

STAR: High-DoF Robotic Manipulation for Memory-Constrained NN Accelerator

Jhao-Ying Chen*, Wen Sheng Lim*, Tei-Wei Kuo*[†], Yuan-Hao Chang*

*Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan

[†]Delta Electronics, Inc.

{joel0115, tundergod1882}@gmail.com {ktw, johnson}@csie.ntu.edu.tw

Abstract—As robotic manipulators adopt increasingly higher degrees of freedom (DoFs) to handle complex tasks, the corresponding growth in neural network (NN) size leads to substantial memory and energy demands, making deployment on low-level controllers increasingly impractical. To overcome this challenge, we propose STAR, a novel framework that enables accurate and energy-efficient high-DoF manipulation under strict memory constraints. STAR introduces a *spherical task-space approximation* strategy to mathematically formulate the manipulator’s reachable space, followed by a *memory-aware training* algorithm that adaptively divides this space into smaller, manageable regions, with each partition assigned a lightweight NN optimized to satisfy memory capacity while preserving high precision. Specifically, STAR employs deep reinforcement learning (DRL) to learn absolute pose-to-joint mappings, allowing each task to be completed with a single NN load, eliminating the need for large networks or frequent NN switching. Experiments demonstrate that STAR achieves up to $8.09\times$ faster execution and $10.93\times$ lower energy consumption, while reducing memory usage by up to $128\times$ compared to state-of-the-art approaches, all without compromising control accuracy.

Index Terms—Robotic manipulation, reinforcement learning, neural network inference, accelerator

I. INTRODUCTION

Robotic systems have recently attracted growing attention, ranging from industrial manipulators to mobile platforms and humanoid assistants [1]–[4]. While high-level planning tasks (to generate a sequence of target poses by an AI/ML model) must be customized to specific applications, the low-level control functionality remains consistent across systems to *operate joint-level motor commands to achieve a given pose*, as shown in Fig. 1. Consequently, there is a growing need for low-level controllers that are not only accurate and fast, but also energy-efficient and cost-effective, especially given the increasing scale and diversity of robotic applications.

Inverse Kinematics (IK) [5] is a fundamental component of low-level control. It involves computing the joint angles required to move a robotic manipulator’s end-effector to a specified pose in Cartesian space (i.e., the three-dimensional coordinate system (x, y, z) task space). Solving the IK problem is challenging due to the nonlinear mapping between joint angles and end-effector poses. For example, shifting the end-effector slightly from pose $(2, 2, 2)$ to $(3, 3, 3)$ may require significant and non-intuitive changes in multiple joint angles, such as $(22^\circ, 33^\circ, 44^\circ)$ in a 3-DoF manipulator.

Traditional robotic systems primarily rely on *analytical* methods [6] and *numerical* methods [7], [8] to solve the inverse

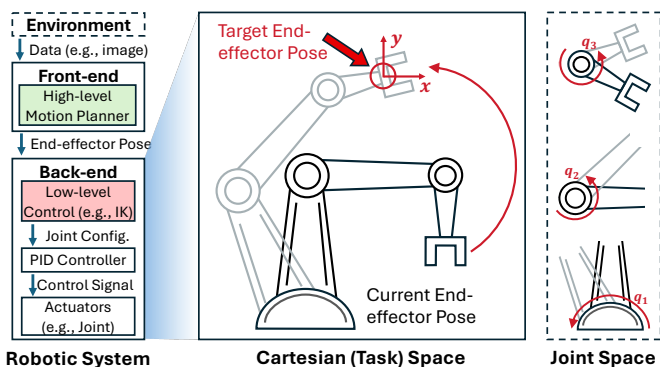


Fig. 1: Illustration of a 3-DoF robotic manipulation system.

kinematics (IK) problem. Analytical methods derive closed-form equations using geometric and trigonometric relationships, offering fast and exact solutions for computing joint angles from a target pose. However, analytical methods are generally applicable only to manipulators with six or fewer DoFs [9]. Modern robots, such as Tesla’s Optimus humanoid with 22 DoFs [10], far exceed this range, rendering analytical solutions intractable. Moreover, these methods demand substantial manual effort to derive the dedicated closed-form equation. Numerical methods, by contrast, employ iterative algorithms to approximate joint angles for a desired end-effector pose. These methods are broadly applicable to high-DoF and complex structures but are computationally intensive, prone to convergence issues, and difficult to parallelize.

To achieve efficient and cost-effective low-level control that can adapt to diverse robot architectures and scale to high-DoF manipulators, recent research has focused on neural network (NN)-based IK solvers [11]–[13]. These methods train NNs on large datasets to learn the *nonlinear mapping between target end-effector poses in task space* (e.g., (x, y) in Fig. 1) and the *corresponding relative joint rotations in joint space* (e.g., (q_1, q_2, q_3) in Fig. 1). However, our evaluation shows that the memory demand of current NN-based methods grows rapidly with the number of DoFs, leading to exponential increases in model size. For example, an 8-DoF manipulator requires more than 1 GB of memory (see Section II-B). Such requirements are impractical for low-level controllers where the on-chip SRAM is prohibitively expensive¹ and energy consuming [16], [17],

¹The cost of SRAM scales rapidly with capacity: an ISSI’s 8 KB SRAM block may cost approximately \$1.08 [14], whereas an ISSI’s 1 MB SRAM block can reach up to \$9.71, i.e., nearly $9\times$ higher [15].

