arXiv:2608.09138v2 [cs.RO] 11 Aug 2026

SPEEDTUNING: Speeding Up Policy Execution with Lightweight Reinforcement Learning

David D. Yuan, Tony Z. Zhao, Kaylee Burns, and Chelsea Finn

Stanford University

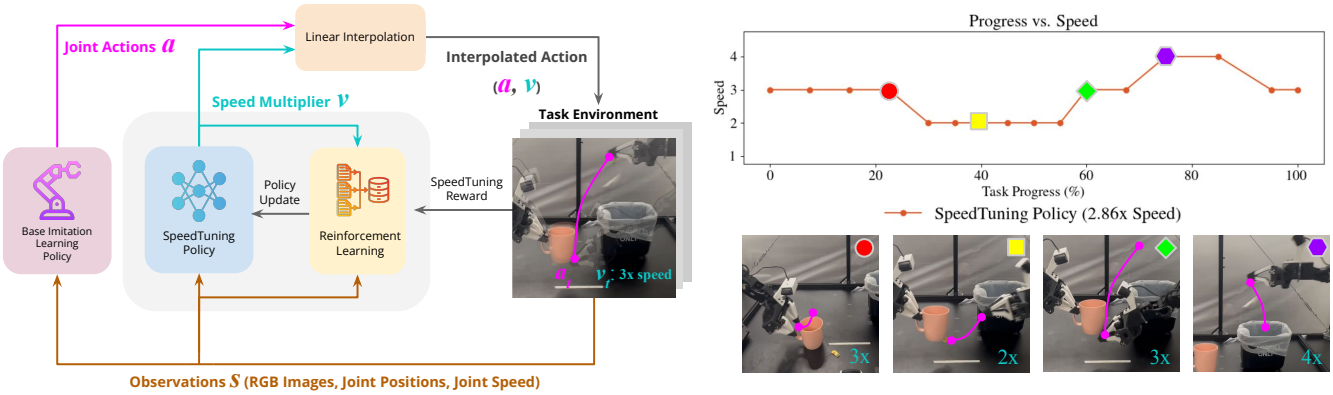

Fig. 1: SPEEDTUNING is a reinforcement learning framework designed to optimize both execution speed and task success for learned manipulation policies. The left panel presents a schematic of SPEEDTUNING augmenting an imitation learning policy, where it outputs a speed multiplier to modulate the execution of predicted actions. This speed policy is optimized using reinforcement learning. The right panel illustrates performance on the Tea Bag Disposal task, with the upper half depicting speed versus task progress and the lower half showing the speed multipliers predicted at different stages. For critical actions, such as grasping the tea bag (yellow marker), the policy selects a lower speed (2x) to ensure precise timing, whereas for less demanding stages, such as the final phase of discarding the tea bag (purple marker), a higher speed (4x) is applied to expedite execution.

***Abstract*— While learned robotic policies hold promise for advancing generalizable manipulation, their practical deployment is often hindered by suboptimal execution speeds. Imitation learning policies are inherently limited by hardware constraints and the speed of the operator during data collection. In addition, there are no established methods for accelerating policies learned via imitation, and the empirical relationship between execution speed and task success remains underexplored. To address these issues, we introduce SPEEDTUNING, a reinforcement learning framework specifically designed to enhance the speed of manipulation policies. SPEEDTUNING learns to predict the optimal execution speed for actions, thereby complementing a base policy without necessitating additional data collection. We provide empirical evidence that SPEEDTUNING achieves substantial improvements in execution speed, exceeding 2.4x speed-up, while preserving an adequate success rate compared to both the original task policy and straightforward speed-up methods such as linear interpolation at a fixed speed. We evaluate our approach across a diverse set of dynamic and precise tasks, including pouring, throwing, and picking, demonstrating its effectiveness and robustness in enhancing real-world robotic manipulation. Videos and code are available at `https://daivdyuan.github.io/speed-tuning/`**

I. INTRODUCTION

Speed is critical in real-world robotic tasks, as faster policies can increase throughput and enhance the user experience. For example, in robotic assembly, high-speed and precise manipulation is essential for tasks like inserting screws or fitting components, where faster execution directly boosts production efficiency. Even though speed is essential in many real-world settings, imitation learning methods largely ignore speed both when training and evaluating a policy.

Training robots to perform tasks quickly and successfully through imitation is challenging. Imitation learning is typically concerned with matching the behavior in the demonstration dataset, while it is hard to collect fast and accurate teleoperation data even from experts. Take ALOHA [1], a widely adopted and performant teleoperation hardware as an example, cutting a piece of tape and sticking it to a box takes more than 15 seconds, and putting a velcro shoe on a foot takes more than 22 seconds. In contrast, humans can perform either of these tasks in under 5 seconds, so simply matching the speed of demonstrations won't result in "human-level" performance. Moreover, many real-world tasks are dynamic, requiring precise coordination of actions over time. For instance, tasks like tossing and pouring demand fine velocity control, while actions such as placing a tea bag into a cup require careful timing to synchronize with the bag's swing. Consequently, naive approaches—such as increasing the underlying speed of the robot by a constant multiplier—are insufficient, as they fail to account for the nuanced relationship between speed and task success.

An ideal speed-up system for imitation learning would

© 2025 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
DOI: 10.1109/ICRA55743.2025.11128753

(1) not require modifications to demonstration collection mechanisms, (2) retrofit existing policies to work at faster speeds, and (3) learn the interaction between execution speed and policy success. Modeling the relationship between speed and performance is especially important in dynamic tasks, where the speed may need to be increased or decreased depending on the stage of the task. For example, when pulling a tea-bag out of a cup, most of the episode can be accelerated, but grasping the tag of the tea-bag must be timed with the swing of the string, as shown in Fig. 1.

We present SPEEDTUNING, a method for accelerating manipulation policies by learning a "speed-policy" that predicts a speed factor for a chunk of future actions. The speed-policy is built on top of a base imitation learning policy. As actions are predicted in the environment, the speed policy takes in the observation history and outputs the velocity of the subsequent action. The speed policy is trained with reinforcement learning and optimizes a task and speed reward. To make this practical and compatible with modern imitation learning methods, most of which predict action chunks $a_{t:t+k}$, we linearly interpolate predicted chunks by the factor predicted by the speed policy. Because the speed policy is trained entirely with reinforcement learning, this acceleration process requires no extra data-collection.

