Robo-DM: Data Management For Large Robot Datasets

[†]Kaiyuan Chen[1], Letian Fu[1], David Huang[1*], Yanxiang Zhang[1*], Lawrence Yunliang Chen[1], Huang Huang[1], Kush Hari[1], Ashwin Balakrishna[2], Ted Xiao[2], Pannag R Sanketi[2], John Kubiatowicz[1], Ken Goldberg[1,3]

***Abstract*— Recent results suggest that very large datasets of teleoperated robot demonstrations can be used to train transformer-based models that have the potential to generalize to new scenes, robots, and tasks. However, curating, distributing, and loading large datasets of robot trajectories, which typically consist of video, textual, and numerical modalities - including streams from multiple cameras - remains challenging. We propose Robo-DM, an efficient open-source cloud-based data management toolkit for collecting, sharing, and learning with robot data. With Robo-DM, robot datasets are stored in a self-contained format with Extensible Binary Meta Language (EBML). Robo-DM can significantly reduce the size of robot trajectory data, transfer costs, and data load time during training. Compared to the RLDS format used in OXE datasets, Robo-DM's compression saves space by up to 70x (lossy) and 3.5x (lossless). Robo-DM also accelerates data retrieval by load-balancing video decoding with memory-mapped decoding caches. Compared to LeRobot, a framework that also uses lossy video compression, Robo-DM is up to 50x faster. In fine-tuning Octo, a transformer-based robot policy with 73k episodes with RT-1 data, Robo-DM incurs 2.6% increase at training performance loss. We physically evaluate a model trained by Robo-DM with lossy compression, a pick-and-place task, and In-Context Robot Transformer. Robo-DM uses 75x compression of the original dataset and does not suffer reduction in downstream task accuracy. Code and evaluation scripts can be found on website `https://github.com/BerkeleyAutomation/fog_x`.**

I. Introduction

Recent work [1–7] suggests Vision-Language-Action models [1, 4, 5] can enhance robot capabilities and generalization in handling multiple settings in diverse environments. A key ingredient for large model training is large and well-curated datasets of teleoperated robot demonstration trajectories such as the Open-X Embodiment (OXE) dataset [2]. However, the current management of robot data is inefficient [2]. Each robot demonstration consists of sequences of actions and observations, making the learning samples much larger and of diverse structrure compared to the images or text tokens in VLMs [8–12] and LLMs [13, 14]. At tera or even penta-byte scale, which is sometimes characterized as Big Data [15], existing robot data storage methods can be inefficient. We propose Robo-DM, an efficient data format with a toolkit for robot data collection, management, and training.

A typical robot dataset includes a number of *episodes*, a sequence of actions performed by an agent from a starting state to a terminal state. Each episode contains multiple sensor data streams in addition to language instructions and other metadata such as robot, task, environment, and control scheme specifications. The size of a typical episode ranges from 1 MB to 400 MB, depending on the episode length, compression level, number of cameras, and camera resolution. Data streams may be recorded at different sampling rates. Episodes are typically stored as a sequence of matrices; for example, data collection with DROID [6] automates data storage with Hierarchical Data Format 5 (HDF5) [16], a format that supports hierarchical storage of matrices. OXE uses Reinforcement Learning Datasets (RLDS) [17], an extension of Tensorflow Datasets (TFDS) to store reinforcement learning demonstrations. Storing image and sensor data directly in matrices is lossless but not space efficient. One emergent framework, LeRobot [18], provides a platform to share robot models and datasets based on lossy video compression and HuggingFace datasets. However, its file structure is complex and loading is generally slower due to decoding.

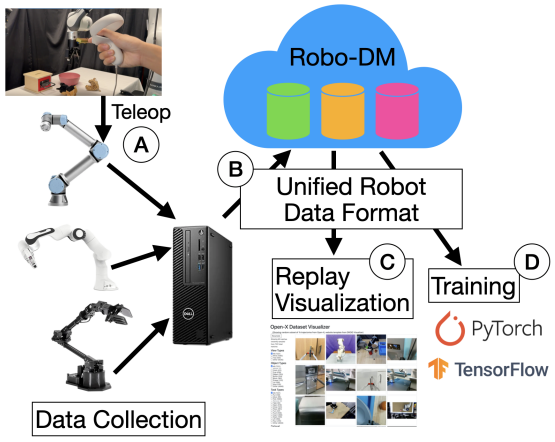

Fig. 1: Robo-DM can streamline robot data collection, management, and learning. (B) Robo-DM uses a unified format for vision, language, and action that does not rely on assumptions about timestamps and data, and supports plug-and-play data collection to integrate with existing setups. (C) Robo-DM can facilitate replay and visualization. (D) Existing training frameworks can load from Robo-DM efficiently with minimal modification.

We observe the following challenges in robot data collection and usage:

(A) Transmission Efficiency: Distributing robotics datasets is costly. Cloud service providers, such as Google Cloud Platform (GCP) and Amazon Web Services (AWS), charge the *data host* for both data storage and outbound data transfers. The cost of transferring data often exceeds the cost of storing it. For example, storing 8.9 TB of Open-X data on Google Cloud costs 172 US dollars per month, but *every full download*

[1]University of California, Berkeley
[2]Google Deepmind
[3]Department of Industrial Engineering and Operations Research
[†]For correspondence and questions: kych@berkeley.edu

costs between 172 US Dollars and 1,540 US dollars. [1] Directly training with cloud storage requires repeated downloads if the local storage cannot store the full dataset, further increasing network traffic and cost to the *data host*. Thus, improving data compression and transmission efficiency can reduce host costs and encourage public sharing of datasets.