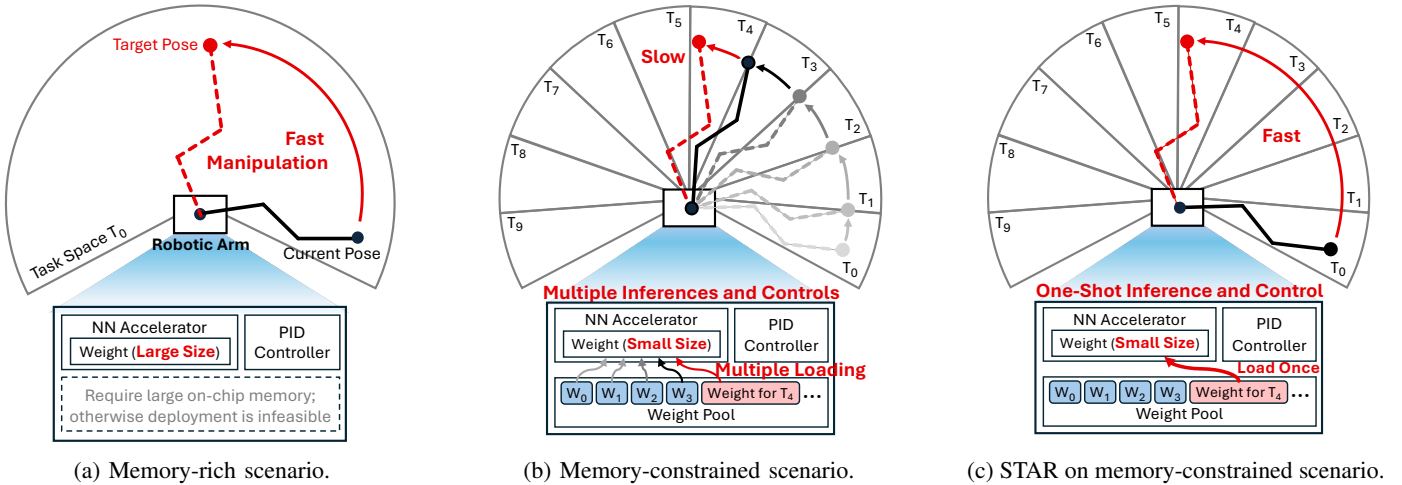


Fig. 2: (a) Conventional methods require a large NN to learn the Cartesian-to-joint mapping; (b) Partitioning reduces NN size but causes frequent loading and latency; (c) STAR achieves one-shot inference by selecting only the NN for the target partition.

while using off-chip access or lower-tier memory introduces high latency and undermines real-time control feasibility.

This paper proposes STAR, a novel NN-based robotic manipulation framework designed to achieve accurate, low-latency, and energy-efficient control for high-DoF manipulators under tight memory constraints. Unlike existing NN approaches, STAR introduces a memory-aware training framework that learns the mapping between the current end-effector pose and the *target joint configurations* within the partitioned task space. At runtime, STAR requires loading only a single lightweight NN corresponding to the target partition, with each model trained under the capacity limits of on-chip memory. Specifically, STAR consists of two key components. First, the Spherical Task space Approximation strategy mathematically formulates the manipulator’s reachable task space. Second, a memory-aware training algorithm based on deep Reinforcement learning (DRL) partitions this space into multiple regions, with each partition assigned a lightweight NN optimized to satisfy memory capacity while preserving high precision. To the best of our knowledge, STAR is the first paper to demonstrate that investigating the Cartesian-to-joint mappings with memory-aware task space partitioning enables accurate robotic manipulation under strict on-chip memory constraints.

STAR is evaluated on Genesis [18] robot simulator with 6-DoF DRV90L [19] and 7-DoF Franka Emika Panda [20] robotic manipulators. STAR achieves up to $8.09\times$ faster execution, $10.93\times$ lower energy consumption, and $128\times$ smaller NN size without sacrificing manipulation accuracy. These results validate STAR’s ability to deliver scalable, low-cost, and energy-efficient robotic control across robotic manipulators on a memory-constrained NN accelerator.

II. BACKGROUND AND MOTIVATION

A. Robotic Manipulation using DRL

Machine learning offers a data-driven alternative to traditional IK solvers by learning the nonlinear mappings between the current end-effector pose (in task space) and the target joint angles (in joint space) using neural networks [11]–[13]. In particular, DRL enables robotic manipulators to learn control policies

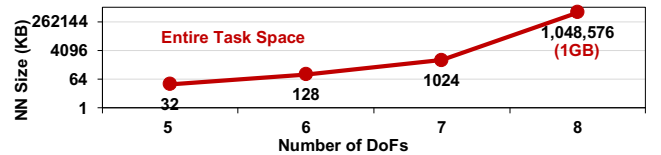


Fig. 3: The impact of DoFs with entire task space.

through direct interaction with the environment [21]–[27]. Among DRL methods, *Proximal Policy Optimization* (PPO) is widely employed for robotic manipulation [21], where its actor-critic structure naturally handles high-dimensional continuous actions (i.e., multi-joint torque commands for robotic arms) while reducing gradient variance for stable training [28].

