ScaRLS: Scalabe Robot Learning System for Robot Manipulation

Agibot Team

***Abstract*—In the field of robotic manipulation, achieving both high generalization capability and high reliability (i.e., high task success rates) remains a central challenge. A critical bottleneck constraining current system performance is the insufficiency of data space coverage—specifically, the robot's exploration and comprehension of the state space (comprising both observations and actions) are severely limited. To overcome this bottleneck, this paper proposes ScaRLS (Scalable Robotic Learning System), a scalable learning framework designed for robotic manipulation tasks. By leveraging a distributed edge computing architecture, ScaRLS facilitates the acquisition of broader data distributions and enables efficient learning from such data.**

ScaRLS establishes a complete system closed-loop featuring asynchronous data collection (incorporating both autonomous model inference and human-in-the-loop intervention), asynchronous training, and asynchronous parameter distribution. This architecture not only accelerates the workflow of individual computing nodes but also enables large-scale data acquisition and high-efficiency learning by scaling up the number of computing nodes. Furthermore, the system demonstrates robust versatility, offering adaptability to diverse algorithmic models and robotic embodiments.

I. INTRODUCTION

Learning-based approaches, including Deep Reinforcement Learning (DRL) [7, 20, 22, 8, 14] and Imitation Learning (IL) [6, 25, 2, 3], have enabled steady progress in robotic manipulation. These methods have been applied to tasks ranging from pick-and-place to contact-rich manipulation [18, 9], such as connector insertion and furniture assembly. Nevertheless, achieving robust and reliable performance outside controlled laboratory settings remains a central challenge for real-world robotic systems.

In practical deployment, robotic policies must operate under distribution shifts caused by unseen objects, varying illumination, background clutter, and changes in physical parameters. Under such conditions, learned policies often exhibit brittle behavior, including performance degradation and unexpected failures [4]. A key contributing factor is that most robotic learning pipelines rely on single-agent physical interaction, where data collection is inherently slow and costly. This limitation prevents sufficient coverage of long-tail regions of the state–action space, resulting in policies that overfit narrow training conditions and generalize poorly to new environments.

From a system-level perspective, these failures are closely tied to how data is generated. In domains such as computer vision and natural language processing, robust generalization has been driven by the combination of large-scale datasets [12, 23] and expressive foundation models [13, 21]. In contrast, robotic manipulation data must be acquired through physical interaction, making scale fundamentally constrained by hardware availability, human supervision, and deployment logistics. As a result, improving generalization in robotic manipulation requires not only advances in learning algorithms, but also learning systems that explicitly account for the process of data acquisition in the physical world.

We view scalable robotic learning through two complementary dimensions of data expansion. The first is **exploration depth**, which focuses on acquiring informative and corrective data within a single-agent setting. Human-in-the-loop learning methods [18, 11, 15, 17] have demonstrated that expert intervention can effectively guide robots out of failure states and back toward valid behaviors [1], enabling efficient exploration of difficult regions of the state space [24]. The second dimension is **exploration breadth**, which emphasizes parallel data collection through multi-agent or cluster-based deployments [10, 16], allowing broader state–action coverage via concurrent physical interaction.

While both exploration depth and breadth have been studied independently, existing learning systems typically address them in isolation. Systems optimized for sample efficiency and human supervision at the single-agent level (e.g., HIL-SERL [19]) are difficult to scale due to limited human bandwidth [5]. Conversely, large-scale learning infrastructures designed for high-throughput data collection often treat scalability primarily as a training-time concern, abstracting away how data is physically generated, corrected, and deployed on real robots. This separation creates a disconnect between single-agent development and multi-agent deployment, making it difficult to iteratively scale robotic learning systems in practice.

To bridge this gap, we introduce **ScaRLS** (**Sca**lable **R**obot **L**earning **S**ystem), a unified robot learning system abstraction that tightly couples data acquisition, human supervision, and distributed learning. ScaRLS is designed to support a seamless transition from single-agent, human-in-the-loop experimentation to large-scale multi-agent deployment, enabling scalable data expansion without decoupling learning from real-world interaction.

ScaRLS is structured around three core components. First, a **foundational algorithm layer** enables flexible integration of diverse learning algorithms and model architectures at the single-agent level. Second, a **cluster expansion layer** supports scalable multi-agent learning through Edge Computing Units (ECUs), which manage local data acquisition, training, and deployment, while synchronizing model parameters across the cluster. Finally, the system is designed for **scalable execution**, such that data throughput and training capacity grow approx-

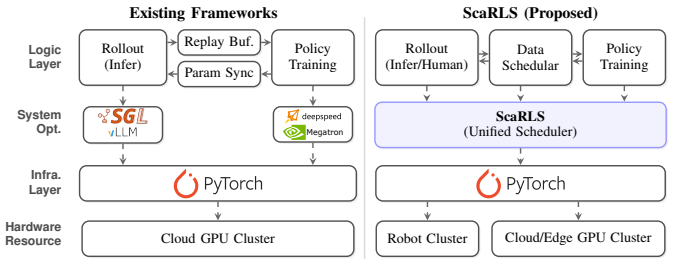

Fig. 1. **Overview of Existing Frameworks vs. ScaRLS.** The left side illustrates traditional fragmented systems where rollout and training are isolated. The right side demonstrates our proposed ScaRLS architecture, which introduces a unified scheduler to orchestrate heterogeneous resources (Robots and Cloud GPUs) for efficient data generation and policy training.

imately linearly with the number of deployed robots.

