i] \quad (12)$$

As a result, the actual input dimension of each node is 7 under the default setting. Six dimensions come from the hand-crafted state features, and the last dimension comes from the learnable joint identity embedding. The final joint-node sequence is written as:

$$X_t = [x_{1,t}, x_{2,t}, \dots, x_{7,t}]^\top \in \mathbb{R}^{7 \times 7} \quad (13)$$

This encoding preserves information at three different levels. It is not just a simple feature split. First, $q_{i,t}$ and $\dot{q}_{i,t}$ describe the instantaneous state of each joint. Second, $q_{i,t}^{\text{cum}}$ reflects the accumulated pose from the base to joint i , which helps characterize the overall manipulator configuration. Third, ρ_t , $\hat{g}_{x,t}$, and $\hat{g}_{y,t}$ broadcast task-related goal information to every joint node. In this way, both the later graph module and the Transformer module can perceive the current control objective from local and global perspectives.

3.4. Local Dependency Modeling Through Gated Graph Aggregation

After joint-level encoding, the seven nodes are connected according to the serial kinematic topology of the manipulator. This gives the model an explicit structural prior. Let $A \in \mathbb{R}^{7 \times 7}$ denote the adjacency matrix with self-loops, and let row-wise normalization be applied to A . With this design, each joint preserves its own feature during propagation while also receiving messages from its upstream and downstream neighbors.

Before graph aggregation, the node features are first projected into a unified hidden space using a two-layer node-wise MLP:

$$h_i^{(0)} = \text{MLP}_{\text{node}}(x_{i,t}) \quad (14)$$

The default hidden dimension is $d_g = 64$. This projection prepares the node features for subsequent local interaction.

A two-layer gated graph aggregation module is then used to capture local dependencies. For the ℓ -th layer, the update is defined as:

$$m_i^{(\ell)} = W_m^{(\ell)} \sum_j A_{ij} h_j^{(\ell)} \quad (15)$$

$$z_i^{(\ell)} = \sigma \left(W_z^{(\ell)} \left[h_i^{(\ell)} \parallel m_i^{(\ell)} \right] \right) \quad (16)$$

$$h_i^{(\ell+1)} = \text{LN} \left(z_i^{(\ell)} \odot h_i^{(\ell)} + (1 - z_i^{(\ell)}) \odot m_i^{(\ell)} \right) \quad (17)$$

Here, $h_i^{(\ell)}$ denotes the hidden feature of the i -th joint node at layer ℓ , $m_i^{(\ell)}$ denotes the aggregated neighborhood message, and $z_i^{(\ell)}$ denotes the element-wise gating vector. The function $\sigma(\cdot)$ is the sigmoid activation, and $\text{LN}(\cdot)$ denotes layer normalization. The learnable matrices $W_m^{(\ell)}$ and $W_z^{(\ell)}$ are used for neighborhood message mapping and gate generation, respectively. The detailed structure of this local dependency modeling module is depicted in Figure 4.

This design allows each joint to adaptively balance two sources of information. One comes from the joint itself, and the other comes from its local kinematic neighborhood. When neighboring information is more informative, the model can place more weight on $m_i^{(\ell)}$. When the current joint state is more critical, it can preserve more of $h_i^{(\ell)}$. This mechanism is especially suitable for the KUKA iiwa, which is a typical serial manipulator with strong local coupling between adjacent joints.

3.5. Kinematic-Aware Transformer with Sinusoidal Positional Encoding

After the local graph aggregation stage, the model obtains joint-level local representations:

$$H_t^G = [h_1^G, \dots, h_7^G]^\top \in \mathbb{R}^{7 \times d_g} \quad (18)$$

These features are not fed into the Transformer directly. To make them suitable as Transformer tokens, they are first projected into a shared hidden space through a linear mapping:

$$Z_t^{(0)} = H_t^G W_{in} + b_{in} \in \mathbb{R}^{7 \times d} \quad (19)$$

where the default hidden dimension is $d = 128$.

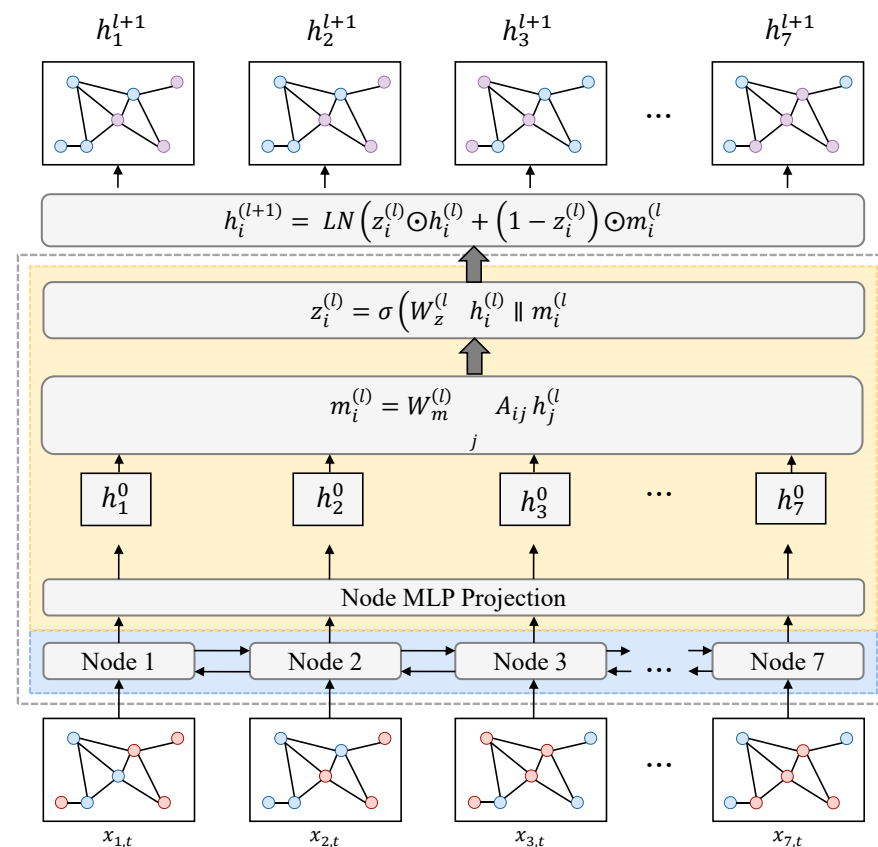


Figure 4. The detailed structure of the GNN-based local dependency modeling module using gated graph aggregation.

One point should be clarified here. The Transformer does not take the raw state as input. Instead, it receives the joint representations that have already been processed by the GNN. The two modules are therefore connected in series rather than in parallel. The model first captures local dependencies and then models global interactions. Before entering the first self-attention layer, standard sinusoidal positional encoding is added to the token sequence:

$$\tilde{Z}_t^{(0)} = Z_t^{(0)} + E_{pos} \quad (20)$$

where $E_{pos} \in \mathbb{R}^{7 \times d}$ denotes the sinusoidal positional encoding matrix.

This step preserves the order information of the joint sequence. It also helps the Transformer distinguish different joint positions within the chain. Under the default setting, the Transformer encoder contains two self-attention layers and four attention heads. It adopts a Pre-LayerNorm architecture and uses a feed-forward network with a fourfold expansion ratio. The internal structure of the kinematics-aware Transformer encoder is presented in Figure 5.