B. Large Memory Demand for High-DoF Manipulation

Prior studies have optimized PPO for mm-scale precision in robotic manipulation [21], [26]. Building on these advances, this work shifts the focus to a system-level challenge by analyzing the memory demands of PPO-trained models for energy-efficient and cost-effective low-level controller deployment. Fig. 3 presents our experiments on memory demand across different DoFs in robotic systems trained with PPO under 5 mm precision. For the 5-DoF case, we disable one joint of a DRV90L arm, while for the 8-DoF case, we adopt a dual-arm configuration with two 6-DoF DRV90L arms, disabling two joints on each. We observe that the NN size required to represent the entire task space grows exponentially with the number of DoFs, reaching nearly 1 GB at 8 DoFs. While existing approaches show fast and accurate manipulation, GB-scale on-chip memory is not realistic for low-level robotic controllers. When such large models cannot fit into on-chip memory, deployment becomes infeasible due to excessive off-chip memory access, which introduces prohibitive latency, energy consumption, and unreliable real-time control (Fig. 2(a)).

C. Task Space Partitioning Incurs High Manipulation Latency

A straightforward solution to reduce memory footprint is to partition the entire task space into multiple sub-spaces and train a dedicated NN model for each. During manipulation,

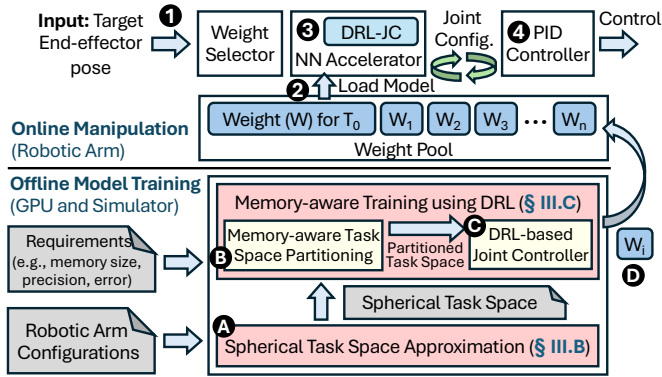


Fig. 4: An overview of STAR’s workflow.

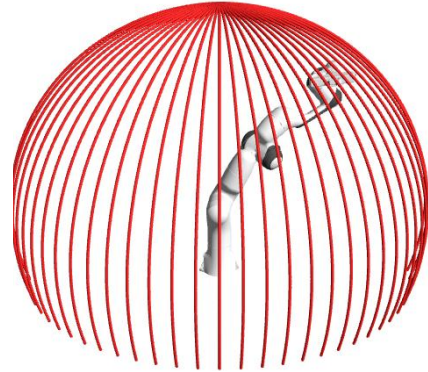


Fig. 5: Partitioned task space of Panda manipulator.

however, as demonstrated in Fig. 2(b), the robot must perform partition-wise inference and control by repeatedly loading the corresponding NN weights (T0 to T4) from a weight pool (e.g., from DRAM to on-chip SRAM memory), executing motion within the current partition, transitions to the next, and continues this process until the end-effector reaches the target pose. While this approach reduces the memory requirement of each individual NN, it introduces frequent NN switching, which increases latency, energy consumption, and control complexity.

D. Motivation

We propose STAR, a novel NN-based robotic manipulation framework tailored to memory-constrained platforms. As illustrated in Fig. 2(c), STAR selects only the model weight corresponding to the target partition (e.g., T4). During inference, the selected model directly outputs the joint commands required to move the end-effector from T0 to T5. This *one-shot inference and control* allows the robot to perform fast, energy-efficient, and accurate manipulation under strict memory budgets. Unlike conventional approaches, STAR learns the *mapping from the target end-effector pose in task space to the corresponding joint configuration*. This design enables each NN to infer the final joint angles directly, independent of the current end-effector pose. By decoupling task space coverage from NN size, STAR supports conceptually “infinite” workspace partitioning while requiring only one NN load and execution per task.

III. THE STAR FRAMEWORK

A. Overview

The STAR framework is designed to enable memory-aware NN-based robotic manipulation on highly constrained low-level controllers. As illustrated in Fig. 4, STAR consists of two major components: a *spherical task space approximation* strategy to mathematically formulate the manipulator’s reachable task space, enabling effective and scalable task space partitioning and avoiding the exponential complexity of high-DoF joint spaces. Second, a *memory-aware DRL training* algorithm partitions this space into multiple regions, with each partition assigned a lightweight NN optimized to learn pose-to-joint mapping while satisfying memory capacity and preserving high precision. During runtime, given an end-effector pose, STAR aims to minimize total execution time and energy consumption while preserving high-precision manipulation performance.

Workflow. Fig. 4 illustrates the STAR system architecture, which operates in two phases. During the offline phase, STAR performs three key tasks. First, it constructs a spherical approximation of the task space (A), based on the kinematic structure and joint limits of the robotic manipulator (Section III-B). Second, STAR applies memory-aware partitioning (B) to segment this task space according to the available SRAM capacity of the NN accelerator. Each partition is then evaluated using an accuracy-aware DRL module to ensure that the chosen number of partitions satisfies precision requirements such as positional error and success rate. To find the appropriate number of partitions, STAR performs a search guided by accuracy feedback. Once the partition count is finalized, STAR (C) trains one NN per partition using the PPO algorithm (Section III-C). These trained NN weights, denoted as W_i , are stored in a weight pool (D) and deployed on the low-level DRL-based joint controller (DRL-JC). In the online manipulation phase, STAR performs inference and control in four steps: (1) the high-level planner provides a target end-effector pose, which is used by a weight selector to choose the appropriate NN weight from the pool; (2) the selected NN is loaded onto the NN accelerator; (3) the DRL-JC performs one-shot task execution to generate the target joint configuration; and (4) the PID controller applies this configuration to drive the actuators. While a single inference is typically sufficient, multiple inference iterations may be needed to reach the final pose, as discussed in Section II-A.

