Data Factory 1 Technical Report

Tutor Intelligence

Data Factory 1 (DF1) is a 100-robot farm of Sonny bimanual manipulator robots, to our knowledge the largest robot data factory in the United States. DF1 enables large-scale real-world teleoperation, evaluation, and online iterative improvement of robot foundation models for stationary dexterous bimanual manipulation.

DF1 supports two primary modes of online data collection, each facilitated by our international team of Tutors, our teleoperators and offline data labelers, using our *proprioceptive VR remote teleoperation interface* (PTeleop):

1. **1:1 control.** Tutors are matched via a queue to a single robot to perform a task demonstration in 3D space, marking completion at the end of the session.
2. **1:N supervision.** Tutors supervise a large number of robots executing policy rollouts, with the ability to proactively or reactively intervene mid-task with a corrective action.

We also share early results on developing Ti0, Tutor's first vision-language-action model trained entirely on DF1 data across multiple tasks. To train performant manipulation policies from data collected from a heterogeneous set of teleoperators, we introduce *velocity normalization* to preprocess demonstrations via time-based trajectory reparametrization. We also use DF1 for post-training by collecting correction data via human interventions and using our own large-scale human reward annotation system to improve policies based on demonstration quality.

Together, DF1 and Ti0 constitute a step towards building real-world robot fleets that experience compounding policy improvement over time.

Figure 1: **Data Factory 1.** Located at Tutor HQ in Watertown, MA.

1 Design goals and contributions

Large volumes of on-distribution data covering the generality of intended tasks and contexts are a critical pre-requisite to training modern large, multi-task deep neural networks [29, 9, 17]. Frontier models require not only large quantities of offline data for initial unsupervised learning (pretraining) but also leverage real-world rollouts collected and batched at scale [8] to drive continuous improvement beyond the pretraining set. This presents a substantial additional operational and technical challenge in robotics, where real-world rollouts require substantial investment in physical hardware to achieve and maintain high availability and data throughput.

Our prior robot embodiment, Cassie, is deployed in factories and warehouses to perform economically meaningful work, leveraging remote humans-in-the-loop (Tutors) for task demonstration and error recovery and using data from the field to build compounding autonomy over time. This approach creates an economic mechanism by which to scale on-distribution data collection. However, Cassie is only capable of a small fraction of human physical labor, limited both by intelligence (vision transformer affordance-generation feeding classical manipulation pipelines [31]) and hardware (one arm and a suction-based gripper).

Recent breakthroughs in behavior cloning methods for robot manipulation have shown the ability to learn highly multi-modal, dense end-to-end dextrous manipulation policies from pixels to joints without classical planning or control [5, 39]. Later work has demonstrated increased robustness and sample efficiency from multi-task data and policies [36, 1], presenting a vision of data collected across a large number of environments and task sets enabling more robust and general manipulation over time. However, these methods have not yet been widely deployed in real-world productive environments, and to our knowledge no robot running this new class of model is commercially generally available to perform economically valuable work.

We built DF1 as a research tool to bootstrap initial data scale and evaluate learning methods for large-scale deployment of bimanual manipulators to perform economically valuable work in the real world. Deploying human-like robots in a commercial setting requires more than task-level capability: systems must sustain uptime and throughput, support low-latency remote human-in-the-loop intervention, and continuously improve from failures observed during operation. We designed DF1 to satisfy these requirements.

DF1 is designed to satisfy the following deployment constraints:

- **Industrial Robustness and Scale.** Commercial robots must operate continuously at production-rate cycle times. Manipulation quality does not make up for poor uptime or low throughput.
- **Rapid Remote Teleoperation.** Because autonomous policies do not yet achieve consistent success at production speed, non-expert operators must be able to intervene remotely with minimal perceptual and control latency. Teleoperation interfaces that slow down the operators' decisions or require substantial training would invariably limit the effective robot speed.
- **Data Collection and Model Improvement.** Practical systems must support frequent intervention, reliably detect and record failures, and feed correction data back into training to improve policy performance over time.

To address these challenges, we present:

- **Data Factory 1 (DF1):** We build DF1, a 100-humanoid-robot data factory that operates at high uptime and collects up to 1,000 hours per day of expert demonstrations on dexterous tasks including picking, folding, and packing.

- **Proprioceptive teleoperation and PTeleop:** We define *proprioceptive teleoperation* as a teleoperation system where the operator views a real-time 3D rendering of the robot's workspace, there is a one-to-one relationship between the (Euclidean) distance the operator's hands move and the distance the robot's grippers move, and the operator receives physical feedback (e.g., haptics) from scene interactions. We introduce PTeleop, our VR implementation of proprioceptive teleoperation that leverages a point cloud stream with sub-1mm surface resolution and sub-50ms update time to enable production-rate remote manipulation, improves teleoperation speed by 35.8% over standard 2D video VR teleoperation, and enables new operators to meet data-quality standards within 6 hours of training.
- **The Ti0 model:** We introduce Ti0, a VLA trained on early DF1 multi-task data. We improve model learning and policy speed using a technique called velocity normalization and design a reward function on in-house annotation data.

For the remainder of the paper we keep each section as self-contained as possible for ease of reading and thus will discuss prior work when it is relevant.

2 DF1: The Data Factory

Figure 1 depicts DF1, a 100 Sonny robot data factory for large-scale data collection. Our custom-made Sonny robots are industrial humanoid robots that are physically robust, move quickly, and support heavy payloads. The data factory can run continuously to produce data at a massive scale. We use DF1 for both expert data collection for pretraining, and for post-training. Figure 2 shows the scaling of the number of active robots over time.

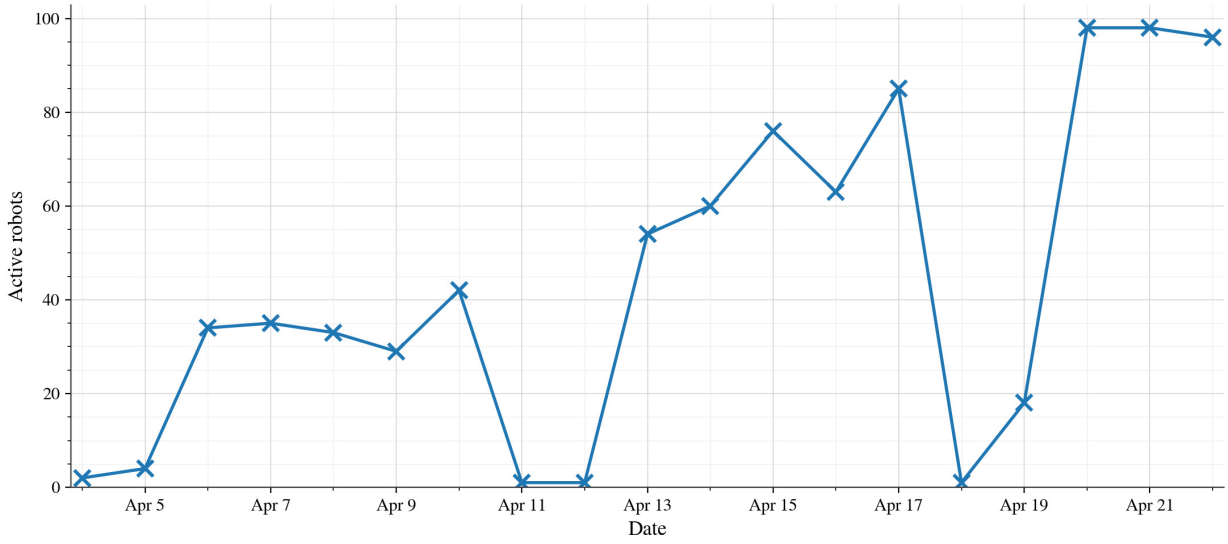


Figure 2: **DF1 active fleet growth.** Active robots over a two-week period in April.

Through our battle-tested Cassie robot embodiment, we solved many common hardware robustness problems such as camera cable failures, motor overheating, and 3D print durability. Our lessons learned here benefit Sonny and enable DF1 to operate at its 100-robot scale without overbearing maintenance requirements.

