VOLT: Vision and Language Trajectory Segmentation for Faster-than-Demonstration Policies

Robert Ramirez Sanchez[1], Daniel J. Evans[1], Dylan P. Losey[1], and Siddarth Jain[2]

***Abstract*—Humans often take longer to demonstrate a task than a robot would need to execute it. Rather than learning to replicate the demonstration at the same pace, many industrial and practical applications require robots to perform tasks as quickly as possible. In this paper, we investigate several hypotheses for learning policies that operate *faster-than-demonstrations*. Our experiments show that the most effective strategy is to downsample recorded demonstrations and train the robot's policy on this accelerated data. However, uniformly downsampling an entire trajectory can be problematic. Some parts of a task can be safely sped up (e.g., unconstrained motion), while others demand slower, more precise motion (e.g., object interactions or fine manipulation). To address this challenge, we introduce VOLT, a vision-and-language trajectory segmentation method that reasons over video demonstrations, and leverages contextual cues to determine when acceleration is appropriate and when careful precision is required. VOLT identifies segments where slow, deliberate motion is necessary, then selectively downsamples the remaining segments. The resulting reformatted trajectories can be used with standard imitation learning approaches, such as diffusion policies. Our results highlight that segmentation quality is critical—baseline methods often misidentify when acceleration is possible, leading to overly cautious or unreliable policies. Compared to state-of-the-art alternatives, VOLT allows robots to execute tasks faster while maintaining strong performance.**

I. Introduction

Robots should be capable of performing tasks with precision and speed. Recent works have enabled robots to learn complex tasks by imitating human experts [1]. Since the quality of demonstrations dictates the performance of imitation learning approaches [2]–[4], operators tend to demonstrate tasks at a lower speed to minimize errors. Indeed, even if the human demonstrates at a normal speed — robots often learn to perform tasks slower than demonstrated. But autonomous systems are capable of completing tasks significantly faster than human demonstrators, and this acceleration is beneficial for industrial applications. So how can robots take a given demonstration and speed-up their learned policy?

Existing works often attempt to accelerate learned policies during either training or execution. For example, *training* approaches include high-frequency controllers that reduce teleoperation time [5], as well as data augmentation methods that resample the provided demonstrations [6], [7]. At the other extreme, *execution* approaches accelerate a learned policy by leveraging high-fidelity controllers to precisely track the original behavior at variable frequencies [8]. Both types of approaches recognize two key problems. First, the robot needs to decide *how* to accelerate its policy, and second, the robot must determine *which parts* of its trajectory are appropriate to accelerate. Speeding up the motion often leads to failures, particularly when the robot must perform precise subtasks. (Figure 1).

In this work we seek a fundamental understanding of both questions: (a) how do we determine which segments of a demonstration to accelerate, and (b) what approaches are most effective for accelerating those segments. We adopt an experimental approach, and test a variety of hypothesis across a standardized testing setup. Our insight — consistent with prior works — is that downsampling the human's demonstration provides an effective method for speeding-up the learned policy. However, unlike prior works, we argue that high-level task knowledge is often necessary to determine which parts of the trajectory to accelerate. Specifically, we hypothesize that:

Reasoning over videos of the entire demonstration provides the context cues needed to decide when to accelerate.

We accordingly apply vision and language models (VLMs) to develop **VOLT**. The key idea for VOLT is that the robot collects its standard visual demonstration, and then passes that demonstration through a VLM to determine the correct segmentation. After determining which parts the robot can safely accelerate, VOLT then downsamples those segments and passes the refined dataset to the given imitation learning algorithm. This enables the robot to learn a policy that is faster than the human's demonstration (e.g., speeding up to quickly reach the cup), but also effective at completing the task (e.g., slowing down to carefully grasp that cup). Overall, this work makes the following contributions:

- We introduce VOLT, an algorithm for enabling faster-than-demonstration policies. To the best of our knowledge, VOLT is the first approach to use a high-level VLM to holistically analyze human demonstrations without explicitly defined features, segmenting them into regions that can be safely accelerated and regions that require precise execution.
- We develop and test a variety of hypotheses for effective faster-than-demonstrator policies. These include accelerating policy execution at test-time and learning faster policies via train-time downsampling. Our results suggest that train-time downsampling is a more consistent approach, particularly when using diffusion policies.
- We compare VOLT to state-of-the-art baselines for faster-than-demonstration policies across a diverse range of tasks. Overall, we find that VOLT achieves up to a x2.57 speedup over policies trained on raw demonstrations while maintaining similar success rates. We conclude with practical guidelines for designers who want to accelerate policies within an imitation learning context.

[1]Collab, Dept. of Mechanical Engineering, Virginia Tech, Blacksburg, VA 24061. {robertjrs, danielevans, losey}@vt.edu

[2]Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA 02139. sjain@merl.com

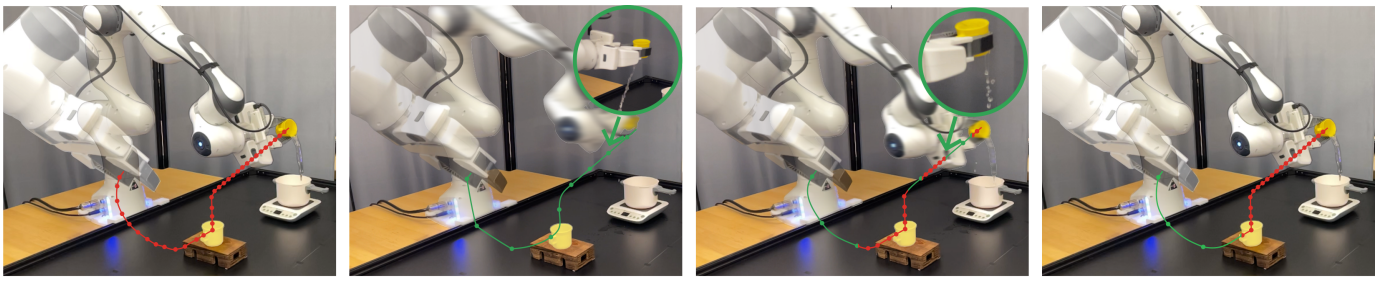

Figure 1. (Left) A human shows the robot how to carry a cup full of water and then pour that water into a pot. (Middle) Naively accelerating the entire demonstration causes the robot to move too quickly (faster motions shown in green), accidentally spilling as it moves. Alternatives slow down the robot in segments (slower motions shown in red). But these segments are often imperfectly chosen, and the robot still accelerates and spills while carrying the water. (Right) VOLT addresses this problem by using context cues from the entire video demonstration to only accelerate relevant parts of the trajectory.