Unlike physically grounded kinematics-aware formulations such as Sheng et al. (2024) [31], the mechanism used here does not explicitly model manipulator dynamics or Jacobian constraints. Instead, it introduces a kinematics-aware structural bias into the self-attention computation by using joint topology, path distance, and joint-range priors derived from the serial manipulator. The structural prior matrix is defined as:

$$P = C - D + S \quad (21)$$

where D is the normalized path-distance matrix, C is the motion-range coupling matrix, and S is the symmetry reward matrix. Among these three terms, D describes how far two

joints are from each other along the real kinematic chain. It is not a simple index distance. For the KUKA iiwa 7, the adjacent joint-link distances are set as:

$$l = [132.9, 87.8, 151.4, 90.6, 122.3, 141.1, 80.4] \text{ mm}$$

and the corresponding joint motion ranges are:

$$r = [170^\circ, 120^\circ, 170^\circ, 120^\circ, 170^\circ, 120^\circ, 175^\circ]$$

For any joint pair (i, j) , the normalized path distance is defined as:

$$D_{ij} = \frac{\sum_{k=\min(i,j)}^{\max(i,j)-1} l_k}{\max_{u,v} \sum_{k=\min(u,v)}^{\max(u,v)-1} l_k}, D_{ij} \in [0, 1] \quad (22)$$

This term penalizes attention between joints that are farther apart along the manipulator chain. In this way, the model can retain a clearer sense of kinematic structure during global interaction.

The motion-range coupling term is defined as:

$$C_{ij} = \frac{r_i r_j}{\max_{u,v} (r_u r_v)}, i \neq j \quad (23)$$

with $C_{ii} = 0$.

This term assigns larger values to joint pairs with broader motion capability. It reflects the obvious intuition that such joints are more likely to play a stronger role in coordinated movement. In addition, the KUKA iiwa 7 shows an alternating pattern of large-range and small-range joints. To describe this pattern, a simple symmetry reward is introduced:

$$S_{ij} = \begin{cases} 0.1, & \text{if } r_i \text{ and } r_j \text{ belong to the same kinematic category} \\ 0, & \text{otherwise} \end{cases} \quad (24)$$

The final structural prior is therefore written as:

$$P_{ij} = C_{ij} - D_{ij} + S_{ij} \quad (25)$$

Based on this prior, the bias term for the h -th attention head is defined as:

$$B_h = \alpha_h P + \tanh(\Delta_h) \quad (26)$$

where α_h is a learnable head-level scaling factor and $\Delta_h \in \mathbb{R}^{7 \times 7}$ is a learnable residual bias matrix.

The attention of the h -th head is then computed as:

$$\text{Attn}_h(Q, K, V) = \text{softmax} \left(\frac{Q_h K_h^\top}{\sqrt{d_h}} + B_h \right) V_h \quad (27)$$

where Q_h , K_h , and V_h are obtained by linear projections of the current layer input.

This design makes the attention mechanism more likely to focus on structurally related joint pairs. At the same time, $\tanh(\Delta_h)$ keeps a learnable correction term, so the model does not rely only on fixed prior information. For this reason, the module is described here as kinematic-aware rather than dynamic-aware. The added bias reflects manipulator structure, but it does not explicitly model dynamic equations, torques, or inertial effects.

Besides the attention bias, the model also introduces node-level hierarchical weights based on joint motion ranges:

$$w_i = 0.6 + 0.6 \cdot \frac{r_i - r_{\min}}{r_{\max} - r_{\min}}, w_i \in [0.6, 1.2] \quad (28)$$

These weights are not applied to the softmax attention probabilities and are only used to rescale the node outputs after attention aggregation. If the attention output of node i is denoted as o_i , the reweighted output is written as:

$$o'_i = w_i o_i \quad (29)$$

This operation gives relatively more emphasis to joints with larger motion ranges. It is a simple design, but it helps reflect structural differences across joints at the feature level. One final point should be noted. In the current implementation, the structural prior parameters are built from fixed constants extracted from the KUKA iiwa URDF.

However, the proposed method itself is not tied to one specific robot platform, and its design is inherently generalizable to different serial manipulators. For any serial manipulator, the same construction procedure can be followed as long as the adjacent joint distances l_k and joint motion ranges r_i are available from the URDF file, DH parameters, or manufacturer specifications. Based on this information, the kinematics-aware prior matrix can be constructed in the same manner and then incorporated into the attention computation of the Transformer encoder. If the target robot does not exhibit an obvious symmetric pattern, the symmetry term can simply be set to $S = 0$. In that case, the structural bias can still be effectively formed using the topology-based distance term and the joint-range-related term. This design gives the proposed method both structural expressiveness and practical flexibility when adapting to different robotic systems. Thus far, the configurations of the graph topology, GNN encoder, Transformer specifications, and activation function have been presented, with detailed settings shown in Table 1.

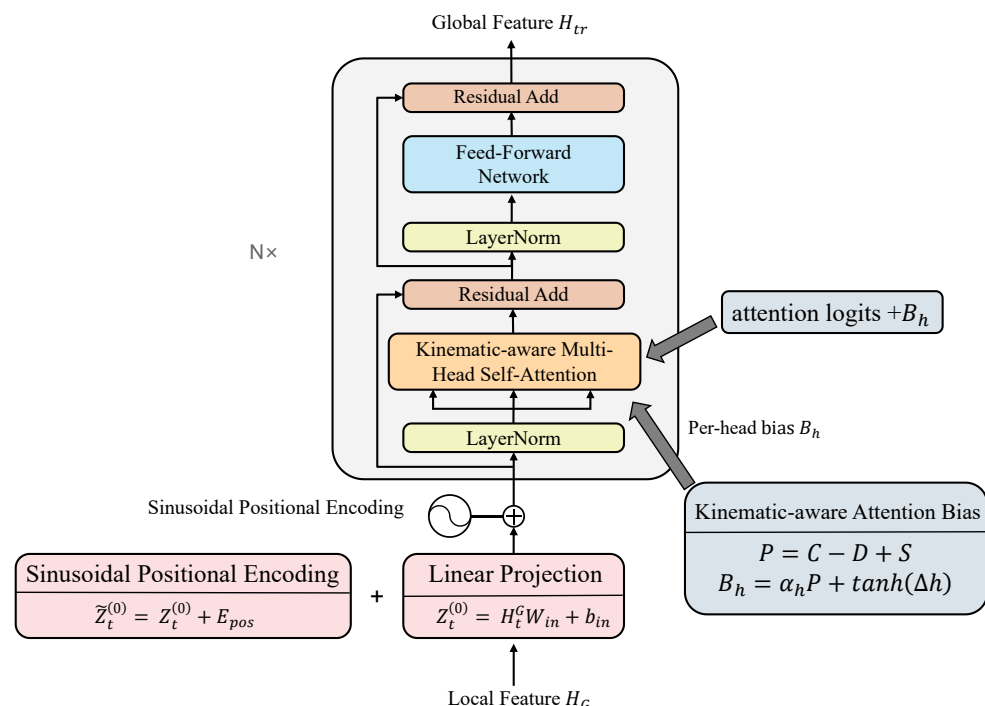


Figure 5. The internal structure of the kinematic-aware Transformer encoder, featuring sinusoidal positional encoding and a kinematic-aware structural bias.