We evaluate SPEEDTUNING across a diverse set of dynamic tasks, including pouring, throwing, and picking, in both simulated and real-world environments. We compare our method against reinforcement learning from scratch and a naive speed-up baseline that uniformly accelerates actions. The results demonstrate that SPEEDTUNING achieves substantial speed-ups (over 2.4x) compared to the baseline imitation-learning policies. Additionally, we conduct ablation studies to analyze the impact of various design choices, such as the choice of task policy, the use of image observations, and the specific image encoder employed, further validating the effectiveness and robustness of our approach.

II. RELATED WORK

Imitation learning. Imitation learning provides a straightforward way to acquire robotic manipulation skills by learning from expert demonstrations [2, 3]. Our work builds on a recent paradigm for imitation learning that combines a puppeteering setup [4, 5, 6, 7], transformer architecture [8], and action chunking to learn dexterous skills [1, 9]. Recent advancements in imitation learning highlight the surprising effectiveness of imitation learning for rapid skill acquisition [9, 10, 11]. However, the quality of learned skills remains inherently tied to the quality of demonstrations, which can be limited by unintuitive teleoperation interfaces and demonstrator sub-optimality [12]. Furthermore, the current paradigm lacks mechanisms to modify demonstrations or learn policies post-hoc that better optimize for task demands. We focus specifically on optimizing for execution speed and introduce a novel framework to accelerate existing imitation-learned policies for higher-speed manipulation tasks without requiring additional demonstrations or sacrificing performance.

Reinforcement learning in robotics. A reinforcement learning (RL) agent takes actions in an environment to maximize cumulative reward [13, 14, 15, 16]. RL in robotic environments [17, 18, 19] evades the issues of sub-optimal demonstration data, but state-of-the art RL algorithms still require extensive real-world interactions to learn complex control policies from scratch [20]. Roboticists have successfully fine-tuned policies pre-trained with imitation learning [21, 22]. Unlike these works, we train a new policy that predicts the *velocity* at which the base policy runs.

Fast robot manipulation. The pursuit of fast robot manipulation, particularly in dynamic tasks, has traditionally centered around optimizing the robot's hardware or integrating speed into the reward function. These approaches have been successful in highly dynamic scenariossuch as table tennis [23, 24]. Dynamic manipulation tasks, such as cable manipulation [25, 26, 27, 28], throwing objects [29, 30], and high-throughput pick-and-place [31, 32, 33] also benefit from speed-focused objectives and hardware solutions. Unlike these works, SPEEDTUNING is compatible with many base policy learning methods, which means that it can be incorporated on top of already performant robotic policies.

III. SPEED TUNING

We introduce SPEEDTUNING, a method for accelerating manipulation policies learned from demonstrations while optimizing for both execution speed and task success. We achieve this by decoupling the policy into two components: a task policy trained with imitation learning to mimic expert demonstrations, which is summarized in Section III-A, and a speed policy trained with reinforcement learning to predict optimal action velocities that balance speed and success. The SPEEDTUNING objective is described in Section III-B. We then present a baseline approach in Section III-F that uniformly accelerates actions to establish the empirical trade-off between speed and success. In Section III-E, we detail how SPEEDTUNING uses RL to predict the velocity of executed actions based on current state to achieve a better combined success and speed than naive speed-up techniques.

A. Preliminaries

We assume access to a set of expert demonstrations $\mathcal{D} = \{(\tau_1, \ldots, \tau_N)\}$ where τ_n is a trajectory defined as a sequence of state-action pairs: $\tau = [(s_1, a_1) \ldots (s_T, a_T)]$. Under the standard imitation learning paradigm, learning a policy, π, parameterized by θ is a matter of minimizing the below loss function to maximize the likelihood of the expert actions:

$$L_\theta = \mathbb{E}_\mathcal{D} \left[d(\pi_\theta(s_i), a_i) \right] \tag{1}$$

where d is any distance metric, usually Manhattan or Euclidean.

These demonstrations are considered "expert" in that they are sampled from a behavior policy optimizing for task reward, r_{task}, which we assume to be a binary reward indicating whether the task was completed during a given transition. This reward signal can be derived either from the structure of the environment or through human-in-the-loop feedback, particularly in real-world reinforcement learning

Algorithm 1 SPEEDTUNING Training

Given: Demonstration dataset $\mathcal{D}_{\text{task}}$, speed reward function $r_{\text{speed}}(v)$ speed reward weight α, frame skip constant k_{skip}, frame stack constant k_{stack}, action chunk size k_c
Initialize task policy $\pi_\theta(a_{t:t+k_c}|s_t)$
Initialize speed policy $\pi_\varphi(v_t|s_t)$
Initialize replay buffer $\mathcal{D}_{\text{replay}}$
Initialize observation buffer $\mathcal{O}$
Train the task policy $\pi_\theta(a_{t:t+k_c}|s_t)$ using the task dataset $\mathcal{D}_{\text{task}}$
for each episode **do**
  Clear observation buffer $\mathcal{O} = []$
  for each timestep t **do**
    Obtain current observation $s_{(t,0)}$
    Append current observation to buffer $\mathcal{O} = \mathcal{O} \cup s_{(t,0)}$
    Predict the speed $v_t = \pi_\varphi(\mathcal{O}[-k_{\text{stack}}:])$
    Predict the task action sequence $A_t := a_{t:t+k_c} = \pi_\theta(s_{(t,0)})$
    Interpolate the action based on speed $a^{(A_t, v_t)}$
    for iteration $i = 1, 2, \ldots, k_{\text{skip}}$ **do**
      Apply interpolated action $a'_i := a_i^{(A_t, v_t)}$
      Obtain next observation $s_{(t,i)}$
      Append new observation to buffer $\mathcal{O} = \mathcal{O} \cup s_{(t,i)}$
      Obtain task reward $r_{\text{task}}(s_{(t,i-1)}, a'_i)$
      Compute the combined reward:

$$r_{ST} = \alpha \cdot r_{\text{speed}}(v_t) + r_{\text{task}}(s_{(t,i-1)}, a'_i)$$

      Store transition in the replay buffer:

$$\mathcal{D}_{\text{replay}} = \mathcal{D}_{\text{replay}} \cup (s_{(t,i-1)}, a'_i, r_{ST}, s_{(t,i)})$$

    end for
  end for
  Sample mini-batches from the replay buffer $\mathcal{D}_{\text{replay}}$ to update the speed policy $\pi_\varphi(v_t|s_t)$
end for

scenarios where automated reward formulation is insufficient.

$$r_{\text{task}}(s_t, a_t) = 1_{\{s_{t+1} \in S_{\text{success}}\}} \tag{2}$$

Absent from this framework is an understanding of how to speed up policy execution. Typically, we assume actions are executed at a constant velocity, v. In the next section, we consider an updated objective where the velocity of actions is a component of the action space.

