Real-Time Execution of Action Chunking Flow Policies

Kevin Black[1,2,] **Manuel Y. Galliker**[1] **Sergey Levine**[1,2]
[1]Physical Intelligence [2]UC Berkeley
{kevin,manuel,sergey}@physicalintelligence.company

Abstract

Modern AI systems, especially those interacting with the physical world, increasingly require real-time performance. However, the high latency of state-of-the-art generalist models, including recent vision-language-action models (VLAs), poses a significant challenge. While action chunking has enabled temporal consistency in high-frequency control tasks, it does not fully address the latency problem, leading to pauses or out-of-distribution jerky movements at chunk boundaries. This paper presents a novel inference-time algorithm that enables smooth asynchronous execution of action chunking policies. Our method, real-time chunking (RTC), is applicable to any diffusion- or flow-based VLA out of the box with no re-training. It generates the next action chunk while executing the current one, "freezing" actions guaranteed to execute and "inpainting" the rest. To test RTC, we introduce a new benchmark of 12 highly dynamic tasks in the Kinetix simulator, as well as evaluate 6 challenging real-world bimanual manipulation tasks. Results demonstrate that RTC is fast, performant, and uniquely robust to inference delay, significantly improving task throughput and enabling high success rates in precise tasks—such as lighting a match—even in the presence of significant latency. See `https://pi.website/research/real_time_chunking` for videos.

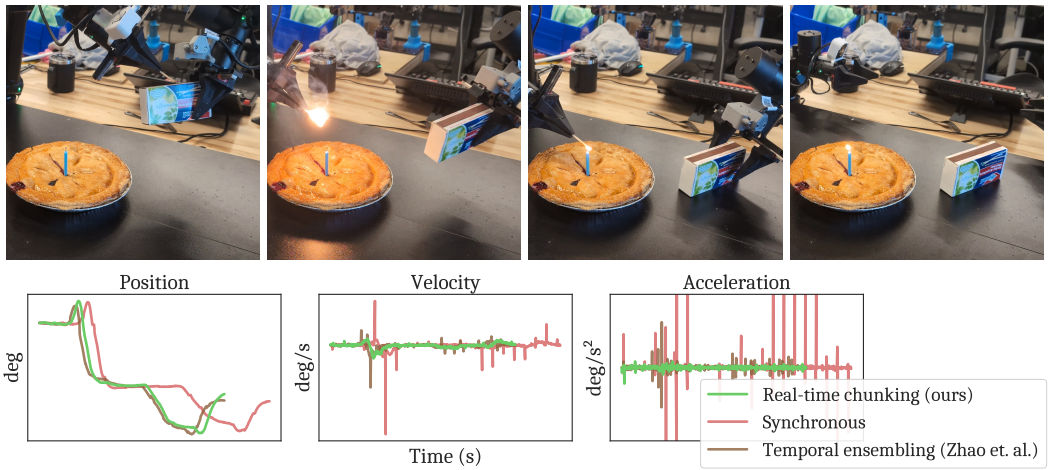

Figure 1: **Top:** Real-time chunking (RTC) enables the robot to perform highly dexterous and dynamic tasks, such as lighting a match—even in the presence of inference delays in excess of 300 milliseconds, corresponding to more than 30% of the model's prediction horizon. **Bottom:** RTC performs the same robot motion 20% faster than synchronous inference [5, 29, 8, 23, 30, 58], and smoother than all competing methods, including temporal ensembling [67]. The shown positions, velocities, and accelerations correspond to the shoulder joint of one arm, and are taken from the first 10 seconds of a real autonomous match-lighting rollout.

Preprint. Under review.

1 Introduction

As AI systems have become more capable, they have also interacted more and more directly with their environment. Whether they're executing terminal commands [44], playing Pokémon on livestream [19], or browsing the web on your behalf [64], recent advances—driven primarily by large-scale deep learning—have enabled these systems to increasingly *control*, rather than merely *process*, the vast heterogeneity of the outside world. Embodied agents, where machine learning models directly control real, physical constructs, are perhaps the quintessential example. The same advances fueling agentic language and vision models are also making great strides in physical intelligence on platforms ranging from humanoid robots [4] to autonomous cars [59].

Cyber-physical systems, unlike chatbots and image generators, always operate in *real time*. While a robot is "thinking", the world around it evolves according to physical laws. Thus, delays between inputs and outputs have a tangible impact on performance. For a language model, the difference between fast and slow generation is a satisfied or annoyed user; for a robot action model, on the other hand, it could be the difference between a robot handing you a hot coffee or spilling it in your lap.

Unfortunately, the effectiveness of modern large-scale machine learning comes with high latency as an unavoidable side effect. Large language models (LLMs), vision-language models (VLMs), and vision-language-action models (VLAs)—the last referring to a class of models designed for visuomotor control—have billions of parameters [8, 29, 5, 4, 57]. These models are not only slow to run, but also require heavy-duty hardware that is difficult to attach to edge devices such as mobile robots, adding even more overhead for remote inference. Edge hardware will improve over time, but as robot datasets grow in size, so will the best VLAs [27].

Thus, applying large models to real-time control problems effectively will require some form of asynchronicity: that is, a model must think about its future actions while executing a previous one. Action chunking [67, 32, 11], where a model outputs and executes a sequence of multiple actions for each inference call, presents a partial solution. Although action chunking has already achieved many state-of-the-art results in dexterous manipulation [5, 4, 57], it still suffers from the latency problem. Chunking sacrifices the reactivity of a system to external stimuli and also introduces discontinuities in the transition points between chunks, as adjacent chunks may jump between different modes (or "strategies") from the learned action distribution. Such anomalies are especially harmful to learning-based systems, as they produce a distribution shift in dynamics that the model is likely not equipped to handle. Naive smoothing strategies, such as averaging multiple predictions together [67], are not guaranteed to produce valid actions and may only make matters worse (e.g., see Figure 2).

A good real-time system must produce a consistent and continuous control signal, incorporating the latest observations without perturbing the environment's natural dynamics or the model's ability to produce correct actions. In this work, we present **real-time chunking (RTC)**, which poses asynchronous action chunking as an inpainting problem. Our algorithm generates the next action chunk while executing the previous one, freezing the actions that are guaranteed to be executed (due to inference delay) and "inpainting" the rest. It is applicable to any diffusion- [21] or flow-based [35] VLA, and operates purely at inference time, requiring no changes to existing training recipes.

Our contributions are as follows. First, we present a novel system for asynchronous, real-time inference of action chunking diffusion- or flow-based policies for continuous control. Since standard simulation benchmarks are quasi-static—and have mostly been saturated with pseudo open-loop inference strategies [11]—we devise a new benchmark based on the Kinetix simulator [42] consisting of 12 highly dynamic manipulation and locomotion tasks. In the real world, we evaluate RTC on 6 challenging bimanual manipulation tasks using the $\pi_{0.5}$ VLA [23] as the base policy. Across both simulation and the real world, we demonstrate that RTC is fast and performant; it is uniquely robust to inference latency, even in highly precise tasks such as lighting a match (Figure 1), and it achieves greatly improved task throughput on all real tasks.

2 Related Work

Action chunking and VLAs. Inspired in part by human motor control [32], action chunking has recently emerged as the de facto standard in imitation learning for visuomotor control [67, 11]. Learning to generate action chunks from human data requires expressive statistical models, such as variational inference [67, 18], diffusion [11, 12, 68, 67, 45, 58], flow matching [5, 6], vector

quantization [33, 3, 43], or byte-pair encoding [46]. Recently, some of these methods have been scaled to billions of parameters, giving rise to VLAs [7, 13, 29, 5, 70, 10, 9, 69, 23, 46, 36], a class of large models built on pre-trained vision-language model backbones. With the capacity to fit ever-growing robot datasets [13, 28, 61, 14, 40, 26], as well as Internet knowledge from vision-language pre-training, VLAs have achieved impressive results in generalizable robot manipulation.