II. Related Work

Emerging work in imitation learning enables robots to complete complex and long-horizon tasks [9]–[11]. In general, these methods focus on matching the action distribution of the human's dataset. Recent research suggests that robots can improve their performance and generalization abilities by learning task-relevant representations [12], leveraging language-based reasoning for task planning [13], and reasoning over large-scale robot datasets [14]. But these methods primarily focus on task success — here we are interested in the task speed. Human demonstrations are typically slow to avoid mistakes; and even if the human moves quickly, for industrial applications we often want to accelerate beyond the human's capabilities. Our VOLT algorithm is not tied to any specific imitation learning algorithm. But to provide results that are consistent with the state-of-the-art methods, we will leverage diffusion policies in our experiments [15].

Unintentional Speed Up. Within the context of imitation learning, several methods alter the given dataset to improve data efficiency and mitigate compounding errors [16]–[19]. The common theme across these works is to reduce the prediction horizon required to complete a task. Although accelerated execution is not the objective, the changes these methods make to the demonstrations often result in a sped-up policy. For example, Belkhale *et al.* [19] propose a hybrid architecture that dynamically switches between waypoints and dense low-level actions; the robot moves more rapidly between the waypoints marked by the human teacher. Similarly, Shi *et al.* [17] introduce Automatic Waypoint Extraction (AWE), a method to automatically extract waypoints from trajectories via linear interpolation within a specified error threshold. A different perspective is to decompose the trajectory into a set of primitives, and then learn a policy to sequence these primitives — again, reducing the prediction horizon [18]. The insight we obtain from these works is that data alterations are an effective way to reach faster-than-demonstration policies. In VOLT we extend these ideas, while specifically focusing on how rapidly the learned policy performs the entire task.

Intentional Speed Up. Recent works seek to maximize the robot's speed without significantly impacting its performance [6]–[8], [20]. In reinforcement learning, existing works include learning speed interpolation policies [20], and performing online fine-tuning to select faster actions [6]. Although these approaches can capture complex speed-up strategies, they typically require millions of online interactions, making them impractical in most real-world tasks. Most similar to our approach are offline imitation learning methods that label each demonstration by classifying states suitable for faster task execution. Guo *et al.* [7] learn a proxy policy from the original dataset to estimate the conditional action entropy distribution. They denote high-entropy actions as "casual" segments and low-entropy actions as "precise". By contrast, [8] estimates motion complexity via AWE [17] waypoints and DBSCAN [21] clustering to identify precise segments. We hypothesize that these approaches can be inaccurate because they do not consider the high-level context of tasks. Towards this objective, [22] explicitly defines 3D gripper-object distances as high-level context extracted from images. They estimate these features using VLMs, then parse them into text for an LLM to label a single demonstration, and propagate these labels via Dynamic Time Warping. But there are human intents not encoded by these designer-chosen features, for example, gripper-object interaction (e.g., grasping vs. pushing), or manipulation of deformable bodies.In VOLT we do not assign intermediate features; instead, we directly use VLMs to reason over video demonstrations.

III. Problem Statement

We consider visual imitation learning from offline datasets of human demonstrations, where a robot arm is teleoperated by a human while the system records environment states and actions. Because the quality of the learned policy is correlated to demonstration fidelity, experts typically perform demonstrations slowly and cautiously to avoid mistakes. This results in policies that can achieve high task success but execute behaviors at relatively low speeds. The central problem we address is how to improve execution efficiency, specifically, increasing task execution speed without degrading task performance or safety. To this end, we aim to (a) autonomously identify which portions of a task can be safely executed at higher speeds, and then (b) develop a strategy for accelerating those segments without degrading overall performance.

Policy. Let $x \in \mathcal{X}$ represent the robot joint and gripper position, and let $y \in \mathcal{Y}$ denote the environment state observed through RGB cameras. We combine both the robot state and RGB images into observation $o_t = (x_t, y_t)$. The robot's learned policy π maps these observations into actions:

$$A_t \sim \pi(\circ \mid O_t) \tag{1}$$

where $A_t = \{a_t, a_{t+1}, ..., a_{t+H}\}$ is a sequence of actions with horizon H, and each action a_t is a commanded robot joint and gripper position. The observation history $O_t = \{o_{t-M}, ..., o_t\}$ is composed of the most recent M observations.

In our experiments we will instantiate Equation (1) as a diffusion policy [15]. To match state-of-the-art performance, we will apply the Denoising Diffusion Implicit Models approach [23]; this requires fewer iterations during inference (as compared to training) for faster action generation. The inference process is defined by the following equation:

$$A_t^{k-1} = \alpha(A_t^k - \gamma\varepsilon_\theta(O_t, A_t^k, k) + \sigma), \quad \sigma \sim \mathcal{N}(0, \sigma^2 I) \quad (2)$$

where ε_θ is the noise prediction network with learnable parameters θ that are optimized to predict the noise added to each sample. Here k represents the denoising iteration, α and γ are the noise scheduler hyperparameters, and $\sigma \sim \mathcal{N}(0, \sigma^2 I)$ is Gaussian noise added at each iteration.