B. Spherical Task Space Approximation

The maximum reachable range of a robotic manipulator is typically defined in joint space based on the joint limits. However, directly partitioning joint space is impractical, as the number of possible joint angle combinations grows exponentially with the number of DoFs. To enable efficient representation and facilitate memory-aware partitioning, we instead approximate the robot’s task space using spherical coordinates². The spherical coordinate system is defined by the radius r , elevation angle θ , and azimuth angle ϕ , as expressed by the following equations:

²Most robotic manipulators are built with rotational joints, resulting in a task space that closely resembles a sphere.

Algorithm 1: Memory-aware Training

Initialize: Memory budget M , threshold τ , target rate ρ , NN architecture \mathcal{A}

- 1 Adjust hidden dims of \mathcal{A} to fit M ;
- 2 **for** $P = \text{SearchAlg}(P)$; ▷ determine the number of partitions
- 3 **do**
- 4 $\pi_0 \leftarrow \text{TrainPPO}(\mathcal{A}, \text{partition}_0)$; ▷ train partition_0
- 5 $r \leftarrow \text{Evaluate}(\pi_0, \tau)$; ▷ evaluate success rate r
- 6 **if** $r \geq \rho$ **then**
- 7 Store W_0 ;
- 8 $n_{\text{part}} \leftarrow P$;
- 9 **break**
- 10 **end**
- 11 **end**
- 12 **for** $i = 1$ to $n_{\text{part}} - 1$; ▷ train remaining partitions
- 13 **do**
- 14 $\pi_i \leftarrow \text{TrainPPO}(\mathcal{A}, \text{partition}_i)$;
- 15 Store W_i ;
- 16 **end**

$$\begin{aligned} x &= r \sin \theta \sin \phi, & 0 \leq r \leq L \\ y &= r \sin \theta \cos \phi, & 0 \leq \theta \leq \pi \\ z &= r \cos \theta, & 0 \leq \phi \leq 2\pi \end{aligned} \quad (1)$$

where L is the length of the manipulator. We assume the robotic manipulator is oriented along the positive x-axis. The ranges of θ and ϕ then define the effective bounds of the spherical task space. Partitioning is performed by segmenting the azimuth angle ϕ , as illustrated in Fig. 5. This representation is independent of the robot’s DoFs, making it well-suited for partitioning complex manipulators with high-dimensional joint spaces.

C. Memory-aware Training using DRL

1) *Memory-aware Task Space Partitioning:* To enable memory-aware training, STAR first searches for the appropriate number of task space partitions that satisfy both the SRAM budget and success rate requirements. As shown in Algorithm 1, for each candidate partition count P , STAR trains an NN on partition_0 and evaluates its success rate (lines 4–5). If the success rate meets the target threshold, the corresponding weight W_0 is stored in the weight pool, and training proceeds for the remaining partitions (lines 12–15). Otherwise, P is increased to halve the partition size, thereby reducing the learning complexity and improving accuracy. Conversely, P is decreased if the NN’s memory usage is significantly below the budget, ensuring efficient utilization of available SRAM. This process repeats until a valid configuration is found. To accelerate convergence, a systematic search method (e.g., binary search) can be employed, where previously evaluated partition counts from other memory budgets serve as the bounds, allowing STAR to narrow the search interval and avoid redundant training runs while rapidly converging to a feasible configuration.

2) *DRL-based Joint Controller (DRL-JC):* DRL-JC determines the joint configurations for the underlying PID controller, and this decision-making process is modeled as a DRL problem in this work. Specifically, the joint configurations are produced by an RL *agent* (i.e., a neural network) that learns low-level control policies by interacting with the robotic manipulation system (*environment*) and generating absolute target joint angles to

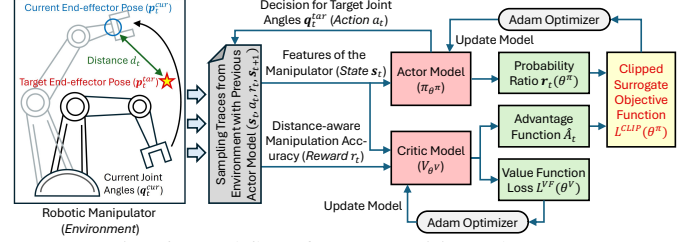


Fig. 6: Workflow for actor-critic style PPO.

enable one-shot loading. The decisions (*actions*) are made based on the *state* observed from the environment. After each inference step, DRL-JC receives a *reward* that reflects the success rate. A task is considered complete once the Euclidean distance between the current and target end-effector poses falls below a predefined threshold (e.g., 5 mm). We adopt PPO as the training algorithm due to its training stability and suitability for high-dimensional continuous actions.

State. The agent observes the device status (from the sampling traces during training) when it is about to make the task manipulation decision (the rotation angle for each joint) in each step t by given a target end-effector pose. For each time step t , the observed state s_t is defined as below:

$$s_t = \{\mathbf{q}_t^{cur}, \mathbf{p}_t^{cur}, \mathbf{p}_t^{tar}, d_t\} \quad (2)$$

where \mathbf{q}_t^{cur} is the current joint angles of the robotic manipulator, \mathbf{p}_t^{cur} and \mathbf{p}_t^{tar} is the current and target end-effector poses, respectively, and d_t is the Euclidean distance between the current and target end-effector poses. $\mathbf{q}_t^{cur} \in \mathbb{R}^n$, where n is the number of DoFs, and both $\mathbf{p}_t^{cur}, \mathbf{p}_t^{target} \in \mathbb{R}^3$ for the Cartesian poses.

