2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Efficient Hierarchical Reinforcement Learning with Dynamic Kolmogorov–Arnold Network for Long-Horizon Robotic Manipulation

Yuke Qu[1], Junkai Ren[1]*, Jiawei Luo[1], Yufeng Xie[1], Huimin Lu[1], Xin Xu[1], Yicong Ye[2]

***Abstract*— Long-horizon robotic manipulation remains a critical challenge in robotics. Hierarchical reinforcement learning offers a promising solution, but often suffers from an imbalance dilemma: simplifying skill learning increases the complexity of planning, thereby expanding the solution space and computational burden of planning. To tackle this challenge, we propose a Hierarchical Reinforcement Learning framework with Dynamic Kolmogorov-Arnold Network (DyKAN) based Actor Critic, named HIKER. Firstly, HIKER innovates with a dual-chain design that decomposes the complex task into two intersecting sub-chains, reducing the optimization conflict across subtasks and alleviating the burden on the planning model. Secondly, we develop DyKAN, a scalable neural network for both actor and critic in the skill model of HIKER. DyKAN adaptively adjusts grids and basis functions while preserving learned knowledge, enabling efficient learning of complex manipulation skills. Furthermore, to optimize DyKAN's performance, we design a per-layer update module that uses Dynamic Tanh (DyT) and low-rank decomposition to ensure stable, low-cost updates during training. Finally, experiments on long-horizon robotic manipulation tasks demonstrate that HIKER significantly improves efficiency and robustness, yielding higher-quality skill models and achieving a 10.9% increase in task success rate under the high noise condition. Further insights are available on the website: https://sites.google.com/view/hikerdykan.**

I. INTRODUCTION

With the rapid development of AI and robotics, manipulation robots are demonstrating significant potential in application fields such as autonomous chemical and materials laboratories [1]. These domains often involve long-horizon tasks, where developing reliable learning methods for robotic manipulation is crucial to enhancing the efficiency, autonomy, and safety of robots.

Imitation Learning (IL) and Reinforcement Learning (RL) are mainstream methods for learning robotic manipulation skills. While both methods have achieved success in tasks such as grasping [2] [3], in-hand manipulation [4], and contact-rich manipulation [5], they each exhibit distinct limitations. IL, which relies on demonstration data, can efficiently learn state-action mappings but suffers from limited generalization, especially in long-horizon tasks, due to its dependence on the coverage of the demonstration data. In contrast, RL learns generalizable policy without demonstration data. However, it struggles with large exploration spaces and sparse rewards in long-horizon tasks, resulting in high training costs and a tendency to converge to local optima.

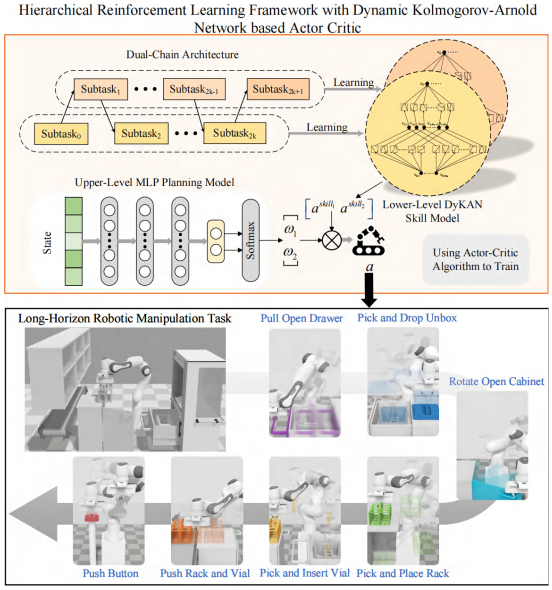

Fig. 1. Dual-chain architecture divides the task chain into two sub-chains, which DyKAN skill models learn. By actions from the skill models and action weights from the planning model, the robot can accomplish the long-horizon manipulation task.

To solve these challenges, hierarchical learning has been applied to long-horizon tasks [6] [7]. Hierarchical learning generally decomposes long-horizon tasks into subtasks or goals via a two-level structure. The lower skill layer focuses on learning manipulation skills for specific subtasks, while the upper planning layer schedules the execution of these skills based on the environment state. This decomposition simplifies the learning process for long-horizon tasks. However, hierarchical learning also brings new challenges, such as an increase in the number of model parameters [8] and instability in upper-layer planning due to state similarities among lower-layer manipulation skills [9]. A more critical issue is the imbalance dilemma: simplifying skill learning at the lower layer can overload the upper-layer planner, especially when a long-horizon task is decomposed into too many subtasks or goals, making the solution space of the upper-layer planner too large.

To address the critical issue mentioned above, we propose HIKER, a hierarchical reinforcement learning framework with a dual-chain architecture and the DyKAN. As shown

*Corresponding Author: Junkai Ren. E-mail: jk.ren@nudt.edu.cn
[1]Yuke Qu, Junkai Ren, Jiawei Luo, Yufeng Xie, Huimin Lu and Xin Xu are with College of Intelligence Science and Technology, National University of Defense Technology, Changsha, 410073, China
[2]Yicong Ye is with College of Aerospace Science and Engineering, National University of Defense Technology, Changsha, 410073, China
Funding: This work is supported by the National Natural Science Foundation of China [grant numbers 62533021 and 62373201] and Innovation Research Foundation of National University of Defense Technology (Innovation Research Foundation of NUDT) [grant number 25-ZZCX-ZXGC-01].

in Fig. 1, the dual-chain architecture divides the long-horizon task chain into two intersecting sub-chains to reduce learning difficulty for the skill models and the planning model. One subtask corresponds to one manipulation behavior. A DyKAN skill model can efficiently master multiple higher-quality manipulations. Through the above components, HIKER can effectively improve the success rates and robustness for the long-horizon robotic manipulation task. The main contributions of our work are as follows:

- HIKER framework: We introduce HIKER, a hierarchical reinforcement learning method for scalable and robust long-horizon robotic manipulation. HIKER employs a novel dual-chain architecture to train the manipulation skill models. This design alleviates conflict optimization objectives across skills and reduces the burden on the task planning model.
- DyKAN architecture: Within HIKER, we propose DyKAN, a scalable neural architecture for both actor and critic networks in reinforcement learning. DyKAN adaptively adjusts grids and basis functions while preserving learned knowledge, enabling efficient learning of skill models.
- Efficient DyKAN updates: To further enhance the efficiency of DyKAN, we design a per-layer update module that achieves stable and low-cost updates by leveraging Dynamic Tanh and low-rank decomposition techniques.
- Empirical validation: We evaluate HIKER on long-horizon robotic manipulation tasks, where it achieves substantially improved efficiency and robustness. HIKER efficiently learns higher-quality skills (reflected by high rewards) and improves long-horizon task success rate by 10.9% under the high-noise condition.

II. RELATED WORK

A. Long-Horizon Robotic Manipulation Task

The long-horizon robotic manipulation task requires the robot to achieve a final goal through a sequence of manipulations. Some studies start with fundamental manipulation skills, such as completing object rearrangement tasks on a desktop or container through combinations of picking, placing and opening [8] [10] [11]. However, these studies cover a limited range of skill categories and do not adequately reflect the task complexity and manipulation diversity encountered in real application scenarios. Other studies enhance task complexity by introducing additional constraints. These constraints include requiring cross-platform manipulations and expanding the range of manipulation skills [9] [12] [13]. Two effective methods for long-horizon tasks are hierarchical learning [8] [11] and Vision-Language-Action (VLA) model [14] [15]. As the emerging method, VLA demonstrates the potential to directly tackle long-horizon tasks by leveraging its powerful representation learning and multimodal fusion capabilities. It is worth noting that hierarchical learning and VLA are not mutually exclusive. For example, π_0 [15] handles long-horizon tasks using the hierarchical framework. Therefore, our research on hierarchical learning can not only solve the problems of the long-horizon robotic manipulation task, but also provide support for the development of VLA.

B. Hierarchical Learning

Hierarchical imitation learning and hierarchical reinforcement learning are two mainstream hierarchical learning methods. Hierarchical imitation learning collects demonstration data for models of both upper and lower layers. To enhance the data fitting ability of models, the transformer [7] [16] and the diffusion model [17] are utilized in hierarchical imitation learning. However, hierarchical imitation learning requires high-quality and large amounts of data, just like IL. In hierarchical reinforcement learning, robots train multiple policies through RL to complete the long-horizon task. Conventional hierarchical reinforcement learning includes the options framework [18], HAM [19], and so on. There are many improvements and optimizations for hierarchical reinforcement learning to cope with various long-horizon robotic manipulation tasks. Examples include sharing a policy network with multiple value networks to reduce network parameter size [8], incorporating curriculum learning [11], and combining IL to form hierarchical hybrid learning [9]. Meanwhile, to alleviate the sparse reward problem in long-horizon tasks, some studies introduce intrinsic rewards, such as the Generative Adversarial Imitation Learning (GAIL) reward [9], the large language model (LLM) [12] reward and the intrinsic curiosity reward [11]. Although hierarchical learning is a popular method for long-horizon tasks, it still faces the dilemma of imbalanced learning difficulty between upper and lower layers, as mentioned earlier. Our HIKER can effectively alleviate this dilemma.

C. Kolmogorov–Arnold Network

KAN is inspired by the Kolmogorov–Arnold representation theorem [20]. Unlike the Multi-Layer Perceptron (MLP), which relies on fixed activation functions, KAN parameterizes activation functions using learnable B-spline basis function parameters to train. KAN can be used to discover new mathematical or physical laws, thus showing great potential for application in physics-informed machine learning. Studies such as PIKAN [21], KINN [22], and KKAN [23] demonstrate its value in this field. KAN also shows application value in multiple other fields. For example, KA-GNNs [24] for molecular property prediction, U-KAN [25] for medical image segmentation, and MP-KAN [26] for magnetic positioning. KAN is also used for various tasks in RL, including but not limited to multi-agent [27], safe autonomous driving [28], and chaos control [29]. However, KAN periodically updates its grids based on the data distribution and simultaneously adjusts its learned B-spline basis function parameters. This process will disrupt learned knowledge to some degree. The stable update is important for RL, and disrupting learned knowledge can reduce training efficiency and stability. By designing residual structure, KAN uses a base structure equivalent to a fully connected layer to alleviate the above problems. Nevertheless, knowledge tends to be dominated by the equivalent fully connected