Table 1. Detailed configuration of the GT-TD3 network architecture, including the graph topology, GNN encoder, transformer specifications, and activation.

Module	Parameter	Value	Configuration Details
Graph Topology	Nodes/Edges	$N = 7$	1-hop chain graph with self-loops $\left[q_{i,t}, \dot{q}_{i,t}, q_{i,t}^{\text{cum}}, \frac{\rho_t}{1.2}, \hat{\delta}_{x,t}, \hat{\delta}_{y,t} \right]$
	Configured dim	$d_{\text{node}} = 6$	
GNN Encoder	Layers	2	Gated message passing with LayerNorm Row-normalized adjacency Global pooling
	Hidden dimension	64	
	Readout	Mean + Max	
Transformer	Encoder layers	2	KAPE-based encoder Multi-head self-attention Latent feature space Expansion ratio = 4
	Attention heads	4	
	Embedding dim	$d_{\text{model}} = 128$	
	FNN dimension	$d_{\text{ff}} = 256$	
Activation	Hidden/Output	LeakyReLU/ELU/Tanh	Implementation-consistent

3.6. Cross-Scale Feature Fusion and Action Output Head

After the graph module, the network captures local kinematic features at the joint level. After the Transformer module, it obtains global features that describe long-range dependencies across joints. These two kinds of information are not equivalent. They should not simply overwrite each other. For this reason, the model does not directly replace the GNN representation with the Transformer output. Instead, it performs a gated cross-scale fusion in a shared hidden space, so that local structure and global context can be preserved at the same time.

More specifically, let $g_i \in \mathbb{R}^d$ denote the local feature obtained by linearly projecting the GNN output, and let $t_i \in \mathbb{R}^d$ denote the global feature produced by the Transformer encoder. Each joint therefore has two feature views. One is local. The other is global. For the i -th joint node, the fusion gate is defined as:

$$\beta_i = \sigma(W_c[g_i \parallel t_i] + b_c) \quad (30)$$

and the fused feature is written as:

$$f_i = \text{LN}(\beta_i \odot g_i + (1 - \beta_i) \odot t_i) \quad (31)$$

Here, β_i is an element-wise gating vector. It controls how much information should be taken from the local branch and how much should be taken from the global branch.

This design allows the fusion process to adapt across joints and states. When the motion of one joint depends more strongly on nearby kinematic constraints, the model can keep more of g_i . When long-range coupling becomes more important, it can place more weight on t_i . The fused node set is then written as:

$$F = \{f_i\}_{i=1}^7. \quad (32)$$

At this stage, the network has already integrated both local and global information. A compact readout is still needed. Rather than flattening all node features directly, the model uses two parallel pooling paths, namely the mean pooling and max pooling:

$$f_{\text{global}} = [\text{MeanPool}(F) \parallel \text{MaxPool}(F)] \quad (33)$$

This readout is more compact than direct flattening. It also provides a more stable global summary of the joint set.

Mean pooling captures the average trend of all fused node features. Max pooling, in contrast, highlights the most salient responses among the joints. By combining the two, the model can preserve both overall information and strong local activations. This makes the final representation more suitable for action generation. The deterministic action is finally produced by a lightweight MLP:

$$a_t = a_{\max} \tanh(\text{MLP}(f_{\text{global}})) \quad (34)$$

In the default implementation, this action head contains one hidden layer with 256 units and uses the ELU activation function. The final output is a 7-dimensional joint velocity command. The outer $\tanh(\cdot)$ and the scaling factor $a_{\max}$ explicitly constrain the action range.

As a result, the actor output can be directly used as the target joint-velocity control signal in the simulation environment. Overall, this output head is intentionally simple. The main modeling burden is placed on the structured encoder and the cross-scale fusion module, while the final action projection is kept lightweight and stable.

3.7. Reward Function

This study uses a dense reward function to encourage target approaching, trajectory tracking, successful arrival, and smooth control. The total reward is defined as:

$$r_t = 0.3 r_{\text{goal}} + 0.4 r_{\text{track}} + 0.3 r_{\text{success}} \quad (35)$$

The goal-reaching term is:

$$r_{\text{goal}} = -\tanh(d_g) + \exp(-10d_g) \quad (36)$$

$$d_g = \| p_{ee} - p_{\text{target}} \|_2 \quad (37)$$

The tracking term is:

$$r_{\text{track}} = \exp(-5 \| P_{ee} - P_{\text{ref}}(t) \|_2) \quad (38)$$

where $P_{\text{ref}}(t)$ is the reference point at the current step. The success reward is:

$$r_{\text{success}} = I(d_g < \epsilon) \quad (39)$$

where ϵ is the target threshold.

This formulation emphasizes tracking accuracy while still considering final target convergence, task completion, and action continuity. The reward function in this study is mainly designed empirically, although it follows well-established reward-shaping principles in robot reinforcement learning [32,33]. This still remains a limitation of the present work.

4. Experiments and Results

4.1. Experimental Setup

For our primary experimental platform, we selected the 7-degree-of-freedom KUKA LBR iiwa 14 R820 manipulator. The simulation environment was constructed using the PyBullet physics engine, which provides a highly realistic framework for simulating rigid-body dynamics and complex collision behaviors. Within this environment, we executed the point-to-point trajectory tracking task (illustrated in Figure 6), leveraging an NVIDIA GeForce RTX 5070 GPU to accelerate the deep reinforcement learning training process.

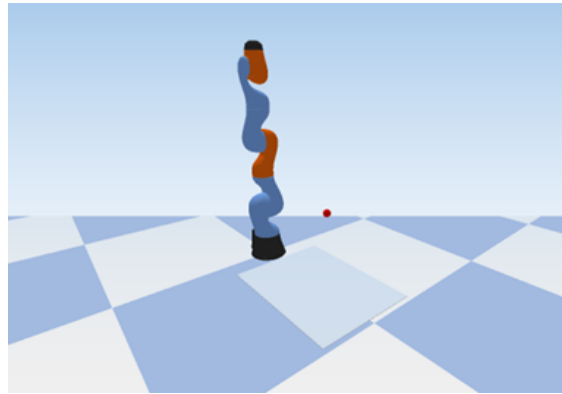


Figure 6. The PyBullet simulation environment for the point-to-point trajectory tracking task using a 7-DoF KUKA LBR iiwa manipulator.

The system's state space, denoted as $s_t \in \mathbb{R}^{20}$, is composed of four distinct components: joint positions, joint velocities, the instantaneous Cartesian coordinates of the end-effector, and the Cartesian position of the target point. During the state encoding phase, the model dynamically derives supplementary features, such as goal direction and Euclidean distance, by calculating the spatial difference between the current end-effector location and the target. Conversely, the action space, $a_t \in \mathbb{R}^7$ directly outputs continuous velocity commands for each of the seven joints. We define one episode as successful when the end effector reaches the target within 200 simulation steps and the position error satisfies $\|p_{ee} - p_{target}\|_2 < 5 \text{ cm}$.

To ensure statistical reliability, each model was evaluated across five random seeds (0, 1, 42, 123, and 2025), with the main text reporting the aggregated results of five primary seeds. Consequently, the reported learning curves and quantitative metrics accurately reflect policy performance across diverse initializations. Furthermore, we enforced strictly fair evaluation protocols across all baseline models by keeping the parameter scales and training conditions identical. The sole distinguishing factor among the tested policies is their underlying backbone network architecture, with all models sharing the standardized hyperparameters detailed in Table 2.