Action. Given an observed state s_t , the actor NN decides the angles of each joint to manipulate the robotic manipulator. The action is defined as $a_t = \mathbf{q}_t^{tar}$, which is the target joint configuration to be applied to the robotic manipulator. To support one-shot NN loading, STAR employs absolute mapping between Cartesian target poses and joint angles. Instead of learning relative joint updates from the current pose, each NN directly predicts the absolute joint configuration needed to reach the target pose. This approach removes dependence on the initial state, significantly reducing runtime overhead.

$$\mathbf{q}^{tar} = \{q_1^{tar}, q_2^{tar}, \dots, q_{DoF}^{tar}\} \quad (3)$$

Reward. The reward function R is designed to encourage the end-effector to approach the target pose. When the Euclidean distance d_t between the current and target end-effector poses falls below a predefined threshold τ , the manipulation is completed and a positive reward c is given, where τ is the error determining the task completion; otherwise (unsuccessful manipulation occurs and require next step of manipulations), a distance-based reward is provided to encourage the agent to reach the goal. The reward function R is defined as:

$$R = \begin{cases} c, & \text{if } d_t \leq \tau \\ e^{-\alpha * d_t}, & \text{if } d_t > \tau \end{cases} \quad (4)$$

3) *Training DRL-JC:* Fig. 6 illustrates the architecture of PPO, which consists of two NNs: the actor and the critic. The

Algorithm 2: TRAINPPO(\mathcal{A} , partition)

Initialize: Actor π_{θ^π} , Critic V_{θ^V} , clip ratio ϵ , horizon length T , initial pose \mathbf{p}_0^{cur} and target pose \mathbf{p}_0^{tar} in partition

```

1 Initialize  $\pi_{\theta_{old}^\pi} \leftarrow \pi_{\theta^\pi}$ ;
2 for iteration = 0, 1, 2, ... do
3   for time step  $t = 0$  to  $T - 1$  do
4     Execute  $a_t$  from  $\pi_{\theta^\pi}(a|s_t)$  to collect data
       ( $s_t, a_t, r_t, s_{t+1}$ );
5   end
6   for  $t = 0$  to  $T - 1$  do
7      $\delta_t = r_t + \gamma V_{\theta^V}(s_{t+1}) - V_{\theta^V}(s_t)$ ; ▷ TD-errors
8      $\hat{A}_t = \sum_{l=0}^{T-t-1} (\gamma\lambda)^l \delta_{t+l}$ ; ▷ GAE
9      $\hat{R}_t = \sum_{l=0}^{T-t-1} \gamma^l r_{t+l}$ ; ▷ estimated returns
10  end
11  foreach minibatch do
12     $\mathbf{r}_t(\theta^\pi) = \frac{\pi_{\theta^\pi}(a_t|s_t)}{\pi_{\theta_{old}^\pi}(a_t|s_t)}$ ; ▷ policy ratio
13     $\mathcal{L}^{CLIP} = \mathbb{E}_t[\min(\mathbf{r}_t \hat{A}_t, \text{clip}(\mathbf{r}_t, 1-\epsilon, 1+\epsilon)\hat{A}_t)]$ ; ▷ objective
14     $\mathcal{L}^{VF} = \mathbb{E}_t[(V_{\theta^V}(s_t) - \hat{R}_t)^2]$ ; ▷ quadratic return error
15     $\pi_{\theta_{old}^\pi} \leftarrow \pi_{\theta^\pi}$ ;
16    Update  $\theta^\pi$  via gradient ascent on  $\mathcal{L}^{CLIP}$ ; ▷ Adam optimizer
17    Update  $\theta^V$  via gradient descent on  $\mathcal{L}^{VF}$ ; ▷ Adam optimizer
18  end
19  Re-initialize  $\mathbf{p}_0^{cur}$ ,  $\mathbf{p}_0^{tar}$  for next iteration
20 end
21 return  $\pi_{\theta^\pi}$ ;

```

actor NN, parameterized by θ^π , defines a policy $\pi_{\theta^\pi}(a_t | s_t)$, which governs the agent’s action selection based on the current state s_t . The critic NN, parameterized by θ^V , estimates the value function $V_{\theta^V}(s_t)$, representing the expected discounted return when the manipulation starts from state s_t under the current policy. The architecture enables the actor to focus on exploration and decision-making, while the critic provides evaluative feedback to guide more stable and efficient learning. Algorithm 2 outlines the training procedures of DRL-JC. The details of the PPO algorithm can be found in [21], [29].

D. Weight Selection and Runtime Manipulation

During runtime, given a target end-effector pose, the DRL-JC loads the NN corresponding to the target region into on-chip memory. Although the pose is conventionally expressed in Cartesian coordinates (x, y, z) , STAR represents the workspace as a sphere parameterized by spherical coordinates (r, θ, ϕ) (see Section III-B). The workspace is partitioned by segmenting the azimuth angle ϕ , with Cartesian coordinates transformed into spherical coordinates using the following inverse trigonometric relations:

$$\begin{aligned}
r &= \sqrt{x^2 + y^2 + z^2} \\
\theta &= \cos^{-1}\left(\frac{z}{r}\right) \\
\phi &= \text{atan2}(x, y)
\end{aligned}$$

This transformation is efficient, requiring only basic inverse trigonometric operations, and allows rapid identification of the NN associated with the target region. Once selected, the NN is loaded onto the accelerator to process the state and generate control actions. When the pose lies at the boundary between partitions, either NN can be used since success rates are equivalent under uniform partitioning.