(B) Usability and Simplicity: Existing robot data frameworks impose restrictions on file structure, data layout, semantics, and alignment. In particular, hybrid approaches rely on framework-specific assumptions to handle multiple formats simultaneously. Extending the current framework or migrating between frameworks can be challenging, resulting in complex structure and file organization. Figure 2 shows a comparison of different storage formats.

(C) Data Loading Performance: Large robot datasets are typically loaded into computationally training applications. In training, decoded frames are frequently reused and randomly accessed, and the decoded data is loaded on demand. Existing frameworks that use heavy compression sometimes lead to high computational resource utilization and interfere with the training performance. Thus, an efficient and perform ant data-loading framework should utilize available resources without contention.

We introduce Robo-DM, an efficient cloud-based toolkit for collecting, sharing, and learning with robot data. Robo-DM streamlines storage for vision, language, and action data via a unified container format with Extensible Binary Meta Language (EBML). Robo-DM efficiently orchestrates heterogeneous data streams, supporting flexible lossless compression and lossy compression for enhanced transmission efficiency. In prior work, LeRobot [18] empirically evaluates how lossy video compression parameters in FFmpeg affect robot policy accuracy. Octo is also pre-trained by compressing image frames in OXE to lossy images [1]. Robo-DM improves the data loading performance for training workloads, which requires repetitive data access by using memory-mapped caching for faster data retrieval and loading. Loading from cache and decoding are load-balanced to maximize the utilization of compute, memory and storage resources. Robo-DM requires minimal integration effort with existing frameworks. It supports plug-and-play data collection, training, replay, and visualization with mainstream frameworks, and can also be easily exported to other formats such as HDF5 and RLDS.

Experiments suggest that Robo-DM can reduce the size of data by up to 70 times with lossy compression compared to how Open-X-Embodiment currently shares the dataset, and up to 50x faster than LeRobot, a framework that also uses lossy video compression to encode vision data. We fine-tune Octo, a transformer-based robot policy trained with an 800k Open-X-Embodiment dataset with 74k training episodes from RT-1. Robo-DM reduces the dataset size by 4.39 times, while being 3.0 times faster in small batch size (data loading intensive) and does not introduce any slowdown in the training pipeline with large batch size (compute intensive).

[1]The rate is calculated with the egress network traffic pricing in Google Cloud Platform (GCP), where the Open-X-Embodiment dataset is hosted. We use the size of Open-X v1.1 dataset with 8,964 GB in total. The rate differs by the downloading source and destination region. The rate does not consider retransmission of lost packets, so the actual cost is higher than the estimation.

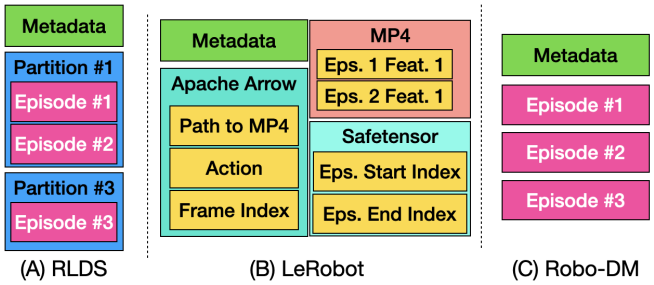

Fig. 2: **A File Structure Comparison of RLDS, LeRobot and Robo-DM** All formats include metadata, storing descriptive information such as authors and dataset summary. (A) Reinforcement Learning Dataset (RLDS) stores episodes in partitions, where each partition is a Tensorflow Dataset Record file. All streams in episode data are compressed matrices that can be directly loaded and trained in Tensorflow. (B) LeRobot combines three formats for robot data. For vision data, it uses one MP4 per video stream in an episode, and uses HuggingFace Dataset (with Apache Arrow as backend[19]) to store language and action streams and the path to the MP4 files. It also uses safetensors [20] to store episode information. All the streams are scattered: to extract an episode, the framework needs to query safetensors for episode information - which is used to find the rest of the non-video streams in the HuggingFace Dataset - and finally use the frame information from the HuggingFace Dataset to find the corresponding MP4 files for vision streams. (C) In Robo-DM, robot data in all the episodes are stored and aligned in a self-contained format. To load an episode, one simply reads from Robo-DM files and load as trainable matrices.

This paper makes the following contributions: (1) an extension of EBML to define a container format that unifies time-based robot data storage; (2) Robo-DM, a framework with 6 new features using this container format; (3) Experimental data that suggests Robo-DM can significantly reduce dataset size, improve loading speed, and incur marginal training performance degradation.