Table 2. Hyperparameters and optimization settings used for training the deep reinforcement learning policies.

Parameter	Symbol	Value	Description
Total Timesteps	T_{total}	500,000	Maximum training timesteps
Warm-up Steps	T_{start}	25,000	Random exploration phase
Min-batch Size	B	512	Increased for stability
Replay Buffer Size	R	500,000	Extended memory capacity
Learning Rate	α	1×10^{-5}	Unified for Actor/Critic
Discount Factor	γ	0.99	Future reward weight
Soft Update Rate	τ	0.003	Target network update
Policy Delay	d	2	Delayed Actor updates
Exploration Noise	ϵ	0.1	Action noise (Gaussian)
Policy Smooth Noise	σ	0.2	Target policy noise
Noise Clip	c	0.5	Noise clipping range

4.2. Learning Dynamics and Convergence Analysis

To rigorously evaluate both learning efficiency and the quality of the learned policies, we monitored eight distinct metrics throughout the training process. The evaluation was divided into two primary analytical dimensions: Figure 7 illustrates the core task-level

performance, while Figure 8 details the metrics associated with trajectory fidelity and operational safety.

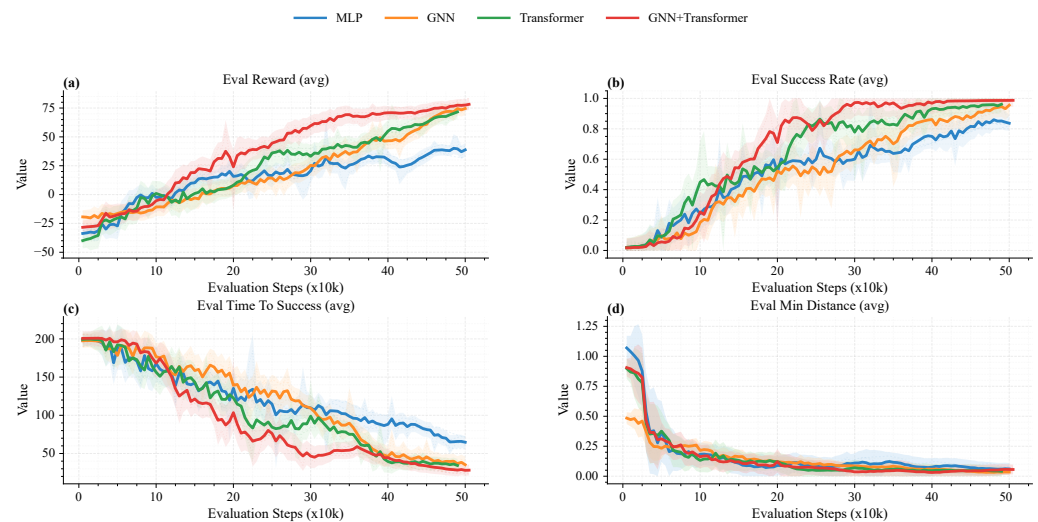


Figure 7. Learning curves for task-level performance metrics during training, including (a) evaluation reward, (b) evaluation success rate, (c) evaluation time to success, and (d) evaluation minimum distance.

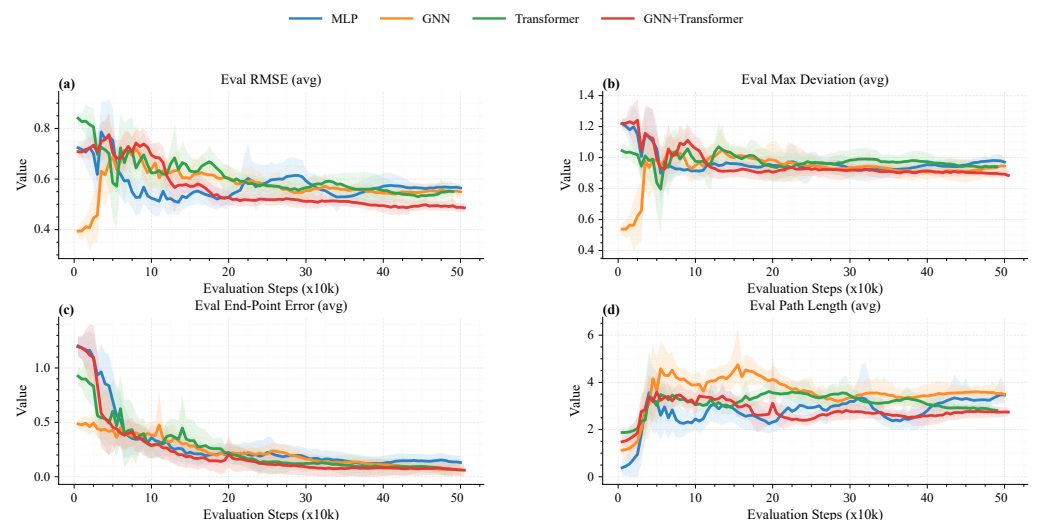


Figure 8. Evaluation curves for trajectory fidelity and kinematic quality during training, including (a) root mean square error (RMSE), (b) maximum deviation, (c) end-point error, and (d) path length.

4.2.1. Task Performance and Sample Efficiency

As illustrated by the task-level performance metrics in Figure 7, GT-TD3 significantly outperformed all baseline methods in both sample efficiency and final asymptotic performance. This superiority is clearly reflected in the learning curves for cumulative reward (Figure 7a) and success rate (Figure 7b). Specifically, our proposed framework not only converged faster but also yielded a substantially higher cumulative reward than the other three configurations. Furthermore, it rapidly achieved and sustained a dominant success rate after 350 k training steps. In stark contrast, the standard MLP baseline suffered from severe fluctuations throughout the entire training process. While the pure GNN and pure Transformer architectures managed to secure higher average success rates than the MLP, they still exhibited pronounced oscillatory behavior during learning.

Secondary evaluation metrics further corroborate these findings. The time-to-success curve (Figure 7c) demonstrates that GT-TD3 eventually converges to the lowest required

temporal steps, indicating that the fused model executes the tracking task with greater speed and decisive action. Finally, the evaluation minimum distance metric (Figure 7d) confirms this steady convergence; GT-TD3 produced the smallest final error variance, proving that the hybrid structure guides the end-effector to the target zone with superior stability and precision.

4.2.2. Trajectory Fidelity and Kinematic Optimization

Beyond mere task completion, we rigorously evaluated the kinematic quality of the generated trajectories, with the quantitative findings detailed in Figure 8. In terms of tracking accuracy, both the Root Mean Square Error (RMSE) (Figure 8a) and end-point error (Figure 8c) demonstrated that GT-TD3 yields significantly smoother and narrower error curves. Ultimately, the fused architecture successfully converges to the lowest possible error threshold, whereas the pure GNN and Transformer baselines stagnate at notably higher error margins. This stark contrast underscores the absolute necessity of feature fusion for achieving precise position control.

From a safety perspective, the maximum deviation metric (Figure 8b) reveals that our proposed method adheres to much tighter spatial constraints along the entire reference path. Furthermore, an analysis of path length (Figure 8d) highlights the superior operational efficiency of GT-TD3. While alternative models successfully reach the target region, their generated paths are frequently circuitous and winding. Conversely, GT-TD3 consistently converges to the shortest valid trajectory.