II. Related Work

A. Single-Agent Learning Paradigms

The central challenge in single-robot learning lies in overcoming the prohibitive costs of physical interaction, necessitating the maximization of information gain from each interaction to enhance sample efficiency. Prior work primarily focused on leveraging static expert datasets, where Imitation Learning (IL) methods sought to circumvent exploration risks by reproducing expert demonstrations. However, classical Behavioral Cloning (BC) is susceptible to the covariate shift problem, which can lead to policy failure during deployment. While the subsequently proposed DAgger algorithm provides theoretical convergence guarantees, its reliance on continuous online expert corrections imposes a significant human bottleneck when applied to physical robotic systems.

To bridge the gap between expert guidance and autonomous exploration, the paradigm of Human-in-the-Loop Reinforcement Learning (HIL-RL) has emerged. Exemplified by systems like HIL-SERL, this approach positions the human as both a real-time instructor and a safety guardian. By integrating three distinct phases—reward function construction, demonstration-guided exploration, and online intervention—such systems have achieved superior sample efficiency in complex, contact-rich tasks. Nevertheless, a fundamental limitation lies in the required 1:1 human-robot ratio. The necessity for the human operator to maintain full attention on a single agent fundamentally constrains system scalability, rendering it ill-suited for large-scale skill acquisition.

More recently, Offline Reinforcement Learning and Vision-Language-Action (VLA) models, grounded in large-scale offline datasets, have emerged as a prominent trend. The RT series, for instance, demonstrates the potential of leveraging massive cross-embodiment data for pre-training to achieve skill generalization. However, these approaches remain heavily dependent on the quality and coverage of static datasets. Their lack of online adaptation and error-correction mechanisms often limits performance when confronting unseen physical dynamics or fine-grained manipulation tasks.

B. Distributed Learning Architectures

To surmount the temporal barriers inherent in single-agent physical learning, Distributed Reinforcement Learning (Distributed RL) leverages parallelized data acquisition to achieve scalability. In simulated environments, the Ape-X architecture achieved orders-of-magnitude improvements in data throughput by decoupling actors from learners and utilizing prioritized experience replay. Building on this, R2D2 addressed the issue of state staleness in Recurrent Neural Networks (RNNs) within distributed settings, laying the groundwork for handling partially observable tasks. However, these simulation-centric architectures often overlook the communication latency and synchronization challenges pervasive in the physical world.

Transferring distributed learning to physical deployments introduces more stringent constraints. QT-Opt and MT-Opt stand as landmark achievements in physical robotic cluster learning. By conducting large-scale, asynchronous data collection across real-world robot farms over several months, they demonstrated that complex manipulation skills can emerge from massive volumes of physical interaction. Empirical studies suggest that in such physical distributed systems, asynchronous architectures typically outperform synchronous update methods due to reduced control latency and higher interaction frequencies [26]. Nevertheless, existing physical cluster systems predominantly rely on fixed task assignments and centralized communication topologies, lacking capabilities for dynamic task orchestration and resource scheduling. This results in prohibitive hardware costs and insufficient system flexibility.

C. Infrastructure and Ecosystem

Algorithmic advancements are inextricably linked to the support of robust infrastructure. In terms of hardware benchmarks, the ROBEL platform has lowered the barrier to entry for physical experimentation by providing low-cost, modular robotic embodiments; however, its software stack lacks native support for distributed clusters. Simulation benchmarks such as RLBench offer extensive task suites to facilitate algorithmic comparison but suffer from a pronounced sim-to-real gap.

Standardization of the data and software ecosystem represents another critical direction. Projects like Open X-Embodiment provide a data foundation for training generalist policy models by aggregating cross-institutional and cross-embodiment robot datasets. While open-source frameworks like LeRobot aim to provide a unified toolchain ranging from data collection to policy training, they remain in nascent stages regarding the real-time orchestration and collaborative training support required for large-scale physical clusters.

III. System Design: ScaRLS

To efficiently acquire and process diverse robotic observational state data, we present ScaRLS, a distributed edge computing system, as illustrated in Fig. [X]. The architecture is defined by two core attributes:

- **Edge-based Online Post-Training**. It enables immediate model refinement directly on edge devices.