B. SPEEDTUNING *Objective*

To extend our framework to account for action execution speed, we introduce an updated objective function that incorporates velocity as a component of the action space. We define the objective $J(\psi)$ as the expected cumulative reward over trajectories sampled from a policy $\pi_\psi(a_t, v_t|s_t)$, where the reward at each time step t consists of a weighted speed reward $\alpha \cdot r_{\text{speed}}(v_t)$ and the task reward $r_{\text{task}}(s_t, a_t)$. We restrict the selection of velocities to a discrete set: $V = \{v_1, v_2, \ldots, v_K\}$ to simplify policy optimization.

$$J(\psi) = \mathbb{E}_{\tau \sim \pi_\psi}\left[\sum_{t=1}^{T} \left(\alpha \cdot r_{\text{speed}}(v_t) + r_{\text{task}}(s_t, a_t)\right)\right] \tag{3}$$

By introducing the weighting term $\alpha \geq 0$ to the velocity reward, we can adjust the trade-off between execution speed and task success. This allows us to describe the Pareto curve of optimality, optimizing for either faster execution (higher α) or higher task success rate (lower α).

C. Factorizing Task and Speed

We decompose the execution policy into two distinct sub-policies: a task policy, $\pi_\theta(a_t|s_t)$, and a speed policy, $\pi_\varphi(v_t|s_t)$. This not only makes optimizing for speed and success more tractable, but also enables us to accelerate any parameterization of π_θ. In our method, the task policy is trained to regress actions with imitation learning. For the speed policy, we explore both constant and non-linear speed policies, which are described in the next two sections.

D. Constant Velocity

In scenarios where the velocity v is constant, the objective simplifies to:

$$J(\psi) = \alpha \cdot vT + \mathbb{E}_{\tau \sim \pi_\psi}\left[\sum_{t=1}^{T} r_{\text{task}}(s_t, a_t)\right]. \tag{4}$$

This formulation enables us to explore different points along the Pareto frontier by varying the weighting term α, thereby balancing the trade-off between speed and task performance. However, in practice, more complex non-linear speed reward functions calculated at each transition are often preferred, which we discuss in the next section.

E. SPEEDTUNING*: Learning a Speed-Adaptive Policy*

While uniformly increasing the velocity provides a simple method for accelerating action sequences, it lacks adaptability to different task contexts. Our aim is to develop a more general closed-loop policy where acceleration is conditioned on the current state. For instance, in a pick-and-place scenario, an optimal speed policy would likely slow down during critical actions, such as grasping or placing, to ensure successful execution, and speed up during less critical movements to maximize overall efficiency.

To achieve this, SPEEDTUNING seeks to balance swift task execution with maintaining a high success rate. Specifically, we use reinforcement learning with the objective in Equation 3 to train a speed policy $\pi_\varphi(v_t|s_t)$, which predicts an optimal speed v_t for a fixed task policy π_θ:

$$J(\varphi) = \mathbb{E}_{\substack{a_t \sim \pi_\theta \\ v_t \sim \pi_\varphi}}\left[\sum_{t=1}^{T} \left(\alpha \cdot r_{\text{speed}}(v_t) + r_{\text{task}}(s_t, a_t)\right)\right] \tag{5}$$

To optimize the objective in Equation 5 we employ Rainbow DQN [34], which is a value-based off-policy algorithm for discrete action spaces, because of its strong performance and high sample efficiency. Adapting the Rainbow DQN objective for our SPEEDTUNING gives the following optimal speed policy and Bellman equation:

$$\pi^*_\varphi(s_t, a_t) = \arg\max_{v_t} Q^*(s_t, a_t, v_t), \tag{6}$$

$$r_{\text{ST}}(s_t, a_t, v_t) = \alpha \cdot r_{\text{speed}}(v_t) + r_{\text{task}}(s_t, a_t), \tag{7}$$

$$Q^{\pi_\varphi}(s_t, a_t, v_t) = r_{\text{ST}}(s_t, a_t, v_t) + \gamma\, \mathbb{E}_{s_{t+1}}\left[Q^{\pi_\varphi}(s_{t+1}, a_{t+1}, v_{t+1})\right], \tag{8}$$

where γ is the discount factor.

We made several careful design choices for the speed network. As described in Section III-B, we limit the output speed v to a discrete set of possible values. We also introduce