Labeling Dataset. To train π the human provides a dataset of task demonstrations $\mathcal{D} = \{\xi_0, \xi_1, ..., \xi_N\}$. Each demonstrated trajectory ξ is a sequence T of observation-action pairs: $\xi = \{((x_0, y_0), a_0), ..., ((x_T, y_T), a_T)\}$. As we will show, learning a policy directly from $\mathcal{D}$ limits the robot's speed to that of the human teacher. One potential way to accelerate the learned policy is to first relabel each demonstration $\xi = \{(\tau_0, l_0), ..., (\tau_k, l_k)\}$ into ordered non-overlapping segments $\bigcap_{i=1}^{k} \tau_i = \emptyset$. The labels $l_i \in L = \{$*speed-up, maintain-speed*$\}$ denote whether the given segment can be executed faster than demonstrated, or whether the segment requires precision or dexterity (e.g., grasping, insertion) and therefore the policy should match the human's original speed.

Hierarchical Control. Actions a indicate the desired joint position and gripper state. To execute the policy quickly, the robot must select control inputs that rapidly perform these actions. In our experiments we update the target robot position at 20 Hz (e.g., we output 20 actions a_t per second). We then reach these target positions using a low-level velocity controller that operates at 1 kHz. The specific controller depends on the robot used (in our experiments we tested with a Franka Emika robot arm). But we generally focus on problem settings where the robot has a high-level diffusion policy — outputting desired positions a — and then a low-level controller — which can accurately track those targets.

IV. How Do We Enable Faster-than-Demonstration Policies?

Our goal is to develop robot policies that maximize task speed while maintaining optimal performance. However, simply training a policy π on collected trajectories $\mathcal{D}$ will inherently limit the robot's motion to the pacing of the human demonstrator. In this section, we therefore test different methods that designers might intuitively consider for speeding up the robot's actions. We divide these methods into two groups: accelerating the policy during *execution* (i.e., reducing the number of high-level actions to shorten the task), or accelerating the policy during *training* (i.e., downsampling the dataset to emulate a faster demonstration). Our experiments in Sections IV-A and IV-B suggest that the training paradigm is more effective for retaining the policy's success rate. We find that naively speeding up *all* the robot's actions exacerbates failures, particularly when the robot requires precise motion. We address this problem through our proposed VOLT algorithm in Section IV-C. VOLT reasons over video demonstrations and segments these trajectories into sections that should either be accelerated or left at the demonstrated speed. We downsample the marked segments and train the robot; our resulting policies yield faster execution while maintaining task performance. However, we note that there are practical limits on how much the robot can accelerate, even after correctly identifying these segments. In Section IV-D we discuss the practical factors for maximizing speed.

Evaluation Setup. We perform all experiments on a Franka Emika robot arm on a tabletop setup as shown in Figure 2. We collect all demonstrations using a GELLO [24] to teleoperate the arm at 20 Hz. For each timestep, we record the robot's joint position, including the gripper state x, the commanded joint and gripper state a_t, and RGB images y from three RealSense D435 cameras. Two cameras are fixed in the environment, and one is directly mounted on the robot's end-effector. Each paradigm is tested on the following manipulation tasks:

- *Pick and Place*: The robot grasps a blue cup located on the table and places it on top of a red plate.
- *Push Cup*: The robot pushes a red cup beside a white bowl, only making contact with the outside of the cup.

Baseline Policy. We train diffusion policies using the original dataset (*Normal Speed*) as our baseline for performance and task speed. Diffusion policies are capable of accurately learning action chunks, which are crucial for speeding up execution. In our experiments, inference delay is about ~ 0.1 seconds because the inference time is greater than the desired 20 Hz high-level control frequency. We eliminate sensing-inference delays through asynchronous inference, and enforce temporal consistency via Error-Adaptive-Guidance [8]. We utilize reached states x_{t+1} instead of the commanded actions a_t to mitigate action tracking errors. These additions to the standard diffusion policies were made to facilitate acceleration.

Acceleration. Within this section we will test two paradigms for faster-than-demonstration policies. These paradigms perform acceleration during either execution or during training:

- *Action Downsampling (Action-D)*: uniformly downsample the predicted action chunk A_t with a fixed factor n. Instead of performing actions a_1, a_2, etc., here the robot performs actions a_1, a_{1+n}, etc. This shortens the task into fewer steps and thereby accelerates the robot's execution.
- *Demonstration Downsampling (Demo-D)*: uniformly downsample each demonstration by factor n. Instead of training on trajectory $\xi = \{(o_1, x_1), (o_2, x_2), ...\}$, here the robot trains with $\xi = \{(o_1, x_{1+n}), (o_{1+n}, x_{1+2n}), ...\}$. Downsampling makes the demonstration shorter, resulting in a faster learned policy.

A. Acceleration During Execution

Given a trained policy on the original dataset, the first strategy we consider is speeding up its execution. We take

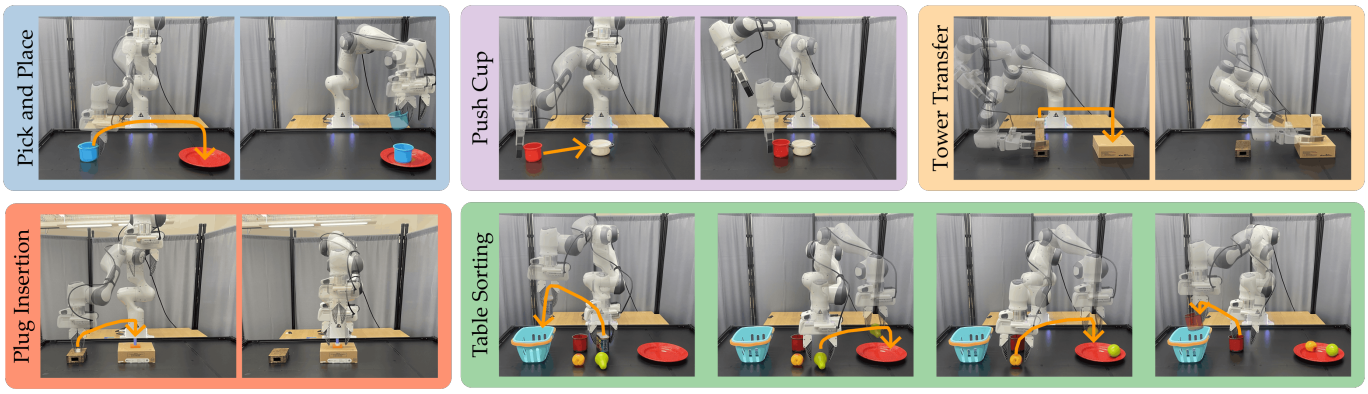

Figure 2. Real-world manipulation tasks used in our experiments. The manipulated objects are randomly initialized in each task, and the target object location is fixed. (Top) In *Pick and Place* and *Push Cup* the robot manipulates a cup, while in *Tower Transfer* the robot to interact with a stack of blocks. (Bottom) *Plug Insertion* a challenging assembly task, and *Table Sorting* a long-horizon task where the robot sorts fruits on a plate and containers in a basket.