Collectively, these evaluations substantiate that the asymmetric fusion structure extends well beyond simply reaching a spatial goal; it inherently facilitates the generation of motion trajectories that are fundamentally more accurate, safer, and highly efficient.

4.3. Test Results Analysis

The comprehensive test results across all evaluated models, as displayed in Figures 9 and 10, demonstrate that GT-TD3 largely mirrors the favorable behavioral trends initially observed during the training phase. With the minor exception of the minimum distance and end-point error metrics, where our framework was marginally outperformed by one specific baseline, GT-TD3 maintains a definitive evaluative advantage.

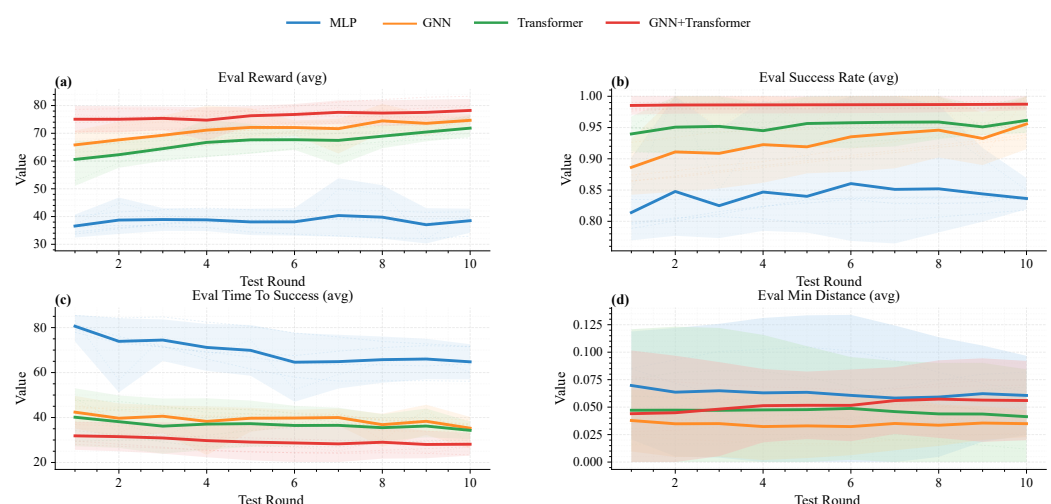


Figure 9. Comprehensive test results for task performance metrics across multiple test rounds. (a) Average evaluation reward; (b) Average evaluation success rate; (c) Average evaluation time to success; (d) Average evaluation minimum distance. The shaded areas represent the standard deviation of the results across multiple trials.

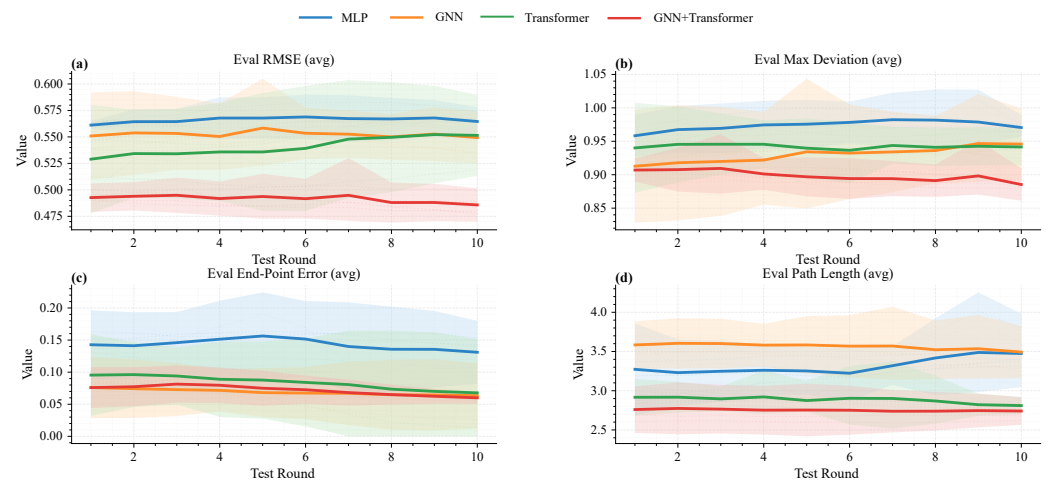


Figure 10. Comprehensive test results for trajectory fidelity metrics across multiple test rounds. (a) Average evaluation root mean square error (RMSE); (b) Average evaluation maximum deviation; (c) Average evaluation end-point error; (d) Average evaluation path length. The shaded areas represent the standard deviation of the results across multiple trials.

Broadly speaking, the proposed architecture not only delivered superior performance across the majority of the testing criteria but also exhibited significantly reduced performance fluctuations. These outcomes collectively validate that GT-TD3 substantially elevates the baseline trajectory tracking capabilities of the robotic manipulator while simultaneously ensuring remarkable model stability and robustness across multiple independent testing iterations.

4.4. Stability Analysis

This study uses Figures 11 and 12 to analyze the stability of different models under disturbance conditions. This study denotes the end-effector tracking error as e_k . This study uses $V(e_k) = \|e_k\|^2$ as the candidate function in the stability analysis. This study further computes $\Delta V_k = V(e_{k+1}) - V(e_k)$. This study uses the negative dV ratio to denote the proportion of time steps with $\Delta V_k < 0$. This metric reflects whether the system error energy keeps decreasing over time. This study also uses the final tracking error, recovery step, and stability success rate as supporting metrics. Figure 11 shows how these metrics change under different levels of initial disturbance. Figure 12 shows the distributions of these metrics under the maximum disturbance, $\sigma = 0.25$ rad. These two sets of results describe model stability from two angles: one shows the overall trend and the other shows the behavior under strong disturbance.

As illustrated in Figure 11, the stability of all evaluated models predictably deteriorates as the magnitude of the initial disturbance increases. For the majority of policies, this heightened control difficulty manifests directly as inflated final tracking errors and correspondingly prolonged recovery steps. Despite this general trend of degradation across the baseline architectures, GT-TD3 distinguishes itself by sustaining a relatively high negative dV ratio across almost the entire perturbation spectrum. By consistently securing lower final errors while demanding fewer recovery steps, our proposed framework demonstrates superior robustness over the full range of disturbance conditions.

To dissect policy performance under the most extreme conditions, Figure 12 isolates the detailed outcomes at the maximum disturbance level. Within this visualization, the embedded boxplots mark the medians and interquartile ranges, while the varying widths of the violin plots explicitly capture the concentration of the outcome distributions. Individual trial results, overlaid as scatter points, further reveal that GT-TD3 yields not only superior median performance but also a highly concentrated data distribution. This tight clustering

signifies that the fused architecture behaves with exceptional steadiness under intense perturbations, thereby minimizing the probability of generating overtly unstable control sequences. Conversely, although baseline models occasionally manage to recover in isolated trials, their broadly scattered outcome distributions betray a profound sensitivity to severe external disturbances.