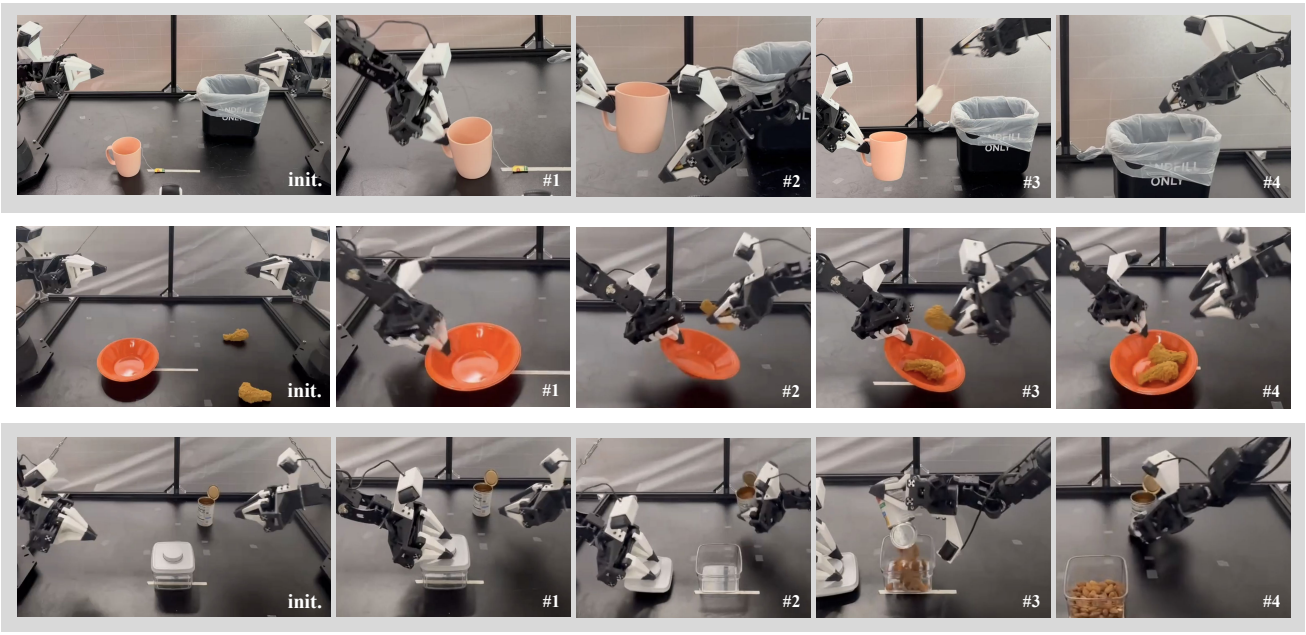

Fig. 3: Initialization and key actions for three real-world tasks (top to bottom): Tea Bag Disposal, Food Preparation, and Almond Pouring.

an additional hyperparameter, β, to further scale the speed reward: $r_{\text{speed}}(v) = v^{\beta}$. To manage complexity in long-horizon tasks, we use a frame skip constant $k_{\text{skip}} \in \mathbb{Z}^{+}$, querying the speed policy only once every k_{skip} steps, while continuing executing the accelerated policy with the latest speed between queries. To better preserve the Markovian properties, we adopt a frame stack of k_{stack} recent observations as input. However, the most important design decision we make is to predict an action *chunk* at each time step. This makes the implementation of the velocity change more straightforward and reduces the effective horizon of the task. We describe the concrete interpretation of velocity when predicting action chunks in the next section.

F. Interpolation over Action Chunks

Instead of manually scaling the velocity at which actions are run, we assume that the base policy predicts chunks of actions of size k for joint positions: $\pi_\theta(A_t|s_t)$ where $A_t = a_{t:t+k}$. To modulate the execution speed of actions generated by the task policy, we employ *linear temporal interpolation*.

With linear temporal interpolation, the i-th accelerated action at speed v for time step t is denoted as $a_i^{(A_t,v)}$, with $a_1^{(A_t,v)}$ selected as the next action to be executed at the current time step.

Formally, we define linear temporal interpolation over a function $f : \mathbb{Z} \mapsto \mathbb{R}^n$ with a step length of v at point t as:

$$\text{interp}_{f,v}(t) = f(\lfloor vt \rfloor) + \frac{vt - \lfloor vt \rfloor}{v}\big(f(\lfloor vt+1 \rfloor) - f(\lfloor vt \rfloor)\big). \tag{9}$$

This linear temporal interpolation effectively accelerates the execution of the function f by a factor of v. For simplicity, we denote a trajectory of actions as $A = a_{0:k}$. The temporal action function $f_a^{(A)} : \mathbb{Z} \mapsto \mathbb{R}^n$ is defined as:

$$f_a^{(A)}(i) = a_i^{(A)}, \quad 0 \le i < k+1. \tag{10}$$

The accelerated action at speed v over the trajectory A at step i is then:

$$a_i^{(A,v)} = \text{interp}_{f_a^{(A)},v}(i), \quad 0 \le i \le \left\lfloor \frac{k+1}{v} \right\rfloor. \tag{11}$$

In summary, this interpolation method allows us to integrate the speed policy's output, v, with the action chunks generated by a task policy by decreasing the effective chunk-size down to $\frac{k+1}{v}$.

IV. Experiments

In this section, we present a comprehensive evaluation of SpeedTuning to demonstrate its effectiveness in accelerating policy execution. In the following sections, we discuss the implementation details and describe our simulated and real-world dynamic manipulation tasks. Then, we present our key results, which show that SpeedTuning achieves significant speed-ups while maintaining solid policy performance. Finally, we ablate the design choices such as the base action policy and the parameterization of the reward function. Overall, we find that our method robustly produces the policy with the best tradeoff between speed and success.

A. Implementation Details

Imitation Learning Details. Action Chunking with Transformers (ACT) [1] models the policy distribution $p(A_t|s_t)$ as a transformer-based Conditional Variational Autoencoder (CVAE) [35, 36]. The decoder outputs a chunk of actions of size k_c. Chunking is particularly important for high-frequency, fine-grained manipulation tasks, with chunk sizes k_c being as high as 100 [1]. 3) The input observations o_t includes both proprioception joint positions and camera image embedding through a pretrained image encoder, which enables the algorithm to be effectively used on real robots while not adding too much computing overhead.

RL details RainbowDQN [34] models the Q function in discrete action space $Q(q_t|s_t, a_t)$, enhancing DQN with several improvements: double Q-learning [37] reduces overestimation bias, prioritized experience replay [38] increases sam-