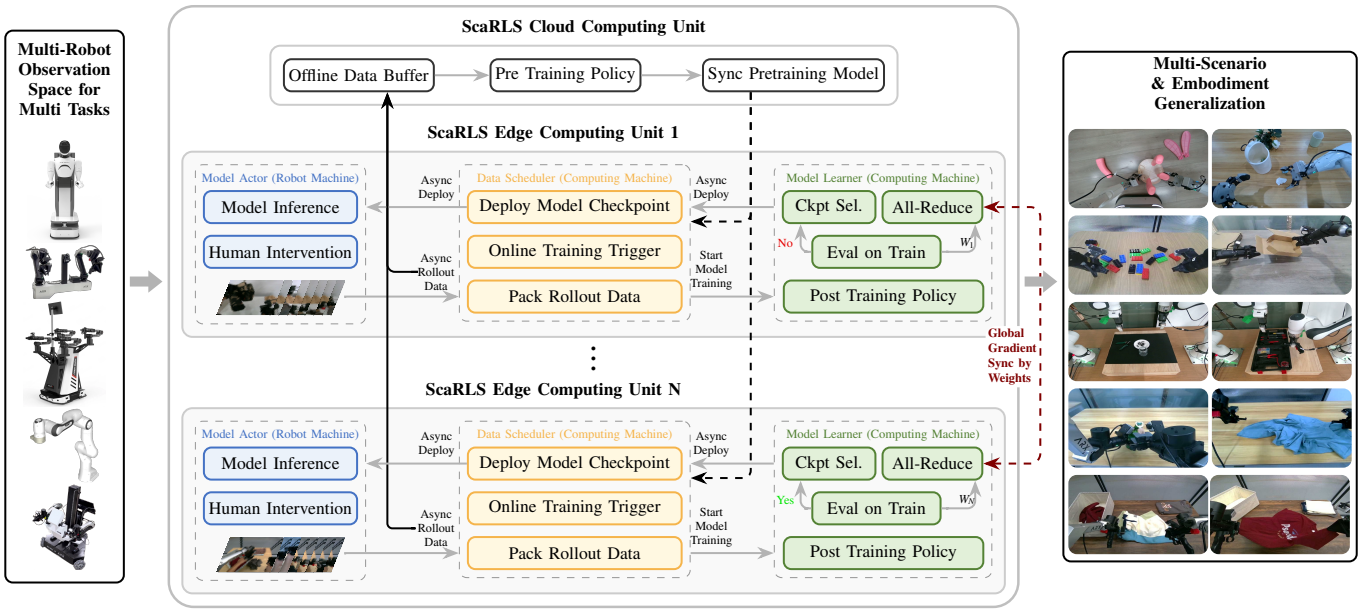

Fig. 2. **Overview of the ScaRLS Framework.** The system is composed of three main modules: (1) **Multi-Robot Observation Space**, which aggregates sensory data from diverse robot embodiments (e.g., mobile manipulators, robotic arms); (2) The core **ScaRLS Computing Architecture**, featuring a hierarchical Cloud-Edge design. The **Cloud Unit** manages the global offline data buffer and synchronizes the pre-training model, while multiple distributed **Edge Computing Units (ECU)** handle local model inference, asynchronous data collection (Rollout), and local gradient updates (All-Reduce); (3) **Multi-Scenario & Embodiment Generalization**, demonstrating the system's capability to deploy learned policies across various real-world tasks (e.g., object manipulation, cloth folding) and hardware platforms through global weight synchronization.

- **Scalable Data Acquisition and Synchronization**. By scaling the number of edge nodes, the system expands the accessible observational state space. When combined with distributed parameter synchronization, this facilitates the continuous evolution of a progressively more robust post-trained policy.

In the following sections, we detail the architectural components and operational workflow of ScaRLS.

A. System Architecture

ScaRLS is composed of multiple Edge Computing Units (ECUs). Architecturally, each ECU comprises a physical robot paired with a computational workstation. The software stack within an ECU is structured into three primary modules: the **Model Actor**, the **Data Scheduler**, and the **Model Learner**.

1) Model Actor: Running on the physical robot (or real-time controller), the Model Actor executes model inference and facilitates human intervention. It is responsible for transmitting data generated from both autonomous inference and human takeover events to the Data Scheduler. The Model Actor is implemented as a hardware-agnostic software framework, enabling seamless deployment across diverse robotic embodiments. Its core control logic relies on a Finite State Machine (FSM) to manage the interleaved execution of autonomous model inference and human intervention loops as shown in Algorithm 1.The detailed transition logic is illustrated in the software architecture diagram.

2) Data Scheduler: Hosted on the workstation, the Data Scheduler serves as the data orchestration hub. It receives rollout data, formatting and persisting it for training purposes.