II. Related Work

Big Robot Data The robot learning community is actively building a number of open-source robot learning datasets [3, 21, 22]. Recent work, such as Octo [1], Open-VLA [5], are trained on large datasets such as RT-1 [3], RT-2 [4], Open-X-Embodiment [2], Distributed Robot Interaction Dataset (DROID) [6]. Their initial results suggest training with large and diverse robotics datasets can enhance robot capabilities and generalization in handling multiple settings in diverse environments. In this work, we present an efficient data pipeline for managing large and diverse robot datasets.

Robot Data Frameworks Existing frameworks for collecting, managing and storing robot data fall into the following three categories: (1) Serialized Log format that preserves timing information. This allows users to directly replay the data, e.g. with the official ROS2 tool, rosbag [23]. (2) Matrix format that can be directly supported by training infrastructrue. For example, DROID [6] automates data storage with HDF5 Hierarchical Data Format (HDF5) [16] and existing OXE datasets use RLDS, an extension of Tensorflow Datasets (TFDS) [24] that store and retrieve the interaction between an agent and an environment with observation, action and reward.

Storing image and sensor data directly in matrices limits the capability of compression, and is thus not space efficient. (3) Hybrid formats that store different features in separate files and require assumptions on how different features are aligned and synchronized, such as LeRobot [18], a platform to share robot models and datasets based on HuggingFace datasets.

Cloud and Fog Robotics Fog Robotics [25] utilizes cloud and edge resources for robotics applications. Existing Fog and Cloud robotics focus on deployment of robotics applications, such as grasp planning [26], motion planning [27], visual servoing [28], and human-robot interaction [29]. FogROS2 [30] automates cloud compute resources for robotics, addressing issues such as connectivity [31], latency [32], and cost [33]. We recognize the cost of the cloud required to distribute large robot datasets, and study how formats affect robotics learning in data collection, loading and management.

III. Robo-DM Features

Six novel features differentiate Robo-DM from existing robot data frameworks:

(1) Self-Contained Robot Data Storage: Robo-DM uses a self-contained file format that integrates and stores heterogeneous robot data streams, ensuring all necessary data is consolidated within a single file.

(2) Vision, Language, Action Data Orchestration: The format of Robo-DM allows diverse binary robot data streams, including sensor data, environment specifications, language instructions, and kinematic controls.

(3) Data Flexibility: Robo-DM is extensible for new different data streams, compression algorithms and video encoding formats. For example, Robo-DM enables users to flexibly choose from storing vision data as a sequence of serialized matrices, images, or encoding with lossy or lossless video codecs. With Robo-DM, one can record all the data with original timestamps without resorting to heuristics on data alignment.

(4) Efficient Dataset Size: Robo-DM efficiently encodes heterogeneous time-aligned streams. It uses video compression to significantly reduce the size of file transfer.

(5) Data Loading Efficiency: Robo-DM efficiently loads data by caching decoded frames and balancing resource utilization across available hardware.

(6) Simple Data Collection, Training and Visualization: Robo-DM adopts a concise interface for data collection that is compatible with existing systems with minimal modification. It integrates seamlessly with TensorFlow and PyTorch interfaces, enabling easy adoption. It also allows for exporting of the collected data to existing state-of-the-art data storage frameworks, such as RLDS and HDF5. Robo-DM supports replaying messages through Robot Operating System (ROS) 2, the de-facto standard for developing robotics applications. One can use off-the-shelf ROS2 tools such as rviz [34] or Foxglove [35] to visualize the replayed streams.

IV. Robo-DM Design

A. Unified and Self-Contained Robot Data Format

Robo-DM uses Extensible Binary Meta Language (EBML) [36] for data structuring. EBML is a versatile and extensible markup language that combines the flexibility of Extensible Meta Language (XML) with the efficiency of binary encoding. It organizes binary data elements in a hierarchical structure similar to XML, allowing for nested elements and coherent data management. This enables EBML to handle data streams from different sources within a single container, using self-describing elements that ensure compatibility and future extensibility. A notable application of EBML is in the MKV [37] video container format, which uses it to store multiple video and audio tracks, along with subtitles, in a time-aligned manner within a single container.

Figure 3 illustrates how Robo-DM encapsulates heterogeneous robot data streams. Robo-DM compresses vision streams and serializes robot data into byte packets. A byte packet encapsulates the raw bytes and descriptive information, such as timestamp and stream information. To efficiently replay the data and keep the relative timing information between data streams with different frequencies, all the data packets are stored with a relative timestamp to the beginning of the episode. Robo-DM extends MKV to store robot data to ensure the synchronization of multiple streams on vision, language, and action within the same container.

Data Collection and Post-Processing Compression can be computationally intensive. To prevent interference with the data collection process, Robo-DM uses its file format flexibility to first store all data in raw serialized form. After the data collection is finished, Robo-DM iterates through the collected data, transcodes data that requires compression and re-arranges the collected data (remux) to arrange the data packets. Because training applications sometimes access the episode at a given time frame, Robo-DM groups time-aligned data streams together. On querying a specific frame, metadata is used to identify the related segments and decode the video starting from the latest keyframe before the start of the slice. All the decoded trajectories are cached to speed up future accesses.