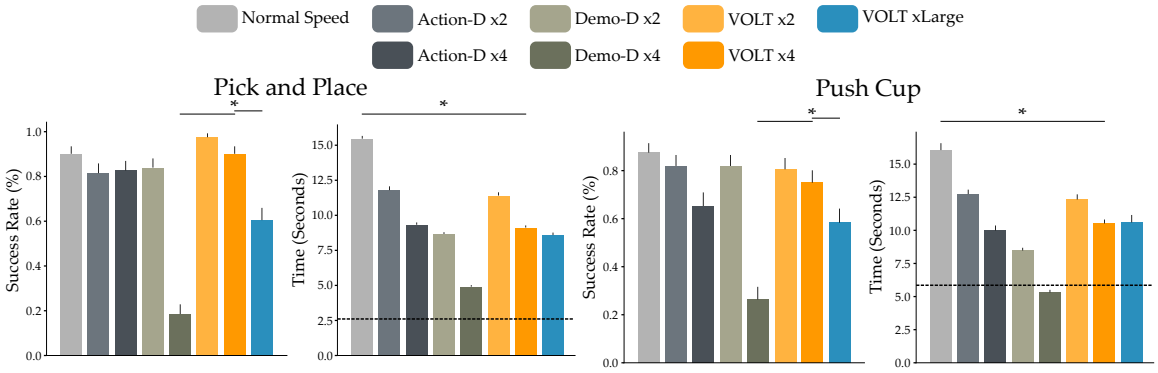

Figure 3. Experimental results for Section IV. Results are averaged across 3 training runs. *Normal Speed* is the policy trained with the default data. In *Action-D* action sequences are downsampled at run time to accelerate task execution. By contrast, in *Demo-D* actions are downsampled at training time to emulate a human with faster teaching motions. *VOLT* is similar to *Demo-D*, but it uses a VLM to determine which segments of the trajectory should or should not be downsampled. The downsampling rate is captured by n; i.e., in $\times 2$ the robot skips every other action. We find that speeding up the policy during execution (*Action-D*) degrades performance as n increases, since the policy misses critical actions (e.g., aligning the grippers). Similarly, speeding up the entire policy during training (*Demo-D*) lowers the success rate. VOLT is able to maintain its success by only accelerating the parts of the task that do not require precise motion. However, the performance of VOLT still degrades if the human increases n to a level where the low-level controller is unable to track the commanded actions (i.e., there are still practical limitations). The dotted lines in Time denote the lower bound for *VOLT* if the robot could instantaneously complete the accelerated segments. An $*$ denotes statistical significance, and error bars indicate standard error.

a policy π and modify its action outputs so that the robot completes the task in less time. The way we modify these actions depends on the policy structure. Referring back to Equation (1), our diffusion policy outputs a sequence A of actions, where each action is a commanded joint position and gripper state. We can accelerate the robot's motion by *skipping* actions within the sequence, i.e., downsampling A by a fixed rate n. After downsampling, the new sequence A' has a reduced horizon; thus, the execution time of a single prediction decreases linearly as n increases. Since the prediction horizon has fixed length H, the downsampling rate n is constrained to at most executing one action ($n \leq H - 1$). Note that omitting parts of the predicted actions makes the scheduled trajectory lose fine-grained details, and the low-level control has to apply larger control signals to track the desired states in A'. Considering that completing a task may not require the robot to visit all states that the human demonstrated, this uniform acceleration could potentially speed up task execution.

Figure 3 compares the performance of *Normal Speed* against test-time downsampling (*Action-D*) at rates $n = 2$ and $n = 4$. As expected, the total time decreases as n increases. However, the performance of the learned policy degrades, particularly in tasks where the robot needs consistent, fine-grained motions. We observed that the robot was skipping actions, and some of these skipped actions were critical to the task. For example, when moving to push the cup, the robot needed to align its gripper behind the cup before initiating the push motion; if the accelerated actions A' skip these steps, then the robot failed the task. A more fundamental issue, however, was the mismatch between the robot's speed during training and testing. The observation history $O_t = \{o_{t-M}, ..., o_t\}$ used to train π was obtained at the human's teaching speed. When we accelerate during execution, the robot collects observations O with larger motions between images, pushing the robot's policy farther out-of-distribution as n increases. This mismatch between behaviors the policy has been trained upon and the behaviors expected at test time limits acceleration during execution.

B. Accelerating During Training

With these issues in mind, we next explore methods that downsample the human's offline dataset. The core idea here is that — by reducing the number of datapoints — we effectively

make the human's demonstration faster. Policies trained on the accelerated dataset expect rapid movement during execution, and thus do not have the out-of-distribution challenge faced in Section IV-A. In practice, we perform downsampling by dividing a single trajectory into n non-overlapping versions. The first version is $\xi^1 = \{(o_1, a_1), (o_{1+n}, a_{1+n}), \ldots\}$, the second version is $\xi^2 = \{(o_2, a_2), (o_{2+n}, a_{2+n}), \ldots\}$, and so on. Similar to before, we train new policies with $n = 2$ and $n = 4$, and test their performance in Figure 3. We find that *Demo-D* has the potential to significantly reduce execution time, and performs the task faster than equivalent versions of *Action-D*. But this speed comes at a cost: increasing to $n = 4$ sharply reduces the success rate, significantly underperforming the alternatives. Observing the robot's motion, we find that *Demo-D* fails because the robot is trying to complete segments of the task *too rapidly*. For example, in the pick and place task the robot often misses the cup, and in the pushing task the robot often knocks the cup instead of sliding it across the table. The policy tries to address these mistakes — but by the time the robot moves to fix errors, the end-effector is already far from the cup and out-of-distribution. Accelerating during training has potential, but naively downsampling the entire motion results in critical failures.