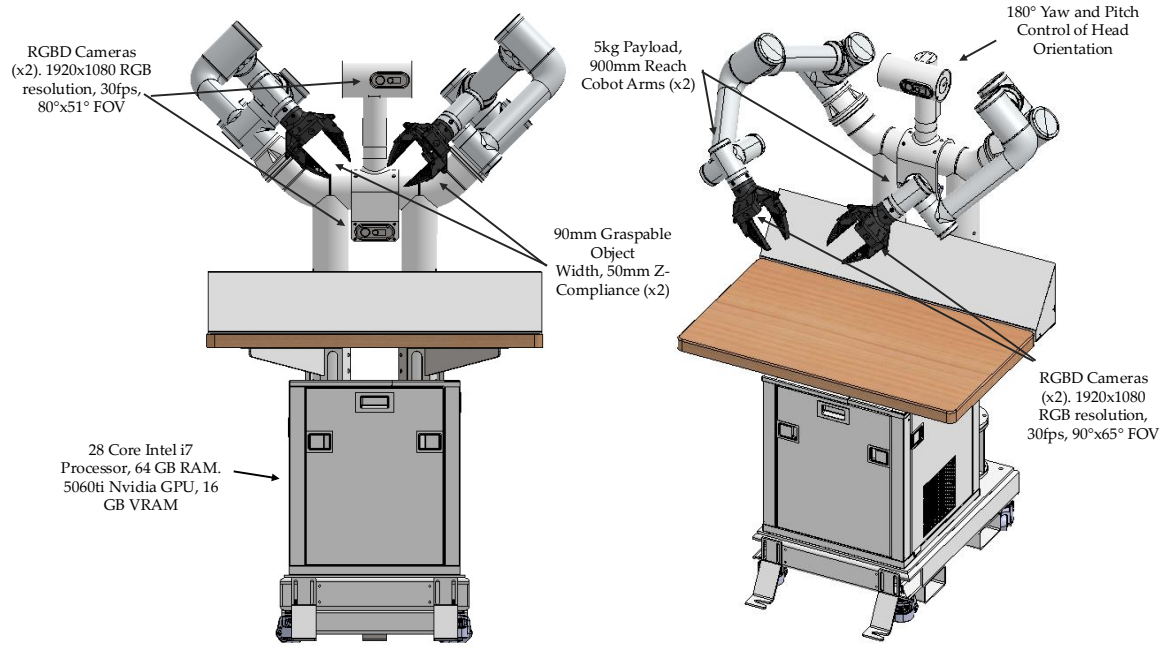


Figure 3: **Sonny.** Sonny is equipped with two 6-DoF arms, each with 5 kg payload capacity and 900 mm of reach, giving it the workspace and payload capacity required for a wide range of bimanual manipulation factory tasks. Two stereo RGBD cameras, one on either wrist, serve as the main drivers for model control, while time-of-flight RGBD cameras on the head and torso work together to provide high-fidelity point clouds that we use for PTeleop. Sonny's grippers are designed using Fin Ray technology [7].

Related work. Researchers have pursued multiple avenues for large-scale data collection. Simulation has shown promise [23] but requires closing the sim-to-real gap [35]. Internet-scale video and egocentric data are plentiful but require bridging the embodiment gap [14]. Portable collection rigs are low-cost and flexible but require solving gripper localization and kinematic feasibility [6]. Real-world robot teleoperation produces on-distribution data with no embodiment gap [22, 25, 1]. DF1 scales real-world teleoperation for industrial robots and requires significant innovation to ensure high-quality, consistent data.

Data quality and consistency. Early model training experiments on DF1 data showed that model performance improves when demonstrations are performed consistently across operators. For a few hundred demonstrations, close communication with robot operators is sufficient, but scaling to 100 continuously operating robots necessitates additional infrastructure. We needed an efficient, tightly integrated system to evaluate and correct individual operators' behaviors, and decided to build an in-house offline data labeling workflow extending our existing Tutor team.

Episode labeling system. After expert demonstrations or autonomous rollouts complete in DF1, episodes are immediately scored by desktop Tutor operators using our offline data labeling workflow *within minutes of the data being collected*, with a total system throughput of *thousands of hours of data per day*. This labeling system is used to close the loop between data collection and quality control by establishing operator-level performance metrics presented as leaderboards, and by giving our quality control staff better visibility into teleoperation mistakes and behavioral drift.

These labels also provide dense supervision for model training, as described in Section 4.3. We find that owning and integrating both collection and labeling is necessary to keep large-scale robot datasets consistent, auditable, and usable for training.

We implement this workflow through a split Tutor organization operating on a shared platform with two interfaces. Our desktop-based Tutor team performs 2D teleoperation on the deployed fleet while also labeling a continuous stream of incoming data in unallocated inter-query time. These Tutors answer task-specific labeling questions, select preferred demonstrations, and assign timestamp-level rewards over tasks and subtasks.

In parallel, our VR-based Tutor team performs real-time 3D demonstrations using PTeleop for behaviors that require precise dexterous manipulation or more advanced recovery actions. Together, these two interfaces let us support deployed robots, collect new demonstrations, and annotate data within a single operational stack split across both research and production.

Most robot learning today relies on manually curated datasets, which scale poorly with fleet size. The labeling system above transforms continuous fleet operation into a continuous training stream—a critical enabler for autonomous, unsupervised improvement at scale.

3 PTeleop: Remote Proprioceptive Teleoperation at Scale

Building a fast, accurate remote teleoperation system is essential for deployment, enabling a single operator to intervene on the execution of different robots in quick succession. Moreover, increasing the speed of execution increases the throughput of DF1 to produce data, and leads to faster policies.

A key reason that remote teleoperation is challenging is that the delay between an operator's movement and the robot's response, combined with delayed visual feedback, creates uncertainty about the robot's state and causes the operator to hesitate. PTeleop addresses these problems by using a virtual 3D scene to communicate to the teleoperator *where the robot is* and *where the robot will be next*, while leveraging the operator's proprioception: their innate sense of the position of their hands and the objects in the scene.

Related approaches. Prior work uses leader-follower rigs to facilitate data collection [11, 39, 38]. In this setup, a teleoperator moves a mimic of the robot, the leader, and another robot, the follower, performs the corresponding trajectory. While these rigs have low latency and guarantee kinematic feasibility, there is a steep learning curve for producing high-quality data, and they typically require an on-site teleoperator, limiting real-world deployability.

To be economically valuable, teleoperation should lead to fast robot movements. A key insight is that in many picking and packing tasks in factories, *most movement occurs in open space*. The speed of these open-space motions therefore dominates teleoperation throughput. As we discuss below, this is precisely where prior systems struggle most.

Point-and-click teleoperation addresses speed by having a user point to a pixel and then using a solver to identify the corresponding robot pose that the robot can move to [16]. This approach has the benefit of fast open-loop behavior, but offloads recovery and obstacle-avoidance behaviors to the solver. Moreover, it introduces unnatural pauses in intermediate motions and requires new scans of the scene when changes occur. For these reasons we do not use point-and-click teleoperation.

Point-and-click teleoperation can be implemented in virtual reality (VR), but VR also supports remote-controller-based teleoperation. In these systems the operator watches a camera stream

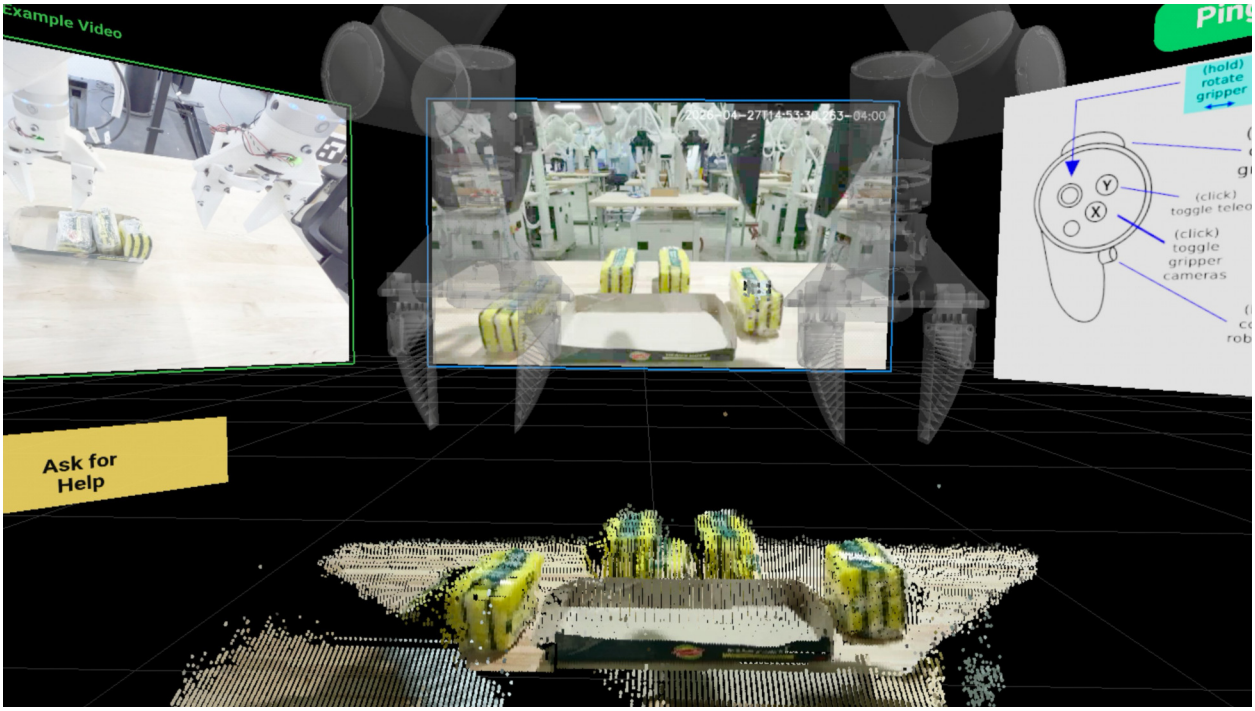


Figure 4: **The PTeleop interface.** RGBD cameras produce real-time point-cloud state with under 50 ms latency.

and moves a game or other controller to issue motor commands to the robot. An advantage is that due to the rise of VR gaming, there is a large pool of expert users. On the other hand, VR teleoperation often results in slow movements, leading to slow policies and less valuable robot labor. Our system, PTeleop, retains the benefits of standard VR teleoperation while addressing the problem of slow teleoperation, by utilizing operator proprioception to speed up intermediate motions. In addition, VR systems support full hand and finger tracking—a path to dexterous manipulation that point-and-click cannot support. For this report we focus on 1-DoF parallel-jaw grasping and leave dexterous control to future work.