Reducing inference latency. A natural approach to improve the real-time capabilities of a model is to simply speed it up. For instance, consistency policy [48] distills diffusion policies to elide expensive iterative denoising. Streaming diffusion policy [22] proposes an alternative training recipe that allows for very few denoising steps per controller timestep. Kim et al. [30] augment OpenVLA [29] with parallel decoding to elide expensive autoregressive decoding. More broadly, there is a rich literature on optimizing inference speed, both for diffusion models [51, 37, 55, 16] and large transformers in general [31, 24, 34]. Unfortunately, these directions cannot reduce inference cost below one forward pass. So long as this forward pass takes longer than the controller's sampling period, other methods will be needed for real-time execution.

Inpainting and guidance. There is a rich literature on image inpainting with pre-trained diffusion and flow models [47, 54, 39, 41]. In our work, we incorporate one such method [47] into our novel real-time execution framework with modifications (namely, soft masking and guidance weight clipping) that we find necessary for our setting. For sequential decision-making, Diffuser [25] pioneered diffusion-based inpainting for following state and action constraints in long-term planning, though their inpainting method is not guidance-based. Diffuser and other work [63, 1] have also guided diffusion models with value functions to solve reinforcement learning (RL) problems. Our work is distinct in that it is the first to apply either inpainting or guidance to real-time control.

Real-time execution. Real-time control has been studied long before the advent of VLAs. Similar to action chunking, model predictive control (MPC; [50]) generates plans over a receding time horizon; like our method, it parallelizes execution and computation, and uses the prior chunk to warm-start planning for the next. Though recent works combining learning methods with MPC have demonstrated real-time control capabilities in narrow domains [52, 20], their reliance on explicit dynamic models and cost functions makes their application difficult in unstructured settings, where VLAs have gained prominence. Separately, in reinforcement learning, a variety of prior works have developed time-delayed decision-making methods [56, 15, 53, 62, 65, 66]. However, these approaches are not always applicable to imitation learning, and none of them leverage action chunking. Most recently, hierarchical VLA designs [57, 4] have emerged where the model is split into a System 2 (high-level planning) and System 1 (low-level action generation) component. The System 2 component contains the bulk of the VLA's capacity and runs at a low frequency, while the System 1 component is lightweight and fast. This approach is orthogonal to ours, and comes with its own tradeoffs (e.g., limiting the size of the System 1 component and requiring its own training recipe).

Bidirectional Decoding. The most closely related prior work is Bidirectional Decoding (BID; [38]), which enables fully closed-loop control with pre-trained action chunking policies via rejection sampling. While Liu et al. [38] do not consider inference delay, the BID algorithm can be used to accomplish the same effect as our guidance-based inpainting. We compare to BID in our simulated benchmark, finding that it underperforms RTC while using significantly more compute.

3 Preliminaries and Motivation

We begin with an action chunking policy denoted by $\pi(\mathbf{A}_t|\mathbf{o}_t)$, where $\mathbf{A}_t = [\mathbf{a}_t, \mathbf{a}_{t+1}, ..., \mathbf{a}_{t+H-1}]$ is a chunk of future actions, $\mathbf{o}_t$ is an observation, and t indicates a controller timestep. We call H the *prediction horizon*. When action chunking policies are rolled out, the first $s \leq H$ actions from each chunk are executed, and then a new chunk is produced by the policy. We call s the *execution horizon*; often it is shorter than the prediction horizon, but still much greater than 1 (e.g., $s \approx H/2$ [11, 5, 23]). Chunked execution ensures temporal consistency at the expense of reactivity. A long execution horizon reduces a policy's responsiveness to new information, while a short one increases the likelihood of mode-jumping, jerky behavior resulting from discontinuities between chunks.

In this paper, we consider policies trained with conditional flow matching [35], though our method can also be used with diffusion policies by converting them to flow policies at inference time [47, 17]. To generate an action chunk from a flow policy, random noise $\mathbf{A}_t^0$ is first sampled from a standard Gaussian, and then the flow's velocity field, $\mathbf{v}_\pi$ (a learned neural network) is integrated from $\tau = 0$

to 1 using the update rule

$$\mathbf{A}_t^{\tau+\frac{1}{n}} = \mathbf{A}_t^{\tau} + \frac{1}{n}\mathbf{v}_\pi(\mathbf{A}_t^{\tau}, \mathbf{o}_t, \tau), \tag{1}$$

where $\tau \in [0, 1)$ denotes a flow matching timestep, and n determines the number of denoising steps.

Now, let Δt be sampling period of a controller, i.e., the duration of a controller timestep, and let δ be the time it takes for the policy to generate an action chunk. We define $d := \lfloor \delta / \Delta t \rfloor$ and call this quantity the *inference delay*, corresponding to number of controller timesteps between when $\mathbf{o}_t$ is received and when $\mathbf{A}_t$ is available.[1] If $d = 0$, then inference can be performed between two timesteps without any interruption. However, this is near impossible to achieve with modern VLAs. For example, with an NVIDIA RTX 4090 GPU, the 3 billion parameter π_0 VLA spends 46ms on the KV cache prefill alone, before any denoising steps [5], and targets a 50Hz control frequency ($\Delta t = 20$ms). Run in remote inference for mobile manipulation, π_0 lists 13ms of network latency, in perfect conditions with a wired connection. In a more realistic setting, the network overhead alone could easily exceed 20ms. Kim et al. [30], who optimize the 7B OpenVLA model [29] specifically for inference speed, achieve no better than 321ms of latency on an NVIDIA A100 GPU.

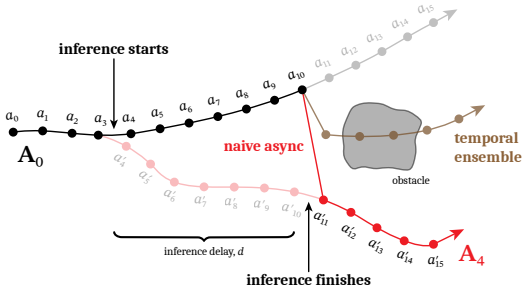

Figure 2: An illustration of a typical bifurcation between consecutive chunks. Inference is started between timesteps 3 and 4. The original chunk that was executing, $\{a_t\}$ (black), had planned to go above the obstacle while the newly generated chunk $\{a'_t\}$ (red) goes below the obstacle. However, $\{a'_t\}$ is not available until $d = 7$ steps later. A naive asynchronous algorithm might jump from a_{10} to a'_{11}, inducing a very high, out-of-distribution acceleration. Temporal ensembling [67], i.e., interpolating between chunks, reduces the acceleration but produces poor actions.

When $d > 0$, chunked execution becomes more complex. Naive synchronous inference, the default in many prior works [5, 29, 8, 23, 30, 58], introduces visible pauses between chunks that not only slow down execution but also change the dynamics of the robot, introducing distribution shift between training and evaluation. The first requirement of a real-time system is *asynchronous* execution, where inference is started early so that an action is guaranteed to be available at every timestep.