C. VOLT: Vision and Language Trajectory Segmentation

Now that we have a method for accelerating the robot's policy, we will determine *when* to speed up the robot's motion. Our intuition is that the videos included within the demonstration provide the context necessary for segmenting the trajectory. For instance, through these videos we can separate between a robot moving close to a cup unintentionally, and a robot moving towards a cup to grasp it. Based on this insight, we propose to separate each demonstration into trajectory segments of types (1) *maintain-speed*: subtasks requiring precision or dexterity, and (2) *speed-up*: simpler motions that a robot can exploit to complete a task in less time than the human demonstrator.

We specifically leverage Qwen3-VL-32B-Instruct-FP8 [25] to autonomously generate the labels for the entire dataset. The VLM is directed to generate a concise reasoning before selecting a label for each segment. We manually adjust inference hyperparameters to balance accuracy, and throughput. Our hyperparameter values and prompt can be found here: project page. We evaluate the computational cost for segmenting for all the tasks shown in Figure 2 using a single H200 GPU. We observe that VOLT generates the labels for all 250 demonstrations in our dataset in $\sim$ 32 minutes with batch inference. To further assess VOLT's applicability to large-scale datasets, we label the 10 longest demonstrations from DROID [26], where each selected demonstration contains over 3500 frames: in this case VOLT requires 12.5 minutes to segment all demos. With demonstrations over 3000 frames is best to downsample the video or separate it into chunks for parallel segmentation over shorter context windows.

The VLM's input (V, p) consists of an entire video demonstration $V = \{y_0, y_1, ..., y_T\}$ along with a text prompt p describing the trajectory segmentation objective with general task-invariant guidelines. We provide a few in-context examples of diverse tasks (excluding the target task) to enhance reasoning and encode optimal strategies. The model outputs an ordered set of segments $\{(\tau_0, l_0), ..., (\tau_k, l_k)\}$, where τ denotes the start and end index for the segment, and l is the label.

During training, the robot now has access to a dataset $\mathcal{D}$ with samples $\{(x, y, l)\}$ and user's chosen downsampling factor n. The robot then trains its policy on the resampled dataset. We apply downsampling only for observations o_t when label l_t = *speed-up*. To directly compare against the previous approaches, we use downsampling factors $n = 2$ and $n = 4$. Our results compare VOLT to the alternatives in Figure 3. For the pick and place task we find that models trained with VOLT maintain or achieve a higher success rate than *Normal Speed*, while for the cup pushing task there is a slight decrease in performance, usually due to errors in gripper placement behind the cup. As expected, only speeding up in the segments identified by VOLT yield polices that are slower than naively downsampling the entire trajectory. But this approach ensures the robot completes the task faster than the *Normal Speed* policy without sacrificing its success rate.

D. What Limits VOLT's Acceleration?

Our previous experiments show that we can leverage VOLT to accurately estimate when to accelerate. But what limits the downsampling factor n? Here we discuss the practical factors that prevent designers from achieving even faster motions.

Policy Inference Delay. The robot's high-level policy π converts its observations into actions. To prevent unintended pauses during evaluation, these actions must be readily available to the robot. However, when using diffusion policies the inference process — i.e., reasoning over images and generating the action sequence — is computationally expensive and typically exceeds the high-level control frequency. We mitigated this challenge by performing policy inference and action execution asynchronously. But there still remains a fundamental restriction on the control frequency f as a function of the inference delay $t_{\text{inference}} < \frac{H}{f}$. Although increasing horizon H allows us to also increase f, executing longer action horizons prevents the robot from reacting to changes in the environment.

Open-Loop vs. Closed-Loop. When the robot's pacing matches the human demonstrator, it can consistently update its actions in response to errors, and make corrections to improve its performance. But if the robot maintains the same control frequency — and accelerates its motions — then the robot may have large errors before it queries its policy for a correction. Taken to an extreme, if we increase n such that each segment is just the start and goal waypoints, then the robot loses its ability to reason over any changes in the environment. Designers must tune n to trade-off between faster, open-loop execution and slower, closed-loop corrections.

Experiment Results. To test these concepts we increased n until the low-level controller was unable to track the reference positions output in A. The results are summarized in Figure 3 as *VOLT xLarge*. Compared to the VOLT alternatives, *VOLT xLarge* has significantly lower success rates in both tasks (decreasing performance for *Pick and Place* by 29.7% and

Push Cup by 29.2%). Interestingly, the completion time with *VOLT xLarge* is only slightly lower: this is because the robot's policy makes several mistakes at high speeds, and takes added time to redo or correct its motions. Example failures include missing the cup when trying to grasp, knocking the cup over by mistake, or pushing the cup at the wrong angle. Overall, we conclude that the designer cannot simply increase n until they reach their robot's physical or control limits. The time delay associated with the high-level policy imposes constraints, and these constraints stack with any delays in the sensors (e.g., cameras are often limited to 60 frames per second).

V. How Does VOLT Compare to Alternatives?

In this section we compare *VOLT* against imitation learning approaches that are designed to accelerate policy execution. We focus on comparing both the performance and speedup magnitude over a standard policy trained on the human's demonstrations. Specifically, we utilize the same tabletop setup as in Section IV and evaluate additional short-horizon and long-horizon tasks (see Figure 2).

Baselines. We compare *VOLT* to the two most similar baselines: *DemoSpeedup* and *SAIL*. Both of these approaches have the same general structure as *VOLT*, in that they use a high-level controller π to select the desired positions and a low-level controller to reach those positions as quickly and accurately as possible. The differences are explained below:

1) *DemoSpeedup*: learns a proxy policy on the original dataset to estimate the conditional action entropy of each demonstration [7]. The authors employ hierarchical density-based spatial clustering (HDBSCAN) [27] to identify low-entropy clusters representing regions where the robot requires precise motion, while high-entropy outliers represent motions that do not require precision.
2) *SAIL*: extracts waypoints from each demonstration using AWE [17] and estimate motion complexity by performing DBSCAN [21] clustering on the selected waypoints. Similar to DemoSpeedup, SAIL identifies clustered regions as precise motion and outliers as the non-precision regions. In the original implementation these labels are predicted at runtime to modulate control frequency. For a direct comparison to our approach, we use the generated labels to downsample the dataset before training.