The waiting-to-act problem. In conventional 2D teleoperation systems with latency between action and robot response, operators naturally move, wait for visual confirmation that the robot has reached the expected state, and only then continue with the subsequent motion. This behavior turns even modest interface latency into substantial task-level delay, since each movement is broken into a sequence of visually verified sub-actions. We call this the waiting-to-act problem. This pattern significantly increases cycle time, even during simple open-space motions.

Proprioceptive teleoperation and PTeleop. *Proprioceptive teleoperation* is a teleoperation system where the operator views a real-time 3D representation of the robot's workspace, there is a one-to-one relationship between the (Euclidean) distance an operator's hands move and the distance the robot's grippers move, and the operator receives physical feedback (e.g., haptics) from scene interactions. Implementing proprioceptive teleoperation requires a precise mapping between user motion and robot motion, as well as a 3D scene representation that preserves the spatial relationship between the robot, the object, and the target. With these properties, the operator can execute movements in an object-centric way, reaching to the object, acquiring it, and moving it to the destination without pausing for continuous video-based confirmation.

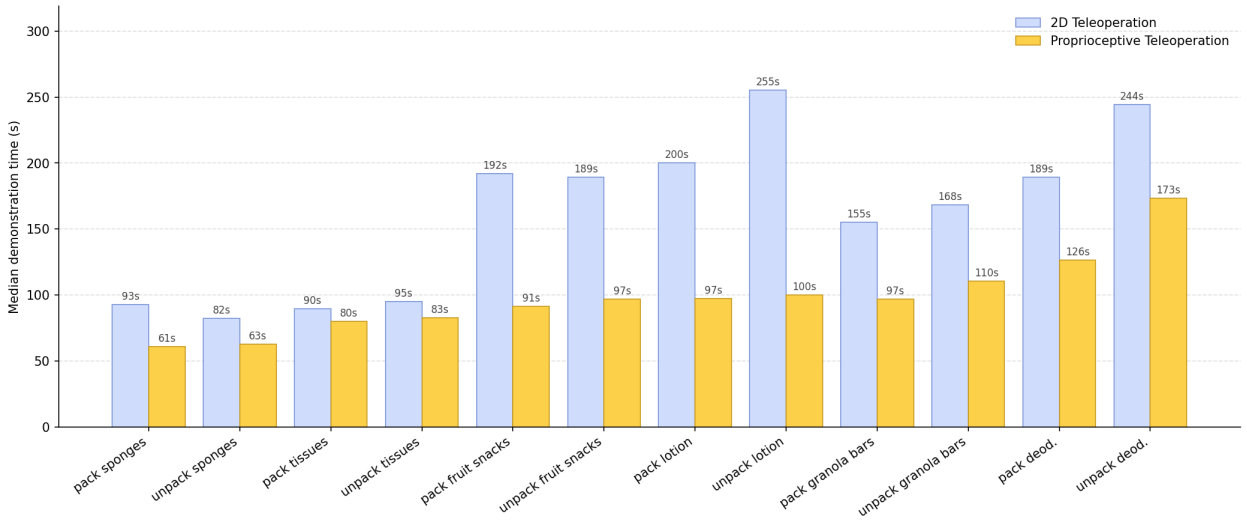


Figure 5: **PTeleop speeds up demonstrations.** Median demonstration time before and after implementing PTeleop. Tasks take less time due to faster motions and fewer mistakes.

We implement PTeleop by rendering a live workspace reconstruction from multi-view point clouds together with a ghost robot that mirrors the state of the physical system in real time. Figure 4 shows the PTeleop interface, where the robot, objects, and scene geometry occupy a shared 3D frame, enabling continuous manipulation without pausing to confirm robot state.

Hardware. PTeleop uses the Meta Quest 3S for teleoperation. Widespread use of the Meta Quest 3S in gaming unlocks low procurement costs, strong software standards support (WebXR), and a pool of hireable international labor familiar with the platform. Using inexpensive, entirely off-the-shelf hardware for our Tutors enables operation at industrial scale: we can easily distribute equipment (either laptops or Quest 3Ss) and find Tutors that already have experience with either desktop or VR gaming.

Results. Figure 5 depicts timing results comparing proprioceptive teleoperation against 2D teleoperation, in which only the camera feed is shown to the operator. Evaluated across twelve different tasks, proprioceptive teleoperation results in over 35% faster teleoperation speeds and over 10% fewer mistakes in demonstrations.

Furthermore, PTeleop provides scene updates within 100 ms, which is fast enough that operators experience their control as effectively closed-loop on tasks such as case packing, folding, and kitting.

Future work. Some more difficult manipulation tasks requiring highly sensitive force application, such as opening jars or screwing in screws, require sub-100 ms reaction loops. Improvements in system-level engineering could further decrease the total latency toward the floor set by network round-trip time. In addition, a richer visual representation beyond raw point clouds, such as neural radiance fields and 3D Gaussian splatting [37], could reduce residual perceptual ambiguity that turns even small latencies into operator hesitation.

4 Ti0: Our Foundation Model Trained with DF1 Data

We built DF1 to support foundation model training and post-training. We provide a report of our first pixels-to-joints robot behavior model, Ti0, trained on one month of DF1 data.

Architecture. We pretrain Ti0 starting from the Qwen3-VL-4B-Instruct backbone [28] and a diffusion transformer action head that predicts action chunks via L2 flow matching [19]. We train the model using the Muon optimizer [13].

Inference. We parameterize Ti0 model inference by the frequency of action execution. We highlight a few techniques we use to speed up model rollouts. First, by using point clouds, our teleoperation system supports highly reactive and precise proprioceptive teleoperation. Second, we use real-time action chunking [2] and asynchronous inference [32], carrying out model inference concurrently with action execution for maximal reactivity. Third, for certain tasks we use action quantization, where the robot only moves on every other predicted action timestep [34]. Lastly, we tune the controller to be appropriately responsive to commanded inputs, depending on the complexity of the task and the base speed of the teleoperated data.

4.1 Learning from correction data

Beyond initial task demonstration, DF1 doubles as an environment where we can deploy our models, identify mistakes that the model makes, issue corrections, and improve performance over time. Concretely, if a model would nominally operate continuously for 8 hours before encountering an edge case, we can instead identify this edge case within 5 minutes of DF1 operation, thanks to the 100x parallelism afforded by the scale of our robot fleet. This rapidly accelerates the rate at which models can be retrained to correct their mistakes.

The end-to-end data generation and quality pipeline is sufficient to produce policies that achieve over 80% success rates on long horizon tasks. However, to produce much higher success rate policies, it is important to collect expert corrections from on-policy rollouts [15].

To this end, PTeleop doubles as a post-training platform, wherein operators can monitor dozens of automated robots at once. As soon as a model hits an edge case (as determined by either a model or a human fleet supervisor), a teleoperator can assume control of that robot within seconds. To an observer, the robot appears to freeze for a moment before figuring out what to do. To the model, this is a vital piece of corrective information that can be used to improve the policy.

4.2 Velocity Normalization

VLA models are typically trained via *behavior cloning*, a supervised learning approach that learns to mimic expert actions by minimizing the difference between predicted and demonstrated actions given observed states. A known problem with behavior cloning is its sensitivity to demonstration speed [30]. For example, two demonstrations of the same pick-and-place task, one performed at twice the speed of the other, may trace nearly identical spatial paths. However, when sampled at a fixed rate (e.g., 20 FPS), they produce dramatically different frame-to-frame action displacements. This forces the VLA policy to reconcile conflicting action labels for visually similar observations, creating harmful multimodality in the training distribution.

