Leveraging OS-Level Primitives for Robotic Action Management

Wenxin Zheng
Shanghai Jiao Tong University
Shanghai, China
wxzheng98@gmail.com

Boyang Li
Southern University of Science and Technology
Shenzhen, China
12111014@mail.sustech.edu.cn

Bin Xu
Shanghai Jiao Tong University
Shanghai, China
levi.xu.bin@gmail.com

Erhu Feng
Shanghai Jiao Tong University
Shanghai, China
fengerhu1@sjtu.edu.cn

Jinyu Gu
Shanghai Jiao Tong University
Shanghai, China
gujinyu@sjtu.edu.cn

Haibo Chen
Shanghai Jiao Tong University
Shanghai, China
haibochen@sjtu.edu.cn

Abstract

End-to-end imitation learning frameworks (e.g., VLA) are increasingly prominent in robotics, as they enable rapid task transfer by learning directly from perception to control, eliminating the need for complex hand-crafted features. However, even when employing SOTA VLA-based models, they still exhibit limited generalization capabilities and suboptimal action efficiency, due to the constraints imposed by insufficient robotic training datasets. In addition to addressing this problem using model-based approaches, we observe that robotic action slices, which consist of contiguous action steps, exhibit strong analogies to the time slices of threads in traditional operating systems. This insight presents a novel opportunity to tackle the problem at the system level.

In this paper, we propose AMS, a robot action management system enhanced with OS-level primitives like *exception*, *context switch* and *record-and-replay*, that improves both execution efficiency and success rates of robotic tasks. AMS first introduces action exception, which facilitates the immediate interruption of robotic actions to prevent error propagation. Secondly, AMS proposes action context, which eliminates redundant computations for VLA-based models, thereby accelerating execution efficiency in robotic actions. Finally, AMS leverages action replay to facilitate repetitive or similar robotic tasks without the need for re-training efforts. We implement AMS in both an emulated environment and on a real robot platform. The evaluation results demonstrate that AMS significantly enhances the model's generalization ability and action efficiency, achieving task success rate improvements ranging from 7× to 24× and saving end-to-end execution time ranging from 29% to 74% compared to existing robotic system without AMS support.

1 Introduction

End-to-end policy learning framework are gaining increasing attention in the field of robotics. Researchers believe that this robotic computing paradigm with imitation learning [3, 13, 17, 60] is the future for rapid task transfer because these models can learn directly from perception to control, eliminating the complex hand-crafted feature extraction and modular design of traditional methods. Through training on large-scale datasets, end-to-end models are expected to autonomously learn to adapt in complex and changing environments, thereby exhibiting greater flexibility and efficiency in perception, decision-making, and action execution.

However, current robotic models still exhibit limitations in their generalization capabilities [56, 66] and action performance. For instance, in a typical robotic task such as pick and place, as illustrated in Figure 1, the success rate declines dramatically once the number of objects exceeds the maximum encountered in the training set, even when utilizing SOTA robotic models [3, 7, 20, 22, 49]. This is primarily due to the limited availability of training data for real robotic platforms, which often leads to model overfitting and consequently results in insufficient generalization capabilities, even for tasks that share similar characteristics. Furthermore, considering the action efficiency, the robot's actions are typically slower than those of human beings. This is primarily due to the limited actions per second (APS) during model inference, as well as the interference of generated actions across different objects, which results in non-optimal trajectory paths.

To address this limitation, the robotics research community focuses on improving the model's generalization ability from an algorithmic perspective. This includes introducing more environmental information [7, 20, 22, 49], using LLM ability [24], optimizing model parameters [29, 41, 46, 65, 69, 72], and designing entirely new network architectures to enhance the model's adaptability in unseen situations [3, 8, 19, 49, 56]. Unfortunately, these methods still face limitations in practical applications, such as the scarcity of training data for real robotic platforms and the retraining overhead associated with different tasks and robotic systems.

After a thorough analysis of the current robotic workflow, we identify a new opportunity to enhance robotic robustness and efficiency from the OS level. Although the tasks executed in robotic systems differ from those of traditional

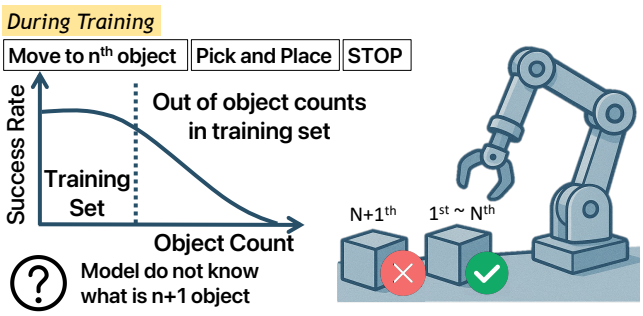

Figure 1. A failure case of the robot's pick-and-place operation. When the number of objects to be picked is not represented in the training set, the action's success rate results in a significant decrease.

applications, we can still discern similarities between these two workloads and leverage the primitives offered by classical operating systems. For example, to ensure the coherence of robotic actions, the current robotic model outputs a fixed number of action steps during each inference. We refer to this sequence of action steps as an "action slice". The action slice is similar to the time slice allocated for application threads in traditional operating systems, yet it lacks certain system capabilities. For example, in the current robotic workload, the action slice cannot be interrupted even when a fault situation arises, leading to a significant degradation in the success rate of robotic tasks. Furthermore, after executing one action slice, all context (e.g., intermediate result for robotic models) are discarded, and the robot generates the next action slice from an initialized state, which limits the performance of robotic actions.

Inspired by traditional operating system primitives such as context switching, exception handling, and record-and-replay mechanisms, AMS introduces **action context**, **action exception**, and **action replay** for robotic fault detection, context saving and restoration across different action slices, and replaying robotic actions within similar environments. With these OS primitives, the efficiency and success rate of robotic actions are significantly enhanced without the necessity of retraining the model. Specifically, AMS proposes three key technical designs:

Firstly, AMS stores and manages all action context within a context pool. The action context encompasses all intermediate inference states, such as KV caches, latent vectors, and output embeddings for robotic models. Unlike traditional robotic systems, where each action is generated from a clean initial state, AMS reuses the context generated from the previous round of inference. This approach significantly reduces redundant computations, accelerates model inference, and decreases the number of actions steps. To mitigate storage overhead, AMS employs hierarchical and differential storage strategies, retaining only the essential and distinctive components closer to the computing resources while evicting/offloading redundant or insignificant action context.

Secondly, AMS implements GPU-Inference with CPU-Action-Fault-Detection to support action exception. It contains multiple rule-based functions that run on the CPU side to detect any hardware/software-defined robotic exceptions, and interrupt the execution of robotic action immediately. By preventing the propagation of faults, robots are more likely to fix erroneous actions with the dedicated exception handler.

Thirdly, to enhance the generalization capability in a typical robotic scenario involving long-horizon tasks that include multiple repetitive jobs, AMS proposes action replay, which leverages previous successful contexts to replay robot actions in similar environments. However, unlike the fixed record-and-replay processes in traditional operating systems, in robotic scenarios, AMS can dynamically regenerate the replay actions based on environmental changes.

Our prototype of AMS is implemented as an OS service with several extensive modules, including context manager and exception tables, etc. We effectively manage context information by creating context pools on both the GPU and CPU, along with the hashed action index, virtual actions, and an automatic action eviction mechanism. Regarding model selection, we utilize a SOTA VLA-based robotic model π_0 [2] without requiring retraining efforts for tasks. We believe that the AMS approach is sufficiently general to be adaptable to various models and robotic configurations.

We evaluate the AMS on a real robot platform and an emulated environment. Our test results show that AMS can effectively improve the model's generalization ability, and reduce the time needed for the model to perform whole tasks, both achieving better and faster performance over time. In real robot tests, the success rate of long-horizon task inference after activating AMS is 7× to 24× higher than that of direct model inference. Regarding the execution time of the whole action, after activating AMS, the number of steps required by the robot decreased by 29% to 74.4% compared to direct model inference. Compared to the time required for the first inference, AMS can reduce the time of subsequent inferences by 5.7% to 20%.

2 Background and Motivation

2.1 Large Models in Robotics

As the size and capabilities of models continue to scale up, end-to-end learning frameworks of large models becomes more and more important in robotic manipulations, especially for current Vision-Language-Action (VLA) Models [1, 2, 6, 23, 35, 38, 58, 64, 67, 70, 73]. These models generally consist of three key components: natural language processing, environment perception, and action planning. Typically, LLMs are used for natural language processing,

multi-modal large models are used as environmental perception components and diffusion model is used for real action planning. Different models have different inference frequency, as shown in Figure 2.

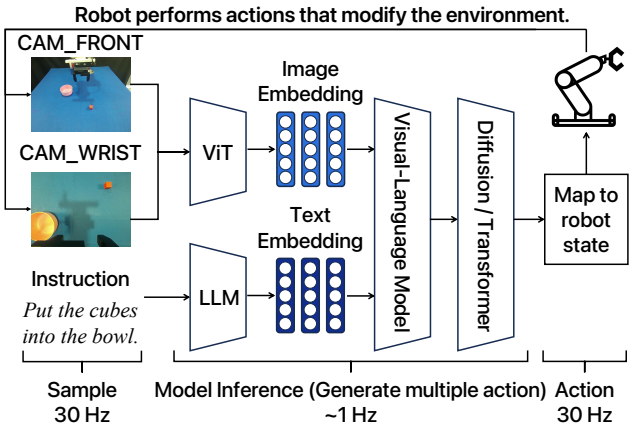

Figure 2. State-of-the-art VLA model structure. It usually combines three models: visual encoders, large language model, and action decoders (e.g., diffusion model).