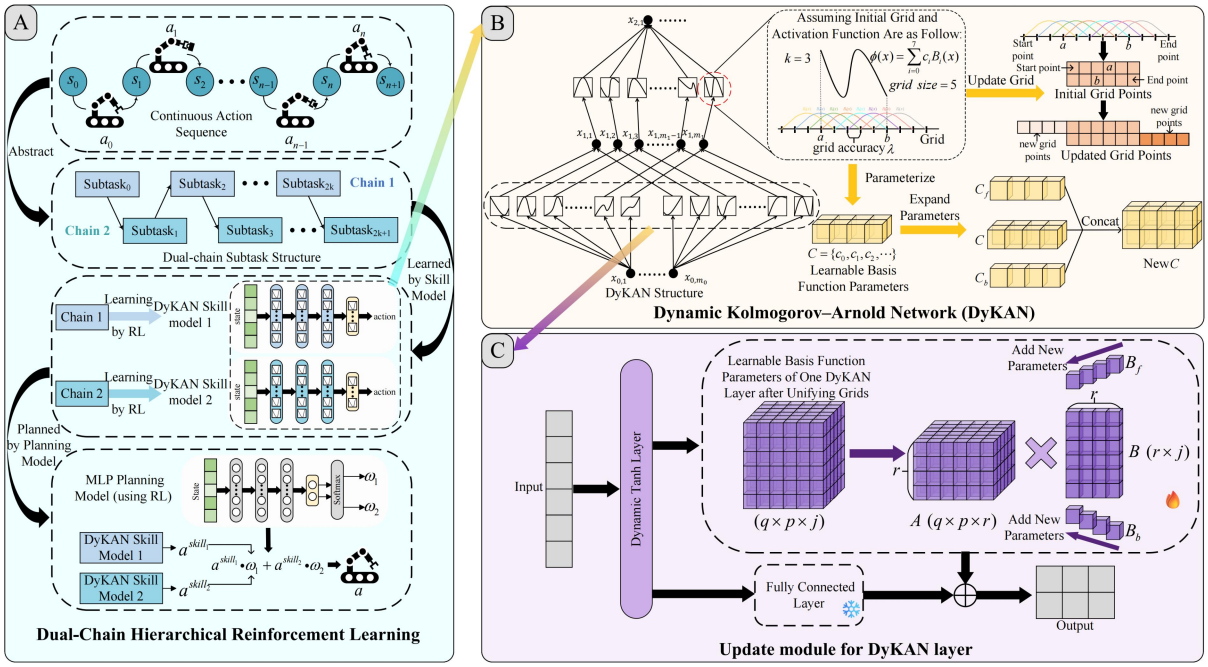

Fig. 2. Schematic diagram of HIKER framework. **A** Overview of dual-chain hierarchical Reinforcement Learning. **B** Schematic process of learnable basis function parameters expansion. **C** DyT ensures stable training, while low-rank decomposition enables low-cost updates to the number of basis functions.

layer, reducing learnable B-spline basis function parameters to merely a fine-tuning extension of them and failing to fully demonstrate the learning capability of B-spline basis function parameters. An alternative approach is to fix the grid of KAN, but this also sacrifices part of its learning potential. Our DyKAN can effectively solve these problems.

III. METHODOLOGY

In long-horizon robotic manipulation tasks, robots need to execute a continuous action sequence $\{a_1, a_2, ..., a_n\}$, which can be abstracted into distinct manipulations, such as pushing, pulling, or picking. These manipulations can be learned either through multiple independent skill models [9] or via a single skill model with task ID embedding [8]. We take the hierarchical learning structure in [9] as an example. Suppose the action space of the upper-level planning model has n dimensions, each action dimension has m discrete values, and the state space of the upper-level planning model has d dimensions, each with k discrete values. The action output of the planning model assigns corresponding action weights to the skill model. The exploration space size of the upper-level planning model is $k^d \times m^n$. The larger the exploration space, the more challenging it becomes for the planning model to learn. As the number of skill models increases, it will lead to rapid expansion of exploration space and increase the burden on the planning model. To reduce the planning burden, we propose an efficient hierarchical reinforcement learning method named HIKER. Firstly, we design a dual-chain hierarchical reinforcement learning method, which reduces optimization conflicts and alleviates planning burden. Secondly, we propose DyKAN for low-level skill models. DyKAN can dynamically expand network parameters based on data distribution. It can stably and efficiently master multiple manipulations without task ID embedding. Finally, we design the update module for the DyKAN layer. To simplify the expanding process of DyKAN, the update module uses low-rank matrix decomposition to unify the grids of B-spline activation functions in the same layer. Meanwhile, the update module adopts DyT [30] to constrain the input data distribution of each layer. Fig. 2 depicts our HIKER framework.

A. Dual-chain hierarchical reinforcement learning

We define a set of manipulation behaviors abstracted from action sequences as a sub-task chain $\{subtask_0, subtask_1, ..., subtask_n\}$. Using one skill model to learn multiple subtasks simultaneously can effectively reduce the action space dimension of the upper-level planning model, thereby reducing the planning burden. We don't adopt the multi-task learning based on task ID embedding. Although task ID embedding can help a single model master multiple manipulations, the planning model needs to output an ID for planning and thus cannot reduce the action space dimension of the planning model.

When manually dividing subtasks, failure to clearly define task transition boundaries can easily lead to state overlap between adjacent tasks. Therefore, we design a dual-chain hierarchical reinforcement learning method that decomposes the subtask chain $\{subtask_0, subtask_1, ..., subtask_n\}$ into two coupled task chains: One is $\{subtask_0, subtask_2, ..., subtask_{2k}\}$, and the other is $\{subtask_1, subtask_3, ..., subtask_{2k+1}\}$. As illustrated in Fig. 2.A, the subtasks within each chain share a skill model based on DykAN for learning, while an

upper-level planning model based on MLP is responsible for scheduling the two skill models. Through this dual-chain design, a clear separation exists between the state distributions of different subtasks within the same model. The requirement for high-precision manual definition of subtask boundaries is significantly reduced. Assuming the state distributions of $subtask_l$, $subtask_{l+1}$, and $subtask_{l+2}$ are denoted as Ds_l, Ds_{l+1}, and Ds_{l+2}, we define the intersection $Ds_{overlap}$ between these state distributions:

$$Ds_{overlap(l,l+1)} = Ds_l \cap Ds_{l+1} \neq \emptyset \tag{1}$$

$$Ds_{overlap(l,l+2)} = Ds_l \cap Ds_{l+2} = \emptyset \tag{2}$$

Taking the policy gradient RL method as an example, for state $s \in Ds_{overlap(l,l+1)}$, the robot executes action a_l in $subtask_l$ and action a_{l+1} in $subtask_{l+1}$. The advantage functions for the two actions are defined as follows:

$$A_l(s, a_l) = Q_l(s, a_l) - V(s) \tag{3}$$

$$A_{l+1}(s, a_{l+1}) = Q_{l+1}(s, a_{l+1}) - V(s) \tag{4}$$

A denotes the advantage function, Q represents the state-action value function, and V is the state value function. The skill model is updated based on the loss function $L(s)$. Thus, for $s \in Ds_{overlap(l,l+1)}$, $L(s)$ of the skill model is defined as Eq. (5):

$$\begin{aligned} L(s) &= L_{subtask_l}(s) + L_{subtask_{l+1}}(s) \\ &= F(A_l(s, a_l)) + F(A_{l+1}(s, a_{l+1})) \end{aligned} \tag{5}$$