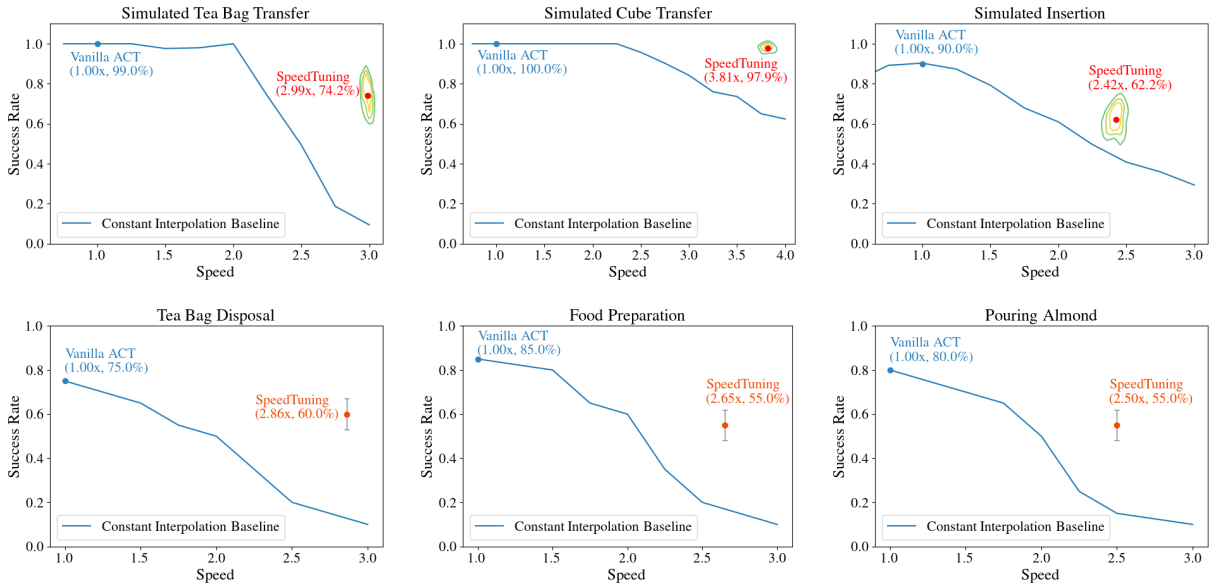

Fig. 4: Results showing the SPEEDTUNING policy on simulated (top row) and real world (bottom row) tasks. For simulated tasks, the baseline curve is obtained by evaluating 100 episodes across various speeds at 0.1 intervals, while the distribution and mean of the SPEEDTUNING policy is calculated over 2000 episodes. For the real-world tasks, the baseline curve is obtained by evaluating 20 episodes across various speeds at 0.25 intervals, while the mean and error of the SPEEDTUNING policy is calculated over 20 episodes. Due to the sample efficiency constraints of reinforcement learning in real-world settings, the policy was not trained to full convergence, which may account for the slightly lower observed success rate.

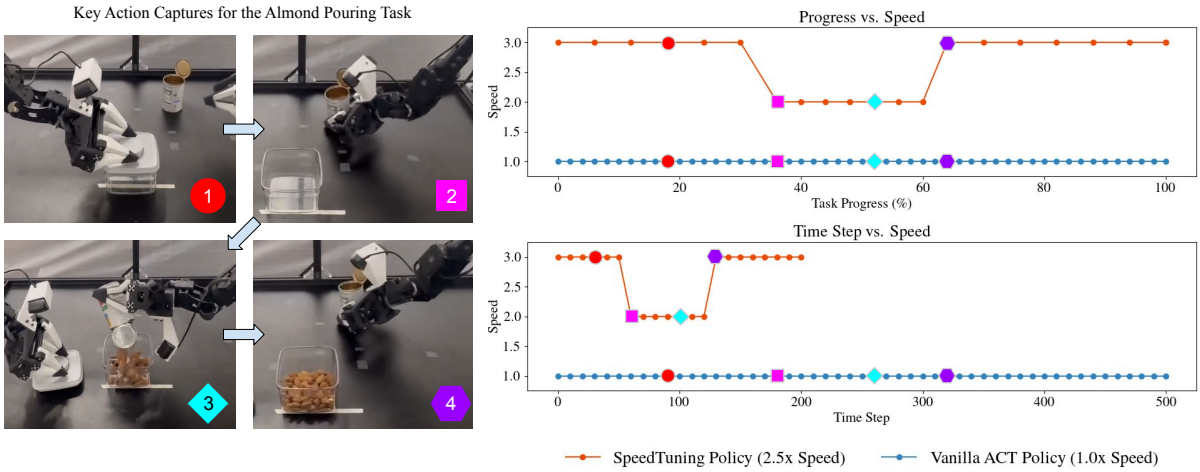

Fig. 5: Comparison of the SPEEDTUNING and vanilla ACT policy trajectories for the Almond Pouring task, featuring key action captures and speed-up values throughout the task. The results highlight SPEEDTUNING's task understanding, significantly accelerating overall execution by 2.5x (200 time steps vs. 500 time steps), while deliberately slowing down during the critical, dynamic pouring phase.

pling efficiency, and the dueling architecture [39] improves performance in environments where action values are state-independent. Additionally, distributional Q-learning [40] captures reward uncertainty by modeling a categorical distribution over Q-values, making it well-suited for high-dimensional, dynamic tasks in fine-grained manipulation.

B. Robot Setup and Tasks

We conducted all our experiments based on the ALOHA setup for simulated and real-world tasks. The bimanual hardware setup operates at 50Hz, consists of a 14-dimensional action space, and the observation states include proprioception (joint position and speed) and 4 RGB cameras.

We conducted evaluations across six fine-grained manipulation tasks, divided equally between simulated and real-world environments. These tasks encompass long-horizon, contact-rich, and dynamic manipulation scenarios, reflecting the complexity of daily robotic applications.

Simulated Cube Transfer: This task requires the robot to pick up and transfer a cube between two grippers, assessing the system's ability to handle basic object manipulation.

Simulated Peg Insertion: In this task, the robot must pick up and accurately insert a peg into a cube with a hole, demanding precise and bimanual manipulation.

Simulated Tea Bag Transfer: This task involves picking up and transferring a tea bag from a desktop into a cup. The dynamic nature of this task, including the swinging motion of the tea bag during transfer, presents significant challenges for our approach.

Tea Bag Disposal: Extending the simulated tea bag transfer to the real world, this task requires more precise

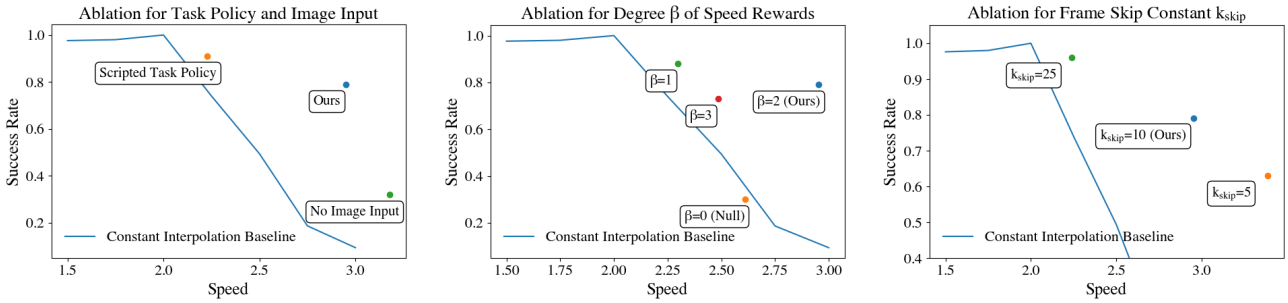

Fig. 6: Ablation study on task policy, speed rewards, and frame skip values. All configurations are evaluated over 2000 episodes on Simulated Tea Bag Transfer.

bimanual and dynamic manipulation. The robot must first pick up a mug containing a tea bag and then carefully dispose of the tea bag into a garbage bin.