Algorithm 1 Robot Control Loop

Require: Init state s_0, Controller C, Model π_θ

$state \leftarrow s_0$
while true do
  if $state =$ INIT **then**
    RESETPOSE()
    $state \leftarrow$ INFERENCE
  else if $state =$ INFERENCE **then**
    $obs \leftarrow env.$GETOBS()
    $a \leftarrow \pi_\theta(obs)$
    $env.$STEP(a)
  else if $state \in \{$SUCCESS, FAIL$\}$ **then**
    $actor.$LOG$(state)$
    if $actor.$NEEDCHECKPOINT() **then**
      $actor.$RELOAD()
    end if
    RESETPOSE()
    $state \leftarrow$ IDLE
  end if
  ▷ Wait for external signal
  $state \leftarrow C.$GETSIGNAL()
end while

Furthermore, it triggers the Model Learner for online training sessions and deploys the latest model parameters from the Learner back to the Model Actor. Functioning as the central orchestration hub, the Data Scheduler governs data persistence, training triggers, and model deployment. It operates as a reactive service handling two primary request types from the Model Actor:

1) Rollout Data Ingestion. The sscheduler receives incoming interaction data and executes high-throughput

serialization to disk.

2) Checkpoint Synchronization. Upon request, it verifies the availability of updated checkpoints from the Model Learner and disseminates them to the Actor. Furthermore, the Data Scheduler dictates the learning cadence by triggering the training process immediately upon episode completion and continuously monitoring the pipeline for the generation of new model checkpoints.

3) Model Learner: Also residing on the workstation, the Model Learner functions as the training engine. It monitors training data availability, executes the model training loop, and performs model validity assessments. Crucially, it manages the synchronization of model parameters across multiple ECUs in the cluster. The Model Learner encapsulates a standard online post-training pipeline. It continuously streams episode data for real-time updates while periodically assessing model validity via Open-Loop Evaluation. The resulting evaluation metrics serve as weighting factors for a distributed Weighted All-Reduce operation across cluster nodes. Subsequently, a Checkpoint Selector validates the synchronized parameters via a secondary Open-Loop test; only models surpassing a predefined performance threshold are flagged to the Data Scheduler for deployment.

B. Operational Workflow

The operational workflow of ScaRLS is depicted in Fig. 3 and proceeds as follows:

Initially, the Model Actor requests the target policy (model) from the Data Scheduler. Upon receipt, the Actor initiates model inference. In instances of inference failure or deviation, human teleoperation intervenes to correct the trajectory. Data generated from both autonomous inference and human intervention is uploaded to the Data Scheduler for packaging and storage.

Upon receiving an episode-completion signal from the Actor, the Data Scheduler generates formatted episode data and triggers a training signal. When the Model Learner receives this signal, it commences the training process. Simultaneously, the Learner performs an online assessment of the current policy's validity by constructing an evaluation metric via open-loop testing. Based on this metric, weighted parameter synchronization is executed across the ECU cluster.

After a designated number of training iterations, the Model Learner issues a checkpoint synchronization signal, prompting the Data Scheduler to update the Model Actor with the latest checkpoint. The entire pipeline of data acquisition and model training operates asynchronously to maximize throughput.

C. System Scalability

Algorithmic Scalability: Enhanced Policy Generalization. Scaling the number of Edge Computing Unit (ECU) nodes inherently expands the distributional coverage of the acquired data. By integrating this broadened data landscape with distributed parameter synchronization, ScaRLS achieves a joint modeling of the global state-action distribution. Consequently, increasing the node count theoretically correlates with improvements in the robustness and capability of the base policy.

Engineering Extensibility: Pipeline Integration. From an engineering perspective, ScaRLS is designed as an open-ended data engine. The interaction data generated by distributed Model Actors can be aggregated to cloud storage, serving as a foundational dataset for subsequent stages of large-scale pre-training or facilitating further downstream model development.

Algorithm 2 System Coordination Protocol

Require: Shared Storage $\mathscr{S}$, Model Registry $\mathscr{R}$

Process 1: Model-Actor (Data Producer)

procedure RUNINFERENCELOOP
  $\theta \leftarrow \mathscr{R}$.FetchLatestModel()
  loop
    $\tau \leftarrow$ InteractWithEnvironment(θ)
    UploadData($\mathscr{S}, \tau$)
    if EpisodeFinished(τ) **then**
      NotifyScheduler(NewEpisode)
    end if
    if $\mathscr{R}$.HasNewVersion() **then**
      $\theta \leftarrow \mathscr{R}$.FetchLatestModel()
    end if
  end loop
end procedure

Process 2: Data-Scheduler (Coordinator)

procedure ONDATANOTIFICATION
  $\mathscr{D} \leftarrow$ ScanAvailableData($\mathscr{S}$)
  if CheckQuality($\mathscr{D}$) $\wedge$ TimeSinceLastTrain() $> T_{\text{interval}}$ **then**
    $\mathscr{B} \leftarrow$ GenerateDataset($\mathscr{D}$)
    TriggerTraining(Learner, $\mathscr{B}$)
  end if
end procedure