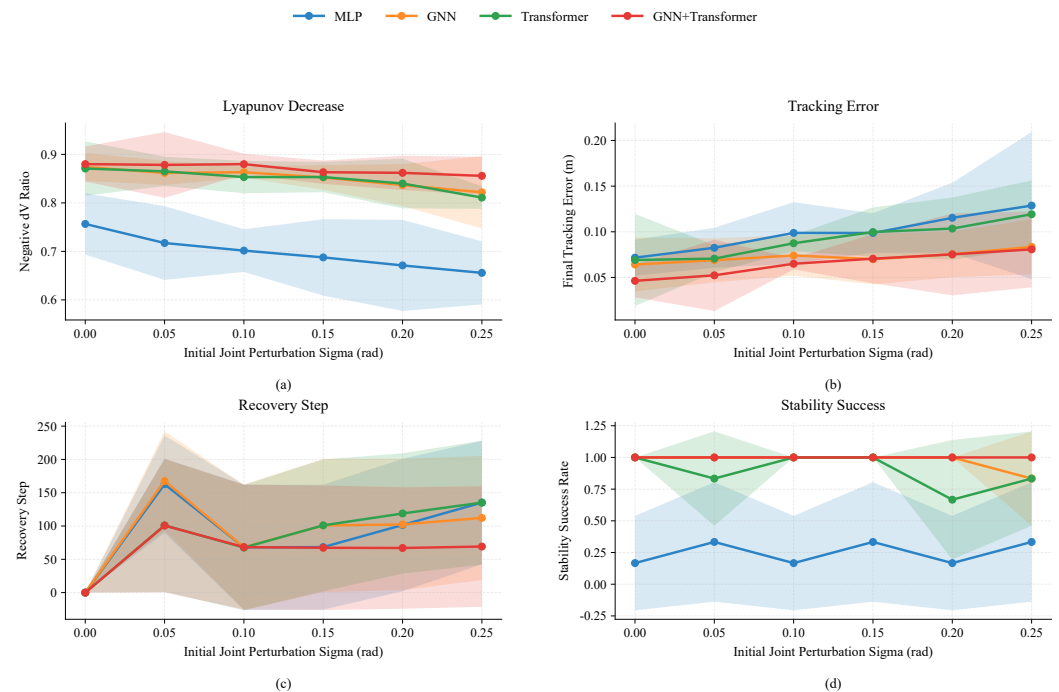


Figure 11. Stability analysis line plots showing the impact of varying initial joint perturbation levels on (a) Lyapunov decrease (negative dV ratio), (b) final tracking error, (c) recovery step, and (d) stability success rate.



Figure 12. Violin and box plots illustrating the detailed distribution of stability metrics under the hardest perturbation setting ($\sigma = 0.25$ rad).

Ultimately, the continuous trajectory evaluations in the line plots and the discrete extreme-case distributions in the violin plots mutually reinforce the same core conclusion.

Synthesizing these analyses, it is evident that GT-TD3 not only elevates average task execution but also provides empirical evidence of improved stability.

4.5. Trajectory Tracking Evaluation

To provide a more granular assessment of trajectory quality, Figure 13 illustrates four representative end-effector tracking tasks. By visually contrasting the actual executed trajectories (solid lines) against their corresponding references (dashed lines), we can intuitively gauge tracking accuracy, motion smoothness, and convergence behavior. For a comprehensive comparison, the figure displays three randomly initialized runs for each evaluated policy.

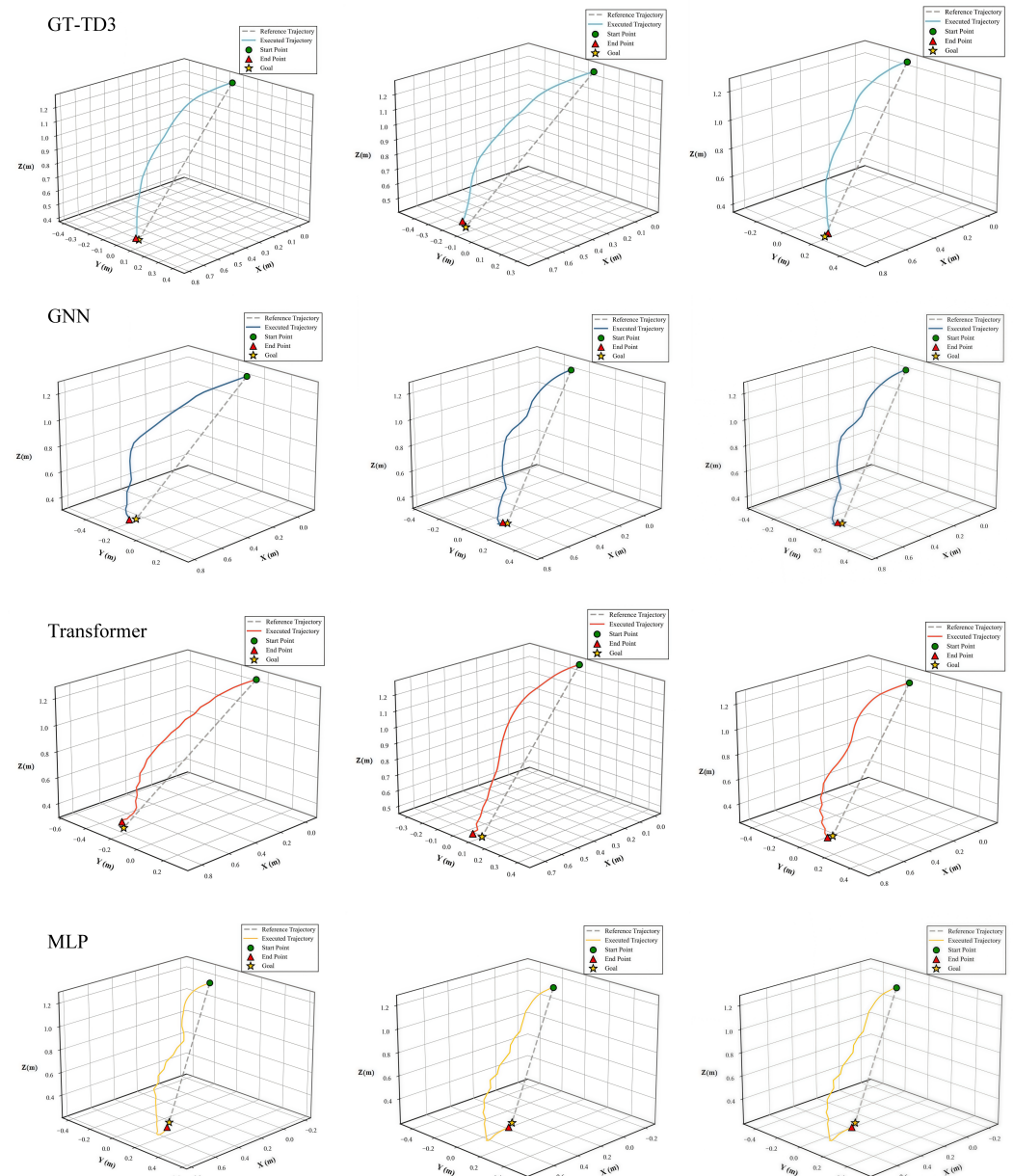


Figure 13. 3D visualizations of four representative end-effector trajectory tracking tasks.

The light blue trajectories, denoting the GT-TD3 control outcomes, showcase the most superior tracking proficiency among all candidates. The end-effector adheres tightly to the intended path with negligible lateral deviation and entirely fluid speed transitions, free from abrupt command jumps. More crucially, this policy achieves stable terminal convergence, effectively sidestepping the severe overshoot phenomena that frequently

plague high-degree-of-freedom robotic control. This exceptional stability validates the premise of our hybrid architecture. By concurrently modeling localized structural couplings and overarching global dependencies, GT-TD3 orchestrates a far more coordinated control behavior across the entire operational workspace.