The natural language processing component is responsible for interpreting the user's natural language commands and converting them to text vectors. The environment perception component gathers information about the surroundings. It collects 2D images or 3D point cloud data with depth information by the robot's sensors, such as cameras and LiDAR. These 2D or 3D images are then transformed into image vectors using Transformer-based models like Vision-in-Transformer (ViT) [11, 48]. Building on this, the action planning component utilizes text and image vectors to generate a series of motion commands suitable for the robot and an optional stop flag. The action planning components are mainly diffusion models [2, 4, 8, 25, 31, 44, 57] or auto-regressive models [9, 18, 32, 35, 45, 61, 68, 71].

Different robotic arms have different degrees of freedom, so the models cannot be directly shared between different robotic arms. The model parameters from the large-scale pre-trained model needs to be fine-tuned on each robotic arm individually to adapt it to each robotic arm.

2.2 Current Limitation in VLA-based Robotic Models

2.2.1 Generalization Problems. Currently, the generalization ability of robotic models relies mainly on fitting limited data during training. They adjust their parameters by learning patterns and features from this training data. However, unlike large language models, robots operate in complex and ever-changing environments. Gathering large and diverse datasets of robot actions is very difficult and requires huge human efforts, which limits the diversity and scale of their training data. This lack of data hinders the development of robotic models, preventing them from showing new abilities in unfamiliar situations like large language models can [15].

Due to these training data limitations, robotic models perform poorly when asked to do actions beyond what they've seen before. For example, a long-horizon task involving multiple repetitive or similar jobs is a typical scenario in robotics, such as the pick-and-place operation of multiple objects. The π_0 models [2] typically don't exceed the maximum number of repetitions they encountered during training. We tested different models with various repetition counts in their training data. The results in Figure 3 showed that models had high success rates up to the maximum repetition seen during training, but their accuracy dropped significantly to zero beyond that point.

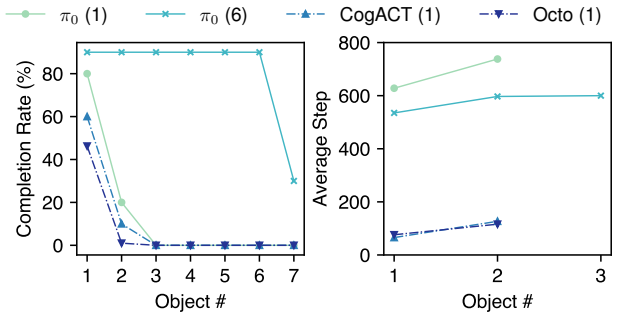

Figure 3. Degrading of different models. The number in the brackets means the maximum repetition counts in the training set.

Key Challenges: To enhance the model's generalization ability, our work mainly focuses on classical robotic scenarios, specifically long-horizon tasks that involve multiple repetitive jobs. We first adopt a straw-man design: resetting the model's internal state and regenerating robotic actions. Figure 4 illustrates a classical robotic task of "picking up all small blocks and placing them into a bow" ("pick and place"). In its training set, the robot learns only to place one block into the bowl and has never encountered a situation involving two blocks. Employing the straw-man method fails to yield even a single successful completion. This failure occurs because the environmental state after placing one block leads the model to mistakenly believe that it has completed the entire task, thereby preventing the robot from executing subsequent actions. Consequently, the key challenge for model generalization in these typical robotic tasks is **how to enable the model to perform repetitive and similar jobs within a long-horizon task without the need for retraining efforts.**

2.2.2 Efficiency Problems. Efficiency is also an important part of a model's performance. The end-to-end time relates to actions per second (APS) and the total number

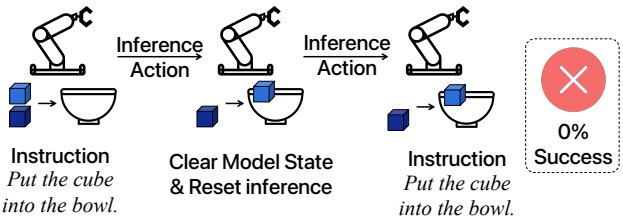

Figure 4. Strawman design of resetting the model inference.

of action steps. The APS is influenced by two factors: the speed of model inference and the performance of the robotic hardware. The minimum value among these two factors determines the actual APS value observed in the test. As for total number of steps, it depends on the results of each inference. Humans learn from repeating tasks, dropping unnecessary actions and correcting mistakes to become more skilled over time. However, robots cannot learn from past tasks. For instance, when the model has already kicked a ball and its position is very close to the shortest path, the model still cannot find that optimal path to kick the ball. They rely heavily on the training data, which can lead to performing tasks slower over time, as shown in Figure 3.

Key Challenges: Previous work [63] has attempted to enhance the efficiency of robot execution by accelerating model inference. However, in real robotic scenarios, hardware performance often constitutes the primary bottleneck, resulting in only minimal improvements from optimizing model inference speed. Consequently, the primary challenge for enhancing robotic efficiency lies in how to enable the model to identify the optimal action path, thereby reducing the total number of action steps.

3 AMS Designs

3.1 Key Insight

To enhance the efficiency and success rate of robotic actions, the core of AMS is to **migrate OS-level primitives into robotic action management**. Similar to the time slice abstraction of threads in traditional operating systems, the action slice in the robotic system, which comprises several contiguous action steps, serves as the minimal and atomic execution unit. However, in traditional operating systems, when a program encounters an exception or fault during execution, the kernel interrupts the target program to handle this exception immediately. Moreover, the kernel also manages the context for each application thread to prevent the need for restarting the application from scratch after scheduling. Inspired by these, AMS introduces a comparable mechanism for robotic model inference. *First*, we propose action context, which stores the model execution state for each action (§3.2). To mitigate storage overhead, we construct a context pool along with a dedicated eviction strategy. *Second*,

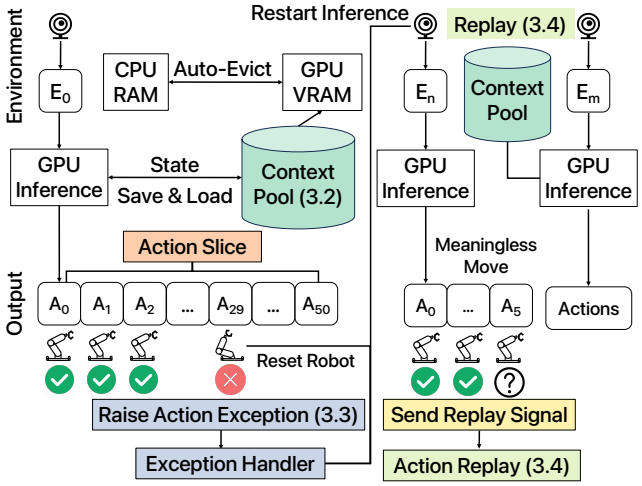

Figure 5. The overall design of AMS enabled inference process for robotic models.

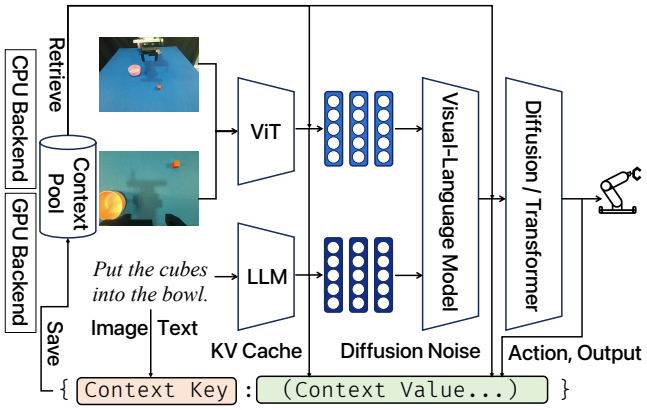

Figure 6. The structure of action context for multi-layer storage and retrieval.

we introduce the concept of action exception for robotic actions (§3.3). It can promptly interrupt the current action slice when any software- or hardware-defined exception is triggered, analogous to the kernel's exception handling. *Third*, to manage repetitive tasks, AMS employs the action replay mechanism (§3.4), which allows the replay of robotic actions in a similar, albeit not identical, environment using the prior action context. The whole process is shown in Figure 5.

3.2 Action Context

The action context is used to represent a complete action for one task. It is a layered storage module that not only stores memories of action execution for repeatability but also speeds up subsequent reasoning processes. The structure of action context is shown in Figure 6.