Let $\mathbf{a}_{t'|t}$ denote the $(t' - t)$-th action of chunk $\mathbf{A}_t$, generated from observing $\mathbf{o}_t$. If $\mathbf{A}_0$ is currently executing, and we wish to switch chunks at step j, then an asynchronous algorithm must start inference at $j - d$. However, since the policy cannot know what will happen between steps $j - d$ and j while generating $\mathbf{A}_{j-d}$, the transition point between $\mathbf{a}_{j-1|0}$ and $\mathbf{a}_{j|j-d}$ may be arbitrarily discontinuous and out-of-distribution. Similar to a too-short execution horizon, this strategy leads to jerky behavior that is worsened dramatically with higher delays; see Figure 2.

4 Real-Time Chunking via Inpainting

The key challenge in real-time execution is to maintain continuity between chunks. By the time a new chunk is available, the previous one has already been executed partway, and therefore the new chunk must be "compatible" with the previous one. At the same time, the new chunk should still incorporate new observations, so that the policy does not lose the ability to react and make corrections.

Our key insight is to pose real-time chunking as an inpainting problem. To make the new chunk "compatible", we must use the overlapping timesteps where we have access to the remaining actions of the previous chunk. The first d actions from the new chunk cannot be used, since those timesteps will have already passed by the time the new chunk becomes available. Thus, it makes sense to "freeze" those actions to the values that we know *will* be executed; our goal is then to fill in the remainder of the new chunk in a way that is consistent with this frozen prefix (see Figure 3), much like inpainting a section of an image that has been removed. We describe this basic inpainting principle in Sec. 4.1. In Sec. 4.2, we introduce a *soft masking* extension that is critical for full cross-chunk continuity; finally, we describe our full real-time chunking system in Sec. 4.3.

[1] For simplicity, we do not consider delays or synchronization issues at the sub-timestep level; we assume that the low-level controller provides $\mathbf{o}_t$ at the same instant that $\mathbf{a}_{t-1}$ is consumed.

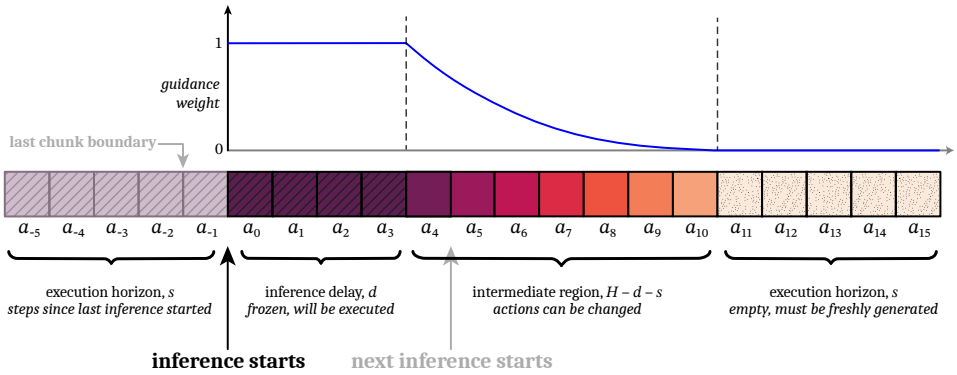

Figure 3: A diagram illustrating how action generation attends to the previous action chunk in real-time chunking. If inference starts after the execution of a_{-1} and the inference delay is $d = 4$, then the newly generated chunk will not be available until after a_3 is consumed. Therefore, $a_{0:3}$ are "frozen" and are attended to with a full guidance weight of 1. In the intermediate region, $a_{4:10}$, actions from the previous chunk are available but may be updated, since inference will have finished before a_4 is needed. This region is attended to with an exponentially decreasing guidance weight. Finally, the last $s = 5$ actions are beyond the end of the previous chunk, and need to be freshly generated. The execution horizon, s, is a hyperparameter constrained by $d \leq s \leq H - d$.

4.1 Inference-Time Inpainting with Flow Matching

Inpainting is a known strength of iterative denoising frameworks such as diffusion and flow matching. We build on the training-free image inpainting algorithm from Pokle et al. [47], which is itself based on pseudoinverse guidance (ΠGDM; [54]). The algorithm operates by adding a gradient-based guidance term to the learned velocity field $\mathbf{v}$ at each denoising step (Equation 1) that encourages the final generation to match some target value, $\mathbf{Y}$, which is a corrupted version of the desired result. In the case of image inpainting, the corruption operator is masking, $\mathbf{Y}$ is the masked image, and the desired result is a full image consistent with $\mathbf{Y}$ in the non-masked areas. The ΠGDM gradient correction, specialized to our setting, is given by

$$\mathbf{v}_{\Pi\text{GDM}}(\mathbf{A}_t^\tau, \mathbf{o}_t, \tau) = \mathbf{v}(\mathbf{A}_t^\tau, \mathbf{o}_t, \tau) + \min\left(\beta, \frac{1-\tau}{\tau \cdot r_\tau^2}\right)\left(\mathbf{Y} - \widehat{\mathbf{A}_t^1}\right)^\top \text{diag}(\mathbf{W}) \frac{\partial \widehat{\mathbf{A}_t^1}}{\partial \mathbf{A}_t^\tau} \tag{2}$$

$$\text{where } \widehat{\mathbf{A}_t^1} = \mathbf{A}_t^\tau + (1-\tau)\mathbf{v}(\mathbf{A}_t^\tau, \mathbf{o}_t, \tau), \tag{3}$$

$$r_\tau^2 = \frac{(1-\tau)^2}{\tau^2 + (1-\tau)^2}. \tag{4}$$

$\widehat{\mathbf{A}_t^1}$ is an estimate of the final, fully denoised action chunk and $\mathbf{W}$ is the mask. We are abusing notation by treating $\mathbf{Y}$, $\mathbf{A}_t$, and $\mathbf{W}$ as vectors of dimension HM where M is the dimension of each action. Thus, the guidance term is a vector-Jacobian product and can be computed using backpropagation. The guidance weight clipping, β, is our addition; we found that without it, the algorithm became unstable with the small number of denoising steps commonly used in control problems (see A.2 for ablation).

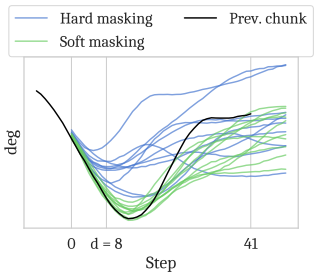

Figure 4: A comparison of naive inpainting (hard masking) and our proposed soft masking method: note that hard masking does not match the frozen region very well and produces faster changes in direction.

4.2 Soft Masking for Improved Cross-Chunk Continuity

In practice, naively inpainting using only the first d timesteps of the previous action chunk is often insufficient to ensure that the new chunk takes a consistent strategy, particularly when d is small (e.g., see Figure 4). The ΠGDM correction is not perfect, and a small d leads to a weak guidance signal, which can allow for the new chunk to still switch strategies and cause discontinuities. Our solution, illustrated in Figure 3, is to give our policy more cross-chunk continuity by considering not just the first d overlapping actions, but all $H - s$ overlapping actions. We do this via *soft masking*, setting $\mathbf{W}$ to real-valued weights rather than 1s and 0s. The first d actions get a weight of 1; the last s actions of the new chunk do not

overlap with the previous chunk, so they get a weight of 0; the actions in between get weights that exponentially decay from 1 to 0, accounting for the fact that actions further in the future should be treated with more uncertainty. The resulting expression for $\mathbf{W}$ is given by

$$\mathbf{W}_i = \begin{cases} 1 & \text{if } i < d \\ c_i \frac{e^{c_i} - 1}{e - 1} & \text{if } d \leq i < H - s \\ 0 & \text{if } i \geq H - s \end{cases} \quad \text{where } \; c_i = \frac{H - s - i}{H - s - d + 1}, \;\; i \in \{0, \dots, H - 1\}. \quad (5)$$

Intuitively, $\mathbf{W}$ modulates the "attention" paid to each corresponding action from the previous chunk.