Process 3: Model-Learner (Consumer)

procedure TRAINANDEVALUATE($\mathscr{B}$)
  $\theta_{\text{train}} \leftarrow$ LoadCurrentPolicy()
  $\theta' \leftarrow$ TrainOnBatch($\theta_{\text{train}}, \mathscr{B}$)
  $s_{\text{eval}} \leftarrow$ EvalOnTrain(θ')
  if $s_{\text{eval}} >$ Threshold **then**
    $\theta_{\text{sync}} \leftarrow$ SyncParam(θ')
    CheckpointSelector(θ_{sync})
    $\mathscr{R}$.Publish(θ_{sync})
  else
    discard θ'
  end if
end procedure

IV. EXPERIMENTS

A. Implementation Details

1) Hardware Configuration: The fundamental building block of our infrastructure is the Edge Computing Unit (ECU), established via a local network connection between a robotic platform and a computational workstation. In our reference implementation, we utilize the Arx-robotics AcOne platform. The robot is equipped with an NVIDIA RTX 5080 GPU for real-time inference and the paired workstation utilizes an NVIDIA RTX 4090D GPU for training and heavy processing. Communication within the ECU is facilitated by a high-speed Gigabit Ethernet interface via a dedicated router. The architecture supports the scalable deployment of multiple such ECU nodes to form a cluster.

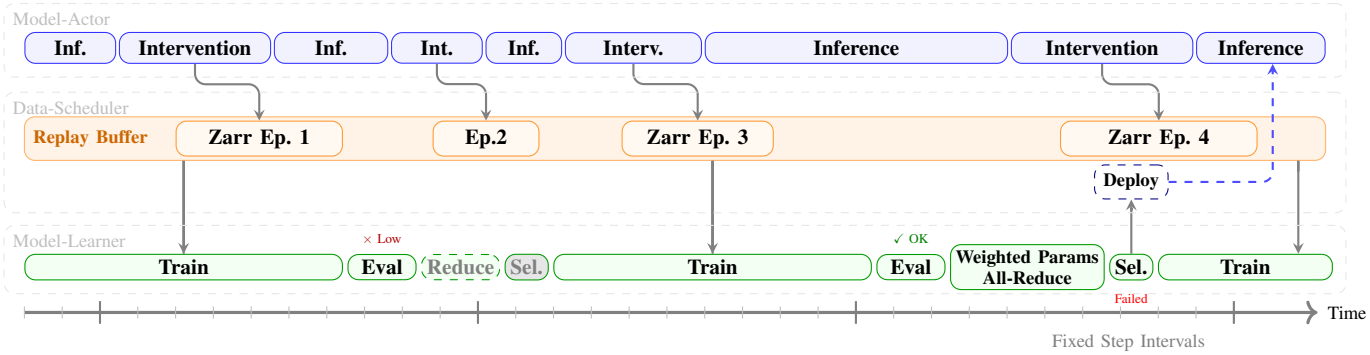

Fig. 3. System architecture with asynchronous periodic deployment. The framework operates asynchronously across three layers: **Model-Actor** (generating experience), **Data-Scheduler** (managing replay buffers), and **Model-Learner** (training). **Abbreviations:** Inf.: Inference; Int./Interv.: Intervention; Ep.: Episode; Sel.: Model Selection; Red.: Parameter Reduction (All-Reduce). The timeline illustrates two update cycles: (1) A **failure scenario** (× Low), where the evaluation metric is insufficient. Consequently, the reduction and selection steps are skipped (indicated by dashed/gray nodes), and no deployment occurs. (2) A **successful update** (✓ OK), where the evaluation meets the threshold, triggering the weighted parameter all-reduce and selection, finally leading to a **Deploy** event that updates the actor's model.

2) Software Environment: Within a single ECU node, the throughput of data serialization (logging) and retrieval (loading) constitutes a critical performance factor. To maximize efficiency, we developed a custom data management backend built upon ***Zarr***. This module is responsible for the high-frequency persistence of heterogeneous data streams, originating from both model inference and human intervention and ensures efficient, chunked data loading during the training phase. The specific data schema is defined as follows:

```
/zarr_data/
└── {task_name}/
    ├── episode_0.zarr
    ├── episode_0_meta.json
    ├── episode_1.zarr
    ├── episode_1_meta.json
    └── ...
```

Fig. 4. Hierarchical data structure implementation.

TABLE I
DATA SCHEMA OF THE ZARR STORAGE BACKEND. T DENOTES THE SEQUENCE LENGTH OF AN EPISODE.

Group / Key	Description	Data Type	Shape
`observation/`	Sensory inputs group	-	-
`top`	Top-down RGB camera	`uint8`	$(T, H, W, 3)$
`left`	Left-side RGB camera	`uint8`	$(T, H, W, 3)$
`right`	Right-side RGB camera	`uint8`	$(T, H, W, 3)$
`obs_state`	Proprioceptive state	`float32`	(T, D_{state})
`action`	Robot control commands	`float32`	(T, D_{action})

We implement the policy learning framework using PyTorch 2.7.1. To ensure system robustness and minimize deployment gaps, we maintain a unified computational backend for both the training and inference phases.