3.2.1 Constructing the Action Context. To efficiently reuse information from each inference, AMS stores all cacheable objects in the action context pool. This includes intermediate

states during inference and the action trajectories produced by inference. Specifically, the intermediate states consist of the KV Cache from the Transformer model in visual recognition, latent vector generated at each time step by the Diffusion model for joint action generation, and the KV Cache from the Transformer model in the language understanding module. These cached intermediate states can be reused in future inference, thus reducing computational overhead and speeding up the time required for a single inference. The inference output includes data from the robot's joints and the output embedding obtained from the final inference. These action trajectories can guide the model to repeat previously executed actions in future tasks, enhancing planning efficiency and action consistency.

During each inference, the system first checks the action context pool to see if any similar and reusable intermediate states exist by comparing the similarity of the instruction and images. If a reusable intermediate state like KV cache is found, the system directly retrieves and uses these states to avoid redundant computations. If similar task trajectories are found, the system uses these previously inference results to guide the model by reusing latent vectors of successful action as the start state. This encourages it to mimic these trajectories in current inference. These approaches reuse previously cache and provide guidance for task completion, both increasing success rate and efficiency. After one single inference, AMS adds the unseen vector including new KV value and latent vector into the context pool for further use.

3.2.2 Layered Storage of Action Context. Saving all cacheable items would consume vast amounts of GPU memory, which is unacceptable for robotic applications that require real-time processing. However, the frequency of data usage are not uniform. As shown in Table 1, the KV cache from the environment recognition vision transformer and LLM KV cache occupies almost one-third of data but is accessed less frequently than the diffusion noise and output embeddings. Therefore, AMS manages data based on their access characteristics by prioritizing storage resources for caches that are frequently used but small in size, while applying optimized storage strategies for caches that are infrequently used but large in size.

Table 1. Storage occupation for different cacheable items for one single action step.

Cacheable Items	π_0	CogACT	Use Frequency
ViT KV Cache	63MB	51MB	Low
LLM KV Cache	165MB	134MB	Medium
Diffusion Noise	165MB	301MB	High
Output Action	14Byte	14Byte	High

Specifically, AMS adopts a hierarchical storage architecture to manage the action context. First, we determine the storage location of data based on its use frequency. For data that is frequently accessed, we store it in GPU memory to minimize data access latency and improve system performance. Conversely, for data that is infrequently accessed, we apply an on-demand recomputation strategy or move it to CPU. Additionally, we do not place data that does not participate in computation near the computing components.

3.2.3 Action Compression. To minimize the storage space occupied by the action context, AMS employs a compression algorithm to reduce the size of the action context. We observed that a single task often includes many repeated actions across different inference instances. For example, the paths for picking up objects might be very similar when the arm is exactly on top of the objects. Thus, we first build the action index for quickly retrieve the action and its corresponding vector and then introduce a "virtual action", which links repeated data with only one copy to reduce redundant storage. Then we perform auto eviction policy to reduce the size of the action context pool on GPU.

Virtual Action. Different action sequences and environments often share common parts that can be reused, such as a next-step action of same task at the same location. Thus, AMS introduces a virtual action mechanism to enable cross-context action reuse, similar to the page table concept in operating systems. To achieve this, AMS first creates an action index for each action to facilitate action indexing. Once AMS collects an input, it serializes that input into a binary form in the background and calculates hash values for each segment of that binary data. In the first phase of hashing, AMS independently calculates a hash value for each segment using parallelism. In the second phase, it merges all the segments' hash values to generate the final hash result, serving as the unique index for action lookup and management. The process is listed in Algorithm 1. To minimize the complexity, the hash functions involved are modular calculation. Since this process is carried out asynchronously, it does not interference the normal operation of foreground tasks, thereby enabling fast action search and reuse without adding extra overhead.

Based on the index, AMS introduces "virtual action" mechanism that minimizes repeated storage by treating the context as a series of uniquely numbered actions with independent reference counts. These actions are stored in a contiguous area. Each context only keeps track of the actions it needs by referencing their IDs. When a new context adds an action, that action's reference count increases. When a context is evicted or deleted, its referenced actions each have their reference count decreased. Any action whose reference count drops to zero is removed. This design eliminates unneeded storage overhead and boosts the AMS's overall reuse efficiency.

Algorithm 1 Two-Phase Hashing for Action Index

Require:
- Binary input data B of length N.
- Segment size s (number of bytes per segment).
- Hash functions *HashFunc1* and *HashFunc2*.

1: **Phase 1: Segment-level Hashing**

1. Partition B into $k = \lceil \frac{N}{s} \rceil$ segments: $B_1, B_2, \ldots, B_k$, where each B_i has size s bytes (except possibly the last one, if N is not divisible by s).
2. **for** $i = 1$ to k **do**
   a. $h_i \leftarrow HashFunc1(B_i)$
   b. Store h_i in a list $H = [h_1, h_2, \ldots, h_k]$

2: **Phase 2: Aggregation**

1. Combine all segment-level hashes from H using *HashFunc2*:

$$H_{\text{final}} \leftarrow HashFunc2(h_1, h_2, \ldots, h_k)$$

2. **return** H_{final}

Auto Eviction. As stated in §3.2.2, different types of data occupy memory in different amount and exhibit different access frequency. To ensure transparent management of GPU memory and CPU memory resources, AMS employs a hybrid cache management strategy that combines least recently used (LRU) with a priority-based approach, using dynamic cache eviction to improve resource utilization. Specifically, AMS keeps a access counter for each vector, storing all vector on the GPU when sufficient space is available. If GPU memory becomes scarce, AMS automatically initiates auto-eviction, giving priority to objects with the smallest effect on outcomes or the lowest computational cost, and moving the least-used portion of data back to CPU memory. For example, the KV Cache in vision models can be easily recalculated. The extra cost is less than 10%. When GPU resources are low, AMS will first evict the KV Cache of vision models. By balancing data distribution across different tiers and optimizing the eviction policy, AMS boosts memory efficiency and overall performance without sacrificing the response speed and accuracy for robots.

3.3 Action Exception

3.3.1 Exception. During the execution of real robots, errors may occur. Subsequent inference steps of the model can only correct errors produced by the algorithm in the model's output after the current slice has finished executing. However, some errors that cannot be corrected or addressed in a timely manner through model inference. We refer to these errors as **exceptions** and categorize them into two types: hardware exceptions and software exceptions. These two types of exceptions are shown in Table 2.

Hardware Exception. The hardware exception is defined as the robot's abnormal mechanical state. The model is unaware of these abnormal mechanical state because it did not take such abnormal state as input to model inference. To address this, AMS monitors the robot's state at runtime, converting the robot's corresponding abnormal state into a hardware exception and raising it to the model to immediately interrupt the execution. Specifically, when AMS detects that the physical hardware returns an abnormal state, it immediately interrupts the remaining actions in the action slice that have not been executed, stopping the execution of the current action slice.

Table 2. Exception types defined in AMS.

#	Type	Exception
1	Hardware	Collision.
2		Unreachable robot state.
3		Robot crash.
4		Too big torque.
5		Too big angular momentum.
6	Software	Not expected action.
7		Software defined condition violation.

Software Exception. Software Exception means the unexpected state in the robot's execution defined by software. It has larger scope than hardware exception. For example, when controlling the gripper for grasping, the gripper may occasionally close without actually gripping anything, resulting in a failed grasp. Although in a complete action slice, a single failed action may not affect the normal execution of subsequent actions, continuing to operate after an uncorrected software exception can further cause hardware exceptions that are difficult to handle.

Table 3. Example check of software defined condition.

Action	Expected Outcomes
Gripper grasping	Gripper's minimum gap exceeds 0.
Move stick	Change in the stick's angle.
Move manipulator	Change in the manipulator's angle.
Apply force	Force sensor exceed 0.
Send stop	All actuators halt.

Following the principle that each action should produce an observable effect on the environment, AMS defines a set of expected outcomes for each action. For example, when the gripper is grasping, the expected outcome is that the gripper's minimum gap exceeds 0. Such expected outcomes can be rule-based or defined from the LLMs. Examples of the expected outcomes is shown in Table 3.

However, raising software exception is not as straightforward as hardware exception since it requires additional time on the critical path of action to determine whether the action is successful or not. If the system performs checks more

frequently, the proportion of correct actions will increase; however, this will also lead to slower operation of the robot. There is a trade-off between the checking frequency and accuracy of the action, as shown in Figure 7.

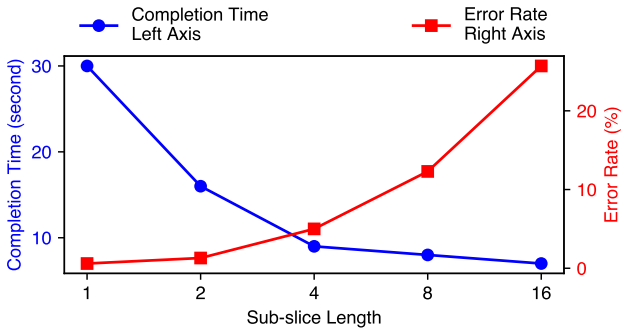

Figure 7. Trade-off between checking frequency and accuracy of the action.