4.3 Real-Time Chunking

We present our full real-time chunking system in Algorithm 1 (complemented by Figure 3). The controller interfaces with our algorithm via GETACTION, which is called every Δt to consume an action $\mathbf{a}_{t-1}$ and provide the next observation $\mathbf{o}_t$. The INFERENCELOOP runs in a background thread so that an action is always available. It forecasts the next delay, d, by keeping a buffer of past delays. The execution horizon, s, can change from chunk to chunk; the user provides a minimum desired horizon, s_{min}, and the actual horizon for a given chunk is $\max(d, s_{\text{min}})$ where d is the delay encountered when computing the *next* chunk. Finally, the algorithm describes the inpainting with soft masking procedure in GUIDEDINFERENCE, which explicitly defines a denoising function (Eq. 3) and computes a vector-Jacobian product, which can be done with reverse-mode autodifferentiation [2].

Algorithm 1 Real-Time Chunking

Require: flow policy π with prediction horizon H, minimum execution horizon s_{min}, mutex $\mathcal{M}$, condition variable $\mathcal{C}$ associated with $\mathcal{M}$, initial chunk $\mathbf{A}_{\text{init}}$, initial delay estimate d_{init}, delay buffer size b, number of denoising steps n, maximum guidance weight β

procedure INITIALIZESHAREDSTATE ▷ Initialize mutex-protected shared variables
  $t = 0$; $\mathbf{A}_{\text{cur}} = \mathbf{A}_{\text{init}}$, $\mathbf{o}_{\text{cur}} = \text{null}$

function GETACTION($\mathbf{o}_{\text{next}}$) ▷ Called at an interval of Δt by controller
  with $\mathcal{M}$ acquired **do**
    $t = t + 1$
    $\mathbf{o}_{\text{cur}} = \mathbf{o}_{\text{next}}$
    notify $\mathcal{C}$
    return $\mathbf{A}_{\text{cur}}[t - 1]$

procedure INFERENCELOOP ▷ Run inference in a looping background thread
  acquire $\mathcal{M}$
  $\mathcal{Q} =$ new Queue([d_{init}], maxlen=b) ▷ Holds a limited buffer of past inference delays
  loop
    wait on $\mathcal{C}$ until $t \geq s_{\text{min}}$
    $s = t$ ▷ s is the number of actions executed since last inference started
    $\mathbf{A}_{\text{prev}} = \mathbf{A}_{\text{cur}}[s, s + 1, \dots, H - 1]$ ▷ Remove the s actions that have already been executed
    $\mathbf{o} = \mathbf{o}_{\text{cur}}$
    $d = \max(\mathcal{Q})$ ▷ Estimate the next inference delay conservatively
    with $\mathcal{M}$ released **do**
      $\mathbf{A}_{\text{new}} = \text{GUIDEDINFERENCE}(\pi, \mathbf{o}, \mathbf{A}_{\text{prev}}, d, s)$
    $\mathbf{A}_{\text{cur}} = \mathbf{A}_{\text{new}}$ ▷ Swap to the new chunk as soon as it is available
    $t = t - s$ ▷ Reset t so that it indexes into $\mathbf{A}_{\text{new}}$
    enqueue t onto $\mathcal{Q}$ ▷ Record the observed delay

function GUIDEDINFERENCE($\pi, \mathbf{o}, \mathbf{A}_{\text{prev}}, d, s$)
  compute $\mathbf{W}$ using Eq. 5; right-pad $\mathbf{A}_{\text{prev}}$ to length H; initialize $\mathbf{A}^0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  for $\tau = 0$ to 1 with step size $1/n$ **do**
    $f_{\widehat{\mathbf{A}^1}} = \mathbf{A}' \mapsto \mathbf{A}' + (1 - \tau)\mathbf{v}_\pi(\mathbf{A}', \mathbf{o}, \tau)$ ▷ Define denoising function (Eq. 3)
    $\mathbf{e} = \left(\mathbf{A}_{\text{prev}} - f_{\widehat{\mathbf{A}^1}}(\mathbf{A}^\tau)\right)^\top \text{diag}(\mathbf{W})$ ▷ Weighted error term from Eq. 2
    $\mathbf{g} = \mathbf{e} \cdot \left.\frac{\partial f_{\widehat{\mathbf{A}^1}}}{\partial \mathbf{A}'}\right|_{\mathbf{A}' = \mathbf{A}^\tau}$ ▷ Compute vector-Jacobian product from Eq. 2 via autodiff
    $\mathbf{A}^{\tau + \frac{1}{n}} = \mathbf{A}^\tau + \frac{1}{n}\left(\mathbf{v}_\pi(\mathbf{A}^\tau, \mathbf{o}, \tau) + \min\left(\beta, \frac{1-\tau}{\tau \cdot r_\tau^2}\right)\mathbf{g}\right)$ ▷ Integration step (Eq. 1)
  return $\mathbf{A}^1$

5 Experiments

In our experiments, we aim to answer the following questions. First, how does RTC compare to existing methods in highly dynamic and stochastic environments, and under increasing inference delays? Second, how important is soft masking (Sec. 4.2) to RTC? Third, how does RTC affect the performance *and* speed of real-world dexterous robots?

We first evaluate RTC using a benchmark of 12 highly dynamic and stochastic environments in the Kinetix [42] simulator. We use this benchmark to compare the performance of RTC to other methods under simulated inference delays, as well as investigate the effect of soft masking. Then, using the $\pi_{0.5}$ VLA [23] as the base model, we evaluate the performance and speed of RTC on 6 challenging bimanual dexterous manipulation tasks, including 2 mobile manipulation tasks (see Figure 6).

5.1 Simulated Benchmark

Most simulated imitation learning benchmarks are quasi-static, and standard chunked execution with a long enough execution horizon can achieve near-perfect success rates [11]. We instead create a benchmark of 12 dynamic tasks in Kinetix [42], which uses force-based control, so inference delay *necessitates* asynchronous execution (there is no concept of "holding position"). We select 10 existing environments and create 2 new ones such that all environments involve dynamic motions like throwing, catching, and balancing. To simulate imperfect actuation, we add Gaussian noise to the actions, making closed-loop corrections crucial for success.

Setup. To generate data for imitation learning, we first train expert policies using RPO [49] and a binary success reward. For each environment, we train 6 expert policies with different seeds and then generate a 1M transition dataset with a different policy selected each episode. We then train action chunking flow policies with a prediction horizon of $H = 8$ and a 4-layer MLP-Mixer [60] architecture for 32 epochs. We report binary success rates with 2048 rollouts per data point, and simulate delays between 0 (fully closed-loop) and 4 (the maximum supported when $H = 8$).

Baselines. We compare against the following baselines:

- *Naive async.* This strategy does not pay attention to the previous action chunk at all when generating a new one, naively switching chunks as soon as the new one is ready.
- *Bidirectional decoding (BID; [38]).* This strategy uses rejection sampling to keep continuity across chunks. We use a batch size of $N = 32$, mode size of $K = 3$, and a checkpoint trained for 8 epochs as the weak policy.
- *Temporal ensembling (TE; [67]).* This strategy involves keeping a buffer of predicted action chunks and executing an average of all actions predicted for a particular timestep.