We found that this speed-induced multimodality significantly reduces sample efficiency, requiring more demonstration data to achieve the same performance. Given our goal of learning effective policies from minimal data across diverse human teleoperators—each with different natural

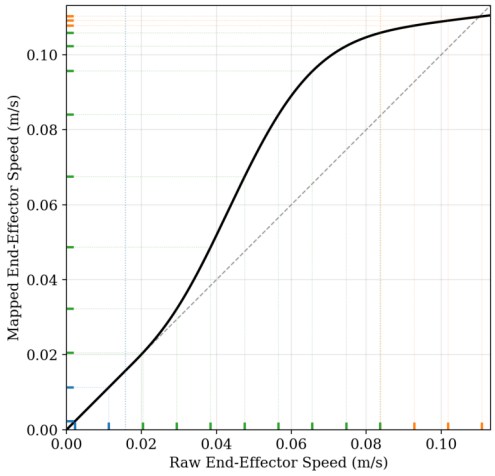


Figure 6: **The velocity-normalization mapping $H(s)$ for the "Pack the Four Sponges" task.** Raw end-effector speeds below the lower threshold m pass through unchanged (blue markers). Frames with end-effector speeds $> m$ are sped up (green markers); those with speeds above M are saturated (orange markers). The blue/green/orange tick markers on the x-axis (equally spaced) project onto the y-axis non-uniformly, illustrating how $H(s)$ clusters varied operator paces around a common mapped speed.

operating speeds—addressing this speed sensitivity becomes critical for practical deployment.

Prior work. Shi et al. [30] define *velocity bias* as the problem where "similar input states are associated with output action chunks exhibiting heterogeneous velocity distributions." Their key insight is that *spatial* variation across demonstrations encodes meaningful task strategies, whereas *velocity* variation poses an obstruction to the model training. However, this approach does not effectively address speed variation *within* a demonstration; a demonstrator moves quickly during transit and slowly during a precision grasp, and this profile differs across demonstrators. Previous works [10, 20] have addressed this variability by conditioning the VLA model on the action execution frequency; however, this effectively requires the model to learn multiple manipulation policies (one for each action execution frequency), slowing down the training process. Tang et al. [34] use temporal-offset augmentation to handle small latencies by training on multiple temporal alignments of the same trajectory. VN addresses an orthogonal problem: large-scale speed variation across different human demonstrators. As such, the two techniques are complementary.

Velocity normalization (VN). VN addresses both inter- and intra-episode speed variation by changing *which frames are selected* during downsampling. Rather than modifying recorded data, VN adjusts the frame-selection so the model perceives a consistent speed distribution across episodes. VN is agnostic to the actual training pipeline, and is specifically a *preprocessing* strategy for training VLA models effectively. It operates in two stages: **Stage 1** aligns *inter-episode* speed (different demonstrators' overall paces) while **Stage 2** compresses the remaining *intra-episode* variation (speed changes within a single demonstration).

Stage 1: Speed Alignment across Human-Demonstrators

Consider the distribution of end-effector speeds within an episode. Empirically, we observe many such distributions to be concentrated around low speeds (associated with fine-grained manipulation), with a long tail at higher speeds (associated with fast, transitional motions); see

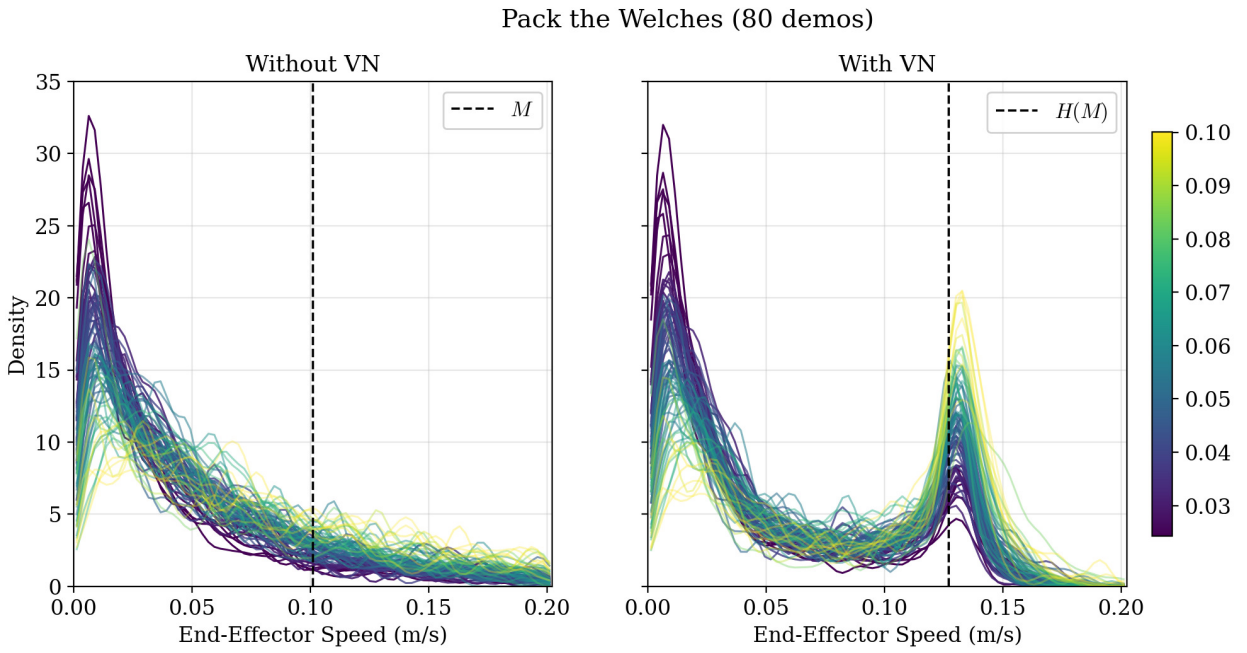


Figure 7: **Distribution of end-effector speeds with and without Velocity Normalization (VN).** We plot each demonstration as a single line. The dashed line in each figure represents the 80th percentile speed M across all demos.

Figure 7. We summarize each episode by its *characteristic speed*: the 30th percentile of its speed distribution, which is an estimate of the cruising pace of the demonstration. Like the *median speed* (which is equal to the 50th percentile speed), the 30th percentile speed is robust to outliers. After computing characteristic speeds, each episode receives a uniform speedup factor: the ratio of the median task speed to the episode's characteristic speed. We clamp this speedup to $[0.75, 1.5]$ to prevent adverse speedups.

Stage 2: Task-Level Speed Curve

Stage 1 applies uniform speedup or slowdown to an episode. Stage 2 marches frame-by-frame through each episode and applies a smooth, monotonic *speed mapping* to each frame. This speed mapping function maps a raw input speed to a desired output speed. It has two key parameters:

- Lower breakpoint (m): median of per-episode characteristic speeds. Speeds lower than m correspond to contact-rich segments of the episode, and must therefore be preserved.
- Upper breakpoint (M): 80th percentile of frame-level speeds, serving as a (soft) upper limit for the downsampled speeds.

Formally, the speed mapping function defines a *diffeomorphism* from $\mathbb{R}^+$ to $\mathbb{R}^+$: a smooth invertible function with a smooth inverse. It maps raw human-demonstration speeds to training (i.e., downsampled) speeds. To preserve fine-grained manipulation segments — where the robot moves slowly and deliberately — the mapping acts as the identity below the cruising speed: these frames pass through unchanged. Above the cruising speed, the curve rises above identity (a slight speedup), then flattens relative to identity (compression), contracting the high-speed region into a narrow output band. The net effect is that diverse input speeds are clustered into a tighter distribution without collapsing them. Figure 6 visualizes this mapping.

Frame-Selection. We apply both stages of VN in a frame-selection loop that marches a cursor through the episode. At each step: apply the Stage 1 speedup/slowdown factor, apply the Stage 2 speed mapping function, and use the result to determine the next selected frame. Fast motion $\implies$ small steps (more frames are selected), slow motion $\implies$ large steps (fewer frames are selected). Output timestamps remain uniform at the target frame rate. This leads to a perceived slowdown or speedup in the training data, respectively, effectively *normalizing* the speed of demonstrations.

Additional Considerations. We further use the telemetry data from the demonstrations to preserve contact-rich segments, and remove idle segments:

- Gripper open/close events receive a 0.5s protection window: frames near open/close events are always sampled at normal rate, as they likely correspond to contact-rich manipulation segments.
- Idle frames (i.e., frames where the robot is inactive and the camera frames do not change beyond a predefined threshold) are skipped. Idle frames at the end of the episode (last 2s) are preserved, so that the trained VLA policy assumes an idle state upon task completion.

Empirical analysis. Figure 7 depicts per-episode speed distributions before and after VN. Before, episodes exhibit very different speed profiles. After, distributions are pulled into tight alignment. Across 30 sampled demonstrations, VN reduced cross-episode speed variance by 62%.