B. Transmission-Efficient Storage, Retrieval and Loading

Transmission-Efficient Compression Robo-DM unifies heterogeneous data streams that require different mechanisms for compression and serialization. Because Robo-DM naturally supports byte streams, it is agnostic to mainstream byte compression algorithms and video encoders. For vision data, three channels (red, green, blue) can be compressed with off-the-shelf video compression algorithms, such as H.264 [38], H.265 [39], AV1 [40]. For large matrices that require full precision, such as stereo depth images, users can alternatively choose to compress them with lossless compression algorithms such as FFV1 [41, 42].

Efficient Decoding Cache For sequential access patterns, compression-based algorithms can reduce space usage by decoding all frames in order. When training, decoded frames

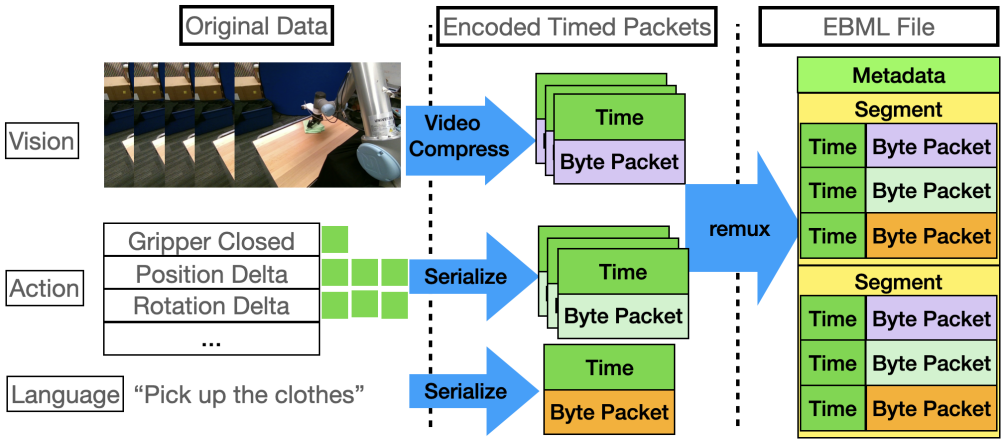

Fig. 3: **How Robo-DM stores an episode of robot data with vision, language and action data** Robo-DM encodes vision, language and action data. For vision data, Robo-DM uses video or image compression; language and action data are serialized into bytes. All the bytes are encapsulated with an intake timestamp. Then Robo-DM multiplexes different streams of data into a self-describing EBML file format.

are frequently reused and randomly accessed, and the decoded data is loaded on-demand. Robo-DM amortizes the random access patterns using memory-mapped files (mmap) [16, 43]. Mmap creates a new mapping in the virtual address space of a process to a cache file. If a slice of data is used, only the portions of the file that are actually used are brought into memory, conserving both I/O bandwidth and physical memory.

Load Balancing For Decoding and Decoding Cache Robo-DM automates the choice of computationally heavy decoding, loading directly cache in memory, and loading the decoded matrices from disk. To prevent overusing a single resource, Robo-DM estimates the potential latency of accessing the data and dynamically balancing the access. Specifically, if the memory resources are underutilized and a prior decoded matrix is available, this means the decoded data is likely in physical memory without being cached to the disk by mmap, and Robo-DM can directly use the decoded cache. In contrast, if the memory is full, cache miss is frequent and the data is not frequently accessed, Robo-DM does not load from cache, and directly decodes the video data instead.

C. Integration with existing Frameworks

Data Collection Interface In order to integrate with custom data collection software stacks, Robo-DM uses a concise programming interface for data collection. Listing 1 shows how the Robo-DM data collection library infers time and the data type from the input vision, action and language data. Due to the simplicity in Robo-DM's data storage format, the data collection library introduces minimal code complexity to the overall custom data collection software stack.

Plug-and-Play Data Collection and Visualization Robo-DM supports integration with ROS2-enabled setups to collect data in a plug and play manner. In ROS2, computational modules, *nodes*, can be deployed on different machines. ROS2 provides an off-the-shelf tool, *rosbag*, to capture data streams from sensors, logs, and various topics during robot operation. Robo-DM supports transcoding from and exporting robot data

```python
import robo_dm

# Data Collection
trajectory = robo_dm.Episode("Autolab-01-02-2024.vla")
trajectory.add(feature = "language_instruction",
    value = "pick up the tiger and place in the bowl")
trajectory.add(feature = "image", value = image)
trajectory.add(feature = "joint_state", value = state)

# Data Loading
trajectory = robo_dm.load(path = "Autolab-01-02-2024.vla")
# [image_1, image_2...]
images = trajectory["image"]
# [joint_state_1, joint_state_2, ...]
joint_states = trajectory["joint_state"]

# Data Exporting
# Support HDF5, RLDS
dataset.export(format = "hdf5")
```

Listing 1: **Code Example** Robo-DM adopts a minimalist data collection, loading and exporting interface that can be easily integrated with existing frameworks.

to rosbag, with all the timing information recorded. Rosbags also can be directly replayed in ROS2. The ROS2 community provides a number of frameworks, such as rviz [34], and Foxglove [35] from the open source community, a browser-based tool that enables visualization of ROS 2 topics. Besides replaying videos, these visualizers also support visualization in 3D, which is helpful for action data such as robot state and motions.