$L_{subtask_l}(s)$ and $L_{subtask_{l+1}}(s)$ respectively represent the loss functions for $subtask_l$ and $subtask_{l+1}$. F denotes the mapping from the advantage function to the loss function. If $A_l, A_{l+1} > 0$ (or $A_l, A_{l+1} < 0$) and $a_l \neq a_{l+1}$, the optimization directions of $L_{subtask_l}(s)$ and $L_{subtask_{l+1}}$ may be in conflict in state s. For example, in a state $s \in Ds_{overlap(l,l+1)}$, $subtask_l$ requires the robot to return to a reset position (because of its completed objective), whereas $subtask_{l+1}$ requires the robot to execute its task objective. Although finely dividing the reset and start positions for each subtask can solve this problem, this approach relies on human intervention. Therefore, our dual-chain hierarchical reinforcement learning method achieves the state space intersection as shown in Eq. (2), which can avoid optimization conflicts in the same state.

B. Dynamic Kolmogorov–Arnold Network

The local support property of B-spline basis functions ensures that input data only affects a subset of basis functions in the activation functions of KAN [20]. We aim to leverage this local influence mechanism to assist multi-task learning in skill models. However, KAN periodically updates the grid range based on input data distribution without changing the grid size, and synchronously adjusts the learned basis function parameters and grid accuracy λ to match the changes of the grid. Under the exploration paradigm of RL, frequent network parameter adjustments will lead to training instability. Therefore, we propose the DyKAN, which maintains the grid accuracy λ when updating the grid, but dynamically adjusts the grid size based on changes in the grid range. DyKAN also adaptively modifies the number of basis functions in the activation functions according to changes in the grid size, thereby achieving dynamic updates to the number of learned basis function parameters. DyKAN matchs the grid range with the data distribution without adjusting the learned basis function parameters.

As shown in Fig. 2.B, assuming grid parameters initialization of a B-spline activation function $\phi(x)$ in DyKAN are as follows:

$$grid\ range = [a, b],\ grid\ size = 5,\ k = 3 \tag{6}$$

k is the order of the B-spline function, and the initial number of basis functions depends on the grid size and order k. The learnable basis function parameters are denoted as $C = \{c_0, c_1, c_2, ...\}$, and each parameter corresponds to a basis function $B_i(x)$. The B-spline activation function $\phi(x)$ is defined as:

$$\phi(x) = \sum_i c_i B_i(x), x \in [a, b], i \in [0, 7] \tag{7}$$

To achieve dynamic expansion of the number of learnable basis function parameters, we design a forward parameter set C_f and a backward parameter set C_b, which are used to store the additional learnable basis function parameters resulting from extensions of the upper and lower bounds of the grid range, respectively. Specifically, after each network update, we adjust the grid range based on the current upper and lower bounds of the data distribution. If the upper bound of the data distribution is greater than the upper bound of the grid range, the upper bound of the grid range is updated, and new learnable basis function parameters are added to C_f. Similarly, if the lower bound of the grid range is updated, the corresponding learnable basis function parameters are incorporated into C_b. During both training and inference, C_b, C, and C_f are concatenated to compute $\phi(x)$. To streamline parameter management, we periodically merge the parameters from C_b and C_f into C, thereby achieving controlled and orderly growth of the learnable basis function parameter scale. When the data distribution range shrinks, we do not scale down the grid range and the number of learnable basis function parameters synchronously. Frequent addition and removal of network parameters (learnable basis function parameters) can easily lead to training instability.

C. Update module for DyKAN layer

As illustrated in Fig. 2.C, We design an update module that unifies the grids of all B-spline activation functions within the same DyKAN layer to a common size and range, and constrains the input data distribution. Assuming a DyKAN layer has the input dimension q and the output dimension p, there are $q \times p$ activation functions in this DyKAN layer. Instead of maintaining an individual grid for each activation function, we define a shared grid for the entire layer based on the upper and lower bounds of the input data distribution corresponding to all activation functions in that layer. Each DyKAN layer maintains only one grid, which can help

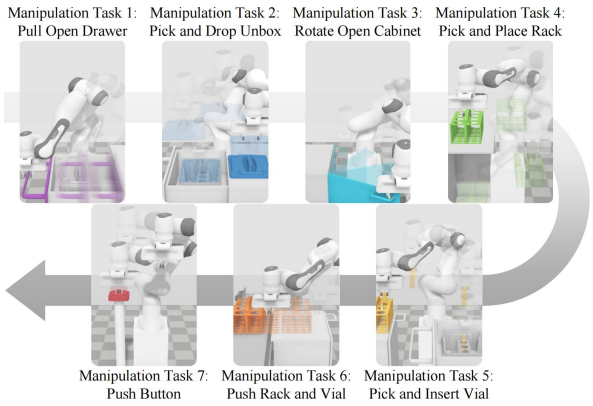

Fig. 3. Seven sequential manipulation tasks in the long-horizon task [9].

expand the basis function parameters at a lower cost. After unifying grids, all learnable basis function parameters for a DyKAN layer will be initialized as a matrix of dimension $q \times p \times j$. Then, we decompose this matrix into two low-rank matrices: A (with dimension $q \times p \times r$) and B (with dimension $r \times j$). r represents the low-rank dimension. When expanding the learnable basis function parameters, only the matrix B needs to be modified. This low-rank matrix decomposition effectively mitigates the potential rapid growth in the number of learnable basis function parameters due to grid expansion, also making network parameter expansion at a lower cost. Low-rank decomposition for the learnable basis function parameter matrix does not affect the local influence mechanism of DyKAN. Because during both the training process and the inference process, DyKAN first computes and restores the complete basis function parameter matrix before proceeding with subsequent calculations.