Summary. The key design choices in VN are

- **Frame Selection.** While downsampling raw robot demonstration data, VN selectively skips or over-samples from different segments of the demonstration to slow down or speed up the corresponding motions.
- **Task-Specific Statistics.** VN computes an assortment of robust statistics per task, with the goal of capturing the latent speed profile that is required to complete a given task successfully.
- **Preserve Manipulation Data.** Frames near gripper transitions are protected from compression.

4.3 Reward from human feedback

We study policies that map observations to actions. Following standard reinforcement learning notation, a stochastic policy π assigns probability $\pi(a \mid s, \theta)$ to taking action a in state s as parameterized by θ [33]. Human demonstrations are often referred to as *expert demonstrations*. Because the trained VLA policy is only ever as good as the underlying behavior cloning policy, there is an emphasis on collecting high-quality demonstrations. Training a VLA policy on a dataset of human demonstrations starts from the assumption that humans approximately follow an underlying expert policy $\pi_E(a \mid s)$. The goal of behavior cloning is then to minimize the discrepancy between $\pi(a \mid s, \theta)$ and $\pi_E(a \mid s)$.

However, the process of collecting high-quality expert demonstrations does not scale well. When collecting large amounts of manipulation data, one typically makes do with a mixed-quality dataset that includes non-expert demonstrations. These non-expert demonstrations may be riddled with low-quality frames, such as actions that are irrelevant or detrimental to the completion of the assigned task, and manifest as spurious signals that pollute the behavior cloning dataset.

We address this by modeling demonstration quality as a scalar reward signal, which lets us turn mixed-quality teleoperation data into a reward-weighted imitation objective rather than treating

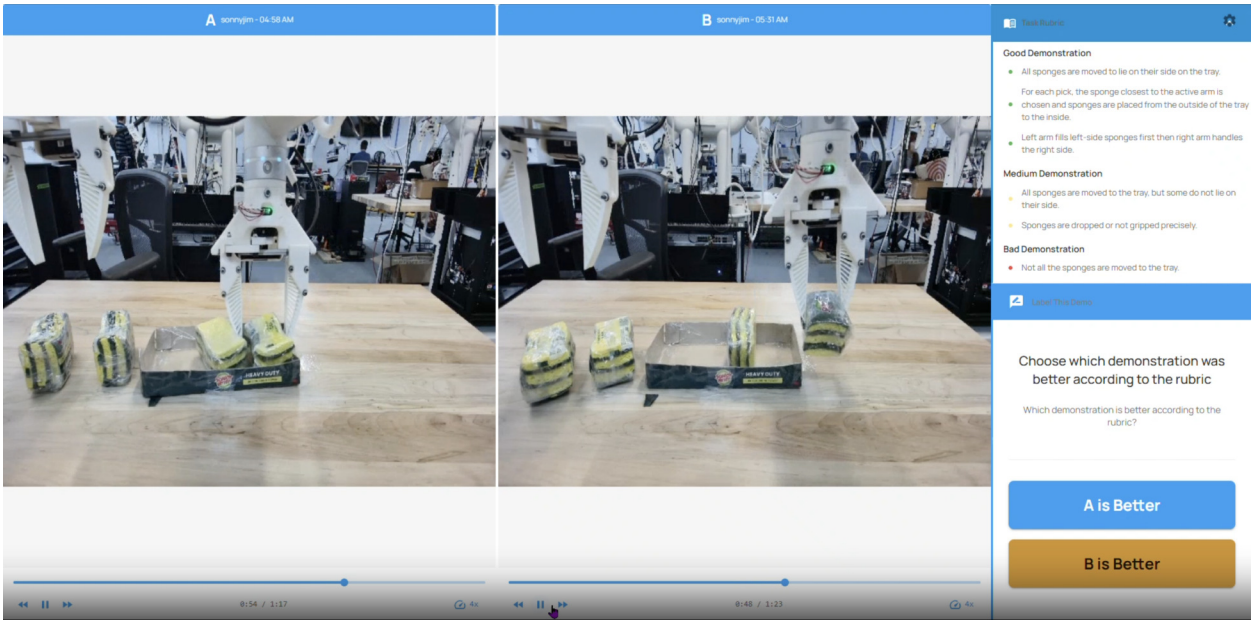


Figure 8: **Reward human data labeling interface.** The labeler is asked to choose which of two demonstrations is better, per a task-specific rubric.

every demonstrated action as equally expert. Human labelers provide structured feedback on recorded demonstrations through a web-based data labeling tool. These labels are fused into a single quality score R per episode, which serves as the *terminal reward* for the episode. The human labelers also provide fine-grained labels that are used to determine progress scores (similar to prior works such as SARM [3], Robometer [18], and $\pi^*_{0.6}$ [27]) as well as for *reward shaping*. The resulting reward trajectory over a demonstration is used to post-train the behavior cloning policy, amplifying the influence of high-quality frames and attenuating the influence of low-quality ones.

In what follows, we describe the labeling interface used to collect human feedback, the procedure for fusing multiple label types into a single quality score, and the reward-shaping step that distributes this score across frames. Figure 8 shows the annotation interface in practice, and Figure 9 summarizes the end-to-end labeling and reward-modeling pipeline.

Label Types. Since demonstration quality is a multifaceted concept, a single labeling modality is insufficient for capturing the holistic quality of robot manipulation within a demonstration. For instance, a robot may complete the task successfully, but exhibit multiple failed grasp attempts that we wish to filter from the behavior cloning dataset. Furthermore, certain label types may be more susceptible to the individual labelers' subjective preferences, making it difficult to interpret the label responses in a consistent and robust manner.

To capture these different aspects of each demonstration, our labeling tool supports five complementary label types, each eliciting a different kind of human judgment. Annotators watch video replays of demonstrations and provide feedback through one or more of the following modalities.

When multiple annotators label the same episode, their scores are averaged. Since not every episode receives every label type, the fusion step (described below) gracefully handles partial coverage by averaging over whichever label types are available for a given demonstration.

Label Fusion. Given a demonstration that has been annotated with one or more reward label

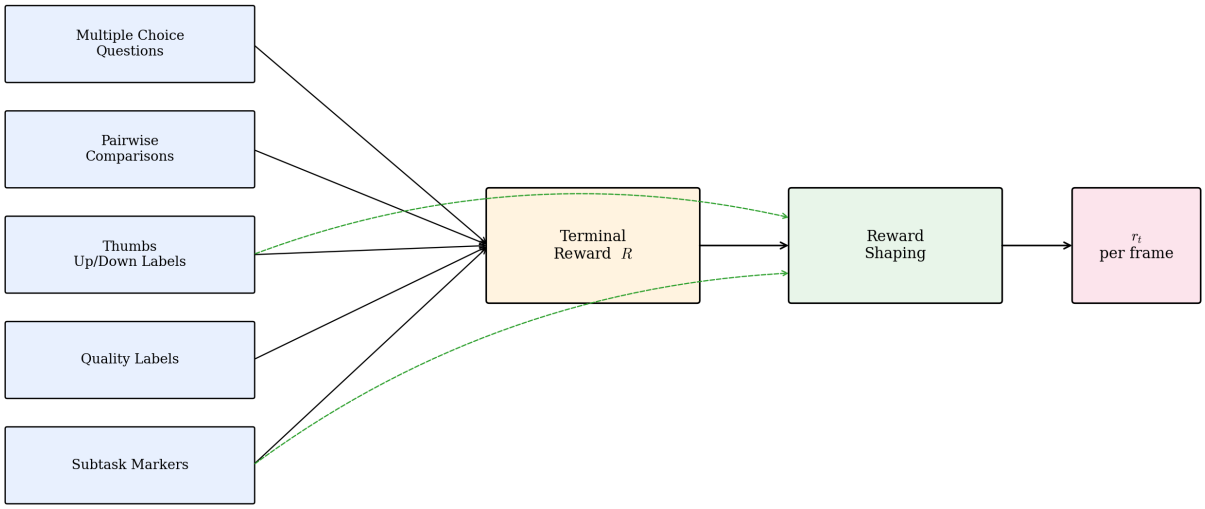


Figure 9: **Reward data labeling and modeling pipeline.** End-to-end overview from raw demonstration to per-frame reward.

types, we wish to compress these annotations into a single scalar *terminal reward* $R \in [0, 10]$. Each label type independently produces a score on the [0, 10] scale (e.g., the multiple choice scores are normalized by the rubric's max possible score). The terminal reward R is defined as the weighted average of these scores across whichever label types are available for the episode:

$$R = \frac{1}{n} \sum_{k=1}^{n} w_k \cdot s_k,$$

where s_k is the score from label type k, w_k is the weight afforded to the k^{th} label type, and n is the number of label types with data for that episode. This design is modular: as more reward label types are introduced or as certain modalities prove more predictive of downstream task success, the weights can be adjusted without changing the pipeline structure. The resulting distribution of terminal rewards is shown in Figure 10.