Conversely, the dark blue trajectories generated by the pure graph neural network (GNN) baseline expose the inherent limitations of relying solely on topological methodologies. While adept at preserving localized joint connections, the GNN policy suffers from pronounced S-shaped deviations during the early and intermediate stages of motion. This observation aligns with established theoretical findings: standard message-passing mechanisms possess restricted receptive fields, severely handicapping the model's capacity to resolve long-range geometric relationships in the absence of a global context [19]. Although continuous feedback eventually allows the policy to correct these detours, the resulting path is marred by redundant turning maneuvers, yielding a cumulative control error substantially higher than that of our fused model.

The pure Transformer baseline (red trajectories) maintains overarching motion continuity, yet is plagued by a conspicuous control lag. This latency forces the end-effector into sluggish reactions during high-curvature maneuvers, precipitating visible spatial tracking errors. Such behavior substantiates recent critiques within the field. Because standard Transformers lack explicit kinematic structural priors despite their prowess in sequence modeling, their localized feature perception becomes alarmingly inefficient during continuous control, ultimately impeding response times [20,28]. Furthermore, the presence of mild oscillations near the final target highlights a secondary weakness: relying exclusively on self-attention mechanisms proves inadequate for the high-frequency fine-tuning actions required during terminal stabilization.

Finally, the yellow trajectories tracking the multilayer perceptron (MLP) baseline exhibit the most degraded performance. The motion is characterized by extreme jitter, severe mid-route deviations, and intense oscillatory behavior as the episode progresses. The model is trapped in a loop of constant directional overcorrections, a chaotic pattern that intensifies as the end-effector approaches its target. This vividly illustrates the classic dilemma faced by unstructured reinforcement learning policies deployed on high-degree-of-freedom hardware: they simply collapse under the weight of high-dimensional state spaces [34]. Bereft of both the structural priors inherent to GNNs and the overarching dependency frameworks provided by Transformers, the MLP cannot adequately filter state noise. Consequently, its heavy reliance on direct state mapping dooms it to an inescapable cycle of violent oscillation and reactive correction.

5. Discussion

Our experiments clearly show that relying on a single neural network architecture just is not enough to capture both the local and global dependencies needed for complex manipulator control. For example, because the MLP baseline uses a simple direct global mapping, it tends to suffer from continuous fluctuations and constant mid-course corrections. The GNN, on the other hand, is great at looking at neighboring joints, but it hits a wall when trying to understand long-range connections. Meanwhile, the pure Transformer is excellent at seeing the big picture globally, but it does not really understand the physical structure of the robotic arm out of the box.

This is exactly why GT-TD3 works so well, it brings together the best of both worlds. By blending local structural awareness with global dependency tracking, it creates a much smoother and more coordinated control process. What this really highlights is how important physical, structural priors are in continuous control tasks. A robotic arm is not just a sequence of numbers; it has a rigid physical chain and strict joint constraints. If we

do not give the network a “heads-up” about these physical rules, it must learn them all from scratch, which makes training unnecessarily hard. Our structure-aware attention bias essentially bakes these rules directly into the model, making trajectory tracking much more stable.

Of course, this approach is not without its trade-offs. The structured design does add some training overhead compared to the simpler baselines. For instance, under identical hardware settings, training the MLP baseline took approximately 1.25 h, whereas the GT-TD3 model required 1.98 h. Though it introduced additional spatio-temporal complexity, we found the extra time to be completely manageable. There are also a few limitations to keep in mind. So far, we have only tested this on a 7-DoF KUKA arm in PyBullet, focusing mainly on simple point-to-point straight-line tracking. We have also not fed the model more complex physical dynamics like torque or contact forces yet, nor have we integrated real-time visual perception systems. Therefore, it definitely needs more testing in messier, real-world scenarios.

Still, we think that this method is very close to being ready for real hardware. In a physical setup, all the system really needs is real-time data on joint positions, velocities, and the end-effector pose, alongside the target path. Since the core of our method just relies on standard robot specs (like joint connections and limits), adapting it to a different manipulator should be pretty straightforward. In a physical setup, the deployment path follows a clear pipeline: state acquisition, target input, policy inference, velocity control output, safety-constrained execution. Looking ahead, we plan to try this out on different robot platforms and more complicated paths. We will also be looking into adding parameter randomization and noise injection to help bridge the sim-to-real gap and get this working reliably in the real world.

6. Conclusions

In this paper, we introduced GT-TD3, a reinforcement learning framework built specifically to handle the tricky task of trajectory tracking for redundant robotic arms. We essentially combined a graph neural network (GNN) and a Transformer inside the actor network of an asymmetric actor–critic setup. The GNN handles the local joint connections, while the Transformer looks at the global picture. By adding structural biases—like kinematic chain distances, joint limits, and symmetry features—we successfully fed the robot’s physical realities directly into the attention mechanism.

When we tested it on a 7-DoF KUKA arm, the results were promising. GT-TD3 outperformed the standard MLP, pure GNN, and pure Transformer models. The data showed that our method is not just more stable; it learns faster and produces much smoother, more precise end-effector movements. Ultimately, it demonstrates that combining local structure with global awareness is a highly robust way to solve high-precision robotic tracking problems. Moving forward, we plan to test this framework on robots with entirely different body structures and push it to handle even more complex real-world tasks.

Author Contributions: Conceptualization, H.M.; methodology, H.M. and H.H.; investigation (experiments), H.M.; writing—original draft preparation, H.M. and H.H.; investigation (literature research), Z.C.; supervision, Z.Z.; project administration, Z.Z.; writing—review and editing, R.Z. All authors have read and agreed to the published version of the manuscript.

Funding: The authors gratefully acknowledge the support of the Research Project of Top Drive System: Key Components of Top Drive System for Deep-Earth Oil and Gas Drilling and Exploration (No. TC240HAJ8-173).

Data Availability Statement: Data will be made available on request.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CNN	Convolutional Neural Network
DDPG	Deep Deterministic Policy Gradient
DH	Denavit–Hartenberg
DQN	Deep Q-Network
GNN	Graph Neural Network
GT-TD3	Graph Transformer-Twin Delayed Deep Deterministic Policy Gradient
MLP	Multilayer Perceptron
KAPE	Kinematic-Aware Positional Encoding
PID	Proportional-Integral-Derivative
RMSE	Root Mean Square Error
SAC	Soft Actor–Critic
TD3	Twin Delayed Deep Deterministic Policy Gradient
URDF	Unified Robot Description Format