To address this, AMS introduces a mechanism that divides the action into smaller sub-slices. After each sub-slice is executed, AMS asynchronously checks the environment gathered at the end of sub-slice to see whether the relevant robot joint states match the expected states. All output checks only check whether the action has taken effect, rather than determining whether the action is semantically correct or meeting its intended goal. The length of one sub-slice is not always fixed. Its length is related to the current action intent. If the robot shows a very large range of movement, AMS will reduce the length of the sub-slice. The final length of the sub-slice is determined to be between 2 to 5 steps. If a sub-slice is judged to have failed, AMS will immediately raise a software action exception to the model.

3.3.2 Handler. The exception handler of AMS is used to recover from the abnormal state of the robotic arm. Due to the different types of exceptions, AMS will take different exception handlers. Among these exception handlers, there are common parts and specific parts.

Common Part. The common part is that AMS immediately interrupts the execution of the current action slice. AMS immediately collects and updates the current environment data and sends it to the GPU to start a new inference. Meanwhile, to prevent previously erroneous actions from polluting future operations, AMS discards any sub-slices that have not yet been executed. It then leverages a newly generated inference result, using the updated environment data to generate subsequent actions. This approach allows the model to promptly correct errors and maintain stability and consistency throughout the execution process.

Specific Part. The specific part is that AMS will handle the exception differently based on the type of exception. For software exceptions, AMS just clear the current action slice and

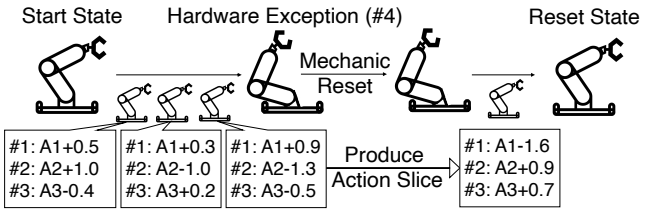

Figure 8. Robot physical state reset. A1, A2, A3 represent the different ankle of the model. When synthesizing action slices, computation is based on robot configuration and might be different from direct addition. #1, #2, #3 represent the different action atomization operations.

do not reset the robot power state. For hardware exceptions, AMS will reset the robot to a safe state first. Then, AMS will decide whether to restart the robot based on the severity of the problem and exception number. For example, if the current action causes too big torque (Exception #4) or too big angular momentum (Exception #5), AMS will not restart the robot. If the robot's mechanical arm collides (Exception #1), AMS will restart the robot.

However, resetting the robot to a safe state cannot be achieved by merely recording every movement trajectory of the robotic arm and rolling back these trajectories, as the movement paths often contain a significant amount of redundancy. AMS designs a smaller atomic action unit to represent the robot's physical state changes called **action atomization**. Different from the traditional action chunking approach, action atomization limit the action to only 1 degree of freedom (DoF) of the robotic arm and it can deconstruct continuous movements into smaller atomic action units that represent micro-angular displacements in joint space and displacement vectors in the coordinate system. During rolling back, AMS follows he concept of spatial composite vectors to combine these atomic actions into a shorter action slice. To implement this mechanism, AMS maintains a buffer on the CPU to record these actions. When the buffer is full, AMS automatically consolidates all actions into one action slice. The entire process is shown in Figure 8.

3.4 Action Replay

Considering a long-horizon task with multiple repetitive jobs, when the model continuously outputs meaningless actions, AMS will trigger an replay signal to prevent the inference from hanging and perform a replay to guide the model to infer based on previously successful experiences. Different from traditional operating systems, when meaningless actions are detected, AMS cannot directly reuse the previous trajectories to resume execution. This is because the current environment and the environment in the context are not exactly the same, which means the steps produced for the last context may not be valid anymore. To address this, AMS adopts a hybrid

approach that combines inference with replay, combining state reset and action regeneration.

3.4.1 Replay Signal. AMS introduces replay signal to indicate the robot is actually hanging and requires additional control. In the traditional operating system, SIGNAL is triggered to force the kernel to reclaim CPU control and switch to other processes/tasks. This prevents a single process from hanging the CPU and exhausting all resources. In robotic models, sometimes the model do not generate any moving action for a long time, making the arm keep idle while the task is not completed. This is similar to the CPU being occupied by a single process for a long time. Specifically, if all actions within an action slice are marked as completed, or if no actions are executed while the task remains unfinished, AMS will send a replay signal. This prevents the inference process from hanging indefinitely. Subsequently, AMS incorporates a description of the unfinished task into the prompt and utilizes the vector from the action context in the context pool as the initial vector for the immediate new inference, rather than employing a random vector.

3.4.2 Action Regeneration. AMS uses replay to stop meaningless actions and use previous successful experiences to guide further inference. However, previous experience cannot be directly applied due to the changing of the environment, as well as factors such as the position and orientation of target objects. When the environment undergoes slight changes or target object positions shift, action trajectories that were previously successful are no longer applicable.

To effectively address this issue and enhance the robustness of action replay, AMS employs a guided inference strategy for the action generation using a diffusion-based action generation model, instead of directly reusing previously successful contextual action data as direct output. Specifically, AMS uses successful action vectors validated through prior environmental interaction as the initial vector for the current action diffusion process, which is equal to performing an additional inference of diffusion steps based on the existing context. This method explicitly guides the model during the initial phase of action generation, making the generated trajectory more likely to leverage previously effective action features, thereby increasing success rates in similar environments that have undergone slight modifications. Additionally, to ensure the adjusted action sequence adapts to changes in the environment and target objects, AMS will also changes the prompt to include the current environment and task description. The added prompt will clearly direct the model to reanalyze the current environment and formulate new action slice, rather than simply reusing previous conclusions or encoded information. With this enhanced prompting mechanism, the LLM is able to proactively utilize the currently perceived state of the environment to re-engage in inference and action generation.

4 AMS Implementation

AMS is a module within the framework. It contains a static auto hooking tool and a runtime library. To maintain maximum compatibility with existing systems and to minimize interference with the inference process, AMS scans the code and inserts hook functions to locate the action slice by static analysis. For the runtime library, AMS include the context cache, the exception handler, and the replay signal trigger. We place the whole control logic of AMS on the CPU. For context manager, we prioritize putting action context on the GPU. If the GPU context cache runs out of space, AMS evicts vectors which are less frequently used and have minimal impact on inference results. For exception handler, AMS will monitor if the action responded as expected to the action sent to the robot. If not, AMS will immediately stop sending further instructions, reset the robot and re-collects environment data to initiate a new inference. For action replay signal trigger, when the model finally outputs a stop signal, AMS will first check whether a replay should be applied. If AMS requires replay, the replay signal is up. Then, AMS will block the stop signal and retrieve stored contexts from the context pool one by one to guide subsequent inferences, helping avoid workflow disruptions caused by misjudgment or intermediate errors. The whole process is shown in Figure 9.

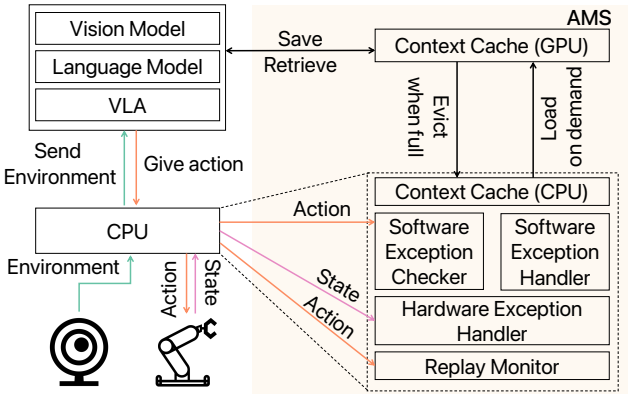

Figure 9. Implementation of AMS. The AMS system contains several key components: Action context pool manager, action exception handler and action replay signal trigger (replay monitor in the figure).

5 Evaluation

To show the performance of AMS, we conducted both real-world robotics and simulation with the models below. We seek to answer the following three questions.

- Can AMS accelerate the task completion? (§5.1)
- Can AMS improve the task success rate? (§5.2)
- How does the AMS perform in comparison to traditional algorithm-based methods? (§5.4)

Setup. We selected real-world robotics (JAKA s5 [28]) and simulation robotics (WidowX [50] in SimplerEnv [37]) as our testbed. We use three SOTA models, Octo [55], CogACT [36] and π_0 [2], to evaluate the performance of AMS. To evaluate the generalization capabilities of the models, these models are not fine-tuned for our test jobs; instead, they are pre-trained to adapt to our robotic platforms. We use direct inference without AMS and VLA-Cache [63] as our baseline. We use three representative tasks from Basic Manipulations and Multiple Object Interactions of RoboMind dataset [59], which is consistent with algorithmic solution evaluation [56]. Given that there may be multiple objects on the desk, these long-horizon tasks can be structured to several repetitive jobs for different objects.

Metrics. To evaluate the performance and efficiency of AMS, we report the success rate, average finish step and end-to-end latency. Success rate refers to the ratio of the number of times a robot successfully completes a task to the total number of trials. Average completion step refers to the average number of action steps required for the robot to successfully complete a task, excluding the first execution. End-to-end latency refers to the time required for one successful execution. It depends on the time taken for each step and the number of steps required. We set a limit of 1500 steps for each task, which is twice the size needed to complete a single task.