As shown in Fig. 2.C, we adopt the residual structure of KAN, which introduces a fully-connected layer to form a residual block together with the DyKAN layer. However, to evaluate the learning ability of basis function parameters, we freeze the fully connected layer during training. Finally, we introduce a DyT layer before each DyKAN layer, whose mathematical formulation is presented in Eq. (8). This function constrains the input data distribution within a bounded range through learnable parameters γ, α, β, thereby preventing drastic changes in network parameters (matrix B) scale caused by sudden shifts in data distribution and ensuring stable training. Additionally, DyT dynamically adjusts the constraining range during training by updating γ, α, β, thus maintaining the scalability of matrix B. Algorithm 1 introduces how DyKAN expands matrix B, where B_f and B_b are used to store new network parameter matrices, similar to C_f and C_b.

$$DyT(x) = \gamma tanh(\alpha x) + \beta \tag{8}$$

IV. EXPERIMENT

A. Task and Training Algorithm

HIKER will be validated in a chemical laboratory environment established in [9]. As shown in Fig. 3, in this environment, the robot (Franka robotic arm) is required to perform sequential manipulations to accomplish the complex long-horizon tasks.

Algorithm 1 The expansion process of matrix B

Initialize: Current training steps $step_{count}$, Step threshold of combining parameter $step_{thre}$, Step increment $step_{incre}$, Upper bound of grid range b, Lower bound of grid range a, Set B_f, Set B_b, Matrix B, γ and β form DyT

for network update episode **do**
  if network parameters have been updated **then**
    if $Max(Abs(\gamma) + \beta) > b$ **then**
      Update $b = Max(Abs(\gamma) + \beta)$
      Add new network parameter matrix to B_f
    end if
    if $Min(-Abs(\gamma) + \beta) < a$ **then**
      Update $a = Min(-Abs(\gamma) + \beta)$
      Add new network parameter matrix to B_b
    end if
    if $step_{count} >= step_{thre}$ **then**
      if $B_f \neq \emptyset$ *or* $B_b \neq \emptyset$ **then**
        Combine the matrices in B_f and B_b with matrix B to form a new matrix B
        Update sets $B_f = \emptyset$, $B_b = \emptyset$
      end if
      Update $step_{thre} = step_{thre} + step_{incre}$
    end if
  end if
end for

Based on dual-chain hierarchical reinforcement learning, we organize Manipulation Tasks 1, 3, 5, and 7 into a subtask chain, which is learned by Skill Model 1. Manipulation Tasks 2, 4, and 6 are learned by Skill Model 2. The definition of each manipulation task can be found in [9]. After the two lower-level skill models have been trained, an upper-level planning model is trained to schedule them, thereby accomplishing the long-horizon tasks. Both the skill models and the planning model employ the Proximal Policy Optimization (PPO) algorithm to train, and GAIL is incorporated to provide intrinsic rewards r_i. In the initial stages of training, we apply behavior cloning to infuse prior knowledge into the actor network. Furthermore, we design independent critic networks to approximate value functions V of extrinsic rewards r_e and internal rewards r_i, respectively.

B. State, Action and Reward

The planning model and skill models have the same state S. S comprises three types of information: the robot proprioceptive information G, target object information M, and specific environment information E. G includes the end effector velocity V and the gripper state $gripper\_state$. M covers the relative polar coordinates m_i of all target objects (e.g., drawers, racks, etc.) with respect to the end effector. E contains information that is not readily quantifiable through relative positions, such as button state. During training, we

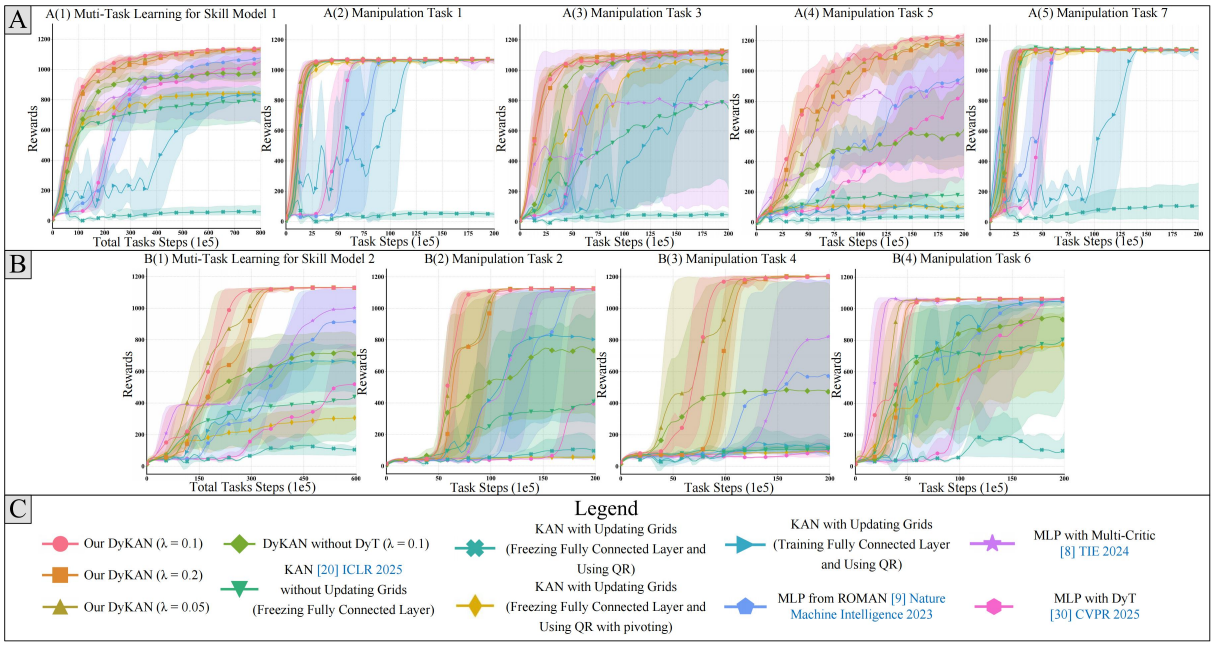

Fig. 4. Reinforcement learning r_e reward curves. **A** The average reward curve for total tasks in Skill Model 1, and the reward curves for each manipulation task in total tasks. **B** The average reward curve for total tasks in Skill Model 2, and the reward curves for each manipulation task in total tasks. **C** Legend.

add $\pm 0.5cm$ Gaussian noise to the positions of objects in the environment. Then we use these positions to calculate S.