Data Loading Interface To support existing training frameworks with minimal modification, Robo-DM supports accessing robot data in the same way as accessing typical HDF5 files (shown in Listing 1). Robo-DM supports converting robot data to other state-of-the-art formats, such as HDF5 and Tensorflow dataset.

V. Evaluation

Our experiments consider three questions: (1) How does Robo-DM's training data loader compare with state-of-the-art data loaders? (2) How does Robo-DM work with training workloads in terms of data loading speed, space saving, and training performance? (3) Does Robo-DM preserve the policy

Dataset	Dataset Description			Total Dataset Size (GB)				
	# Image Streams	Resolution	Avg. Frames per Episode	Original RLDS	HDF5	Robo-DM-Lossless	LeRobot	Robo-DM
Bridge	1 RGB	(480, 640)	34	387.49 (73x)	779.24 (147x)	114.63 (22x)	16.34 (3x)	**5.31 (1x)**
Cable Routing	3 RGB	(128, 128)	25	4.67 (18x)	7.38 (28x)	1.67 (6x)	0.36 (1.4x)	**0.26 (1x)**
Door Opening	1 RGB	(720, 960)	42	7.12 (71x)	35.35 (354x)	2.89 (29x)	0.38 (4x)	**0.10 (1x)**
AutoLab UR5	2 RGB, 1 Depth	(480, 640)	97	76.39 (23x)	258.33 (88x)	23.45 (7x)	(-)	**3.26 (1x)**

TABLE I: **Dataset information and Size Comparison with Different Formats in Gigabytes (GB)**. Compression ratios differ by the number of image streams and resolution. Robo-DM and LeRobot use lossy compression, while the rest are lossless. Both LeRobot and Robo-DM use AV1 codec with 30 Constant Rate Factor (CRF), a factor that balances compression and decoded video quality. These parameters are suggested by LeRobot video benchmark [44]. (-) LeRobot omits depth stream and some action streams at its conversion from RLDS [17].

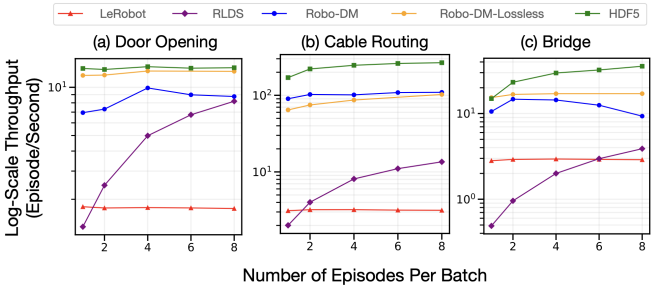

Fig. 4: **Episode Per Second Throughput of Robo-DM on Three OXE datasets with Different Characteristics** We compare Robo-DM with baseline data loading Methods RLDS, HDF5 and LeRobot. Complete episodes are loaded concurrently as a batch, and we record the average throughput with 200 batches.

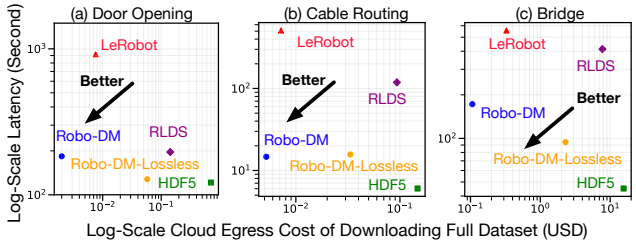

Fig. 5: **Concurrent Loading Latency with respect to Episode Size of Robo-DM** We compare Robo-DM with baseline data loading Methods RLDS, HDF5 and LeRobot. Complete episodes are loaded concurrently as a batch, and we record the average latency of 200 batches with batch size 8 episodes. We use the lowest GCP cost of 0.02 US Dollars (USD) per GB.

performance?

Setup We evaluate Robo-DM with a standard workstation setup: Intel i9-13900K Processor with 96GB RAM and NVidia 4070 Ti Super GPU. The workstation is equipped with 6TB NVMe M.2 SSD with the reading throughput up to 5000 MB/s and writing throughput up to 2500 MB/s. It connects Internet with a 1 Gbps Ethernet connection that can download from Open-X-Embodiment Google Cloud Bucket with 10 Mbps. We make sure the batch can fit in RAM without swap space. The video streams in Robo-DM are decoded with CPU without specialized GPU or additional hardware decoder.

A. Data Loading Benchmarks with Open-X-Embodiment

We evaluate the data loading performance of Robo-DM with a number of exemplar datasets from Open-X-Embodiment (OXE). In the experiments, we concurrently load multiple entire episodes into memory, and we explicitly cast the data into in-memory numpy arrays. We measure the latency of issuing a number of concurrent reads (i.e. a batch) to the time that all the episodes are loaded. For each run, we measure the average latency over 200 data loads.

Datasets We use 1) *Bridge* [22]: two WidowX arms interact with household environments including kitchens, sinks, and tabletops. Skills include object rearrangement, sweeping, stacking, folding, and opening/closing doors and drawers. In the dataset,there are 4 RGB streams and 1 depth stream with 25,460 training episodes. 2) *UC Berkeley Cable Routing*: [45] one Franka robot arm routes a cable through a number of tight-fitting clips mounted on the table with 1,482 training episodes. 3) *NYU Door Opening*: [46] A Hello Stretch robot opens cabinet doors for a variety of cabinets with 435 training episodes. 4) *Berkeley AUTOLab UR5 [47]*: A UR5 robot arm pick-and-place of a stuffed animal between containers, sweeping a cloth, stacking cups with 896 training episodes.