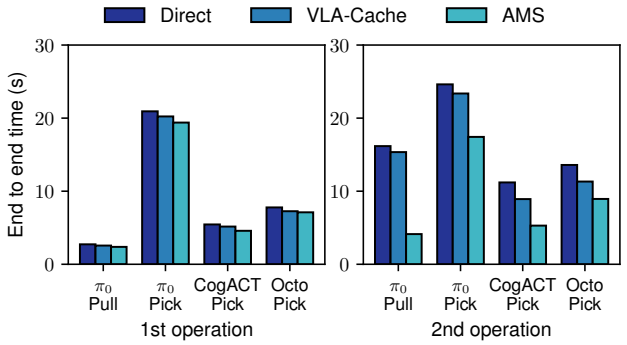

Figure 10. End-to-end efficiency evaluation for the first and second time execution.

5.1 End-to-end efficiency evaluation

We tested the AMS for end-to-end execution time in both real-world and simulation scenario. The overall end-to-end efficiency evaluation is shown in Figure 10.

The total execution time is related to the Actions Per Second (APS) metric and the overall number of action steps required to complete the task. The overall APS is defined as the minimum value between the maximum hardware APS of the robot and the APS of the model inference. In practical scenarios, the hardware APS of the robot is usually lower than that of the model in most situations. Therefore, enhancing the model's APS exerts a lesser influence on the results compared to reducing the number of action steps. To comprehensively evaluate AMS, we measured the execution time for both the first and second executions of the robotic task. In the first execution, AMS achieves a modest acceleration, reducing the execution time by 7% to 15%. This limited improvement is attributed to the fact that the cache can enhance model inference but does not decrease the total number of action steps during the first execution. For the second execution, AMS shows significant acceleration due to the reused context, reducing execution time by 29% to 74.4%. This is because the combination of action context and action interrupt, which allows for direct use of cached previous context information during inference and reduces invalid hanging steps.

To show why AMS can accelerate the execution of robotics, we breakdown the performance with the effects of the number of total steps and the actions per second of AMS.

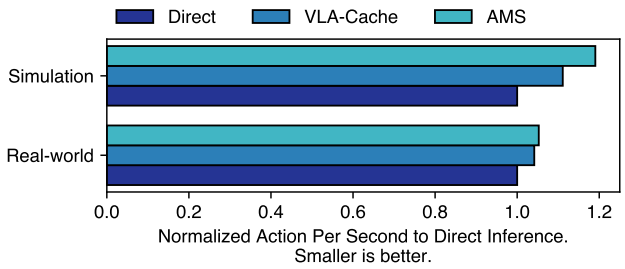

Figure 11. Actions Per Second test. Evaluate the overall APS of the robotic system with and w/o AMS support.

APS (Actions Per Second) We measured the APS to test how AMS accelerates individual step inference. The results are shown in Figure 11. In real-world environments, the inherent limitations of hardware in robotic arms restrict the overall performance. Accelerating model inference speed has a minimal impact on the actual speed of robotic actions; therefore, AMS only achieves an average improvement of 5%. In simulation environments or future robots equipped with higher hardware APS, the acceleration effect achieved by increasing the model inference APS becomes increasingly significant. AMS demonstrates a 19% improvement in single-step performance compared to the baseline model, as it effectively manages the action context, which reduces redundant computations for the KV cache and latent vectors in VLA-based robotic models.

The number of steps Another factor that influences end-to-end efficiency is the total number of action steps. We conducted steps count for both real-world and simulation case, and the result is shown in Figure 12 and 13. For the real-world case, AMS sustains excellent performance compared to direct inference. In the number of steps needed, the AMS system reduces the steps by 29% for picking the cubes and

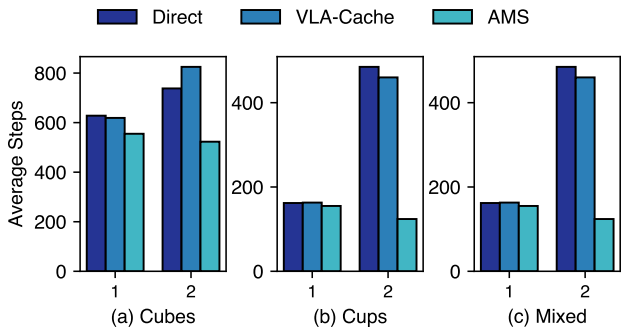

Figure 12. Execution steps of π_0 model for different tasks in real-world scenario. "Cubes" represents picking all cubes and placing into the bowl. "Cups" represents pulling down all cups. "Mixed" represents arranging the whole desk which contains cubes and cups.

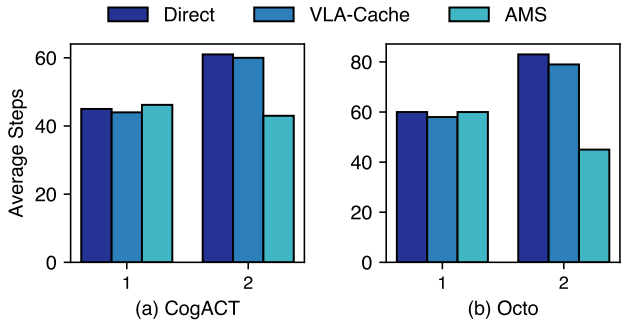

Figure 13. Execution steps in the simulation scenario with CogACT and Octo model for picking the spoons.

74.4% for pulling the cups in the second step compared to a inference without AMS. Additionally, compared to the first execution of the task using AMS, the average number of steps in the second execution decreased by 5.7% for picking up the cubes and by 20% for pulling the cups. For the simulation case, AMS also performs significant steps improvement. Compared to the baseline model using direct inference, AMS can reduce the number of steps required to complete the task by 29% to 45%, and compared to the solution using VLA-cache, AMS can reduce the number of steps by 28% to 43%. This performance improvement is attributed to the caching and reuse of the latent vector for VLA models, which enables the robot's subsequent actions to align with previous trajectories, thereby minimizing hanging and disorderly actions.

5.2 Task success rate evaluation

Another important factor affecting the model's performance is the task success rate. To evaluate the generalization capability of repetitive/similar tasks for AMS, we test the robot on tasks that are both within and outside the training dataset. In this section, the original model is trained on the manipulation of a single cup or cube, yet it is expected to handle multiple objects during operation.

In real-world scenarios, the AMS can achieve 7× in success rate for desk arranging task via picking. It can also achieve up to 24× increase in accuracy for single-arm movement job via pulling. As shown in Figure 14, for objects more than two or for different shaped objects, AMS can still improve the task success rate significantly.

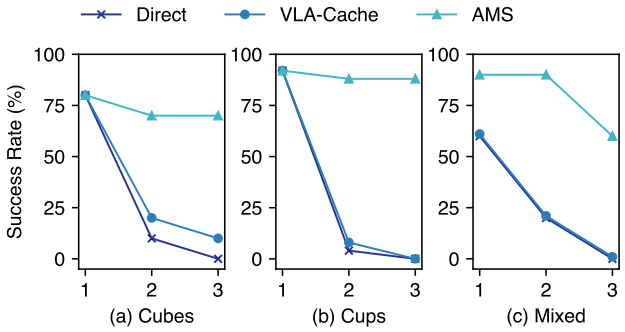

Figure 14. Success rates of π_0 model for different tasks in real-world scenario. "Cubes" represents picking all cubes and placing into the bowl. "Cups" represents pulling down all cups. "Mixed" represents arranging the whole desk which contains cubes and cups.

In simulation environment, the success rate of the task is not as high as in the real-world environment, but compared to the baseline model using direct inference, AMS still achieves a 5 to 12× improvement in accuracy, and a 2× performance improvement compared to the baseline model. The results are shown in Figure 15. We analyzed the execution process of the task and identified that this drop in success rate is primarily attributable to two factors. First, the simulation environment is unable to detect all hardware exceptions. Consequently, AMS cannot promptly trigger an interrupt for fault actions, which may cause the error propagation. Second, the limited space within the simulation environment increases the interference from other objects during model inference, in contrast to the real-world environment.

5.3 Case Study: Picking two cubes on the desk

We further conducted a detailed step-level evaluation of AMS's performance in object-picking tasks, as illustrated in Figure 16. Completing the first task, which is picking up the first small cube, the baseline model and AMS performed similarly, both successfully accomplishing the task between steps 1 and 301. At step 361, AMS detected that all actions in the current action slice were producing no valuable effects, and AMS immediately triggered an replay signal and performed a context switch. This process was immediately

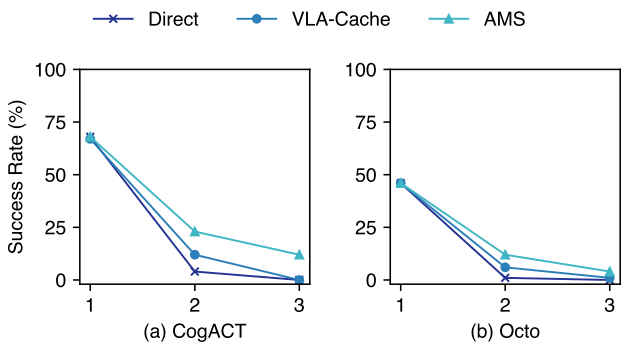

Figure 15. Success rates in simulation scenario with CogACT and Octo model for picking the spoons.

reflected at step 361 as a system reset: the robotic arm returned to its initial position. Subsequently, in the next action slice, AMS successfully reoriented towards the yellow cube using the prior action context, allowing the task to continue. By reusing the action context, AMS can pick up the second cube more efficiently, requiring fewer action steps compared to the first pickup. In contrast, the baseline model became stuck, with no changes in its actions from step 361, unable to further progress the task, ultimately leading to failure. Furthermore, when AMS utilized the previous context for inference, it identified a fault at step 451. AMS promptly interrupted the subsequent actions and executed an exception handler to revert to step 446, ultimately enabling the successful completion of the entire task.