References

1. Billard, A.; Kragic, D. Trends and challenges in robot manipulation. *Science* **2019**, *364*, eaat8414. [[CrossRef](#)]
2. Verl, A.; Valente, A.; Melkote, S.; Brecher, C.; Ozturk, E.; Tunc, L.T. Robots in machining. *CIRP Ann.* **2019**, *68*, 799–822. [[CrossRef](#)]
3. Taylor, R.H.; Menciassi, A.; Fichtinger, G.; Fiorini, P.; Dario, P. Medical robotics and computer-integrated surgery. *IEEE Trans. Biomed. Eng.* **2016**, *63*, 2079–2094.
4. Su, Y.; Zheng, C.; Mercorelli, P. Robust approximate fixed-time tracking control for uncertain robot manipulators. *Mech. Syst. Signal Process.* **2020**, *135*, 106379. [[CrossRef](#)]
5. Khan, H.; Lee, M.C.; Suh, J.; Kim, R. Enhancing robot end-effector trajectory tracking using virtual force-tracking impedance control. *Adv. Intell. Syst.* **2025**, *7*, 2400380. [[CrossRef](#)]
6. Qiu, X.; Cai, Z.; Peng, H. Path planning of a continuum robot's end-effector for assembly missions in unstructured environments. In Proceedings of the 2022 IEEE 5th Advanced Information Management, Communication, Electronic and Automation Control Conference (IMCEC), Chongqing, China, 16–18 December 2022; pp. 539–543.
7. Hart, P.E.; Nilsson, N.J.; Raphael, B. Correction to “A formal basis for the heuristic determination of minimum cost paths”. *ACM SIGART Bull.* **1972**, *37*, 28–29. [[CrossRef](#)]
8. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing Atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602. [[CrossRef](#)]
9. Van Hasselt, H.; Guez, A.; Silver, D. Deep reinforcement learning with double Q-learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; Volume 30.
10. Fujimoto, S.; van Hoof, H.; Meger, D. Addressing function approximation error in actor-critic methods. In Proceedings of the 35th International Conference on Machine Learning (ICML), Stockholm, Sweden, 10–15 July 2018; pp. 1587–1596.
11. Tang, C.; Abbatematteo, B.; Hu, J.; Chandra, R.; Martín-Martín, R.; Stone, P. Deep reinforcement learning for robotics: A survey of real-world successes. *arXiv* **2024**, arXiv:2408.03539. [[CrossRef](#)]
12. Susanto, E.; Sumaryo, S.; Rahmat, B. Neural network control for dynamics of a 3DOF robot arm. In Proceedings of the 2024 IEEE 10th International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA), Bandung, Indonesia, 30–31 July 2024; pp. 196–200.
13. Yan, Z.; Chang, Y.; Yuan, L.; Wei, F.; Wang, X.; Dong, X.; Han, H. Deep learning-driven robot arm control fusing convolutional visual perception and predictive modeling for motion planning. *J. Organ. End User Comput.* **2024**, *36*, 1–29. [[CrossRef](#)]
14. Keppler, M.; Lakatos, D.; Ott, C.; Albu-Schäffer, A. Minimally model-based trajectory tracking and variable impedance control for flexible-joint robots. In Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 16–21 May 2016; pp. 3314–3320.
15. Khan, A.; Tolstaya, E.; Ribeiro, A.; Kumar, V. Graph policy gradients for large scale robot control. In Proceedings of the Conference on Robot Learning (CoRL), Osaka, Japan, 30 October–1 November 2019; pp. 823–834.
16. Scarselli, F.; Gori, M.; Tsoi, A.C.; Hagenbuchner, M.; Monfardini, G. The graph neural network model. *IEEE Trans. Neural Netw.* **2009**, *20*, 61–80. [[CrossRef](#)]
17. Hart, P.; Knoll, A. Graph neural networks and reinforcement learning for behavior generation in semantic environments. *arXiv* **2020**, arXiv:2006.12576. [[CrossRef](#)]
18. Kazemi, E.; Soltani, I. MarineFormer: A Transformer-based navigation policy model for collision avoidance in marine environment. *arXiv* **2024**, arXiv:2410.13973.

19. Alon, U.; Yahav, E. On the bottleneck of graph neural networks and its practical implications. In Proceedings of the International Conference on Learning Representations (ICLR), Vienna, Austria, 3–7 May 2021.
20. Li, W.; Luo, H.; Lin, Z.; Zhang, C.; Lu, Z.; Ye, D. A survey on transformers in reinforcement learning. *arXiv* **2023**, arXiv:2301.03044. [[CrossRef](#)]
21. Ang, K.H.; Chong, G.; Li, Y. PID control system analysis, design, and technology. *IEEE Trans. Control Syst. Technol.* **2005**, *13*, 559–576.
22. Ma, L.; Xue, J.; Kawabata, K.; Zhu, J.; Ma, C.; Zheng, N. A fast RRT algorithm for motion planning of autonomous road vehicles. In Proceedings of the 17th International IEEE Conference on Intelligent Transportation Systems (ITSC), Qingdao, China, 8–11 October 2014; pp. 1033–1038.
23. Huang, S.; Chen, Q.; Zhang, X.; Sun, J.; Schwager, M. ParticleFormer: A 3D point cloud world model for multi-object, multi-material robotic manipulation. *arXiv* **2025**, arXiv:2506.23126.
24. Wang, T.; Liao, R.; Ba, J.; Fidler, S. NerveNet: Learning structured policy with graph neural networks. In Proceedings of the International Conference on Learning Representations (ICLR), Vancouver, BC, Canada, 30 April–3 May 2018.
25. Khan, A.; Ribeiro, A.; Kumar, V.; Francis, A.G. Graph neural networks for motion planning. *arXiv* **2020**, arXiv:2006.06248. [[CrossRef](#)]
26. Zhang, Q.; Zhang, X.; Ye, Z.; Mi, J. MSTT: A multi-spatio-temporal graph attention model for pedestrian trajectory prediction. *Sensors* **2025**, *25*, 4850. [[CrossRef](#)] [[PubMed](#)]
27. Parisotto, E.; Song, F.; Rae, J.; Pascanu, R.; Gulcehre, C.; Jayakumar, S.; Jaderberg, M.; Kaufman, R.L.; Clark, A.; Noury, S.; et al. Stabilizing transformers for reinforcement learning. In Proceedings of the 37th International Conference on Machine Learning (ICML), Virtual, 12–18 July 2020; pp. 7487–7498.
28. Tejomurtula, S.; Kak, S. Inverse kinematics in robotics using neural networks. *Inf. Sci.* **1999**, *116*, 147–164. [[CrossRef](#)]
29. Gao, R. Inverse kinematics solution of robotics based on neural network algorithms. *J. Ambient Intell. Humaniz. Comput.* **2020**, *11*, 6199–6209. [[CrossRef](#)]
30. Cagigas-Muñoz, D. Artificial neural networks for inverse kinematics problem in articulated robots. *Eng. Appl. Artif. Intell.* **2023**, *126*, 107175. [[CrossRef](#)]
31. Sheng, Z.; Huang, Z.; Chen, S. Kinematics-aware multigraph attention network with residual learning for heterogeneous trajectory prediction. *J. Intell. Connect. Veh.* **2024**, *7*, 138–150. [[CrossRef](#)]
32. Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, P.; Zaremba, W. Hindsight experience replay. In *Advances in Neural Information Processing Systems (NeurIPS)*; NeurIPS Foundation: San Diego, CA, USA, 2017.
33. Peng, X.B.; Abbeel, P.; Levine, S.; van de Panne, M. DeepMimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Trans. Graph.* **2018**, *37*, 143. [[CrossRef](#)]
34. Rajeswaran, A.; Kumar, V.; Gupta, A.; Vezzani, G.; Schulman, J.; Todorov, E.; Levine, S. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In Proceedings of the Robotics: Science and Systems (RSS), Pittsburgh, PA, USA, 26–30 June 2018.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.