Results. Figure 5 shows the simulated results. In the delay plots (right): TE performs poorly across the board, even with an inference delay of $d = 0$, illustrating the multi-modality of our benchmark—averages of valid actions are not necessarily valid. RTC shows the most robustness to inference delays, outperforming BID, and the gap widens with increasing delay; note that BID uses significantly more compute than RTC by sampling batches of 64 action chunks, 32 from a strong model and 32 from a weak model. Additionally, we find that hard masking somewhat underperforms soft masking, particularly when d is smaller, supporting our claims in Sec. 4.2. Finally, in the execution horizon plot (left), we find that thanks to its continuity across chunks, RTC is better able to take advantage of closed-loop corrections, always performing better with a decreasing execution horizon.

5.2 Real-World Results

Next, we deploy our full real-time chunking system to the real world. We use the $\pi_{0.5}$VLA [23] as our base policy, and evaluate RTC on a bimanual system with two 6-DoF arms and parallel jaw grippers. Unlike our simulated benchmark, the robots use position control, and so synchronous inference—stopping between chunks—is a reasonable default strategy, used in many prior works [5, 23, 30, 46]. Our goal is to improve upon synchronous inference in a combination of both performance *and* speed.

Setup. We use $\pi_{0.5}$ ($H = 50$, $\Delta t = 20$ms) with $n = 5$ denoising steps, giving a model latency of 76ms for the baselines and 97ms for RTC. We use remote inference over LAN, which adds 10-20ms of latency, giving a starting inference delay around $d \approx 6$ for RTC. However, we would like to understand how the system behaves with higher inference latencies, simulating, e.g., scaling up the

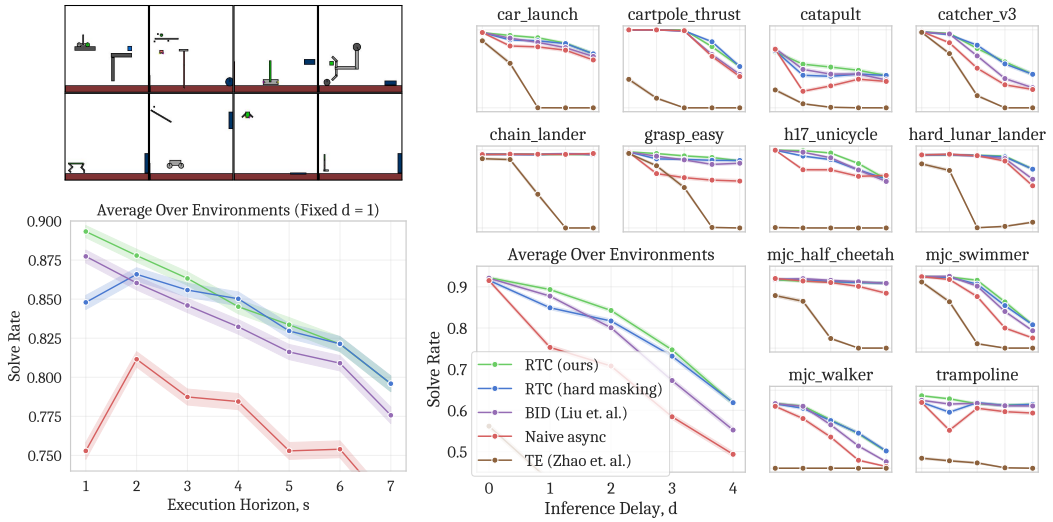

Figure 5: **Top left:** Kinetix environments; each involves getting a green object on the left to touch a blue one on the right. **Bottom left:** Execution horizon vs. solve rate with a fixed inference delay of 1. Only RTC and BID take full advantage of faster updates, showing strictly increasing performance with decreasing execution horizon. **Right:** Inference delay vs. solve rate with a fixed execution horizon of $s = \max(d, 1)$. RTC outperforms all baselines. Furthermore, soft masking (Sec. 4.2) improves performance at lower inference delays and execution horizons. Each data point represents 2048 trials, and 95% Wilson score intervals are shaded in.

model size or running inference on a distant cloud server. Thus, we also evaluate all methods with +100ms and +200ms of injected latency, corresponding to $d \approx 11$ and $d \approx 16$, respectively.

Tasks and scoring. Each episode gets an integer score corresponding to how many substeps of the task it completed successfully. We evaluate the following tasks:

- *Light candle (5 steps, 40s cutoff).* Pick up a match and matchbox, strike the match, use it to light a candle, and drop it in a bowl.
- *Plug ethernet (6 steps, 120s cutoff).* Pick up the end of an ethernet cable, reorient it, plug it into a server rack, and repeat the process for the other end.
- *Make bed, mobile (3 steps, 200s cutoff).* Move the corner of a blanket and 2 pillows from the foot to the head of a bed.
- *Shirt folding (1 step, 300s cutoff).* Fold a shirt from a flattened position.
- *Batch folding (4 steps, 300s cutoff).* Take a varied, crumpled clothing item out of a bin, flatten it, fold it, then place it neatly on a pile.
- *Dishes in sink, mobile (8 steps, 300s cutoff).* Move 4 varied items from a counter into a sink.

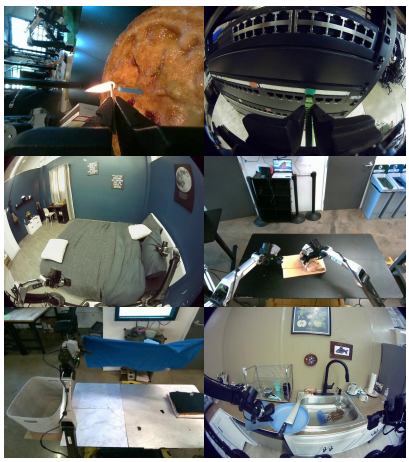

Figure 6: The six real-world tasks, including 2 mobile manipulation tasks. See the blog post for videos.

See Figure 6 for images of tasks, and the blog post for videos. We evaluate each task and method for 10 trials for a total of 480 episodes, adding up to 28 hours of pure robot execution time. We also post-hoc annotate the score for each episode and the timestamp at which each step is achieved.

Baselines. We compare to the following baselines:

- *Synchronous.* This corresponds to the default inference strategy in prior work [5, 23, 30, 46], which executes $s = 25$ actions and then pauses while the new chunk is generated.
- *TE, sparse.* This is similar to *naive async* in our simulated results; it executes $s = 25$ actions at a time while computing the next chunk in parallel. We found it significantly reduced jerkiness to also apply TE, even though only the first $H - s - 2d$ executed steps of each chunk have overlapping actions to ensemble.

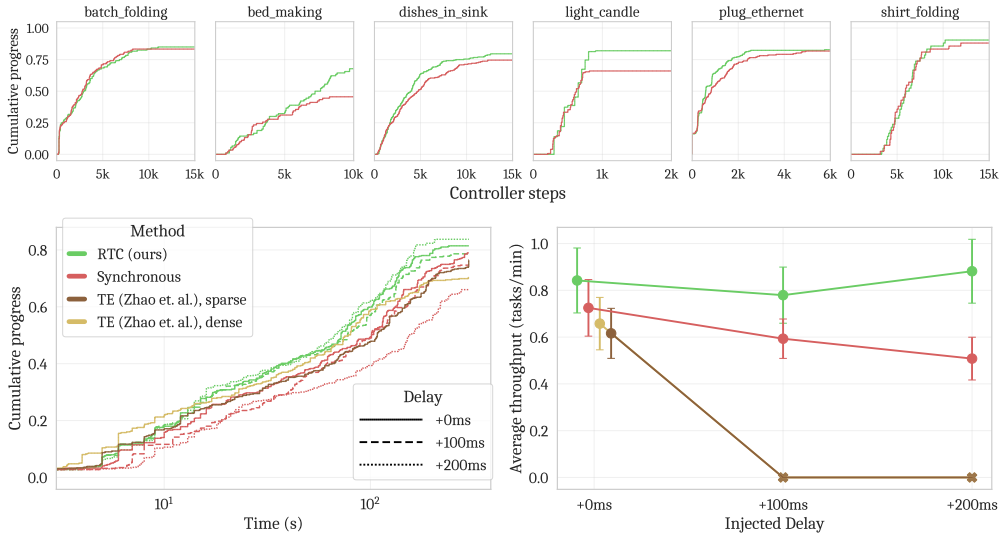

Figure 7: **Top:** Controller steps (equivalent to elapsed time with inference pauses removed multiplied by 50Hz) vs. cumulative progress for each task, aggregated across all delays. Progress is measured in discrete steps corresponding to the subsections of each task. **Left:** Time (including inference pauses) vs. cumulative progress aggregated across all tasks. The x-axis is log scale to better show progress during both short and long-horizon tasks. **Right:** Inference delay vs. average throughput, defined as the proportion of task completed divided by duration of episode averaged over episodes. Error bars are ± 1 SEM. Average throughput gives a balanced view of both speed and performance for each method. Neither TE variant can run at +100 or +200ms of injected latency, causing such high oscillations that the robot's protective stop is triggered.