TABLE I: Hyperparameters for PPO.

Hyperparameter	Value	Hyperparameter	Value
Learning rate	3×10^{-4}	Batch size	30,720
Num. steps	15	Discount factor	0.8
GAE lambda	0.9	Clip range	0.1
VF coefficient	0.5	Entropy coefficient	0.0
Max. gradient norm	0.5		

IV. EXPERIMENT RESULTS

A. Experiment Setup

STAR is evaluated on two robotic platforms with different DoFs on Genesis simulator [18]: the 6-DoF Delta DRV90L [19] and the 7-DoF Franka Emika Panda [20]. Their physical constraints reported by the manufacturers are used to configure the spherical task space and ensure the actions outputted by NNs are within the joint limits.

Training is performed using an NVIDIA GeForce RTX 3090 GPU [30]. During training, each task samples initial and target end-effector poses from a given task space partition. A task is considered successful if the end-effector reaches within a threshold distance of $\tau \leq 5\text{mm}$ from the target pose. Both the actor and critic NNs in STAR’s PPO-based DRL agent are implemented as fully connected networks with four hidden layers. Each layer contains K units, followed by tanh activations. The input and output dimensions are determined by the robot’s DoFs (15 inputs and 6 outputs for DRV90L; 16 and 7 for Panda). To satisfy memory constraints, K is selected such that the total NN size fits within a predefined budget. Training is performed using the Adam optimizer. The PPO hyperparameters are listed in Table I.

We evaluate STAR on an NN accelerator equipped with 16 Kb on-chip SRAM memory that can perform both multiply-accumulate (MAC) operations and analog-to-digital (A/D) conversion [31]. The NN loading time is estimated from the write speed of the Serial Peripheral Interface (SPI) [32], assuming a conservative data transfer latency of 25 ns. The inference latency is derived from the NN architecture and the latency of MAC operations. For the activation function, we adopt a simulated latency of 2 ns, following the hardware implementation reported in [33].

B. Performance Evaluation

To verify the effectiveness of STAR, we evaluate it on two robotic manipulators with differing kinematic complexities, i.e., DRV90L (6-DoF) and Panda (7-DoF). We compare STAR against two baseline approaches. The first is *Memory-rich*, a large NN trained to cover the entire task space without considering memory constraints. This setup represents the state-of-the-art related works [21]–[27]. The second is *Heuristic*, a memory-aware method extended based on related works. It uses multiple small NNs to reduce memory usage, but relies on relative mappings between initial poses and joint angle changes. As a result, it requires frequent NN loading to complete a task. This behavior is illustrated in the motivating example in Fig. 2.

Fig. 7 presents the execution time and energy consumption for 20 manipulation tasks on two robotic manipulators

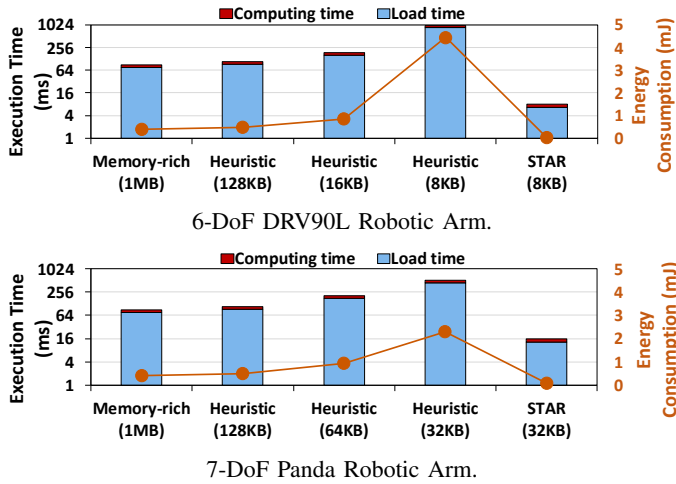


Fig. 7: Execution time and energy consumption comparison for executing 20 tasks across two robot platforms.

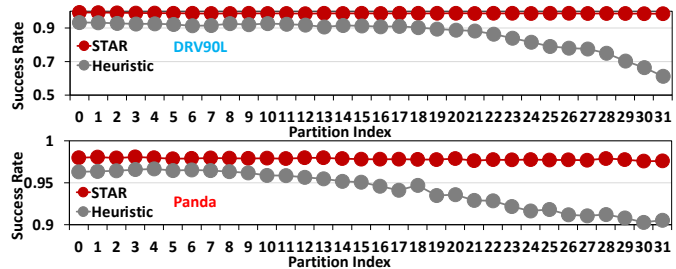


Fig. 8: Success rate comparison of STAR and Heuristic on DRV90L and Panda when crossing task space partitions.

(DRV90L and Panda). STAR outperforms both baselines. Compared to the Memory-rich baseline, STAR achieves $8.09\times$ faster execution, $10.93\times$ lower energy consumption, and $128\times$ lower runtime memory. Compared to the Heuristic baseline, STAR is inherently more scalable with respect to the number of partitions. For example, in the DRV90L case, as the number of partitions increases from 8 to 512, the NN size of the Heuristic baseline decreases from 128 KB to 8 KB; however, execution time and energy consumption increase due to multiple NN loadings required when start and target end-effector poses lie in different partitions. In contrast, STAR’s absolute pose-to-joint learning avoids NN switching during execution, while its memory-aware task space partitioning ensures each NN only learns a local absolute mapping, thereby minimizing runtime memory usage.