Food Preparation: This task involves a sequence of precise maneuvers: picking up a plate with one gripper, picking up and throwing two pieces of chicken from the desktop onto the plate, and finally placing the served food at a designated location on the desktop.

Pouring Almond: This highly precise and dynamic task involves opening the lid of a container and pouring almonds from a can into the container.

C. Key Results

SPEEDTUNING significantly accelerates the policy. As illustrated in Fig. 4 and 4, SPEEDTUNING achieves over a $2.4\times$ speed-up across all six tasks compared to the original ACT task policy. We observed a trend where higher speed-ups are attained in tasks requiring less precision, such as Simulated Cube Transfer and Simulated Tea Bag Transfer. This demonstrates that achieving higher speed-ups necessitates a substantial level of precision in task execution.

SPEEDTUNING effectively retains the success rate at high speed. While accelerating the base policy, SPEEDTUNING maintains a high success rate even at increased speeds, significantly outperforming the universal interpolation baseline. Notably, SPEEDTUNING emerges as an outlying point beyond the Pareto curve defined by the baseline. This trend is particularly pronounced in more dynamic tasks, such as Simulated Tea Bag Transfer, real-world Tea Bag Disposal, and Almond Pouring. This highlights that naive acceleration, as in the baseline, is insufficient for dynamic tasks, requiring a deeper understanding of task-specific dynamics—such as selecting the appropriate speed for transferring a swinging tea bag—something SPEEDTUNING effectively accomplishes through reinforcement learning.

SPEEDTUNING learns the critical parts of the tasks to act accordingly. To achieve high acceleration while maintaining success rates, SPEEDTUNING intelligently segments the task based on dynamic and contact-rich phases, adjusting its speed accordingly. For instance, as shown in Fig. 5, the resulting policy for Almond Pouring demonstrates this capability: SPEEDTUNING learns to maintain a high speed during the lid-opening and can-picking phases, deliberately slows down during the transfer and pouring stages to prevent spilling, and then resumes high speed to place the can back.

V. ABLATIONS

In this section, we conduct ablation studies on various design choices for SPEEDTUNING. All experiments are carried out using Simulated Tea Bag Transfer due to its precise and dynamic nature, as well as its efficiency in simulation.

Choice of Task Policy: Replacing the learned ACT task policy in SPEEDTUNING with an open-loop, scripted policy eliminates the learned generalization capabilities of the task policy. As shown in Fig. 6, the scripted policy results in lower acceleration and a slightly higher success rate. This underscores the importance of generalizing across various speeds for the underlying task policy in SPEEDTUNING.

Image Observations: The speed policy uses both proprioceptive and visual observations for speed predictions. To evaluate whether proprioceptive data alone suffices to represent the environment state, we removed the image input from the policy. Shown in Fig. 6, omitting image observations led to less stable training and lower success rates and acceleration. These results highlight the necessity of incorporating visual information for more efficient task execution.

Degree of Speed Rewards: The speed reward is formulated as $r_{\text{speed}}(v) = v^{\beta}$. As shown in Fig. 6, varying the degree of the speed reward, β, has a significant impact on SPEEDTUNING performance. When β is large, the speed policy prioritizes velocity over task completion, resulting in lower success rates. Conversely, when $\beta = 0$, there is no direct incentive for speed, and acceleration relies solely on the standard discount factor γ, leading to unstable training.

Frame Skip: In SPEEDTUNING, a frame skip of 10 is used to shorten the horizon of the MDP process. While smaller frame skips provide finer granularity, the extended horizon diminishes the emphasis on terminal success rewards. Conversely, larger frame skips improve short-horizon properties but reduce control precision over speed adjustments. As shown in Fig. 6, smaller frame skips with longer horizons lead to suboptimal speed policies, while larger skips sacrifice the granularity needed for achieving higher acceleration.

VI. CONCLUSION

We introduced SPEEDTUNING, a reinforcement learning framework designed to optimize both execution speed and task success for learned manipulation policies. By training a speed policy to dynamically adjust action velocities based on the current state, SPEEDTUNING achieved over 2.4x

speed-ups across various dynamic and precise tasks while maintaining adequate success rates compared to baseline methods. Overall, we hope our solution provides valuable insights into the relationship between execution speed and task success in robotic manipulation. Future work should focus on enhancing sample efficiency to make the method more practical for real-world applications.

REFERENCES

[1] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn, "Learning fine-grained bimanual manipulation with low-cost hardware," *arXiv preprint arXiv:2304.13705*, 2023.

[2] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *NIPS*, 1988.

[3] S. Schaal, "Is imitation learning the route to humanoid robots?" *Trends in Cognitive Sciences*, vol. 3, pp. 233–242, 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:7124120

[4] T. Hulin et al., "The dlr bimanual haptic device with optimized workspace," *2011 IEEE International Conference on Robotics and Automation*, pp. 3441–3442, 2011.

[5] B. Katz, "A low cost modular actuator for dynamic robots," 2018.

[6] M. Schwarz, C. Lenz, A. Rochow, M. Schreiber, and S. Behnke, "Nimbro avatar: Interactive immersive telepresence with force-feedback telemanipulation," *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5312–5319, 2021.

[7] P. Wu, Y. Shentu, Z. Yi, X. Lin, and P. Abbeel, *Gello: A general, low-cost, and intuitive teleoperation framework for robot manipulators*, 2023.

[8] A. Vaswani et al., "Attention is all you need," *ArXiv*, vol. abs/1706.03762, 2017.