4.4 Reward Shaping

The terminal reward R tells us *how good* a demonstration is, but not *when* it is good. To produce a training signal at the frame level, we first distribute R across the T frames of the episode to obtain a *progress reward* r_t^{Prog} satisfying $\sum_{t=1}^{T} r_t^{\text{Prog}} = R$. The actual per-frame reward is then defined as

$$r_t = \begin{cases} r_t^{\text{Prog}} - \dfrac{1}{T} \cdot R_{\text{Max}} & \text{if } t < T \\ r_t^{\text{Prog}} - \dfrac{1}{T} \cdot R_{\text{Max}} + R & \text{if } t = T \end{cases}$$

where $R_{\text{Max}} = 10$ is the maximum possible quality score. The subtraction of R_{Max}/T centers the reward so that a hypothetical perfect (i.e., $R = R_{\text{Max}}$) episode with uniform per-frame progress has

Table 1: **Label types used in our reward labeling system.** Each modality contributes a different kind of human judgment, scored on a [0, 10] scale where applicable.

Label Type	Input	Output per Episode	Notes
Multiple Choice Questions	Task-specific weighted yes/no questions	Score based on a task-specific rubric, normalized to $[0, 10]$	In practice, we ask a series of yes/no questions for each subtask.
Pairwise Comparisons	Side-by-side demos; labeler picks the better one	A rating for each demo (computed using Elo or TrueSkill), rescaled to $[0, 10]$	Can be used to impart a *total order* (i.e., a ranking) on the set of demonstrations.
Thumbs Up/Down Labels	Timestamped like/dislike events collected during video replay	Dense temporal annotations (used for reward shaping)	Provides dense temporal supervision within an episode.
Quality Labels	Three-category overall quality rating	Good $= 10$, Medium $= 5$, Bad $= 0$ (averaged across labelers)	Ablated in later experiments; found redundant with the Multiple Choice Questions and lacking in precision.
Subtask Markers	During demonstration, teleoperator presses a button to indicate "subtask completed"	Segmentation of the demonstration into subtasks	Used together with Multiple Choice Questions to assign scores to subtasks.

zero mean per-frame reward, and only a terminal reward. The terminal bonus R on the last frame provides a sparse signal at the episode boundary.

In the simplest case, one can set $r_t^{\text{Prog}} = R/T$ to indicate that each frame contributed equally towards task progress. Since we have access to subtask segmentation and thumbs up/thumbs down labels, we use a two-step procedure to shape r_t^{Prog}:

1. Use a piecewise-linear r_t^{Prog} based on the individual subtasks' contribution to the total reward R.
2. Use the thumbs-up and thumbs-down labels to add positive and negative Gaussian bumps centered at the event timestamp, respectively.

In short, reward shaping helps us indicate that certain frames were conducive or detrimental to task progress, and is an integral part of reinforcement learning [24, 12].

Figure 11 illustrates how subtask markers define the piecewise progress curve, and Figure 12 shows the resulting reward/progress function variants.

To illustrate the effect of reward trajectory generation concretely, Figure 13 shows value-function curves for 25 demonstrations each of two packing tasks. Each curve traces the cumulative reward over time, with thumbs-up events (▲) and thumbs-down events (▼) marked at their timestamps.

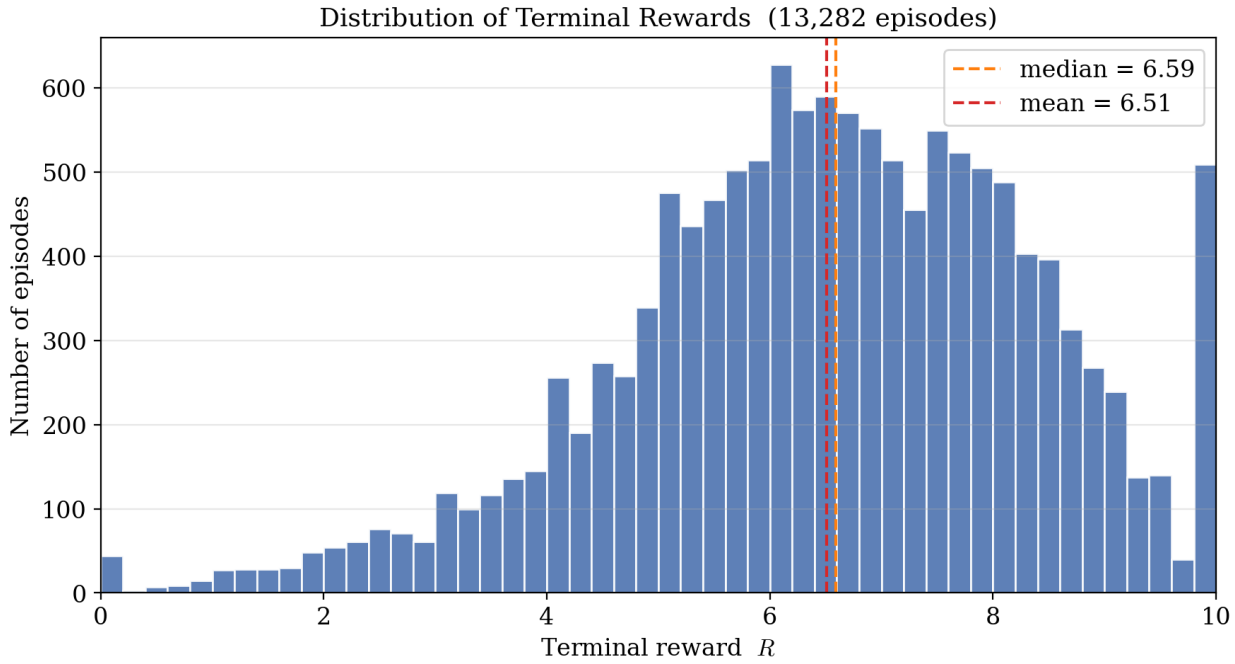


Figure 10: **Distribution of terminal reward R across 13,282 demonstrations.** $R \in [0, 10]$ indicates the quality of overall task completion.

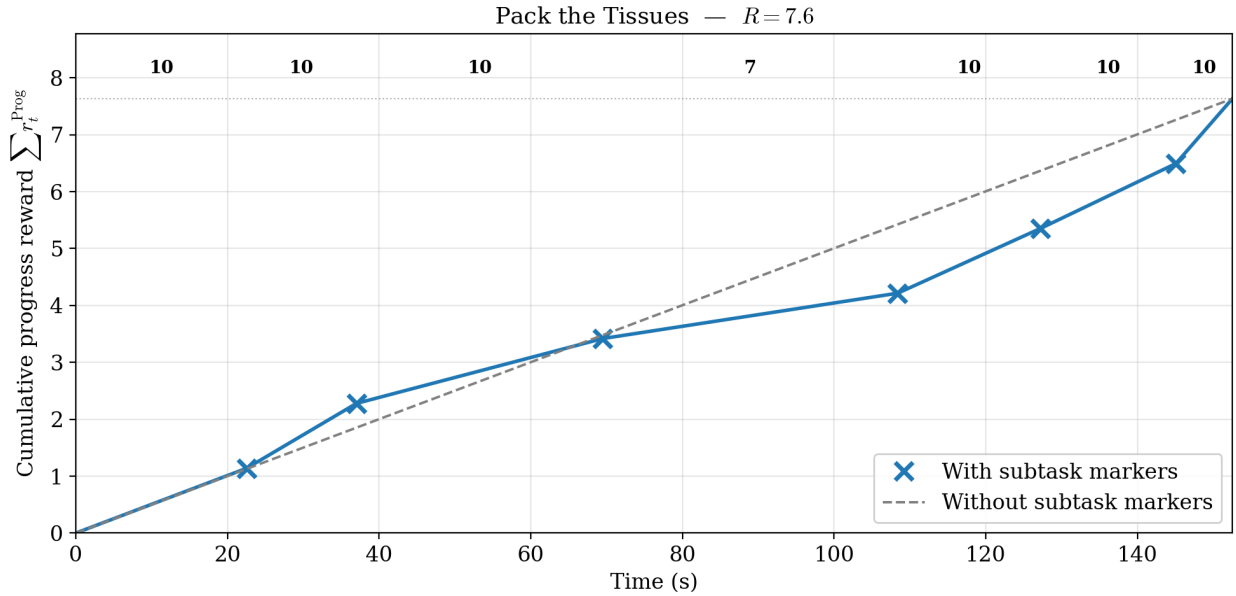


Figure 11: **Subtask markers define a piecewise-linear progress curve**, which is further refined using the other labels (e.g., thumbs up/thumbs down). Each subtask is assigned a score via Multiple Choice Question labels, used as a heuristic for the progress made during the subtask.

Low-quality runs show visible drops at thumbs-down events and often plateau well below the maximum, while high-quality runs climb smoothly to a high terminal reward.