B. System Performance Analysis

In this section, we characterize the I/O and latency performance of a standalone ECU node.

TABLE II
COMPARISON OF DATA LOADING EFFICIENCY.

Method	Avg. Load Time (s) ↓	Speedup ↑
Lerobot (v0.3.3)	0.2052	1.0×
Lerobot (v0.4.1)	0.2083	∼ 1.0×
Ours (Zarr)	**0.0250**	**8.21**×

1) Data Persistence Throughput: We benchmarked the data loading efficiency of our proposed Zarr-based data loader against the widely adopted Lerobot (v0.3.3) framework in Table II. Experiments were conducted under identical hardware and operating system configurations. The data samples consisted of image tensors with shape $(3, 3, 480, 640)$ (`int8`) and action vectors of shape $(50, 14)$ (`float32`). We report the mean wall-clock time averaged over 100 iterations.

2) Data Loading Throughput: We analyzed the system latency using the `xVLA` model (0.9B parameters) as a case study(Table III). With `bfloat16` precision, the model weights occupy 1.7 GB. The robot operates at a control frequency of 30 Hz. Model inference is performed asynchronously with chunked execution, resulting in an inference latency of 30 ms. Similarly, the latency for human teleoperation takeover aligns with the control loop at 30 ms. Model training was conducted on an NVIDIA RTX 4090D GPU, where a single training iteration (forward and backward pass, batch size $= 1$) requires 90 ms.

TABLE III
SYSTEM LATENCY BREAKDOWN. THE EVALUATION INVOLVES 500 ALTERNATING STEPS OF MODEL INFERENCE AND HUMAN INTERVENTION, FOLLOWED BY 3000 TRAINING STEPS.

Stage	Description	Time
Data Collection & Storage		
Rollout	500 steps (Inference/ Intervention)	15 s
Persistence	Saving rollout data to disk	5 s
Cloud Training (3000 Steps)		
Loading	Loading data from disk	75 s
Computation	Forward & Backward pass	270 s
Deployment		
Sync	Weight sync to edge	15 s
Total Loop Latency		
Sync.	Sequential: $T_{roll} + T_{train} + T_{sync}$	380 s
Async.	Parallel: $\max(T_{roll}, T_{train}) + T_{ovhd}$	**345 s**

C. Multi-Machine Experiment

In this subsection, we present the results of the multi-machine experiments conducted alongside single-machine tests. The objective was to evaluate the scalability and efficiency of our proposed framework under varying operational conditions.

In this study, we utilized the ACT model with a ResNet-18 backbone as the underlying architecture. The optimization was carried out using the Adam optimizer, configured with an action chunk size of 30 and a learning rate of 0.0001. Both the single-machine and multi-machine training setups were conducted for a total of 15,000 steps. This configuration allowed us to thoroughly assess the performance of our approach across different experimental conditions.

Table IV presents a comparison of the experimental results obtained from the single-machine and multi-machine setups. Specifically, it reports four key metrics: the number of online samples, training time, open-loop validation error, and task success rate.

- For the **Online Samples**, the single-machine setup produced 20 samples, while the multi-machine setup doubled this to 40 samples, indicating an advantage in sample availability with the multi-machine approach.
- In terms of **Training Time**, the single-machine configuration required 3150 seconds to complete training, whereas the multi-machine configuration took slightly longer at 3160 seconds, suggesting that the time overhead is relatively comparable between the two setups.
- The **Open-Loop Validation Error** showed that the single-machine model had an error rate of 0.02075, compared to a significantly lower rate of 0.01130 for the multi-machine model, highlighting the improved accuracy achieved with the multi-machine training.
- Lastly, the **Task Success Rate** indicated that the single-machine approach yielded a success rate of only 10%, while the multi-machine method achieved a notable success rate of 70%, demonstrating its effectiveness in accomplishing the task objectives.

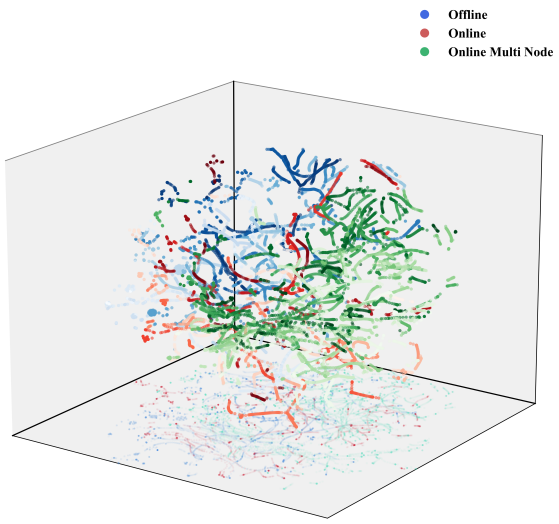

Fig. 5. 3D t-SNE visualization of ScaRLS data collection, illustrating the data distribution across different acquisition methods. Blue represents offline samples, red represents online samples, and green indicates online multi-node samples.