[9] C. Chi et al., "Diffusion policy: Visuomotor policy learning via action diffusion," *arXiv preprint arXiv:2303.04137*, 2023.

[10] P. R. Florence et al., "Implicit behavioral cloning," *ArXiv*, vol. abs/2109.00137, 2021.

[11] E. Jang et al., "Bc-z: Zero-shot task generalization with robotic imitation learning," in *Conference on Robot Learning*, 2022.

[12] A. Kumar, J. Hong, A. Singh, and S. Levine, "Should i run offline reinforcement learning or behavioral cloning?" In *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=AP1MKT37rJ

[13] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[14] R. S. Sutton and A. G. Barto, *Reinforcement learning - an introduction* (Adaptive computation and machine learning). MIT Press, 1998, ISBN: 978-0-262-19398-6.

[15] R. S. Sutton, D. A. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Neural Information Processing Systems*, 1999.

[16] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Neural Information Processing Systems*, 1999.

[17] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

[18] S. S. Gu, E. Holly, T. P. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3389–3396, 2016.

[19] M. Andrychowicz et al., "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, pp. 20–3, 2018.

[20] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: Lessons we have learned," *The International Journal of Robotics Research*, vol. 40, pp. 698–721, 2021.

[21] R. C. Julian, B. Swanson, G. S. Sukhatme, S. Levine, C. Finn, and K. Hausman, "Never stop learning: The effectiveness of fine-tuning in robotic reinforcement learning," in *Conference on Robot Learning*, 2020.

[22] J. Yang, M. S. Mark, B. Vu, A. Sharma, J. Bohg, and C. Finn, "Robot fine-tuning made easy: Pre-training rewards and policies for autonomous real-world reinforcement learning," *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4804–4811, 2023.

[23] D. Büchler, S. Guist, R. Calandra, V. Berenz, B. Scholkopf, and J. Peters, "Learning to play table tennis from scratch using muscular robots," *IEEE Transactions on Robotics*, vol. 38, pp. 3850–3860, 2020.

[24] D. B. D'Ambrosio et al., "Achieving human level competitive robot table tennis," 2024.

[25] Y. Yamakawa, A. Namiki, M. Ishikawa, and M. Shimojo, "Knotting manipulation of a flexible rope using a high-speed multifingered hand and high-speed visual and tactile sensory feedback," *Journal of the Robotics Society of Japan*, vol. 27, pp. 1016–1024, 2009.

[26] Y. Yamakawa, A. Namiki, and M. Ishikawa, "Motion planning for dynamic knotting of a flexible rope with a high-speed robot arm," *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 49–54, 2010.

[27] Y. Yamakawa, A. Namiki, and M. Ishikawa, "Dynamic high-speed knotting of a rope by a manipulator," *International Journal of Advanced Robotic Systems*, vol. 10, 2013.

[28] H. Zhang, J. Ichnowski, D. Seita, J. Wang, and K. Goldberg, "Robots of the lost arc: Learning to dynamically manipulate fixed-endpoint ropes and cables," *ArXiv*, vol. abs/2011.04840, 2020.

[29] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," *Ieee Transactions On*

Robotics, 2019. DOI: 10.1109/TRO.2020.2988642

[30] H. Ha and S. Song, "Flingbot: The unreasonable effectiveness of dynamic manipulation for cloth unfolding," in *Conference on Robot Learning*, 2021.

[31] V. Nabat, M. de la O Rodríguez, O. Company, S. Krut, and V. Pierrot, "Par4: Very high speed parallel robot for pick-and-place," *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 553–558, 2005.

[32] S. D. Han, S. W. Feng, and J. Yu, "Toward fast and optimal robotic pick-and-place on a moving conveyor," *IEEE Robotics and Automation Letters*, vol. 5, pp. 446–453, 2019.

[33] H. Pham and Q.-C. Pham, "Critically fast pick-and-place with suction cups," *2019 International Conference on Robotics and Automation (ICRA)*, pp. 3045–3051, 2018.

[34] M. Hessel et al., "Rainbow: Combining improvements in deep reinforcement learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[35] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[36] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," *Advances in neural information processing systems*, vol. 28, 2015.

[37] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16, Phoenix, Arizona: AAAI Press, 2016, pp. 2094–2100.

[38] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *CoRR*, vol. abs/1511.05952, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:13022595

[39] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16, New York, NY, USA: JMLR.org, 2016, pp. 1995–2003.

[40] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17, Sydney, NSW, Australia: JMLR.org, 2017, pp. 449–458.

APPENDIX
SIMULATION ROLLOUTS AND ADDITIONAL ABLATIONS

A. Simulated Rollout Progressions

Figure 7 provides a qualitative view of SPEEDTUNING on the three simulated tasks. Cube Transfer proceeds from grasping to a bimanual handoff; Peg Insertion requires grasping, alignment, and insertion; and Tea Bag Transfer requires transporting the suspended tea bag into the receptacle. All frames are taken from the original evaluation rollouts.

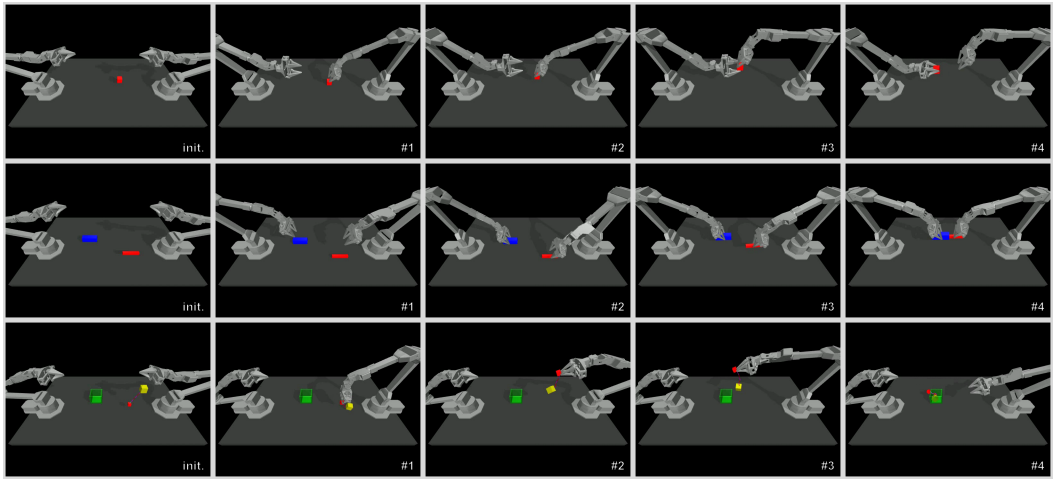

Fig. 7: Representative SPEEDTUNING rollouts in simulation. Rows show Cube Transfer, Peg Insertion, and Tea Bag Transfer; columns show the initial state followed by four successive task stages.