Thus, the reward data labeling system can be used to preferentially train the behavior cloning

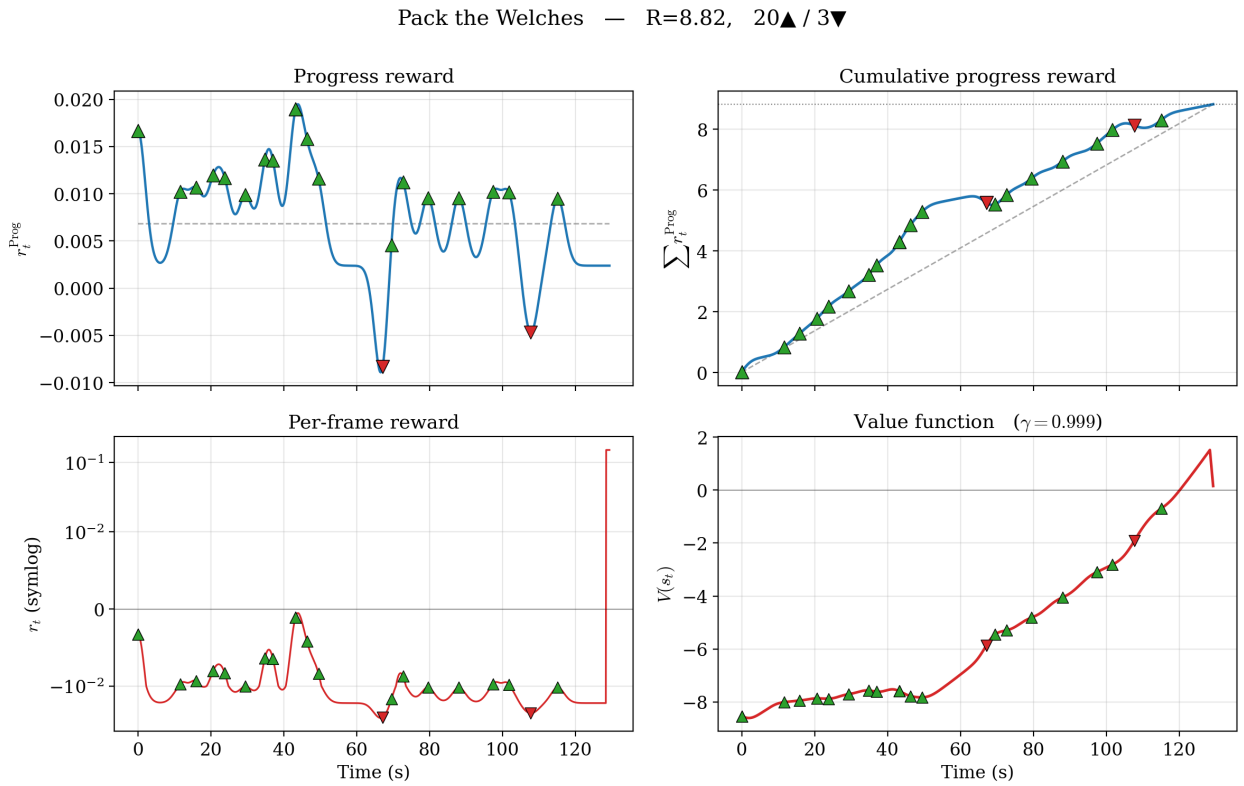
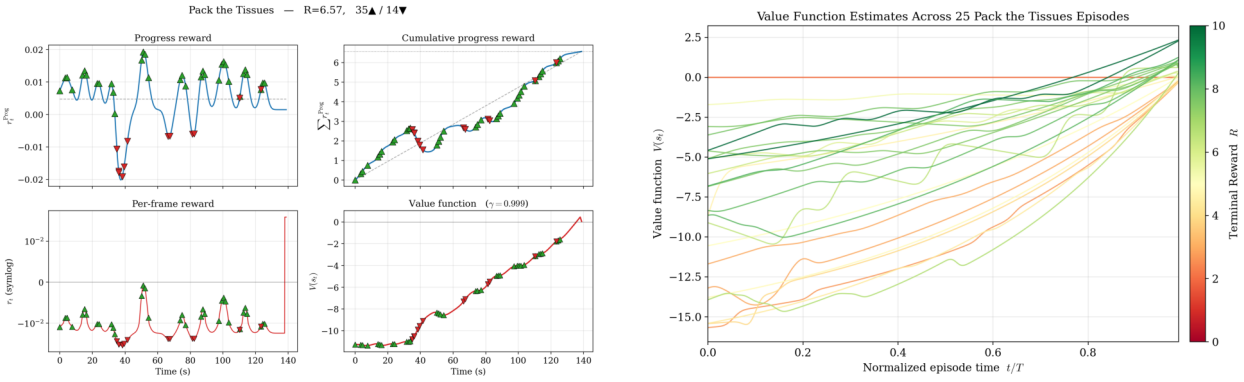


Figure 12: **Reward and progress functions derived from human-provided labels.** The plot on the bottom left uses a symlog (symmetric logarithm) scale for clarity.

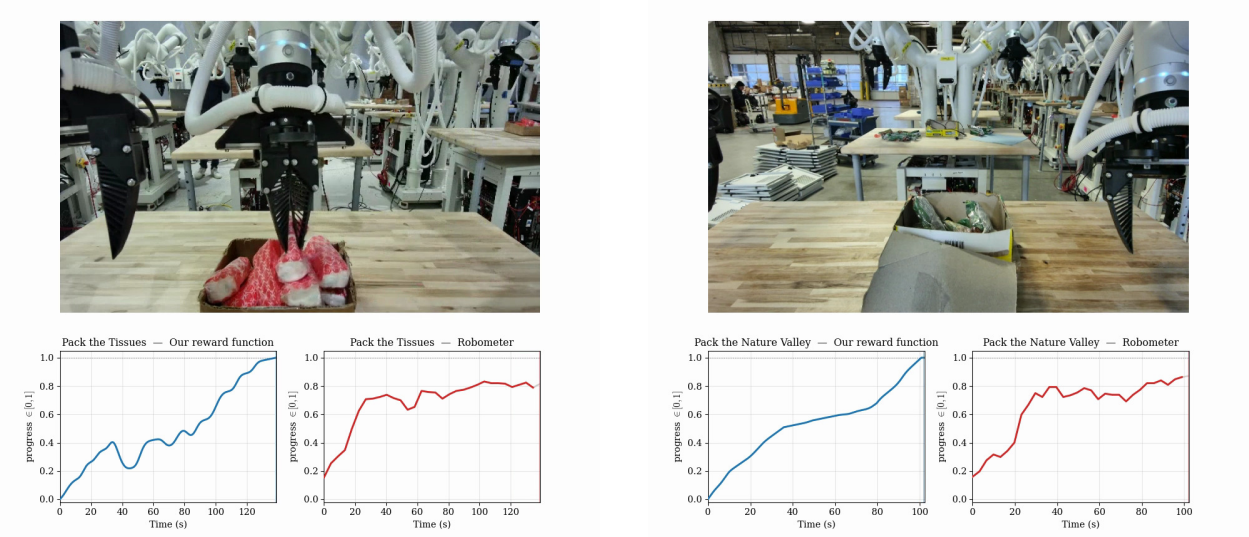


(a) Pack the Welches.

(b) Pack the Tissues.

Figure 13: **Value function plots for 25 demonstrations each of two packing tasks.** Terminal rewards differ due to overall episode quality, and dense feedback controls the curvature.

policy on high-quality demonstrations (e.g., using advantage-weighted regression [26]). It also provides a strong foundation for downstream offline and online reinforcement learning (RL) efforts, enabling Tutor to leverage the massive scale of its DF1 robot fleet.



(a) Pack the Tissues.

(b) Pack the Nature Valley.

Figure 14: Our reward function based on human feedback scales well due to dense human labels. Robometer plateaus after three pick-places and does not reach a progress value of one at the end of a successful demonstration.

4.5 Comparing our reward from human feedback to a general-purpose reward model

We compare our reward trajectories from human feedback with existing general-purpose approaches to reward modeling. We do this to validate our reward labeling pipeline and better understand the strengths and weaknesses of existing general-purpose techniques.

Prior work. Recent work uses VLMs to extract a reward signal from video and language. Zero-shot VLM methods such as TOPReward and GVL extract per-frame progress curves from token probabilities [4, 21]. These values require per-trajectory normalization, so they can rank frames within a rollout but do not compare rollouts to each other, posing a challenge for downstream applications including RL, preference-based curation, and reward-weighted filtering. Trained reward models such as Robometer represent the current state of the art in terms of cross-dataset performance due to the scale and diversity of their training data [18].

The generalization gap. Robometer provides a general-purpose reward model, but our data does not precisely match the camera viewpoints, object sets, and scene clutter from their training data. When we evaluate Robometer on our trajectories, the predicted progress curves are noisy and poorly correlated with actual task progress (Figure 14). For example, Robometer plateaus after three pick-places and does not reach a progress value of one at the end of a successful demonstration, while our reward function continues to increase monotonically through task completion. We also fine-tune Robometer on our own labeled demonstrations to see whether in-domain data would close the gap; it did not meaningfully close the gap. This suggests that there is an opportunity for improved progress modeling when in-domain labels are available.