5.4 Comparison with algorithmic solutions

We further compared AMS with algorithmic solutions, such as retraining the model to address a greater number of repeated tasks. Specifically, we evaluated the success rates of AMS against those achieved by using retrained models to accomplish varying numbers of tasks. For single-arm movement job, AMS is compared with scenarios where the model was trained to pull down 1, 3 or 6 objects.

The results in Figure 17 show that AMS effectively improves the model's ability to generalize when performing repetitive tasks. When the required number of repetitions exceeds the maximum included in the training data, the success rate of the model declines sharply. In the task, the model's accuracy decreases from 92% to 16% for the original model or fine-tuned model (3 objects). The model's accuracy also decreases from 96% to 72% for the fine-tuned model with 6 objects. However, with AMS support, the model's accuracy only decreases by up to 8%, indicating that AMS effectively maintains the model's accuracy as the number of repetitions increases. Furthermore, even when employing the original model (1 object), the AMS demonstrates a higher success rate compared to the fine-tuned model (6 objects) when attempting to pull the seventh cube (out of dataset), and a comparable success rate when pulling the first to sixth cubes (in dataset).

5.5 Ablation Study

To demonstrate the impact of each module on the model's final inference results, we conducted an ablation study to show how different modules affect accuracy and the number of steps completed.

5.5.1 Performance breakdown. In this part, we will analyze impact of different parts on AMS. The results are shown in Figure 18.

For correctness, AMS's context reuse effectiveness varies by action type. In "pick and place" tasks, it improves success by 20%, while in "single-arm movement" tasks, it boosts success by 48%, quadrupling the original rate. After adding an action fault mechanism, "pick and place" success rose by 40%, reaching 5 times the original rate, but "single-arm movement" only increased to 72% due to already low error rates for such job. With an action interrupt mechanism, AMS detects task termination signals and switches context, raising success rates to 70% for "pick and place" and 92% for "single-arm movement", increasing to 7 to 23 ×, respectively.

For efficiency, the main improvement of AMS in real-world environments comes from reducing the number of steps. The reduction varies by task. For pick and place tasks (like picking up cubes or arranging mixed objects), action context can reduce steps by 10.1%. For single-arm movement tasks (like pulling down cups), AMS can improve by 20%. This is because action context can more easily reuse previous experiences for pull tasks. Adding action exception can further reduce steps by 24.2% and 40.3%, respectively, while adding action replay can reduce steps by 29.2% and 74%.

5.5.2 Sensitivity of Replay Signal Threshold. Judging when the model should replay is an important part for action replay. The model does not output whether the current action is completed. We need to determine whether the current model has completed current task based on the current state of the robotic arm. Therefore, we analyzed various parameters for raising replay signal and the results are shown in Table 4. We use a task that involves picking up and placing balls using a gripper. There are two balls on the table in total that need to be placed.

We tested that the setting of the replay signal threshold affects the model's success rate and average number of steps, and that the replay signal settings for different physical components are not the same. We divided the test into two categories based on the type of action: joint movement (Case 1-3) and gripper movement (Case 4-6). The setting of the threshold is not simple, and there is a trade-off between false positive and false negative. When the model's threshold is set too low, it results in a low occurrence of false positives, but there may be some undetected false negatives. When the model's threshold is set too high, it can almost avoid

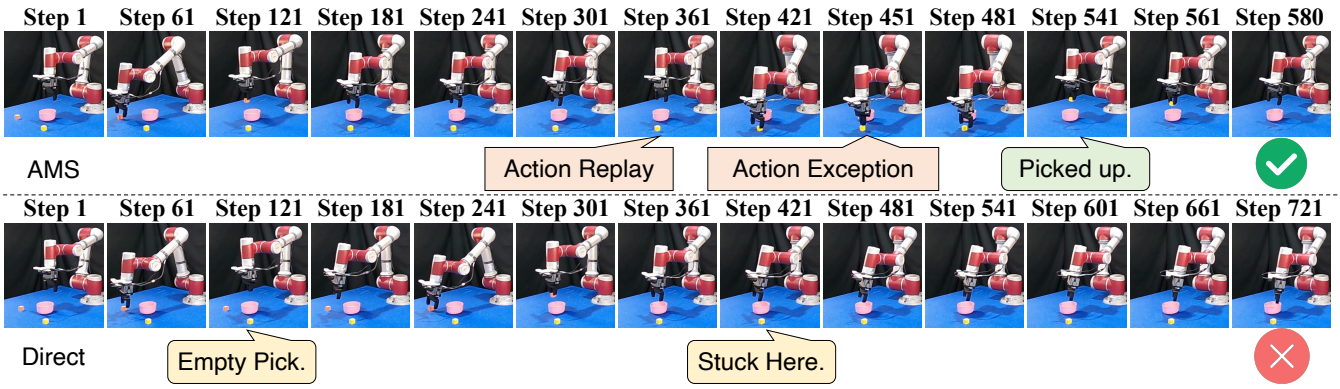

Figure 16. Model output action for "picking all cubes and placing into the bowl" job by model π_0.

Table 4. Sensitivity of action replay signal threshold. "Arm" refers to the rotation angles of each joint in the robotic arm, which are passed as parameters to control the robotic arm. "Gripper" refers to the minimum distance between the gripper fingers.

No.	Signal Threshold		Trial ↑	Average Steps		Replay Signal		
	Arm	Gripper	Success	1st↓	2nd↓	Total	False Positive↓	False Negative↓
1	1e-5	1	7 / 10	555.5 (288 - 1387)	523.7 (299 - 1010)	8	0	2
2	1e-4	1	6 / 10	709.57 (304 - 1590)	614 (576 - 602)	37	31	1
3	1e-1	1	6 / 10	482.83 (317 - 654)	449.67 (311 - 601)	40	34	0
4	1e-5	0.5	6 / 10	1129.25 (299 - 1659)	515.8 (353 - 752)	4	3	7
5	1e-5	5	3 / 10	691.28 (277 - 1470)	322.3 (293 - 363)	49	43	0
6	1e-5	10	3 / 10	1283.7 (250 - 3358)	519.6 (354 - 652)	68	61	0

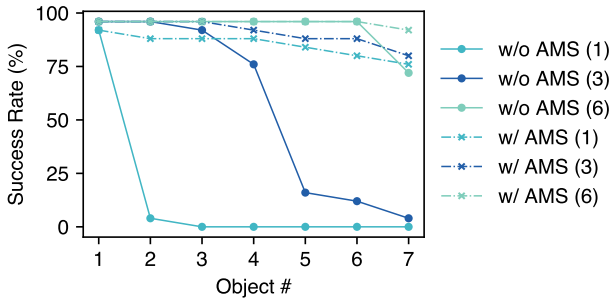

Figure 17. Comparison between AMS and algorithm-based solution in real-world robots. Number in the brackets means the maximum repetition counts of pulling in the training set.

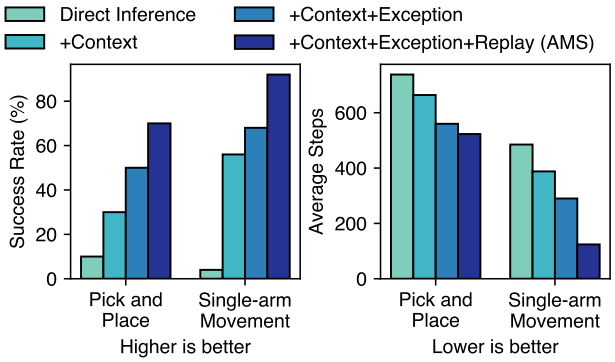

Figure 18. Impact on success rate and average steps of AMS. "Single-arm movement" is pulling down the cups.

false negatives, but the occurrence of false positives is high, which can affect performance and even lead to an up to 50% decrease in accuracy due to excessive replay action.

6 Discussion

Integrating with LLMs. Currently, AMS uses a numerical threshold method to determine replay signals. Although this approach is straightforward and effective, it still demands significant human effort for setup and calibration. In the future, we plan to integrate AMS with LLMs to automatically identify and add new action exception event classifications. Such integration requires no change to AMS. Specifically, AMS will utilize LLMs and MLLMs during action to intelligently fuse

multi-sensor data and semantic information, allowing for real-time judgment and decision-making on replay signals.

Context Transfer capability. Relying on context alone doesn't help with unfamiliar tasks due to the crucial role of LLM and ViTs mapping in fine-tuning. Without a unified representation for new visual inputs and language instructions, the model struggles with unseen tasks. While AMS successfully enhances overall capabilities in long-horizon known tasks, improving performance on previously unseen scenarios demands targeted algorithmic modifications.