The skill model employs a four-dimensional continuous action $a_{skill} = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)$. α_1, α_2, and α_3 respectively control the accelerations of the end effector along the x, y, and z axes. α_4 controls the state of the end effector's gripper. The planning model uses a two-dimensional action $a_{planning} = (\beta_1, \beta_2)$ to output the action weight β of each skill model. We use softmax to normalize the two action weights to obtain ω_1, ω_2. The actual action executed by the robot is $\omega_1 a_{skill_1} + \omega_2 a_{skill_2}$. As described in the section IV-A, the reward function is $r = 0.1 * r_e + r_i$. Our reward design draws on [9] and is adjusted to meet practical requirements.

V. EVALUATION AND RESULT

We aim to answer the following three core questions by evaluating the experiment results: 1) Can the skill model based on DyKAN successfully and efficiently master multiple manipulations? 2) Can DyKAN automatically distinguish between different subtasks? 3) How does HIKER perform in the long-horizon manipulation task?

A. Learning Performance of Lower-Level Skill Model

We use the PPO algorithm for training. Our critic and actor networks consist of three 16-dim DyKAN hidden layers and one DyKAN action/value output layer. The trained Actor network is the skill model we need. As indicated in [30], DyT can functionally replace the normalization layer due to their similar roles. To ensure fair comparison, we apply a normalization layer to state S inputs in methods that do not include DyT. As shown in Fig. 4, to evaluate the learning capability of DyKAN in multiple manipulation tasks, we conduct ablation studies and comparative experiments. First, we examine the impact of different grid precisions $\lambda \in \{0.2, 0.1, 0.05\}$ on training. DyKAN shows stable and efficient learning ability across all precisions. Second, the ablation study on the DyT layer reveals that constraining the input data is important for DyKAN training. Meanwhile, we compare DyKAN with KAN [20]. The grid initialization of KAN is the same as that of DyKAN. We conduct experiments under different configurations of KAN: (1) freezing the fully connected layers without updating the grids; (2) updating the grids using QR decomposition while keeping the fully connected layers frozen; (3) updating the grids using QR decomposition with pivoting while keeping the fully connected layers frozen; and (4) updating the grids using QR decomposition while training the fully connected layers. It is worth noting that our DyKAN also freezes the fully connected layers during training. In all configurations of KAN involving grid updates, grids are only updated during the first half of the training phase. The results indicate that all KAN configurations exhibit inferior learning performance compared to DyKAN. DyKAN can update grids without affecting learned knowledge, making training more stable and efficient. DyKAN's unique network parameter extension and grid update mechanism make it more suitable for complex RL tasks.

Furthermore, we compare DyKAN with the MLP-based network in ROMAN [9] and the multi-critic model with asymmetric inputs in GTHSL [8]. The multi-critic model also uses the MLP-based network, and we omit the task ID encoding in the actor network to meet the action dimensionality reduction requirements of the upper-level planning model. Considering the difficulty of multi-task learning, both the critic and actor networks in the two compared methods consist of three 256-dim hidden layers and one

action/value output layer. The results show that the two compared methods can not stably learn all subtasks. MLP is difficult to handle multi-task gradient conflict stably without additional task ID encoding input. Finally, to investigate whether combining DyT with other network can improve network learning performance in RL, we combine MLP with DyT [30] for the experiment. The results indicate that there is no improvement in learning performance compared to MLP, and its learning performance is inferior to MLP in Skill Model 2 tasks. The results prove that other components of DyKAN are also important. Overall, DyKAN demonstrates stable and efficient learning ability in RL, while HIKER can efficiently learn higher-quality skill models.

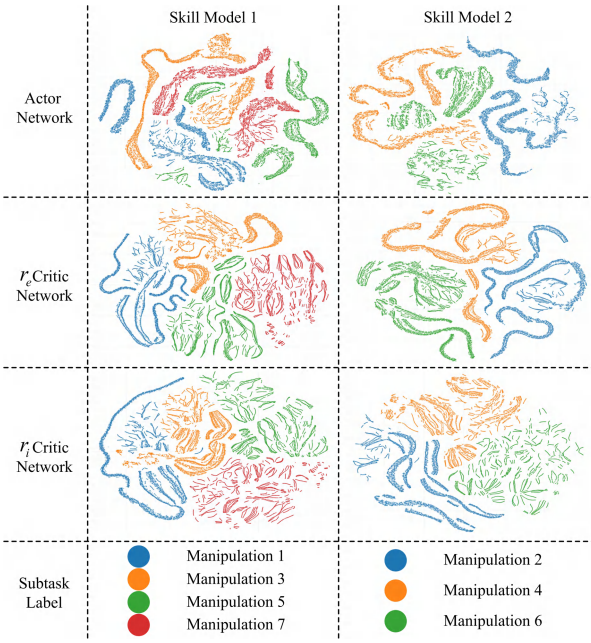

Fig. 5. T-SNE analysis of hidden states in different networks. Each subtask undergoes 20000 samples.