Tasks. In addition to the tasks from Section IV, we evaluate the following additional manipulation tasks:

- *Tower Transfer*: The robot transfers a tower of freely stacked wooden blocks to a new platform.
- *Plug Insertion*: The robot picks a plug and inserts it into a socket that is fixed in place. We test on assembly ID #00186 from the *AutoMate* assembly benchmark [28].
- *Table Sorting*: a long-horizon task with fruits and empty containers are scattered on a table. The robot's goal is to sort the fruits on a plate and the containers in a basket.

Training. We collect demonstrations using the robot arm setup from Section IV. For each short-horizon task we collect ~ 50 demonstrations, and 100 demonstrations for the long-horizon *Table Sorting* task. Across demonstrations, we randomize the initial position of the manipulated objects within a pre-defined region, and keep the target object positions fixed. The same training data is provided to *VOLT* and the baselines. We then use each separate algorithm to determine which segments of these demonstrations to downsample. Once the segments are assigned by *VOLT*, *DemoSpeedup*, and *SAIL*, we then train separate models with identical network architectures and hyperparameters for every method to assess how the label affect policy speedup and performance. In the short-horizon tasks, we apply a downsampling rate of $\times 2$ and $\times 4$ for *maintain-speed* and *speed-up* segments, respectively. For *Table Sorting* we apply $\times 1$ and $\times 3$ downsampling rates.

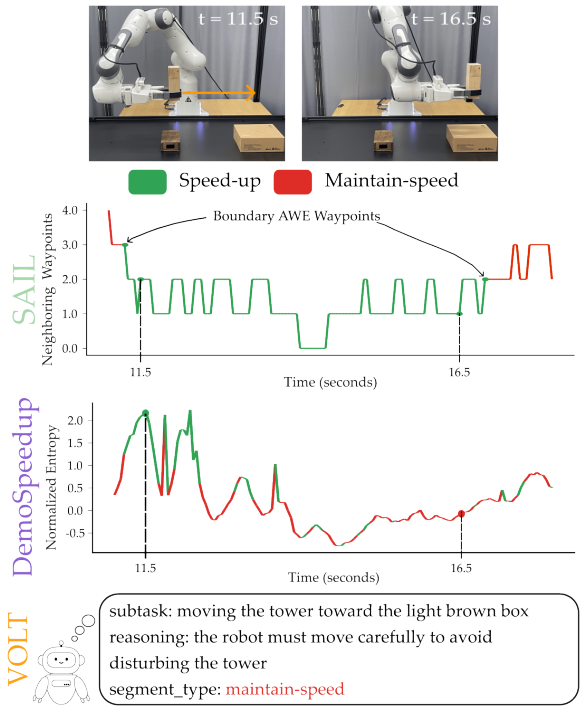

Figure 4. Example segments for transporting the block tower in the Tower Transfer task. Ideally, the robot should maintain speed during the segment from 11.5s – 16.5s, since speeding up causes the blocks to fall over. *SAIL* labels the entire sequence as *speed-up* because a straight-line motion has low complexity. *DemoSpeedup* oscillates between modes while focusing on the entropy in the human's demonstrations. Both of these approaches cause the tower to fall apart during transfer. *VOLT* correctly recognizes that speeding up could lead to failure, and accordingly labels the segment as *maintain-speed*.

A. How Accurately Does VOLT Segment Demonstrations?

The key difference between *VOLT* and the baselines is how they segment the trajectory. When comparing methods, we focused on how accurately each approach decomposes the human's demonstrations into labeled subtasks: l = *speed-up* for simple segments and l = *maintain-speed* for precise motions. In Figure 4 we show an example for Tower Transfer. Intuitively, the robot should move quickly to pick up the blocks, and then maintain the demonstrated speed while transferring the tower. Given the exact same demonstration, *SAIL* incorrectly speeds up the entire motion of transferring the tower. *DemoSpeedup* alternates between accelerating and maintaining speed; only *VOLT* correctly recognizes that the transfer motion should not be accelerated. Note that this