- *TE, dense.* This strategy is the closest to the original TE in Zhao et al. [67]. We run inference as often as possible by setting $s_{\min} = 1$ in Algorithm 1, meaning $s = d$ for every chunk. This results in there always being at least 2 overlapping action chunks to ensemble, and often more.

We do not compare to BID [38] in the real world, as we found in simulation that it underperforms RTC while using significantly more compute—when applied to $\pi_{0.5}$with a batch size of 16, BID has 2.3 times the latency of our method (see A.3 for latency measurements).

Results. We present the results in Figure 7. In average task throughput, a measurement of both speed and performance, RTC achieves the best score at all inference delays with a statistically significant result at +100 and +200ms. RTC is completely robust to injected delay, showing no degradation, whereas synchronous degrades linearly and both TE variants do not run at all due to causing such high oscillations that the robot's protective stop is triggered (see videos). Inspecting the per-task results (Figure 5, top), we can conclude that RTC helps with more than just execution speed: it completes tasks faster than synchronous inference *even when inference pauses are removed*. All tasks, except for *light candle*, allow for retrying until the time limit (and $\pi_{0.5}$ does, in general, exhibit robust retrying behavior). Even though synchronous inference often reaches a similar final score, RTC often completes more of the task earlier in the episode, reflecting fewer mistakes and less retrying. In *light candle*, the most precision-sensitive task—and also the only one without retrying—RTC shows a large advantage in final score, reflecting a higher overall success rate. Interestingly, the same is true in *bed making*, even though that task does elicit retrying. The policy particularly struggles to manipulate the pillows, and *bed making* is the hardest task overall, which may be why RTC has a strong effect.

6 Discussion and Future Work

Real-time chunking is an inference-time algorithm for asynchronous execution of action chunking policies that demonstrates speed and performance across simulation and real-world experiments, including under significant inference delays. However, this work is not without limitations: it adds some computational overhead compared to methods that sample directly from the base policy. Additionally, while our real-world experiments cover a variety of challenging manipulation tasks, there are more dynamic settings that could benefit even more from real-time execution. One example is legged locomotion, which is represented in our simulated benchmark but not our real-world results.

7 Acknowledgements

We thank Charles Xu and Kay Ke for designing the Ethernet plug-in task. We thank Brian Ichter for suggesting the cumulative progress plots and for later feedback on figures. We thank Dibya Ghosh for suggesting the throughput metric to measure a combination of speed and performance. We thank Ury Zhilinsky, Karan Dhabalia, Haohuan Wang, and Dibya Ghosh for help with training infrastructure; Noah Brown, Szymon Jakubczak, Adnan Esmail, Tim Jones, Mohith Mothukuri, James Darpinian, and James Tanner for robot infrastructure; Adrian Li-Bell for evaluation infrastructure; Anna Walling, Chelsea Finn, and Karol Hausman for robot, data and evaluation operations; and Michael Equi, Quan Vuong, and Jost Tobias Springenberg for training some of the $\pi_{0.5}$ policies used in the real-world experiments. We also thank Claudio Guglieri and Alex Krasikov for their help with visualizations for the blog post, and Jessica Dai for helpful copy editing of the paper manuscript. Finally, we are grateful to the whole team of robot operators at Physical Intelligence for their enormous contributions to running data collection and policy evaluations.

References

[1] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making? *arXiv preprint arXiv:2211.15657*, 2022.

[2] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18(153):1–43, 2018.

[3] Suneel Belkhale and Dorsa Sadigh. Minivla: A better vla with a smaller footprint, 2024. URL `https://github.com/Stanford-ILIAD/openvla-mini`.

[4] Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

[5] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[6] Max Braun, Noémie Jaquier, Leonel Rozo, and Tamim Asfour. Riemannian flow matching policy for robot motion learning. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5144–5151. IEEE, 2024.

[7] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alex Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *arXiv preprint arXiv:2307.15818*, 2023.

[8] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

[9] Chi-Lam Cheang, Guangzeng Chen, Ya Jing, Tao Kong, Hang Li, Yifeng Li, Yuxiao Liu, Hongtao Wu, Jiafeng Xu, Yichu Yang, Hanbo Zhang, and Minzhao Zhu. Gr-2: A generative video-language-action model with web-scale knowledge for robot manipulation. *arXiv preprint arXiv:2410.06158*, 2024.

[10] An-Chieh Cheng, Yandong Ji, Zhaojing Yang, Xueyan Zou, Jan Kautz, Erdem Biyik, Hongxu Yin, Sifei Liu, and Xiaolong Wang. NaVILA: Legged Robot Vision-Language-Action Model for Navigation. *arXiv preprint arXiv:2412.04453*, 2024.

[11] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.

[12] 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. *arXiv preprint arXiv:2402.10329*, 2024.

[13] OX-Embodiment Collaboration, A Padalkar, A Pooley, A Jain, A Bewley, A Herzog, A Irpan, A Khazatsky, A Rai, A Singh, et al. Open X-Embodiment: Robotic learning datasets and RT-X models. *arXiv preprint arXiv:2310.08864*, 1(2), 2023.

[14] Hao-Shu Fang, Hongjie Fang, Zhenyu Tang, Jirong Liu, Chenxi Wang, Junbo Wang, Haoyi Zhu, and Cewu Lu. Rh20t: A comprehensive robotic dataset for learning diverse skills in one-shot. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 653–660. IEEE, 2024.

[15] Vlad Firoiu, Tina Ju, and Josh Tenenbaum. At human speed: Deep reinforcement learning with action delay. *arXiv preprint arXiv:1810.07286*, 2018.

[16] Kevin Frans, Danijar Hafner, Sergey Levine, and Pieter Abbeel. One step diffusion via shortcut models. *arXiv preprint arXiv:2410.12557*, 2024.

[17] Ruiqi Gao, Emiel Hoogeboom, Jonathan Heek, Valentin De Bortoli, Kevin P. Murphy, and Tim Salimans. Diffusion meets flow matching: Two sides of the same coin. 2024. URL `https://diffusionflow.github.io/`.

[18] Abraham George and Amir Barati Farimani. One act play: Single demonstration behavior cloning with action chunking transformers. *arXiv preprint arXiv:2309.10175*, 2023.

[19] Anthony Ha. Google's gemini has beaten pokémon blue (with a little help). `https://techcrunch.com/2025/05/03/googles-gemini-has-beaten-pokemon-blue-with-a-little-help/`, May 2025. Accessed May 8, 2025.

[20] Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control, 2022. URL `https://arxiv.org/abs/2203.04955`.