STAR’s absolute relationship effectiveness. We test its robustness to varying initial end-effector poses. The task space is divided into 32 partitions. In partition 0, we train NNs with both absolute and conventional relative mappings. Tasks are generated by sampling initial poses from all partitions and target poses within partition 0. As shown in Fig. 8, relative mappings show a clear decline in success rate as the initial partition moves farther from the trained region, while absolute mappings remain consistently high. This demonstrates that STAR eliminates dependence on the initial pose and ensures reliable manipulation across the task space.

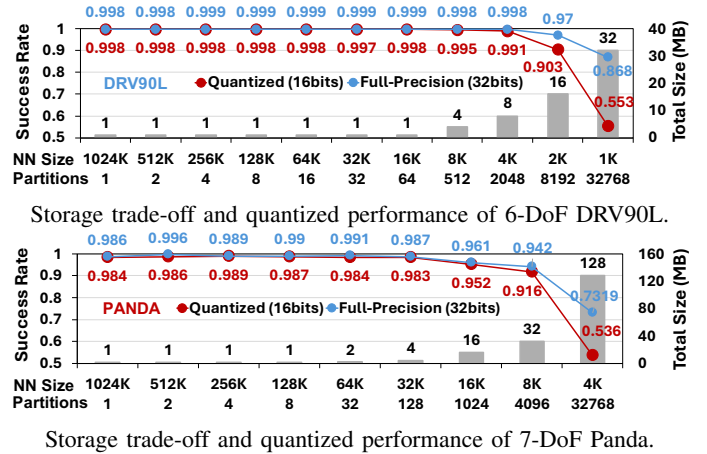


Fig. 9: The storage trade-off and quantized performance of STAR on two robot platforms.

STAR’s storage trade-off. We evaluate the storage trade-off and scalability limits of STAR on DRV90L and Panda. Starting from a 1 MB NN that covers the entire task space, we iteratively halve the NN size and determine the partitions needed to reach comparable success rates. Since lightweight accelerators often lack full-precision support, we also report 16-bit fixed-point results supported by [31]. As shown in Figure 9, total storage stays at 1 MB until 16 KB (DRV90L) and 128 KB (Panda), after which more partitions are needed to maintain success rate, increasing the total. STAR’s practical limit occurs at 8 KB for DRV90L (0.995 success, 512 partitions) and 32 KB for Panda (0.983 success, 128 partitions), where further partitioning no longer helps. These results highlight the trade-off between storage efficiency and accuracy under extreme memory constraints. Nevertheless, STAR reduces runtime memory by up to $128\times$ for DRV90L and $32\times$ for Panda, enabling accurate, low-cost manipulation while preserving the speed and energy benefits discussed in Section IV-B.

V. CONCLUSION

This paper proposes STAR, a novel framework for low-latency, energy-efficient and accurate robotic manipulation on memory-constrained neural network (NN) accelerators. STAR includes two key innovations: a spherical task space approximation that supports efficient and scalable partitioning of the robotic manipulator’s reachable task space, and a memory-aware, DRL-based training strategy that learns absolute pose-to-joint mappings for each partition. Experimental results show that compared to state-of-the-art works, STAR achieves up to $8.09\times$ faster execution and $10.93\times$ lower energy consumption, while reducing the SRAM usage by up to $128\times$ without compromising manipulation accuracy.

ACKNOWLEDGMENT

This work was supported in part by the National Science and Technology Council under grant nos. 114-2927-I-002-525, 114-2927-I-002-532, 114-2223-E-002-011, 114-2221-E-002-219-MY3, 114-2221-E-002-222-MY3, and by Ministry of Education under grant no. 114L900903.