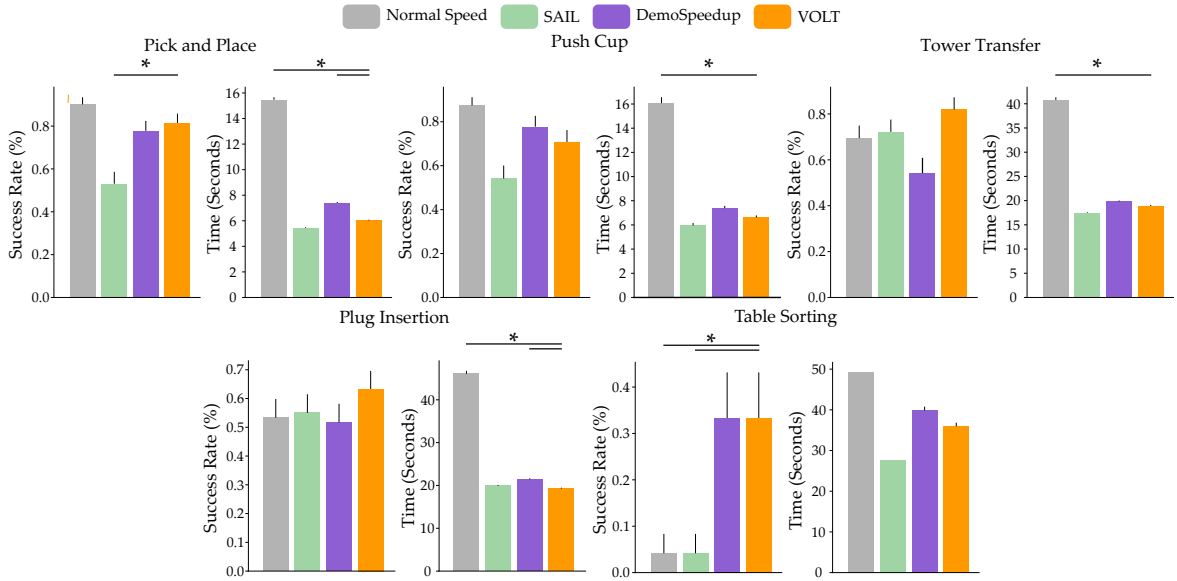

Figure 5. Experimental results for Section V. We compare VOLT against state-of-the-art baselines across five manipulation tasks: Pick and Place, Push Cup, Tower Transfer, Plug Insertion, and Table Sorting (see Figure 2). All methods use the same network architecture and downsampling rate. We report results for a single model in *Table Sorting* and average results over 3 training runs for the short-horizon tasks. In our results *SAIL* overestimates *speed-up* segments (leading to faster but less successful policies), while *DemoSpeedup* often misses *speed-up* segments (leading to slower but consistent policies). We find that *VOLT* balances between these extremes: significantly reducing task completion time, and achieving the second largest speedup across all tasks while maintaining similar performance to *Normal Speed*. An $*$ denotes statistical significance, and error bars indicate standard error.

labeling is not possible with alternatives [22], since they rely on distance features — and here the relevant feature is how the block tower may collapse. In general, while we observe that *SAIL* correctly labels gripper actuation as *maintain-speed*, this method more aggressively categorizes regions as *speed-up* compared to the alternative methods. On the other hand, *DemoSpeedup* is overly conservative and labels parts of the task as *maintain-speed*; this most likely occurs because a fixed goal position for the manipulated object produces a lower entropy value around these repetitive movements.

B. Does VOLT Successfully Accelerate Policy Execution?

Our experimental results are summarized in Figure 5. A one-way ANOVA showed that policies trained with *VOLT* complete tasks significantly faster than *Normal Speed* across all tasks ($p < 0.001$ for all), achieving on average x2.18 speedup. In particular, *VOLT* significantly outperforms training with the original demonstration in Table Sorting ($p < 0.05$), where *VOLT* exhibits a more consistent predicted behavior. We attribute the performance gain to the reduced execution horizon [17]. As compared to the baselines *VOLT* consistently obtains the second largest speedup. *SAIL* is the fastest — because it assigns the most *speed-up* segments — but its overly aggressive choices cause the robot to make mistakes that detrimentally affect performance in Pick and Place and Push Cup. By contrast, *DemoSpeedup* most frequently labels segments as *maintain-speed*, resulting in a slower policy with similar success rates. Overall, our results suggest that *VOLT* is capable of correctly estimating when to accelerate.

C. Ablation Study

The key components in our method to enable VLMs to segment demonstrations are task-invariant guidelines, a few in-context examples, and the elicited reasoning. We ablated them on the *Plug Insertion* task.

Table I
ABLATION STUDY RESULTS FOR SECTION V. *Normal Speed* POLICIES ACHIEVE 53.33% SUCCESS RATE AND TIME OF 46.1 SECONDS.

Method	Success Rate (%)	Time (s)
No Examples	48.33	19.27
No Examples + Simple Prompt	31.67	18.93
Smaller VLM	46.67	**17.88**
VOLT (Ours)	**63.33**	19.27

- *No Examples*: We test our method without in-context examples to test the VLM's ability to identify segments without including visual guidance.
- *No Examples + Simple Prompt*: We additionally remove the task-invariant textual guidance and generated reasoning to evaluate the VLM's baseline performance.
- *Smaller VLM*: We generate labels with our proposed configuration and only change the VLM to Qwen3-VL-8B-Instruct-FP8 to test how VLM size affects performance.

Results. We find that removing the in-context examples slightly degrades the VLM's ability to estimate the start and end frames for short segments. When only providing the labeling objective to the VLM, the model recognizes the correct segments, but often fails at estimating their timing. Finally, employing a smaller VLM has minor negative impacts for recognizing when the robot begins interacting with an object. Our quantitative results are summarized in Table I. Overall, we find that each design choice significantly contributes to balancing the speed-performance trade-off.

VI. PRACTICAL CONSIDERATIONS

We summarize key takeaways for accelerating faster than demonstration policies in practice.

- **Visual cues are necessary.** While utilizing the robot's information to segment demonstrations can be effective, throughout our experiments, we found that video reasoning is crucial to identify when to *maintain-speed* for simple but precise motions (e.g., transferring the tower).
- **Eliminate sensor-inference delays.** Effective faster-than-demonstration policies require the robot to always have actions available to execute, but synchronous inference introduces unintended pauses. We therefore recommend using asynchronous inference, along with methods that ensure continuity between asynchronous actions [8], [29].
- **Reduce the receding control horizon.** As the robot increase is speed, it will become less reactive. Instead of executing a long sequence of open-loop actions, it is better to reduce the policy action horizon and query π more frequently. We can then pass the output actions A to the low-level controller as soon as they are updated.
- **Optimizing the low-level controller.** In our hierarchical paradigm, the policy sets goal positions, and the low-level controller tracks them via velocity or torque commands. Tuning low-level gains for fast, accurate tracking is essential to maximize acceleration.
- **Smooth transition between fast and precise segments.** Abrupt changes in acceleration cause motion jittering, which prevents the robot from reliably executing task near the boundaries of segments. A possible solution is to apply smooth action interpolation near these boundaries. In practice, we recommend downsampling rates 2x-4x, since these have been tested to perform consistently well.