Baselines We compare Robo-DM with the following baselines 1) *RLDS* [17] Open-X-Embodiment is stored and shared in RLDS format. In the evaluation, we directly download and load the datasets with official instructions. 2) *LeRobot* [18] We convert Open-X-Embodiment datasets in LeRobot datasets with the provided official script. Some features in Open-X-Embodiment are omitted in the conversion. We sequentially extract episodes suggested by the example instructions. 3) *HDF5* [16] We use Robo-DM to convert Open-X-Embodiment datasets to HDF5 formats. Since one HDF5 file per trajectory, we implement pre-fetch buffer and pytorch loader with the same setup as Robo-DM. We use a pre-fetch buffer of 50 episodes.

Episode Size Table I shows that Robo-DM significantly reduces file size (18x, 73x, 23x and 73x) per episode compared to the RLDS, a format in which these datasets are originally stored and shared. The episode size reduction leads to high accessibility to large robot datasets, transmission efficiency, and cost efficiency, shown in Figure 5.

Loading Latency Figure 4 compares the throughput difference of Robo-DM compared against LeRobot, RLDS, and HDF5. The lossless version of Robo-DM has similar throughput as Robo-DM. It is faster than LeRobot by 33x, 20x and 5x. Robo-DM is slower than HDF5 because the HDF5 data is uncompressed and loaded in high disk throughput.

Limitation Because Robo-DM extensively uses RAM as a decoding cache to prevent repetitive decoding of the data, it leads to higher RAM usage and potentially degrades the performance when the per-episode data is large. For example,

for the bridge dataset, we see Robo-DM reduces the overall throughput when the batch size increases. This may lead to performance degradation for decoding for finer granularity, such as only sampling one frame for each episode.

B. Case Study: Fine-tuning Octo with Robo-DM

Octo [1] is a transformer-based robot policy trained on 800k robot episodes from Open-X-Embodiment. We fine-tune the pre-trained Octo-small model with 25.6M trainable parameters. We fine-tune the entire model conditioned with both images and language instructions. For each configuration, we train with 50,000 iterations and measure the per-iteration average latency.

Dataset Compression We use RT-1 [3] dataset, a dataset containing 73,499 episodes. The dataset involves picking, placing, and moving 17 objects with Google Robot. The dataset contains 1 RGB video stream with resolution (320, 480). The original dataset is 111.06 GB. The final dataset size of Robo-DM is 36.50 GB with 4.39 times size reduction. The reason why the size reduction is smaller than other datasets from Open-X-Embodiment is that the per-trajectory size is small, with 1.51 MB on average per trajectory in RLDS. Robo-DM needs more space to store metadata for seeking and decoding.

Training Performance We run the training workload with batch size 64. Dataloader in Octo loads from Tensorflow dataloader and Robo-DM and lead to similar data loading latency (0.02 seconds) per iteration and overall training latency (0.10 seconds) per iteration. In validating the effect of lossy compression to the training outcome, we use lossless dataset for validation. The final image-conditioned Mean Squared Error of validation dataset is 1.86 with original lossless data and 1.91 with lossy data. Thus Robo-DM leads to 2.6% increase in validation loss with lossy compression.

C. Case Study: Robo-DM with In-Context Robot Transformer Training

Task We evaluate the training performance of Robo-DM, hypothesizing the lossy compression of Robo-DM, despite a high compression rate, could reduce the accuracy of the trained model. Thus, we evaluate a model trained with 335 human-demonstrated trajectories with the lossy compression of Robo-DM. The trained model is tasked to pick up a stuffed toy tiger. Figure 6 shows the task setup with the Franka Emika robot.

Data We collect 335 human-demonstrated trajectories with one hand camera and one left-side-view camera. All video streams are recorded at resolution (320, 180). The trajectories were originally collected in HDF5 with gzip compression, with a total size of 5.8G. Stored in Robo-DM's format, the dataset with lossless codec leads to 1.7G (3.41x space reduction), and the size of lossy compression is 77MB (75.3x space reduction).

Model We use the ICRT [7], a transformer model that performs autoregressive prediction on sensorimotor trajectories. We train for 200 epochs with image brightness and contrast augmentation and a small proprioception noise ($\mathbf{N}(0, 0.01)$).

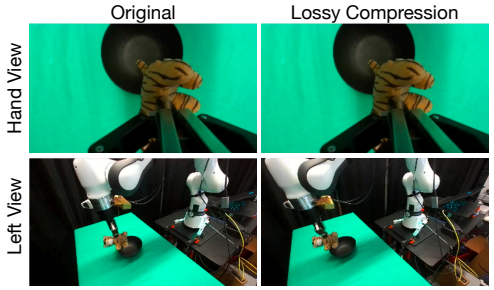

Fig. 6: **ICRT Physical Experiment Setup with Robo-DM** We setup ICRT to pick up a stuffed toy tiger and place it into a black bowl with a Franka Emika robot arm. The Figure shows the view from the left camera and wrist camera used for training, for both the original dataset and reconstructed images from Robo-DM.