Overall, the results presented in this table emphasize the substantial advantages of multi-machine training in terms of sample handling capacity, model accuracy, and task success rate, providing strong support for the findings of this study.

TABLE IV
COMPARISON OF SINGLE AND MULTI-MACHINE EXPERIMENTAL RESULTS

Metric	Single-Machine	Multi-Machine
Online Samples	20	40
Training Time	3150s	3160s
OpenLoop Validation Error	0.02075	0.01130
Task Success Rate	10%	70%

The 3D t-SNE visualization provides critical insights into the data distribution associated with the ScaRLS data collection. Notably, the online acquisition method yields a richer data distribution, as evidenced by the diverse clustering of data points represented in red. This variety signifies a broad spectrum of scenarios encountered during training, which is essential for developing robust models.

In contrast, when assessing the multi-machine setup (depicted in green), we observe that it not only retains this enhanced data distribution but amplifies it. By enabling data collection from multiple nodes simultaneously, the multi-machine configuration facilitates a broader range of data modalities within the same timeframe. This richness in data is directly linked to the observed improvements in task success rates. The concurrent processing of information in multi-machine scenarios provides a significant edge over single online collections, translating into a marked enhancement in performance for real-world applications.

Overall, these observations underscore the benefits of employing online data collection and multi-machine configurations in cultivating more effective and adaptable models.

Analysis: As demonstrated in the results, the multi-machine setup significantly decreased the OpenLoop validation error to 0.01130 compared to 0.02075 for the single-machine configuration. Moreover, the task success rate increased from 10% to 70%, indicating a marked improvement in performance due to the parallel capability of multiple machines.

This clear differentiation between single and multi-machine setups illustrates the efficiency of parallelization in enhancing overall system performance.

V. Discussion

1) Sources of Efficiency: Multi-robot parallel exploration not only accelerates data acquisition but also enhances data diversity through interactions within heterogeneous environments. This facilitates the learning of policies with superior generalization capabilities.

2) System Overhead: While communication and synchronization overheads remain negligible with a small number of robots, further optimization is requisite as the system scales up.

3) Role of Human Intervention: In both single-agent and multi-agent learning scenarios, human intervention effectively guides the exploration process, mitigates ineffective interactions, and thereby elevates the quality of collected samples.

4) Scalability for Large Models: To accommodate computationally intensive large models, the system's local computational workstation can be seamlessly substituted with high-performance cloud computing infrastructure.

5) Comparison with Contemporary RL Training Frameworks: The primary distinction between our proposed system and general frameworks (e.g., RLlib) lies in the unified consistency between edge and cloud environments. Our framework ensures that model rollout and training can be executed interchangeably on either the cloud or the edge. In contrast, conventional architectures typically employ edge/cloud rollout combined with cloud-based training. This separation often introduces inconsistencies—particularly critical for online learning—stemming from discrepancies in edge-cloud environments and inference operators.

6) Comparison with DAgger: Relative to the standard DAgger algorithm, our approach incorporates distributed computing capabilities, enabling scalable and parallelized data aggregation.

VI. Conclusion and Future Work

In this work, we presented ScaRLS, a scalable robotic learning framework designed to facilitate seamless expansion from single-agent to multi-agent configurations. By leveraging distributed experience collection and collaborative training, the system significantly enhances both the sample efficiency and generalization capabilities of robotic manipulation policies. Future directions include

- Heterogeneous Robot Coordination: Extending the framework to support collaborative tasks among heterogeneous robotic fleets;
- Hybrid Sim-and-Real Training: Incorporating parallel training pipelines that simultaneously utilize simulation and real-world environments;
- Multi-Task and Transfer Learning: Investigating multi-task joint learning architectures and advanced mechanisms for skill transfer.

VII. Conclusion

ScaRLS represents a significant advancement in the quest for scalable robotic learning systems. Future work will explore heterogeneous coordination among robotic fleets, hybrid training methods combining simulation and real-world data, and mechanisms for multi-task and transfer learning.

References

[1] Michael Ahn, Debidatta Dwibedi, Chelsea Finn, Montserrat Gonzalez Arenas, Keerthana Gopalakrishnan, Karol Hausman, Alex Irpan, Nikhil J Joshi, Ryan Julian, Sean Kirmani, et al. Autort: Embodied foundation models for large scale orchestration of robotic agents. In *First Workshop on Vision-Language Models for Navigation and Manipulation at ICRA 2024*, 2024.

[2] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π0: A visionlanguage-action flow model for general robot control, 2024a. *URL https://arxiv. org/abs/2410.24164*, 2024.

[3] Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y Galliker, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *9th Annual Conference on Robot Learning*, 2025. URL https://openreview.net/forum?id=vlhoswksBO.

[4] Joseph Oluwatobiloba Bolarinwa, Manuel Giuliani, and Paul Bremner. Should we get involved? impact of human collaboration and intervention on multi-robot teams. *Frontiers in Robotics and AI*, 12:1526287, 2025.