Resource Limitation. AMS doesn't significantly raise computational costs during inference because it only caches intermediate states, needing no more memory than a single inference. More GPU memory allows us to store additional action sequences, enhancing efficiency by quick action reuse. Limited GPU memory might prompt extra CPU-GPU data transfers over PCIe, but these have minimal impact. As shown in Figure 2, a single inference takes roughly 200 milliseconds, whereas executing these action takes about 1 second. Therefore, the PCIe transfer delay is negligible and unlikely to affect overall task performance.

7 Related Work

VLA Acceleration. VLA-Cache [63] accelerate inference by reusing similar vectors from similar inputs. PD-VLA [54] uses parallel decode with mathematical properties to accelerate VLA inference. FAST [47] accelerates VLA training by using a new form of action tokenization and transform the training to auto-regressive training. RoboMamba [42] reduces the complexity of model by adapting Mamba [21] to robotics. TinyVLA [58] introduces compat vision-language-action model to reduce the calculation size. QAIL [46] and quantization [41] reduce the robotic model size by quantizing the model weights. Deer-vla [65] adjusts depth during inference to reduce redundant computation. Token-pruning [5, 72] prunes the tokens to reduce the inference time. SparseVLM [69] uses sparsity to reduce the computation. OFT [34] enhances inference efficiency and policy performance by using a new form of fine-tuning. Actra [43] gives an optimized Transformer architecture to reduce the computation.

Robotic Generalization. Some work [10, 16, 62] introduce agent to improve the generalization ability of robotic models. Some work [10, 51, 53] adds interactive environment to enhance the robot action in unseen states. Papras [33] designs a plug-and-play system for robot arm system to enhance the ability across different arms. BC-Z [30] introduces zero-shot task generalization method for robotic imitation learning by algorithm. Perceiver [52] designs a multi-task transformer for robotic manipulation. Copa [26], Rise [56] and Rh20t [14] use spatial constraints to improve generalization. RST [40] designed another data generalization method by self-teaching. Some work [12, 27, 39] optimize the diffusion policy to make it more robust.

8 Conclusion

This paper introduces AMS, enhancing robotic efficiency and success rate via OS-level primitives. It uses action exception for quick action interruption, action context to avoid redundant computations, and action replay for effortless repetition. Evaluations reveal AMS boosts performance by 7-24 × compared to the robotic system without AMS support.

References

[1] Adilzhan Adilkhanov, Amir Yelenov, Assylkhan Seitzhanov, Ayan Mazhitov, Azamat Abdikarimov, Danissa Sandykbayeva, Daryn Kenzhebek, Dinmukhammed Mukashev, Ilyas Umurbekov, Jabrail Chumakov, et al. 2025. Survey on Vision-Language-Action Models. *arXiv preprint arXiv:2502.06851* (2025).

[2] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. 2024. π_0: A Vision-Language-Action Flow Model for General Robot Control. arXiv:2410.24164 [cs.LG] https://arxiv.org/abs/2410.24164

[3] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. 2022. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817* (2022).

[4] Joao Carvalho, A Le, Piotr Kicki, Dorothea Koert, and Jan Peters. 2024. Motion planning diffusion: Learning and adapting robot motion planning with diffusion models. *arXiv preprint arXiv:2412.19948* (2024).

[5] Liang Chen, Haozhe Zhao, Tianyu Liu, Shuai Bai, Junyang Lin, Chang Zhou, and Baobao Chang. 2024. An image is worth 1/2 tokens after layer 2: Plug-and-play inference acceleration for large vision-language models. In *European Conference on Computer Vision*. Springer, 19–35.

[6] Sijin Chen, Xin Chen, Anqi Pang, Xianfang Zeng, Wei Cheng, Yijun Fu, Fukun Yin, Billzb Wang, Jingyi Yu, Gang Yu, et al. 2024. Meshxl: Neural coordinate field for generative 3d foundation models. *Advances in Neural Information Processing Systems* 37 (2024), 97141–97166.

[7] Shizhe Chen, Ricardo Garcia, Cordelia Schmid, and Ivan Laptev. 2023. Polarnet: 3d point clouds for language-guided robotic manipulation. *arXiv preprint arXiv:2309.15596* (2023).

[8] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. 2023. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research* (2023), 02783649241273668.

[9] Suhyung Choi, Youngseok Joo, Jun Ki Lee, and Byoung-Tak Zhang. 2025. Mixture of Action Expert Embeddings: Multi-Task ACT. (2025).

[10] Sharmita Dey. 2025. Redefining Robot Generalization Through Interactive Intelligence. *arXiv preprint arXiv:2502.05963* (2025).

[11] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv:2010.11929 [cs.CV] https://arxiv.org/abs/2010.11929

[12] Oluwami Dosunmu-Ogunbi, Aayushi Shrivastava, and Jessy W Grizzle. 2024. Demonstrating a Robust Walking Algorithm for Underactuated Bipedal Robots in Non-flat, Non-stationary Environments. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 11210–11217.

[13] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, et al. 2023. Palm-e: An embodied multimodal

language model. (2023).

[14] Hao-Shu Fang, Hongjie Fang, Zhenyu Tang, Jirong Liu, Chenxi Wang, Junbo Wang, Haoyi Zhu, and Cewu Lu. 2024. Rh20t: A comprehensive robotic dataset for learning diverse skills in one-shot. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 653–660.

[15] Roya Firoozi, Johnathan Tucker, Stephen Tian, Anirudha Majumdar, Jiankai Sun, Weiyu Liu, Yuke Zhu, Shuran Song, Ashish Kapoor, Karol Hausman, et al. 2023. Foundation models in robotics: Applications, challenges, and the future. *The International Journal of Robotics Research* (2023), 02783649241281508.

[16] Dayuan Fu, Keqing He, Yejie Wang, Wentao Hong, Zhuoma Gongque, Weihao Zeng, Wei Wang, Jingang Wang, Xunliang Cai, and Weiran Xu. 2025. AgentRefine: Enhancing Agent Generalization through Refinement Tuning. *arXiv preprint arXiv:2501.01702* (2025).

[17] Zipeng Fu, Tony Z Zhao, and Chelsea Finn. 2024. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. *arXiv preprint arXiv:2401.02117* (2024).

[18] Abraham George and Amir Barati Farimani. 2023. One act play: Single demonstration behavior cloning with action chunking transformers. *arXiv preprint arXiv:2309.10175* (2023).

[19] Theophile Gervet, Zhou Xian, Nikolaos Gkanatsios, and Katerina Fragkiadaki. 2023. Act3d: 3d feature field transformers for multi-task robotic manipulation. *arXiv preprint arXiv:2306.17817* (2023).

[20] Ankit Goyal, Jie Xu, Yijie Guo, Valts Blukis, Yu-Wei Chao, and Dieter Fox. 2023. Rvt: Robotic view transformer for 3d object manipulation. In *Conference on Robot Learning*. PMLR, 694–710.

[21] Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752* (2023).

[22] Pierre-Louis Guhur, Shizhe Chen, Ricardo Garcia Pinel, Makarand Tapaswi, Ivan Laptev, and Cordelia Schmid. 2023. Instruction-driven history-aware policies for robotic manipulations. In *Conference on Robot Learning*. PMLR, 175–187.

[23] Yanjiang Guo, Jianke Zhang, Xiaoyu Chen, Xiang Ji, Yen-Jen Wang, Yucheng Hu, and Jianyu Chen. 2025. Improving Vision-Language-Action Model with Online Reinforcement Learning. *arXiv preprint arXiv:2501.16664* (2025).

[24] Huy Ha, Pete Florence, and Shuran Song. 2023. Scaling up and distilling down: Language-guided robot skill acquisition. In *Conference on Robot Learning*. PMLR, 3766–3777.

[25] Zhi Hou, Tianyi Zhang, Yuwen Xiong, Haonan Duan, Hengjun Pu, Ronglei Tong, Chengyang Zhao, Xizhou Zhu, Yu Qiao, Jifeng Dai, et al. 2025. Dita: Scaling Diffusion Transformer for Generalist Vision-Language-Action Policy. *arXiv preprint arXiv:2503.19757* (2025).

[26] Haoxu Huang, Fanqi Lin, Yingdong Hu, Shengjie Wang, and Yang Gao. 2024. Copa: General robotic manipulation through spatial constraints of parts with foundation models. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 9488–9495.

[27] Yutaro Ishida, Yuki Noguchi, Takayuki Kanai, Kazuhiro Shintani, and Hiroshi Bito. 2024. Robust Imitation Learning for Mobile Manipulator Focusing on Task-Related Viewpoints and Regions. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2885–2892.

[28] JAKA. 2025. JAKA Robots. https://www.jaka.com/en/index [Online; accessed 2025-04-18].

[29] Stephen James, Kentaro Wada, Tristan Laidlow, and Andrew J Davison. 2022. Coarse-to-fine q-attention: Efficient learning for visual robotic manipulation via discretisation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 13739–13748.

[30] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. 2022. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*. PMLR, 991–1002.

[31] Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. 2022. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991* (2022).