Results We randomize the position of the stuffed toy tiger at different places on the tabletop. We evaluate with consecutive 15 trials on the model trained with lossy data. The model is able to reliably identify the object, pick it up, and place it in a bowl with a 15 out of 15 success rate (100%).

VI. Conclusion

In this paper we propose Robo-DM, which includes a new format for robot data, and a toolkit for data collection, management, and loading. Robo-DM significantly outperforms Open-X-Embodiment in terms of space saving. It also shows performant loading speed compared to LeRobot, a framework that also uses video compression. In the task of fine-tuning Octo and policy training, Robo-DM reduces dataset size with minimal training performance and accuracy degradation.

The file size reduction is mainly due to video compression. In future work, we will accelerate video compression and analyze the tradeoffs between parameters. In the evaluation, we used the off-the-shelf video processing library, pyav [48], without GPU acceleration. Recent works such as Decord [49] and GPU acceleration by Nvidia NVDEC [50] are demonstrated to be faster than pyav. Also in future work, we will integrate and evaluate Robo-DM with larger-scale of existing and prospective Open-X-Embodiment datasets.

VII. Acknowledgement

This project benefited from discussions with Peter Schafhalter, Silvery Fu, and You-Liang Tan. This work is supported in part by donations from Google.

References

[1] Octo Model Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, C. Xu, J. Luo, T. Kreiman, Y. Tan, *et al.*, *Octo: An open-source generalist robot policy*, `https://octo-models.github.io`, 2023.

[2] O. X.-E. Collaboration *et al.*, *Open X-Embodiment: Robotic learning datasets and RT-X models*, 2024.

[3] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, B. Zitkovich, *et al.*, "RT-1: Robotics transformer for real-world control at scale," *Robotics: Science and Systems (RSS)*, 2023.

[4] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, B. Zitkovich, *et al.*, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," in *Conference on Robot Learning*, PMLR, 2023, pp. 2165–2183.

[5] M. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, Q. Vuong, T. Kollar, *et al.*, "Openvla: An open-source vision-language-action model," *Conference on Robot Learning (CoRL)*, 2024.

[6] A. Khazatsky, K. Pertsch, S. Nair, A. Balakrishna, S. Dasari, S. Karamcheti, S. Nasiriany, M. K. Srirama, L. Y. Chen, *et al.*, "Droid: A large-scale in-the-wild robot manipulation dataset," in *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024.

[7] L. Fu, H. Huang, G. Datta, L. Y. Chen, W. C.-H. Panitch, F. Liu, H. Li, and K. Goldberg, "In-context imitation learning via next-token prediction," *International Conference on Robotics and Automation*, 2025.

[8] "Gpt-4v(ision) system card," 2023.

[9] Google, "Gemini: A family of highly capable multimodal models," *arXiv preprint arXiv:2312.11805*, 2023.

[10] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," *Advances in neural information processing systems*, vol. 36, 2024.

[11] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, R. Ring, E. Rutherford, *et al.*, "Flamingo: A visual language model for few-shot learning," *Advances in neural information processing systems*, vol. 35, pp. 23 716–23 736, 2022.

[12] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, W. Huang, Y. Chebotar, *et al.*, "Palm-e: An embodied multimodal language model," in *International Conference on Machine Learning*, PMLR, 2023, pp. 8469–8488.

[13] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[14] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[15] S. Sagiroglu and D. Sinanc, "Big data: A review," in *2013 international conference on collaboration technologies and systems (CTS)*, IEEE, 2013, pp. 42–47.

[16] The HDF Group, *Hierarchical Data Format, version 5*, 1997-2024.

[17] S. Ramos, S. Girgin, L. Hussenot, D. Vincent, H. Yakubovich, D. Toyama, A. Gergely, P. Stanczyk, R. Marinier, J. Harmsen, O. Pietquin, and N. Momchev, "Rlds: An ecosystem to generate, share and use datasets in reinforcement learning," *arXiv preprint arXiv:2111.02767*, 2021.

[18] R. Cadene, S. Alibert, A. Soare, Q. Gallouedec, and T. Wolf, *Lerobot: Making ai for robotics more accessible with end-to-end learning*, `https://github.com/huggingface/lerobot`, 2024.

[19] *Apache Arrow*, `https://arrow.apache.org/`, 2024-09-14.

[20] *HuggingFace SafeTensors*, `https://github.com/huggingface/safetensors`, Accessed: 2024-09-14.

[21] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *Conference on robot learning*, PMLR, 2018, pp. 651–673.

[22] H. Walke, K. Black, A. Lee, M. J. Kim, M. Du, C. Zheng, T. Zhao, P. Hansen-Estruch, Q. Vuong, A. He, V. Myers, K. Fang, *et al.*, "Bridgedata v2: A dataset for robot learning at scale," in *Conference on Robot Learning*, PMLR, 2023, pp. 1723–1736.

[23] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "ROS: An open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, 2009.

[24] *TensorFlow Datasets, a collection of ready-to-use datasets*, `https://www.tensorflow.org/datasets`.