B. Task Policy

The following plots show the original training curves underlying the endpoint ablations in Fig. 6; all configurations are evaluated on Simulated Tea Bag Transfer. In Fig. 8, learned ACT supports substantially higher acceleration, whereas the scripted policy maintains a higher success rate.

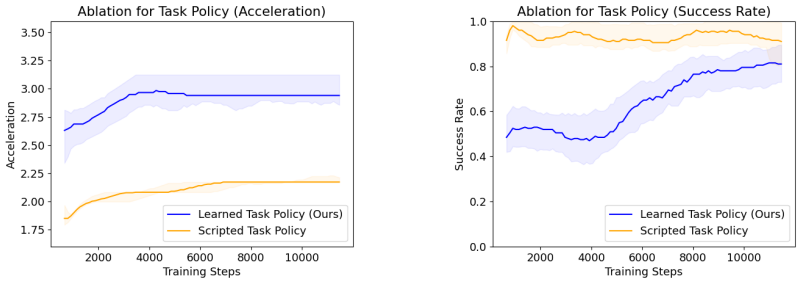

Fig. 8: Base-policy ablation: acceleration (left) and success rate (right). Learned ACT is compared with an open-loop scripted policy.

C. Image Observations

Removing image observations produces high nominal acceleration but a much lower success rate (Fig. 9). Visual input is therefore necessary for identifying task phases in which aggressive acceleration would compromise completion.

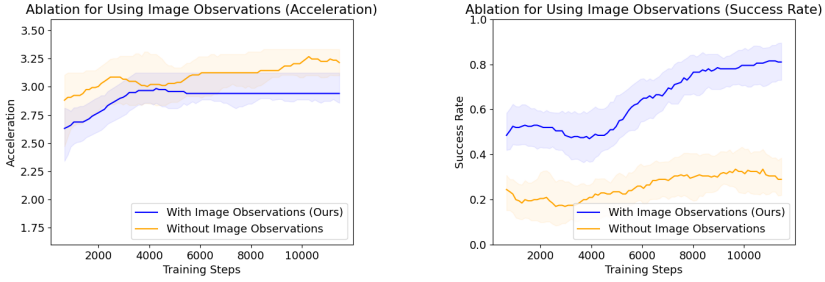

Fig. 9: Visual-observation ablation: acceleration (left) and success rate (right). Both variants retain proprioception.

D. Speed-Reward Exponent

The exponent β controls how strongly the reward favors increased speed. Figure 10 shows that $\beta = 2$ provides the strongest combined acceleration and final success; $\beta = 0$ is less stable, while $\beta = 1$ and $\beta = 3$ favor lower-acceleration operating points.

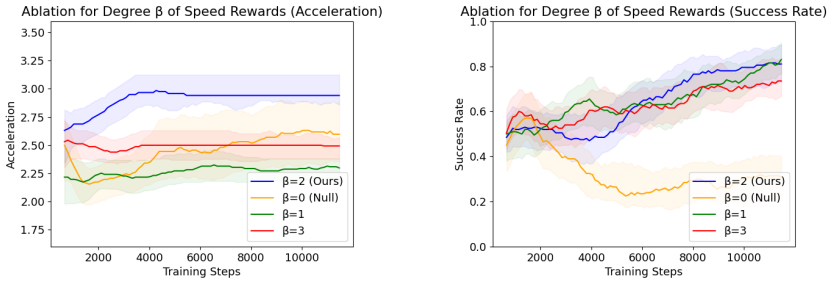

Fig. 10: Speed-reward ablation: acceleration (left) and success rate (right). The main experiments use $\beta = 2$.

E. Frame Skip

Frame skip f_s sets the interval between speed-policy decisions. In Fig. 11, $f_s = 5$ favors acceleration, $f_s = 25$ favors success, and the selected value $f_s = 10$ balances the two objectives.

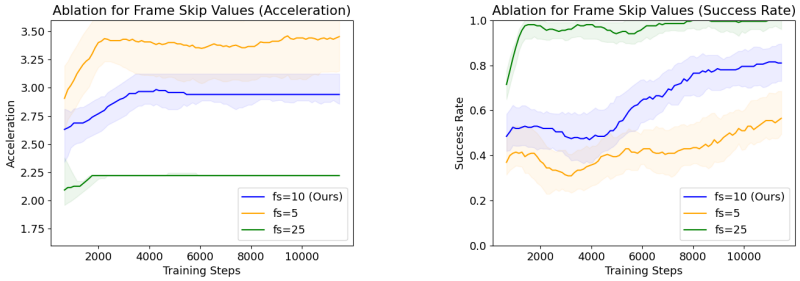

Fig. 11: Frame-skip ablation: acceleration (left) and success rate (right). The main experiments use $f_s = 10$.

F. Image Encoder

All three encoders reach similar final acceleration (Fig. 12). The randomly initialized encoder learns success more slowly, while the pretrained ResNet-18 attains the highest final success, showing that the speed policy need not reuse the task-policy encoder.

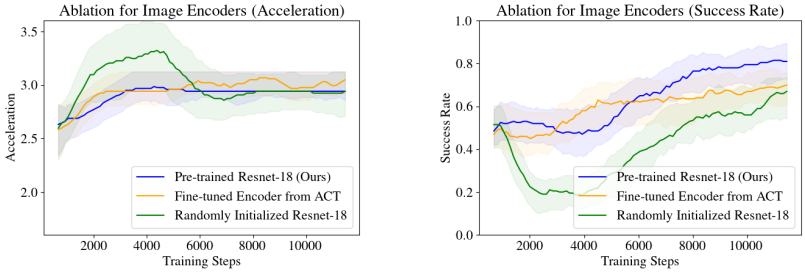

Fig. 12: Image-encoder ablation: acceleration (left) and success rate (right).