REFERENCES

- [1] G.-Z. Yang, J. Bellingham, P. E. Dupont, P. Fischer, L. Floridi, R. Full, N. Jacobstein, V. Kumar, M. McNutt, R. Merrifield *et al.*, “The grand challenges of science robotics,” *Science robotics*, vol. 3, no. 14, p. eaar7650, 2018.
- [2] R. Goel and P. Gupta, “Robotics and industry 4.0,” *A roadmap to industry 4.0: Smart production, sharp business and sustainable development*, pp. 157–169, 2020.
- [3] F. Rubio, F. Valero, and C. Llopis-Albert, “A review of mobile robots: Concepts, methods, theoretical framework, and applications,” *International Journal of Advanced Robotic Systems*, vol. 16, no. 2, p. 1729881419839596, 2019.
- [4] T. B. Sheridan, “Human–robot interaction: status and challenges,” *Human factors*, vol. 58, no. 4, pp. 525–532, 2016.
- [5] S. Kucuk and Z. Bingul, *Robot kinematics: Forward and inverse kinematics*. INTECH Open Access Publisher London, UK, 2006.
- [6] J. Q. Gan, E. Oyama, E. M. Rosales, and H. Hu, “A complete analytical solution to the inverse kinematics of the pioneer 2 robotic arm,” *Robotica*, vol. 23, no. 1, pp. 123–129, 2005.
- [7] J. Angeles, “On the numerical solution of the inverse kinematic problem,” *The International Journal of Robotics Research*, vol. 4, no. 2, pp. 21–37, 1985.
- [8] S. R. Buss, “Introduction to inverse kinematics with jacobian transpose,” *Pseudoinverse and Damped Least Squares methods*, p. 19, 2004.
- [9] A. Aristidou, J. Lasenby, Y. Chrysanthou, and A. Shamir, “Inverse kinematics techniques in computer graphics: A survey,” in *Computer graphics forum*, vol. 37, no. 6. Wiley Online Library, 2018, pp. 35–58.
- [10] I. Tesla, “Tesla ai day 2022 – optimus: Tesla bot reveal,” 2022, available at <https://www.tesla.com/AI>.
- [11] A. R. Almusawi, L. C. Dülger, and S. Kapucu, “A new artificial neural network approach in solving inverse kinematics of robotic arm (denso vp6242),” *Computational intelligence and neuroscience*, vol. 2016, no. 1, p. 5720163, 2016.
- [12] R. Bensadoun, S. Gur, N. Blau, and L. Wolf, “Neural inverse kinematic,” in *International Conference on Machine Learning*. PMLR, 2022, pp. 1787–1797.
- [13] C.-K. Ho and C.-T. King, “Automating the learning of inverse kinematics for robotic arms with redundant dofs,” *arXiv preprint arXiv:2202.07869*, 2022.
- [14] Integrated Silicon Solution, Inc. (ISSI), “61C64AL: 8K x 8 High-Speed CMOS Static RAM Datasheet,” <https://www.mouser.tw/datasheet/2/198/61C64AL-258428.pdf>, 2020, accessed: Apr. 20, 2025.
- [15] —, “61/64WV10248EDBLL: 128K x 8 CMOS SRAM Datasheet,” https://www.mouser.tw/datasheet/2/198/61_64WV10248EDBLL-276598.pdf, 2020, accessed: Apr. 20, 2025.
- [16] G. Indumathi and V. A. alias Ananthakirupa, “Energy optimization techniques on sram: A survey,” in *2014 International Conference on Communication and Network Technologies*. IEEE, 2014, pp. 216–221.
- [17] C.-J. Jhang, C.-X. Xue, J.-M. Hung, F.-C. Chang, and M.-F. Chang, “Challenges and trends of sram-based computing-in-memory for ai edge devices,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 5, pp. 1773–1786, 2021.
- [18] G. Authors, “Genesis: A universal and generative physics engine for robotics and beyond,” December 2024. [Online]. Available: <https://github.com/Genesis-Embodied-AI/Genesis>
- [19] Delta Electronics, Inc. (2024) Articulated robot. Accessed: 2025-04-15. [Online]. Available: <https://www.deltaww.com/en-US/products/Articulated-Robot/3816>
- [20] Franka Emika GmbH. (2018) Panda robot arm datasheet. Accessed: 2025-04-15. [Online]. Available: <https://www.chenlux.com/wp-content/uploads/2018-05-datasheet-panda.pdf>
- [21] C. Zhao, Y. Wei, J. Xiao, Y. Sun, D. Zhang, Q. Guo, and J. Yang, “Inverse kinematics solution and control method of 6-degree-of-freedom manipulator based on deep reinforcement learning,” *Scientific Reports*, vol. 14, no. 1, p. 12467, 2024.
- [22] S. Luo, H. Kasaei, and L. Schomaker, “Accelerating reinforcement learning for reaching using continuous curriculum learning,” in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–8.
- [23] A. Perrusquía, W. Yu, and X. Li, “Multi-agent reinforcement learning for redundant robot control in task-space,” *International Journal of Machine Learning and Cybernetics*, vol. 12, pp. 231–241, 2021.
- [24] D. F. Leguizamo, H.-J. Yang, X. Y. Lee, and S. Sarkar, “Deep reinforcement learning for robotic control with multi-fidelity models,” *IFAC PapersOnLine*, vol. 55, no. 37, pp. 193–198, 2022.
- [25] S. James and E. Johns, “3d simulation for robot arm control with deep q-learning,” *arXiv preprint arXiv:1609.03759*, 2016.
- [26] M. S. Nazeer, C. Laschi, and E. Falotico, “RI-based adaptive controller for high precision reaching in a soft robot arm,” *IEEE Transactions on Robotics*, 2024.
- [27] A. Franceschetti, E. Tosello, N. Castaman, and S. Ghidoni, “Robotic arm control and task training through deep reinforcement learning,” in *International Conference on Intelligent Autonomous Systems*. Springer, 2021, pp. 532–550.
- [28] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, “A survey of actor-critic reinforcement learning: Standard and natural policy gradients,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.
- [29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [30] NVIDIA Corporation, “Nvidia ampere ga102 gpu architecture,” NVIDIA Corporation, Whitepaper v2.1, 2020, accessed: Apr. 20, 2025. [Online]. Available: <https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.1.pdf>
- [31] C.-Y. Yao, T.-Y. Wu, H.-C. Liang, Y.-K. Chen, and T.-T. Liu, “A fully bit-flexible computation in memory macro using multi-functional computing bit cell and embedded input sparsity sensing,” *IEEE Journal of Solid-State Circuits*, vol. 58, no. 5, pp. 1487–1495, 2023.
- [32] Texas Instruments, *TMS320C6678 Multicore Fixed and Floating-Point Digital Signal Processor (Rev. A)*, Texas Instruments, October 2011, user Guide. [Online]. Available: <https://www.ti.com/lit/ug/sprugp2a/sprugp2a.pdf>
- [33] F. Liu, B. Zhang, G. Chen, G. Gong, H. Lu, and W. Li, “A novel configurable high-precision and low-cost circuit design of sigmoid and tanh activation function,” in *2021 IEEE International Conference on Integrated Circuits, Technologies and Applications (ICTA)*, 2021, pp. 222–223.