[25] S. L. K. C. Gudi, S. Ojha, B. Johnston, J. Clark, and M.-A. Williams, "Fog robotics: An introduction," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[26] A. K. Tanwani, N. Mor, J. Kubiatowicz, J. E. Gonzalez, and K. Goldberg, "A fog robotics approach to deep robot learning: Application to object recognition and grasp planning in surface decluttering," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, IEEE, 2019, pp. 4559–4566.

[27] J. Ichnowski, W. Lee, V. Murta, S. Paradis, R. Alterovitz, J. E. Gonzalez, I. Stoica, and K. Goldberg, "Fog robotics algorithms for distributed motion planning using lambda serverless computing," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2020, pp. 4232–4238.

[28] N. Tian, A. K. Tanwani, J. Chen, M. Ma, R. Zhang, B. Huang, K. Goldberg, and S. Sojoudi, "A fog robotic system for dynamic visual servoing," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 1982–1988.

[29] S. L. K. C. Gudi, S. Ojha, B. Johnston, J. Clark, and M.-A. Williams, "Fog robotics for efficient, fluent and robust human-robot interaction," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, IEEE, 2018, pp. 1–5.

[30] K. E. Chen, Y. Liang, N. Jha, J. Ichnowski, M. Danielczuk, J. Gonzalez, J. Kubiatowicz, and K. Goldberg, "FogROS: An adaptive framework for automating fog robotics deployment," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2021, pp. 2035–2042.

[31] K. Chen, R. Hoque, K. Dharmarajan, E. Llontop, S. O. Adebola, J. Ichnowski, J. D. Kubiatowicz, and K. Goldberg, "FogROS2-SGC: A ROS2 cloud robotics platform for secure global connectivity," *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–8, 2023.

[32] K. Chen, M. Wang, M. Gualtieri, N. Tian, C. Juette, L. Ren, J. Kubiatowicz, and K. Goldberg, "FogROS2-LS: A location-independent fog robotics framework for latency sensitive ROS2 applications," *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2024.

[33] K. Chen, K. Hari, R. Khare, C. Le, T. Chung, J. Drake, S. Adebloa, J. Ichnowski, J. Kubiatowicz, and K. Goldberg, "FogROS2-Config: A toolkit for choosing server configuration for cloud robotics," *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2024.

[34] H. R. Kam, S.-H. Lee, T. Park, and C.-H. Kim, "Rviz: A toolkit for real domain data visualization," *Telecommunication Systems*, vol. 60, no. 2, pp. 337–345, 2015.

[35] Foxglove Technologies Inc, *Foxglove*, `https://foxglove.dev`.

[36] S. Lhomme, D. Rice, and M. Bunkus, "Extensible binary meta language," RFC Editor, RFC 8794, Jul. 2020.

[37] *Matroska Video Container*, `https://www.matroska.org/index.html`, Accessed: 2024-09-14.

[38] ITU-T, "Advanced video coding for generic audiovisual services," International Telecommunication Union, Geneva, Switzerland, Recommendation H.264, 2003.

[39] ITU-T, "High efficiency video coding," International Telecommunication Union, Geneva, Switzerland, Recommendation H.265, 2023, Version 9.

[40] Alliance for Open Media, *Av1 bitstream & decoding process specification*, `https://aomediacodec.github.io/av1-spec/`, Accessed: [Insert Date], 2019.

[41] Library of Congress, *Ff video codec 1, version 0, 1 and 3*, `https://www.loc.gov/preservation/digital/formats/fdd/fdd000341.shtml`, Accessed: [Insert Date], 2024.

[42] M. Niedermayer, D. Rice, and J. Martinez, "Ffv1 video coding format versions 0, 1, and 3," RFC Editor, RFC 9043, Aug. 2021.

[43] *Linux mmap(2) Manual*, `https://man7.org/linux/man-pages/man2/mmap.2.html`, Accessed: 2024-09-14.

[44] *LeRobot Video Benchmark*, `https://github.com/huggingface/lerobot/tree/main/benchmarks/video`, Accessed: 2024-09-13.

[45] J. Luo, C. Xu, X. Geng, G. Feng, K. Fang, L. Tan, S. Schaal, and S. Levine, "Multi-stage cable routing through hierarchical imitation learning," *IEEE Transactions on Robotics*, 2024.

[46] J. Pari, N. M. Shafiullah, S. P. Arunachalam, and L. Pinto, "The surprising effectiveness of representation learning for visual imitation," *Robotics: Science and Systems*, 2018.

[47] L. Y. Chen, S. Adebola, and K. Goldberg, *Berkeley UR5 demonstration dataset*, https://sites.google.com/view/berkeley-ur5/home.

[48] *Pyav: Pythonic bindings for FFmpeg's libraries*, `https://github.com/PyAV-Org/PyAV`, Accessed: 2024-09-14.

[49] *Decord: An efficient video loader for deep learning with smart shuffling that's super easy to digest*, `https://github.com/dmlc/decord`, Accessed: 2024-09-14.

[50] *NVIDIA Video Codec SDK*, `https://developer.nvidia.com/video-codec-sdk`, Accessed: 2024-09-14.