[5] Hongpeng Chen, Shufei Li, Junming Fan, Anqing Duan, Chenguang Yang, David Navarro-Alarcon, and Pai Zheng. Human-in-the-loop robot learning for smart manufacturing: A human-centric perspective. *IEEE Transactions on Automation Science and Engineering*, 22:11062–11086, 2025. doi: 10.1109/TASE.2025.3528051.

[6] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Robotics: Science and Systems*, 2023. URL https://doi.org/10.15607/RSS.2023.XIX.026.

[7] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*,

pages 1861–1870. Pmlr, 2018. URL https://proceedings.mlr.press/v80/haarnoja18b.html.

[8] Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: Scalable, robust world models for continuous control. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Oxh5CstDJU.

[9] Minho Heo, Youngwoon Lee, Doohyun Lee, and Joseph J Lim. Furniturebench: Reproducible real-world benchmark for long-horizon complex manipulation. *The International Journal of Robotics Research*, 44(10-11):1863–1891, 2025.

[10] Ryan Hoque, Lawrence Yunliang Chen, Satvik Sharma, Karthik Dharmarajan, Brijen Thananjeyan, Pieter Abbeel, and Ken Goldberg. Fleet-dagger: Interactive robot fleet learning with scalable human supervision. In *Conference on Robot Learning*, pages 368–380. PMLR, 2023.

[11] Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Hg-dagger: Interactive imitation learning with human experts. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8077–8083. IEEE, 2019.

[12] Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024.

[13] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard, editors, *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pages 2679–2713. PMLR, 06–09 Nov 2025. URL https://proceedings.mlr.press/v270/kim25c.html.

[14] Kun LEI, Zhengmao He, Chenhao Lu, Kaizhe Hu, Yang Gao, and Huazhe Xu. Uni-o4: Unifying online and offline deep reinforcement learning with multi-step on-policy optimization. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=tbFBh3LMKi.

[15] Kun Lei, Huanyu Li, Dongjie Yu, Zhenyu Wei, Lingxiao Guo, Zhennan Jiang, Ziyu Wang, Shiyu Liang, and Huazhe Xu. Rl-100: Performant robotic manipulation with real-world reinforcement learning, 2025. URL https://arxiv.org/abs/2510.14830.

[16] Zhaoxing Li, Yue Wang, Wenbo Wu, Yanran Xu, and Sebastian Stein. Hmcf: A human-in-the-loop multi-robot collaboration framework based on large language models. In *International Conference on Principles and Practice of Multi-Agent Systems*, pages 150–167. Springer, 2025.

[17] Huihan Liu, Soroush Nasiriany, Lance Zhang, Zhiyao Bao, and Yuke Zhu. Robot learning on the job: Human-in-the-loop autonomy and learning during deployment. *The International Journal of Robotics Research*, 44(10-11):1727–1742, 2025.

[18] Jianlan Luo, Zheyuan Hu, Charles Xu, You Liang Tan, Jacob Berg, Archit Sharma, Stefan Schaal, Chelsea Finn, Abhishek Gupta, and Sergey Levine. Serl: A software suite for sample-efficient robotic reinforcement learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 16961–16969, 2024. doi: 10.1109/ICRA57147.2024.10610040.

[19] Jianlan Luo, Charles Xu, Jeffrey Wu, and Sergey Levine. Precise and dexterous robotic manipulation via human-in-the-loop reinforcement learning. *Science Robotics*, 10 (105):eads5033, 2025.

[20] Yecheng Jason Ma, William Liang, Hung-Ju Wang, Yuke Zhu, Linxi Fan, Osbert Bastani, and Dinesh Jayaraman. DrEureka: Language Model Guided Sim-To-Real Transfer. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, July 2024. doi: 10.15607/RSS.2024.XX.094.

[21] Oier Mees, Dibya Ghosh, Karl Pertsch, Kevin Black, Homer Rich Walke, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, et al. Octo: An open-source generalist robot policy. In *First Workshop on Vision-Language Models for Navigation and Manipulation at ICRA 2024*, 2024.

[22] Mitsuhiko Nakamoto, Simon Zhai, Anikait Singh, Max Sobol Mark, Yi Ma, Chelsea Finn, Aviral Kumar, and Sergey Levine. Cal-ql: Calibrated offline rl pre-training for efficient online fine-tuning. *Advances in Neural Information Processing Systems*, 36:62244–62269, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/c44a04289beaf0a7d968a94066a1d696-Paper-Conference.pdf.

[23] Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903. IEEE, 2024.

[24] Tony Z Zhao, Jonathan Tompson, Danny Driess, Pete Florence, Seyed Kamyar Seyed Ghasemipour, Chelsea Finn, and Ayzaan Wahid. Aloha unleashed: A simple recipe for robot dexterity. In *8th Annual Conference on Robot Learning*, 2024.

[25] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR, 2023.