VII. Conclusion

In this paper we presented a data augmentation framework to speed up task execution without degrading task performance. Our approach (VOLT) recognizes which segments of a task a robot can complete faster than demonstrated, and segments where the robot should operate similarly to the demonstrator. Specifically, we identify these segments based on the context of the task and the images of the environment by leveraging the video understanding capabilities of VLMs. We then trained accelerated robot policies by downsampling the identified regions at different rates. Our experiments suggest that this approach enables robots to execute task more efficiently by increasing execution speed, while maintaining similar performance to models trained on the original dataset. Our results suggest that VOLT is a step towards learning policies that balance between task performance and execution speed. However, our current implementation requires downsampling rates to be manually defined and applied uniformly. Future work should automate this process, possibly guided by the VLM's own uncertainty, employing trajectory compression algorithms, or a hybrid approach combining VOLT with similar methods that have access to the low-level robot information.

References

[1] M. Zare, P. M. Kebria, A. Khosravi, and S. Nahavandi, "A survey of imitation learning: Algorithms, recent developments, and challenges," *IEEE Transactions on Cybernetics*, 2024.

[2] H. Xu, X. Zhan, H. Yin, and H. Qin, "Discriminator-weighted offline imitation learning from suboptimal demonstrations," in *International Conference on Machine Learning*, 2022.

[3] S. Belkhale, Y. Cui, and D. Sadigh, "Data quality in imitation learning," in *Advances in Neural Information Processing Systems*, 2023.

[4] A. S. Chen, A. M. Lessing, Y. Liu, and C. Finn, "Curating demonstrations using online experience," *Robotics: Science and Systems*, 2025.

[5] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn, "Learning fine-grained bimanual manipulation with low-cost hardware," *Robotics: Science and Systems*, 2023.

[6] T. Nam and S. J. Hwang, "SpeedAug: Policy acceleration via tempo-enriched policy and RL fine-tuning," *arXiv preprint arXiv:2512.00062*, 2025.

[7] L. Guo, Z. Xue, Z. Xu, and H. Xu, "DemoSpeedup: Accelerating visuomotor policies via entropy-guided demonstration acceleration," in *Conference on Robot Learning*, 2025.

[8] N. R. Arachchige, Z. Chen, W. Jung, W. C. Shin *et al.*, "SAIL: Faster-than-demonstration execution of imitation learning policies," in *Conference on Robot Learning*, 2025.

[9] K. Black, N. Brown, D. Driess, A. Esmail *et al.*, "π_0: A vision-language-action flow model for general robot control," *Robotics: Science and Systems*, 2025.

[10] Y. Dai, R. Ramirez Sanchez, R. Jeronimus, S. Sagheb, C. M. Nunez, H. Nemlekar, and D. P. Losey, "CIVIL: Causal and intuitive visual imitation learning," *arXiv preprint arXiv:2504.17959*, 2025.

[11] C. Wang, L. Fan, J. Sun, R. Zhang, L. Fei-Fei, D. Xu, Y. Zhu, and A. Anandkumar, "MimicPlay: Long-horizon imitation learning by watching human play," in *Conference on Robot Learning*, 2023.

[12] R. Ramirez Sanchez, H. Nemlekar, S. Sagheb, C. M. Nunez, and D. P. Losey, "RECON: Reducing causal confusion with human-placed markers," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2025.

[13] J. Clark, S. Mirchandani, D. Sadigh, and S. Belkhale, "Action-free reasoning for policy generalization," in *ICRA Workshop on Foundation Models and Neuro-Symbolic AI for Robotics*, 2025.

[14] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao *et al.*, "OpenVLA: An open-source vision-language-action model," in *Conference on Robot Learning*, 2025.

[15] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song, "Diffusion policy: Visuomotor policy learning via action diffusion," *The International Journal of Robotics Research*, 2025.

[16] S. Parekh, H. Nemlekar, and D. P. Losey, "Towards balanced behavior cloning from imbalanced datasets," *Autonomous Robots*, 2026.

[17] L. X. Shi, A. Sharma, T. Z. Zhao, and C. Finn, "Waypoint-based imitation learning for robotic manipulation," in *Conference on Robot Learning*, 2023.

[18] T. Gao, S. Nasiriany, H. Liu, Q. Yang, and Y. Zhu, "PRIME: Scaffolding manipulation tasks with behavior primitives for data-efficient imitation learning," *IEEE Robotics and Automation Letters*, 2024.

[19] S. Belkhale, Y. Cui, and D. Sadigh, "HYDRA: Hybrid robot actions for imitation learning," in *Conference on Robot Learning*, 2023.

[20] D. D. Yuan, T. Z. Zhao, K. Burns, and C. Finn, "Speedtuning: Speeding up policy execution with lightweight reinforcement learning," in *IEEE International Conference on Robotics and Automation*, 2025.

[21] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Knowledge Discovery and Data Mining*, 1996.

[22] B. Kim, J. Pahk, C. Lee, J. Kim, J. Lee, T. T. Kim, K. Shim, J. K. Lee, and B.-T. Zhang, "ESPADA: Execution speedup via semantics aware demonstration data downsampling for imitation learning," *arXiv preprint arXiv:2512.07371*, 2025.

[23] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," in *International Conference on Learning Representations*, 2021.

[24] P. Wu, Y. Shentu, Z. Yi, X. Lin, and P. Abbeel, "GELLO: A general, low-cost, and intuitive teleoperation framework for robot manipulators," in *IEEE/RSJ International Conf. on Intelligent Robots and Systems*, 2024.

[25] S. Bai, Y. Cai, R. Chen, K. Chen *et al.*, "Qwen3-VL technical report," *arXiv preprint arXiv:2511.21631*, 2025.

[26] A. Khazatsky, K. Pertsch, S. Nair, A. Balakrishna *et al.*, "DROID: A large-scale in-the-wild robot manipulation dataset," in *RSS 2024 Workshop: Data Generation for Robotics*, 2024.

[27] L. McInnes, J. Healy, S. Astels *et al.*, "hdbscan: Hierarchical density based clustering," *Journal of Open Source Software*, 2017.

[28] B. Tang, I. Akinola, J. Xu, B. Wen, A. Handa *et al.*, "AutoMate: Specialist and generalist assembly policies over diverse geometries," *Robotics: Science and Systems*, 2024.

[29] K. Black, A. Z. Ren, M. Equi, and S. Levine, "Training-time action conditioning for efficient real-time chunking," *arXiv preprint arXiv:2512.05964*, 2025.