[21] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[22] Sigmund H Høeg, Yilun Du, and Olav Egeland. Streaming diffusion policy: Fast policy synthesis with variable noise diffusion models. *arXiv preprint arXiv:2406.04806*, 2024.

[23] Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

[24] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[25] Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

[26] Zhenyu Jiang, Yuqi Xie, Kevin Lin, Zhenjia Xu, Weikang Wan, Ajay Mandlekar, Linxi Fan, and Yuke Zhu. Dexmimicgen: Automated data generation for bimanual dexterous manipulation via imitation learning. *arXiv preprint arXiv:2410.24185*, 2024.

[27] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[28] Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, Peter David Fagan, Joey Hejna, Masha Itkina, Marion Lepert, Yecheng Jason Ma, Patrick Tree Miller, Jimmy Wu, Suneel Belkhale, Shivin Dass, Huy Ha, Arhan Jain, Abraham Lee, Youngwoon Lee, Marius Memmel, Sungjae Park, Ilija Radosavovic, Kaiyuan Wang, Albert Zhan, Kevin Black, Cheng Chi, Kyle Beltran Hatch, Shan Lin, Jingpei Lu, Jean Mercat, Abdul Rehman, Pannag R Sanketi, Archit Sharma, Cody Simpson, Quan Vuong, Homer Rich Walke, Blake Wulfe, Ted Xiao, Jonathan Heewon Yang, Arefeh Yavary, Tony Z. Zhao, Christopher Agia, Rohan Baijal, Mateo Guaman Castro, Daphne Chen, Qiuyu Chen, Trinity Chung, Jaimyn Drake, Ethan Paul Foster, Jensen Gao, David Antonio Herrera, Minho Heo, Kyle Hsu, Jiaheng Hu, Donovon Jackson, Charlotte Le, Yunshuang Li, Kevin Lin, Roy Lin, Zehan Ma, Abhiram Maddukuri, Suvir Mirchandani, Daniel Morton, Tony Nguyen, Abigail O'Neill, Rosario Scalise, Derick Seale, Victor Son, Stephen Tian, Emi Tran, Andrew E. Wang, Yilin Wu, Annie Xie, Jingyun Yang, Patrick Yin, Yunchu Zhang, Osbert Bastani, Glen Berseth, Jeannette Bohg, Ken Goldberg, Abhinav Gupta, Abhishek Gupta, Dinesh Jayaraman, Joseph J Lim, Jitendra Malik, Roberto Martín-Martín, Subramanian Ramamoorthy, Dorsa Sadigh, Shuran Song, Jiajun Wu, Michael C. Yip, Yuke Zhu, Thomas Kollar, Sergey Levine, and Chelsea Finn. Droid: A large-scale in-the-wild robot manipulation dataset. In *Proceedings of Robotics: Science and Systems*, 2024.

[29] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[30] Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

[31] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[32] Lucy Lai, Ann Zixiang Huang, and Samuel J Gershman. Action chunking as policy compression. 2022.

[33] Seungjae Lee, Yibin Wang, Haritheja Etukuru, H Jin Kim, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Behavior generation with latent actions. *arXiv preprint arXiv:2403.03181*, 2024.

[34] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100, 2024.

[35] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

[36] Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation. *arXiv preprint arXiv:2410.07864*, 2024.

[37] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

[38] Yuejiang Liu, Jubayer Ibn Hamid, Annie Xie, Yoonho Lee, Maximilian Du, and Chelsea Finn. Bidirectional decoding: Improving action chunking via closed-loop resampling. *arXiv preprint arXiv:2408.17355*, 2024.

[39] Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11461–11471, 2022.

[40] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung, Julian Gao, John Emmons, Anchit Gupta, Emre Orbay, et al. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning*, pages 879–893. PMLR, 2018.

[41] Morteza Mardani, Jiaming Song, Jan Kautz, and Arash Vahdat. A variational perspective on solving inverse problems with diffusion models. *arXiv preprint arXiv:2305.04391*, 2023.

[42] Michael Matthews, Michael Beukman, Chris Lu, and Jakob Foerster. Kinetix: Investigating the training of general agents through open-ended physics-based control tasks. *arXiv preprint arXiv:2410.23208*, 2024.

[43] Atharva Mete, Haotian Xue, Albert Wilcox, Yongxin Chen, and Animesh Garg. Quest: Self-supervised skill abstractions for learning continuous control, 2024. URL `https://arxiv.org/abs/2407.15840`.

[44] OpenAI. Introducing openai codex, August 2021. URL `https://openai.com/index/introducing-codex/`. Accessed on May 27, 2025.

[45] Tim Pearce, Tabish Rashid, Anssi Kanervisto, Dave Bignell, Mingfei Sun, Raluca Georgescu, Sergio Valcarcel Macua, Shan Zheng Tan, Ida Momennejad, Katja Hofmann, et al. Imitating human behaviour with diffusion models. *arXiv preprint arXiv:2301.10677*, 2023.

[46] Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

[47] Ashwini Pokle, Matthew J Muckley, Ricky TQ Chen, and Brian Karrer. Training-free linear image inverses via flows. *arXiv preprint arXiv:2310.04432*, 2023.

[48] Aaditya Prasad, Kevin Lin, Jimmy Wu, Linqi Zhou, and Jeannette Bohg. Consistency policy: Accelerated visuomotor policies via consistency distillation. *arXiv preprint arXiv:2405.07503*, 2024.

[49] Md Masudur Rahman and Yexiang Xue. Robust policy optimization in deep reinforcement learning. *arXiv preprint arXiv:2212.07536*, 2022.

[50] J.B. Rawlings, D.Q. Mayne, and M. Diehl. *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017. ISBN 9780975937730. URL `https://books.google.ch/books?id=MrJctAEACAAJ`.

[51] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

[52] Tim Salzmann, Elia Kaufmann, Jon Arrizabalaga, Marco Pavone, Davide Scaramuzza, and Markus Ryll. Real-time neural mpc: Deep learning model predictive control for quadrotors and agile robotic platforms. *IEEE Robotics and Automation Letters*, 8(4):2397–2404, 2023.

[53] Erik Schuitema, Lucian Buşoniu, Robert Babuška, and Pieter Jonker. Control delay in reinforcement learning for real-time dynamic systems: A memoryless approach. In *2010 IEEE/RSJ international conference on intelligent robots and systems*, pages 3226–3231. IEEE, 2010.

[54] Jiaming Song, Arash Vahdat, Morteza Mardani, and Jan Kautz. Pseudoinverse-guided diffusion models for inverse problems. In *International Conference on Learning Representations*, 2023.

[55] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. 2023.

[56] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

[57] Gemini Robotics Team, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Travis Armstrong, Ashwin Balakrishna, Robert Baruch, Maria Bauza, Michiel Blokzijl, et al. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*, 2025.

[58] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

[59] Waywe Research Team et al. Lingo-2: Driving with natural language, 2024.

[60] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34: 24261–24272, 2021.

[61] Homer Rich Walke, Kevin Black, Tony Z Zhao, Quan Vuong, Chongyi Zheng, Philippe Hansen-Estruch, Andre Wang He, Vivek Myers, Moo Jin Kim, Max Du, et al. BridgeData v2: A dataset for robot learning at scale. In *Conference on Robot Learning*, pages 1723–1736. PMLR, 2023.

[62] Thomas J Walsh, Ali Nouri, Lihong Li, and Michael L Littman. Planning and learning in environments with delayed feedback. In *Machine Learning: ECML 2007: 18th European Conference on Machine Learning, Warsaw, Poland, September 17-21, 2007. Proceedings 18*, pages 442–453. Springer, 2007.