[32] Kai Jiang and Jiaxing Huang. 2024. A Survey on Vision Autoregressive Model. *arXiv preprint arXiv:2411.08666* (2024).

[33] Joohyung Kim, Dhruv C Mathur, Kazuki Shin, and Sean Taylor. 2023. Papras: Plug-and-play robotic arm system. *arXiv preprint arXiv:2302.09655* (2023).

[34] Moo Jin Kim, Chelsea Finn, and Percy Liang. 2025. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645* (2025).

[35] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. 2024. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246* (2024).

[36] Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, et al. 2024. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650* (2024).

[37] Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. 2024. Evaluating Real-World Robot Manipulation Policies in Simulation. *arXiv preprint arXiv:2405.05941* (2024).

[38] Xiang Li, Cristina Mata, Jongwoo Park, Kumara Kahatapitiya, Yoo Sung Jang, Jinghuan Shang, Kanchana Ranasinghe, Ryan Burgert, Mu Cai, Yong Jae Lee, et al. 2024. Llara: Supercharging robot learning data for vision-language policy. *arXiv preprint arXiv:2406.20095* (2024).

[39] Yinghui Li, Jinze Wu, Xin Liu, Weizhong Guo, and Yufei Xue. 2024. Experience-Learning Inspired Two-Step Reward Method for Efficient Legged Locomotion Learning Towards Natural and Robust Gaits. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 13297–13302.

[40] Yunfei Li, Ying Yuan, Jingzhi Cui, Haoran Huan, Wei Fu, Jiaxuan Gao, Zekai Xu, and Yi Wu. 2024. Robot Generating Data for Learning Generalizable Visual Robotic Manipulation. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 5813–5820.

[41] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Guangxuan Xiao, and Song Han. 2025. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. *GetMobile: Mobile Computing and Communications* 28, 4 (2025), 12–17.

[42] Jiaming Liu, Mengzhen Liu, Zhenyu Wang, Lily Lee, Kaichen Zhou, Pengju An, Senqiao Yang, Renrui Zhang, Yandong Guo, and Shanghang Zhang. 2024. Robomamba: Multimodal state space model for efficient robot reasoning and manipulation. *arXiv preprint arXiv:2406.04339* (2024).

[43] Yueen Ma, Dafeng Chi, Shiguang Wu, Yuecheng Liu, Yuzheng Zhuang, Jianye Hao, and Irwin King. 2024. Actra: Optimized transformer architecture for vision-language-action models in robot learning. *arXiv preprint arXiv:2408.01147* (2024).

[44] Cheng Pan, Kai Junge, and Josie Hughes. 2024. Vision-language-action model and diffusion policy switching enables dexterous control of an anthropomorphic hand. *arXiv preprint arXiv:2410.14022* (2024).

[45] J Hyeon Park, Wonhyuk Choi, Sunpyo Hong, Hoseong Seo, Joonmo Ahn, Changsu Ha, Heungwoo Han, and Junghyun Kwon. 2024. Hierarchical Action Chunking Transformer: Learning Temporal Multimodality from Demonstrations with Fast Imitation Behavior. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 12648–12654.

[46] Seongmin Park, Hyungmin Kim, Wonseok Jeon, Juyoung Yang, Byeongwook Jeon, Yoonseon Oh, and Jungwook Choi. 2024. Quantization-Aware Imitation-Learning for Resource-Efficient

Robotic Control. *arXiv preprint arXiv:2412.01034* (2024).

[47] Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. 2025. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747* (2025).

[48] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. arXiv:2103.00020 [cs.CV] https://arxiv.org/abs/2103.00020

[49] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. 2017. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3577–3586.

[50] Trossen Robotics. 2025. Trossen Robotics. https://www.trossenrobotics.com/ [Online; accessed 2025-04-18].

[51] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems* 36 (2023), 8634–8652.

[52] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. 2023. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Conference on Robot Learning*. PMLR, 785–799.

[53] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768* (2020).

[54] Wenxuan Song, Jiayi Chen, Pengxiang Ding, Han Zhao, Wei Zhao, Zhide Zhong, Zongyuan Ge, Jun Ma, and Haoang Li. 2025. Accelerating Vision-Language-Action Model Integrated with Action Chunking via Parallel Decoding. *arXiv preprint arXiv:2503.02310* (2025).

[55] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. 2024. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213* (2024).

[56] Chenxi Wang, Hongjie Fang, Hao-Shu Fang, and Cewu Lu. 2024. Rise: 3d perception makes real-world robot imitation simple and effective. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2870–2877.

[57] Junjie Wen, Minjie Zhu, Yichen Zhu, Zhibin Tang, Jinming Li, Zhongyi Zhou, Chengmeng Li, Xiaoyu Liu, Yaxin Peng, Chaomin Shen, et al. 2024. Diffusion-VLA: Scaling Robot Foundation Models via Unified Diffusion and Autoregression. *arXiv preprint arXiv:2412.03293* (2024).

[58] Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Zhibin Tang, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, et al. 2025. Tinyvla: Towards fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters* (2025).

[59] Kun Wu, Chengkai Hou, Jiaming Liu, Zhengping Che, Xiaozhu Ju, Zhuqin Yang, Meng Li, Yinuo Zhao, Zhiyuan Xu, Guang Yang, et al. 2024. Robomind: Benchmark on multi-embodiment intelligence normative data for robot manipulation. *arXiv preprint arXiv:2412.13877* (2024).

[60] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2025. The rise and potential of large language model based agents: A survey. *Science China Information Sciences* 68, 2 (2025), 121101.

[61] Shenghao Xie, Wenqiang Zu, Mingyang Zhao, Duo Su, Shilong Liu, Ruohua Shi, Guoqi Li, Shanghang Zhang, and Lei Ma. 2024. Towards Unifying Understanding and Generation in the Era of Vision Foundation Models: A Survey from the Autoregression Perspective. *arXiv preprint arXiv:2410.22217* (2024).

[62] Weimin Xiong, Yifan Song, Xiutian Zhao, Wenhao Wu, Xun Wang, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. 2024. Watch every step! llm agent learning via iterative step-level process refinement. *arXiv preprint arXiv:2406.11176* (2024).

[63] Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. 2025. VLA-Cache: Towards Efficient Vision-Language-Action Model via Adaptive Token Caching in Robotic Manipulation. *arXiv preprint arXiv:2502.02175* (2025).

[64] Xinyu Xu, Yizheng Zhang, Yong-Lu Li, Lei Han, and Cewu Lu. 2024. Humanvla: Towards vision-language directed object rearrangement by physical humanoid. *arXiv preprint arXiv:2406.19972* (2024).

[65] Yang Yue, Yulin Wang, Bingyi Kang, Yizeng Han, Shenzhi Wang, Shiji Song, Jiashi Feng, and Gao Huang. 2024. Deer-vla: Dynamic inference of multimodal large language models for efficient robot execution. *Advances in Neural Information Processing Systems* 37 (2024), 56619–56643.

[66] Yanjie Ze, Zixuan Chen, Wenhao Wang, Tianyi Chen, Xialin He, Ying Yuan, Xue Bin Peng, and Jiajun Wu. 2024. Generalizable humanoid manipulation with improved 3d diffusion policies. *arXiv preprint arXiv:2410.10803* (2024).

[67] Jianke Zhang, Yanjiang Guo, Yucheng Hu, Xiaoyu Chen, Xiang Zhu, and Jianyu Chen. 2025. UP-VLA: A Unified Understanding and Prediction Model for Embodied Agent. *arXiv preprint arXiv:2501.18867* (2025).

[68] Xinyu Zhang, Yuhan Liu, Haonan Chang, Liam Schramm, and Abdeslam Boularias. 2025. Autoregressive action sequence learning for robotic manipulation. *IEEE Robotics and Automation Letters* (2025).

[69] Yuan Zhang, Chun-Kai Fan, Junpeng Ma, Wenzhao Zheng, Tao Huang, Kuan Cheng, Denis Gudovskiy, Tomoyuki Okuno, Yohei Nakata, Kurt Keutzer, et al. 2024. Sparsevlm: Visual token sparsification for efficient vision-language model inference. *arXiv preprint arXiv:2410.04417* (2024).

[70] Qingqing Zhao, Yao Lu, Moo Jin Kim, Zipeng Fu, Zhuoyang Zhang, Yecheng Wu, Zhaoshuo Li, Qianli Ma, Song Han, Chelsea Finn, et al. 2025. CoT-VLA: Visual Chain-of-Thought Reasoning for Vision-Language-Action Models. *arXiv preprint arXiv:2503.22020* (2025).

[71] Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. 2023. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705* (2023).

[72] Wangbo Zhao, Jiasheng Tang, Yizeng Han, Yibing Song, Kai Wang, Gao Huang, Fan Wang, and Yang You. 2024. Dynamic tuning towards parameter and inference efficiency for vit adaptation. *arXiv preprint arXiv:2403.11808* (2024).

[73] Haoyu Zhen, Xiaowen Qiu, Peihao Chen, Jincheng Yang, Xin Yan, Yilun Du, Yining Hong, and Chuang Gan. 2024. 3d-vla: A 3d vision-language-action generative world model. *arXiv preprint arXiv:2403.09631* (2024).