B. Visualization Analysis of Task Hidden States in DyKAN

We respectively collect the hidden states from the first DyKAN hidden layer in the actor network, r_e critic network, and r_i critic network. To evaluate whether DyKAN can distinguish between different subtasks, we use T-SNE for dimensionality reduction analysis on the hidden states. The results shown in Fig. 5 indicate that the hidden states of different subtasks exhibit clearly clustered distributions. There are some exceptions, such as the hidden states of the actor network in Manipulation 5, which deviate from the main cluster and are closer to the states of other tasks. This phenomenon may stem from shared characteristics among different manipulations, such as the process of approaching the goal object. Nevertheless, these states do not significantly overlap with those of other subtasks. The result suggests that DyKAN can effectively distinguish between different subtasks. Based on the local influence mechanism in Section III-B, different hidden states can activate distinct basis functions. Although some basis functions are shared, some basis functions are still independent. The independence of these basis functions enables them to effectively extract the distinctive features of subtasks, thereby providing crucial support for DyKAN to alleviate multi-task optimization conflicts and accurately distinguish tasks. Therefore, the DyKAN skill model can autonomously determine which manipulation to execute based on the current state, without requiring any additional information.

C. Long-Horizon Manipulation Task Inference Results

To evaluate the performance of HIKER in the long-horizon manipulation task, we follow the Inference experiment from [9] and compare HIKER with ROMAN (ROMAN with GAIL Rewards) [9] and its variants [12]. ROMAN trains seven independent MLP-based skill models and an MLP-based planning model. For fairness, HIKER uses an MLP-based planning model with the same hidden layers. As shown in Fig. 6, HIKER achieves the highest success rate in tasks.

In ROMAN and its variants [9] [12], ROMAN with intrinsic reward shows higher success rates than ROMAN without intrinsic reward. ROMAN with LLM reward [12] achieves high success rates under low noise levels, but its success rates drop significantly at noise levels of 1.5 cm and 2.0 cm, indicating poor robustness. ROMAN with Gail reward [9] is slightly inferior to the former under low noise but maintains relatively high success rates even under high noise, demonstrating stronger robustness. Since all ROMAN methods use absolute position for state representation, we retrain the ROMAN using our state representation and reward. The results indicate that HIKER outperforms ROMAN, which uses our state representation. For example, HIKER improves the success rate by 10.9% in Case 7 with a 2.0 cm noise level. Overall, HIKER not only achieves higher success rates in long-horizon manipulation tasks but also better robustness.

VI. CONCLUSIONS

We propose HIKER, a novel hierarchical reinforcement learning framework designed for efficient and robust learning of long-horizon robotic manipulation tasks. At its core, the DyKAN autonomously distinguishes subtasks and enables the skill model to master multiple manipulations. Furthermore, DyKAN dynamically adjusts grids and basis functions while preserving learned knowledge, thereby supporting stable and efficient learning performance in multi-task RL tasks. The dual-chain architecture in HIKER mitigates optimization conflicts among subtasks and reduces the burden on the planning model. Experimental results show that HIKER efficiently learns high-quality skill models and achieves more effective and robust performance in long-horizon robot manipulation. Overall, these findings demonstrate the value of HIKER in advancing long-horizon robotic manipulation.

REFERENCES

[1] N. J. Szymanski, B. Rendy, Y. Fei, R. E. Kumar, T. He, D. Milsted, M. J. McDermott, M. Gallant, E. D. Cubuk, A. Merchant, *et al.*, "An autonomous laboratory for the accelerated synthesis of novel materials," *Nature*, vol. 624, no. 7990, pp. 86–91, 2023.

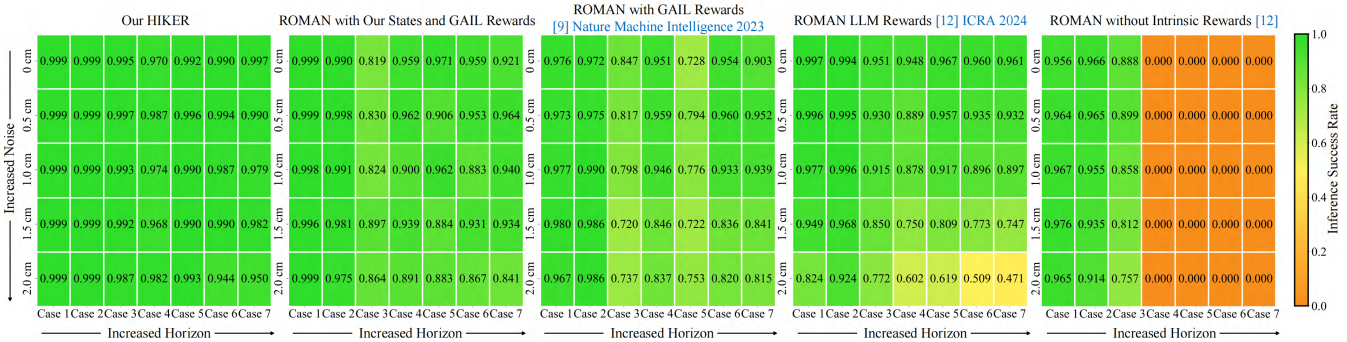

Fig. 6. Inference results of five different methods. X-axis represents different task cases: Case 1 only requires the execution of manipulation 7, while Case 7 requires the sequential execution of manipulations 1 to 7. Y-axis represents the Gaussian noise in positions. Each cell stems from 1K episodes.

[2] J. Zhang, N. Gireesh, J. Wang, X. Fang, C. Xu, W. Chen, L. Dai, and H. Wang, "Gamma: Graspability-aware mobile manipulation policy learning based on online grasping pose fusion," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1399–1405, IEEE, 2024.

[3] Y. Fujita, K. Uenishi, A. Ummadisingu, P. Nagarajan, S. Masuda, and M. Y. Castro, "Distributed reinforcement learning of targeted grasping with active vision for mobile manipulators," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9712–9719, IEEE, 2020.