Contributors

Core Contributors (alphabetical order): Kunal Kapoor, Shiraz Khan, Joseph McCalmon, Jesse Michel, Arif Mohammed, Katia Nikiforova, Adam Oppenheimer, Victor Szabo.

Contributors (alphabetical order). David Bascom, William Burnham, Joshua Fishman, Josh Gruenstein, Alon Kosowsky-Sachs, Yinglong Miao, Ellie Ramos, John Strang.

Acknowledgments

We thank the Tutor Intelligence build and manufacturing team for assembling and maintaining DF1: Stephen Bouwens, Stephen DeBenedictis, Thomas Eckenfels, Carlos Gonzalez, James Hardy, Najique Henry, Andrew Jeffery, Marc Joseph, Jessel Jones, Praneeth Katikala, Adeel Rehman, Yogi Shah, Verona Simms, Nic Tosches, Charles Tsakrios, Jacob Tucker-Figueroa, Arturo Vasquez, Jessica Xie, Vivian Xie, and Tim Zylicz, as well as Tim Foldy-Porto and Anoosh Kumar for their assistance in the build.

We thank our Boston-based Tutor robot operations team, whose demonstrations and labels make this work possible: Roberto Bonilla, Trevor Brake, Marcellus Collins, Christopher Faneuff, Nicholas Faneuff, Janett Lemus, Levi Logan, Ricardo Mathurin, Onur Ozdemir, Melinda Szikora, and Caitlin Thomas. We also thank the 30+ Tutors on our international staff in Mexico City and Manila.

References

[1] 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. π_0: A vision-language-action flow model for general robot control, 2024. arXiv:2410.24164.

[2] Kevin Black, Marco Y. Galliker, and Sergey Levine. Real-Time Execution of Action Chunking Flow Policies, 2025. arXiv:2506.07339.

[3] Qiyu Chen, Jialu Yu, Mac Schwager, Pieter Abbeel, Yide Shentu, and Philip Wu. SARM: Stage-aware reward modeling for long horizon robot manipulation. In *ICLR*, 2026. arXiv:2509.25358.

[4] Shirui Chen, Cole Harrison, Ying-Chun Lee, Angela Jin Yang, Zhongzheng Ren, Lillian J. Ratliff, Jiafei Duan, Dieter Fox, and Ranjay Krishna. TOPReward: Token probabilities as hidden zero-shot rewards for robotics, 2026. arXiv:2602.19313.

[5] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin C. M. Burchfiel, and Shuran Song. Diffusion Policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems*, 2023. doi: 10.15607/RSS.2023.XIX.026.

[6] Cheng Chi, Zhenjia Xu, Chuer Pan, Eric Cousineau, Benjamin Burchfiel, Siyuan Feng, Russ Tedrake, and Shuran Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots. In *RSS*, 2024. arXiv:2402.10329.

[7] Whitney Crooks, Gabrielle Vukasin, Maeve O'Sullivan, William Messner, and Chris Rogers. Fin Ray effect inspired soft robotic gripper: From the RoboSoft grand challenge toward optimization. *Frontiers in Robotics and AI*, 3:70, 2016. doi: 10.3389/frobt.2016.00070.

[8] Cursor Research Team. Composer 2 Technical Report, 2026. arXiv:2603.24477.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, 2019.

[10] Z. Du, B. Liu, Y. Liang, Y. Shen, H. Cao, X. Zheng, and Y.-G. Jiang. HiMoE-VLA: Hierarchical mixture-of-experts for generalist vision-language-action policies, 2025. arXiv:2512.05693.

[11] Raymond C. Goertz. Master-slave manipulator. Technical Report ANL-4311, Argonne National Laboratory, 1949.

[12] Yi Hu, Wei Wang, Hao Jia, Yuxin Wang, Yiran Chen, Jianye Hao, Fei Wu, and Changjie Fan. Learning to utilize shaping rewards: A new approach of reward shaping, 2020. arXiv:2011.02669.

[13] Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

[14] Simar Kareer, Dhruv Patel, Ryan Punamiya, Pranay Mathur, Shuo Cheng, Chen Wang, Judy Hoffman, and Danfei Xu. EgoMimic: Scaling imitation learning via egocentric video. In *CoRL*, 2024. arXiv:2410.24221.

[15] Michael Kelly, Christian Sidrane, Katherine Driggs-Campbell, and Mykel J. Kochenderfer. HG-DAgger: Interactive imitation learning with human experts. In *ICRA*, 2019. arXiv:1810.02890.

[16] David Kent, C. Saldanha, and Sonia Chernova. A comparison of remote robot teleoperation interfaces for general object manipulation. In *HRI*, 2017.

[17] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollar, and Ross Girshick. Segment Anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4015–4026, 2023.

[18] Anthony Liang, Yigit Korkmaz, Jiahui Zhang, Minyoung Hwang, Abrar Anwar, Sidhant Kaushik, Aditya Shah, Alex S. Huang, Luke Zettlemoyer, Dieter Fox, Yu Xiang, Anqi Li, Andreea Bobu, Abhishek Gupta, Stephen Tu, Erdem Biyik, and Jesse Zhang. Robometer: Scaling general-purpose robotic reward models via trajectory comparisons, 2026. arXiv:2603.02115.

[19] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Yann LeCun. Flow matching for generative modeling. In *ICLR*, 2023. arXiv:2210.02747.

[20] S. Liu, L. Wu, B. Li, H. Tan, H. Chen, Z. Wang, and J. Zhu. RDT-1B: A diffusion foundation model for bimanual manipulation, 2024. arXiv:2410.07864.

[21] Yecheng Jason Ma, Joey Hejna, Ayzaan Wahid, Chuyuan Fu, Dhruv Shah, Jacky Liang, Zhuo Xu, Sean Kirmani, Peng Xu, Danny Driess, Ted Xiao, Jonathan Tompson, Osbert Bastani, Dinesh Jayaraman, Wenhao Yu, Tingnan Zhang, Dorsa Sadigh, and Fei Xia. Vision language models are in-context value learners, 2024. arXiv:2411.04549.

[22] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung, Julian Gao, John Emmons, Anchit Gupta, Emre Orbay, Silvio Savarese, and Li Fei-Fei. RoboTurk: A crowdsourcing platform for robotic skill learning through imitation. In *CoRL*, 2019. arXiv:1811.02790.

[23] Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. MimicGen: A data generation system for scalable robot learning using human demonstrations. In *CoRL*, 2024. arXiv:2310.17596.

[24] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, 1999.

[25] Abhishek Padalkar, Ayzaan Wahid, Pieter Abbeel, et al. Open X-embodiment: Robotic learning datasets and RT-X models, 2023. arXiv:2310.08864.

[26] X. B. Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning, 2019. arXiv:1910.00177.

[27] Physical Intelligence. $\pi^*_{0.6}$: a vla that learns from experience. https://pi.website/blog/pistar06, 2025.

[28] Qwen Team. Qwen3-VL technical report, 2025. arXiv:2511.21631.

[29] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. OpenAI technical report, 2018.

[30] Modi Shi, Li Chen, Jin Chen, Yuxiang Lu, Chiming Liu, Guanghui Ren, Ping Luo, Di Huang, Maoqing Yao, and Hongyang Li. Is diversity all you need for scalable robotic manipulation?, 2025. arXiv:2507.06219.

[31] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. CLIPort: What and where pathways for

robotic manipulation. In *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 894–906. PMLR, 2022.

[32] Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Remi Cadene. SmolVLA: A vision-language-action model for affordable and efficient robotics, 2025. arXiv:2506.01844.

[33] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2 edition, 2018.

[34] Jiawei Tang, Yuxuan Sun, Yichen Zhao, Sheng Yang, Yixuan Lin, Zhen Zhang, Jiayi Hou, Yuxiang Lu, Zihan Liu, and Song Han. VLASH: Real-time VLAs via future-state-aware asynchronous inference, 2025. arXiv:2512.01031.

[35] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS*, 2017. arXiv:1703.06907.

[36] TRI LBM Team. A careful examination of large behavior models for multitask dexterous manipulation. *Science Robotics*, 11, 2026. doi: 10.1126/scirobotics.aea6201.

[37] Maximum Wilder-Smith, Vaishakh Patil, and Marco Hutter. Radiance fields for robotic teleoperation. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 13861–13868, 2024. doi: 10.1109/IROS58592.2024.10801345.

[38] Philip Wu, Yide Shentu, Ziyi Yi, Xilun Lin, and Pieter Abbeel. GELLO: A general, low-cost, and intuitive teleoperation framework for robot manipulators. In *IROS*, 2024. arXiv:2309.13037.

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