[4] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar, "Deep dynamics models for learning dexterous manipulation," in *Conference on robot learning (CoRL)*, pp. 1101–1112, PMLR, 2020.

[5] T. Zhao, V. Kumar, S. Levine, and C. Finn, "Learning fine-grained bimanual manipulation with low-cost hardware," in *Robotics: Science and Systems (RSS)*, 2023.

[6] C. Li, F. Xia, R. Martin-Martin, and S. Savarese, "Hrl4in: Hierarchical reinforcement learning for interactive navigation with mobile manipulators," in *Conference on Robot Learning (CoRL)*, pp. 603–616, PMLR, 2020.

[7] X. Huang, D. Batra, A. Rai, and A. Szot, "Skill transformer: A monolithic policy for mobile manipulation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 10852–10862, 2023.

[8] R. Jiang, X. Cheng, H. Sang, Z. Wang, Y. Zhou, and B. He, "Gthsl: A goal-task-driven hierarchical sharing learning method to learn long-horizon tasks autonomously," *IEEE Transactions on Industrial Electronics*, 2024.

[9] E. Triantafyllidis, F. Acero, Z. Liu, and Z. Li, "Hybrid hierarchical learning for solving complex sequential tasks using the robotic manipulation network roman," *Nature Machine Intelligence*, vol. 5, no. 9, pp. 991–1005, 2023.

[10] H. Zhou and X. Lin, "Intelligent redundant manipulation for long-horizon operations with multiple goal-conditioned hierarchical learning," *Advanced Robotics*, vol. 39, no. 6, pp. 291–304, 2025.

[11] Z. Sun, B. Pang, X. Yuan, X. Xu, Y. Song, R. Song, and Y. Li, "Hierarchical reinforcement learning with curriculum demonstrations and goal-guided policies for sequential robotic manipulation," *Engineering Applications of Artificial Intelligence*, vol. 153, p. 110866, 2025.

[12] E. Triantafyllidis, F. Christianos, and Z. Li, "Intrinsic language-guided exploration for complex long-horizon robotic manipulation tasks," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7493–7500, IEEE, 2024.

[13] C. Zhang, Z. Sun, and G. S. Chirikjian, "Goal-guided reinforcement learning: Leveraging large language models for long-horizon task decomposition," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10744–10750, IEEE, 2025.

[14] O. Mees, J. Borja-Diaz, and W. Burgard, "Grounding language with visual affordances over unstructured data," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11576–11582, IEEE, 2023.

[15] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, *et al.*, "π_0: A vision-language-action flow model for general robot control," *arXiv preprint arXiv:2410.24164*, 2024.

[16] Z. Jia, V. Thumuluri, F. Liu, L. Chen, Z. Huang, and H. Su, "Chain-of-thought predictive control," in *International Conference on Machine Learning (ICML)*, pp. 21768–21790, PMLR, 2024.

[17] D. Wang, C. Liu, F. Chang, and Y. Xu, "Hierarchical diffusion policy: manipulation trajectory generation via contact guidance," *IEEE Transactions on Robotics*, 2025.

[18] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, 2017.

[19] R. Parr and S. Russell, "Reinforcement learning with hierarchies of machines," *Advances in neural information processing systems*, vol. 10, 1997.

[20] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljacic, T. Y. Hou, and M. Tegmark, "Kan: Kolmogorov–arnold networks," in *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

[21] K. Shukla, J. D. Toscano, Z. Wang, Z. Zou, and G. E. Karniadakis, "A comprehensive and fair comparison between mlp and kan representations for differential equations and operator networks," *Computer Methods in Applied Mechanics and Engineering*, vol. 431, p. 117290, 2024.

[22] Y. Wang, J. Sun, J. Bai, C. Anitescu, M. S. Eshaghi, X. Zhuang, T. Rabczuk, and Y. Liu, "Kolmogorov–arnold-informed neural network: A physics-informed deep learning framework for solving forward and inverse problems based on kolmogorov–arnold networks," *Computer Methods in Applied Mechanics and Engineering*, vol. 433, p. 117518, 2025.

[23] J. D. Toscano, L.-L. Wang, and G. E. Karniadakis, "Kkans: Kurkova-kolmogorov-arnold networks and their learning dynamics," *Neural Networks*, vol. 191, p. 107831, 2025.

[24] L. Li, Y. Zhang, G. Wang, and K. Xia, "Kolmogorov–arnold graph neural networks for molecular property prediction," *Nature Machine Intelligence*, pp. 1–9, 2025.

[25] C. Li, X. Liu, W. Li, C. Wang, H. Liu, Y. Liu, Z. Chen, and Y. Yuan, "U-kan makes strong backbone for medical image segmentation and generation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, pp. 4652–4660, 2025.

[26] Z. Gao and M. Kong, "Mp-kan: An effective magnetic positioning algorithm based on kolmogorov-arnold network," *Measurement*, vol. 243, p. 116248, 2025.

[27] Z. Lin, J. Lan, and X. Zhao, "Kan-lstm enhanced multi-agent advantage actor-critic reinforcement learning for autonomous ramp merging," *IEEE Transactions on Vehicular Technology*, 2025.

[28] Z. Lin, Z. Tian, J. Lan, Q. Zhang, Z. Ye, H. Zhuang, and X. Zhao, "A conflicts-free, speed-lossless kan-based reinforcement learning decision system for interactive driving in roundabouts," *IEEE Transactions on Intelligent Transportation Systems*, 2025.

[29] T. Liu and Y. Zhang, "Kan-enhanced deep reinforcement learning for chaos control: Achieving rapid stabilization via minor perturbations," *Physica D: Nonlinear Phenomena*, p. 134915, 2025.

[30] J. Zhu, X. Chen, K. He, Y. LeCun, and Z. Liu, "Transformers without normalization," in *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*, pp. 14901–14911, 2025.