[63] Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. *arXiv preprint arXiv:2208.06193*, 2022.

[64] Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025.

[65] Ted Xiao, Eric Jang, Dmitry Kalashnikov, Sergey Levine, Julian Ibarz, Karol Hausman, and Alexander Herzog. Thinking while moving: Deep reinforcement learning with concurrent control. *arXiv preprint arXiv:2004.06089*, 2020.

[66] Bin Xu, Farzam Malmir, Dhruvang Rathod, and Zoran Filipi. Real-time reinforcement learning optimized energy management for a 48v mild hybrid electric vehicle. Technical report, SAE Technical Paper, 2019.

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

[68] Tony Z Zhao, Jonathan Tompson, Danny Driess, Pete Florence, Kamyar Ghasemipour, Chelsea Finn, and Ayzaan Wahid. Aloha unleashed: A simple recipe for robot dexterity. *arXiv preprint arXiv:2410.13126*, 2024.

[69] Haoyu Zhen, Xiaowen Qiu, Peihao Chen, Jincheng Yang, Xin Yan, Yilun Du, Yining Hong, and Chuang Gan. 3d-vla: 3d vision-language-action generative world model. *arXiv preprint arXiv:2403.09631*, 2024.

[70] Ruijie Zheng, Yongyuan Liang, Shuaiyi Huang, Jianfeng Gao, Hal Daumé III, Andrey Kolobov, Furong Huang, and Jianwei Yang. Tracevla: Visual trace prompting enhances spatial-temporal awareness for generalist robotic policies. *arXiv preprint arXiv:2412.10345*, 2024.

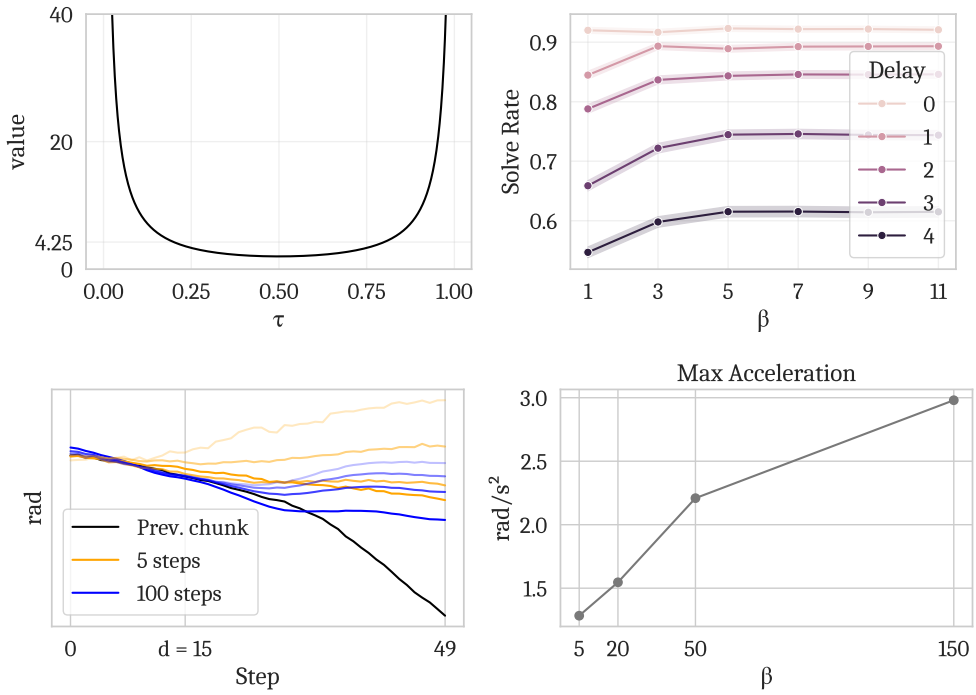

Figure 8: **Top left:** The graph of the value $\frac{1-\tau}{\tau \cdot r_\tau^2}$ from Eq. 2, which we clip at β. At $\tau = 0$, clipping is needed to make the value finite. With 5 denoising steps, if $\beta \geq 4.25$, the clipping only determines the guidance weight for the first step ($\tau = 0$). **Top right:** An ablation of β in our simulated benchmark. Increasing β provides no marginal benefit beyond $\beta = 5$. **Bottom left:** Example real robot action chunks generated from the same noise with 5 denoising steps ($n = 5$) and 100 denoising steps ($n = 100$), with lower opacities corresponding to higher guidance weight clipping ($\beta = \{5, 20, 50, 150\}$). With 5 denoising steps, the generated action chunks diverge when β is too high. **Bottom right:** β vs. maximum acceleration (second discrete difference) for a batch of 325 action chunks generated with $d = 15$ and $n = 5$. Higher β leads to more jerkiness, a proxy for out-of-distribution actions.

A Appendices

A.1 Broader Impacts

The goal of our work is to improve the speed and performance of learned policies for control tasks, and our experiments primarily deal with household robots. This technology has great potential to improve lives, e.g., by automating dangerous and difficult jobs, or assisting the disabled and elderly. Like any technology, it also has the potential for harm—e.g., in military applications, or by displacing physical labor.

A.2 The Necessity of Guidance Weight Clipping (β)

In Section 4.1, we describe how we adapt the inpainting algorithm from Pokle et al. [47] and Song et al. [54] to our setting. One modification we make is to add a clipping value, β, which limits weight applied to the guidance term (Eq. 2), and is necessary to make the weight finite at $\tau = 0$[2]. While image inpainting typically uses a high number of denoising steps (e.g., $n = 100$ in [47]), control problems often use very few steps (e.g., $n = 5$ in our experiments). In this case, we found that high guidance weights led to diverging action chunks, as shown in Figure 8, bottom left. Based on a simulated ablation (Figure 8, top right), we set β to a conservative value of 5.

[2]An alternative approach to avoid the infinite weight at $\tau = 0$ is to start denoising from $\tau > 0$, used in [47], which we did not try.

A.3 Latency Measurements

Method	Latency
RTC (ours)	97ms
BID with $N = 16$ (no forward model)	115ms
BID with $N = 16$ (shared backbone)	169ms
BID with $N = 16$ (full)	223ms
Vanilla $\pi_{0.5}$	76ms

Table 1: Latency measurements for various inference-time methods applied to $\pi_{0.5}$ [23]. Numbers include on-GPU neural network inference only, and are averaged over 10 inference calls after 5 warmup calls. Inference runs on an NVIDIA RTX 4090 GPU using bfloat16 precision and $n = 5$ denoising steps. BID [38] slows down inference due to sampling batches of actions, whereas RTC slows down inference due to backpropagating through each denoising step. BID (no forward contrast) refers to a version of BID without the forward contrast loss, which elides the need for a second model. BID (shared backbone) refers to a version of BID optimized specifically for the π_0 architecture, where the VLM backbone (3B parameters) is shared between the strong and weak model, so only two copies of the action expert (300M parameters) are needed. Full BID requires two copies of the entire model.

Component	Time (mobile)	Time (non-mobile)
Model	96.89 ± 0.16ms	97.43 ± 0.28ms
Network	21.20 ± 3.12ms	6.89 ± 2.39ms
Image resize	11.22 ± 5.00ms	1.44 ± 0.27ms
Other	9.67 ± 3.20ms	3.00 ± 0.68ms
Total	138.98 ± 6.71ms	108.76 ± 2.34ms

Table 2: Breakdown of total inference latency by component for RTC. The image resizing component happens on the CPU of the robot computer. In the mobile manipulation case, this computer is an Intel NUC portable computer with a 12th Gen Intel i7-1260P processor. In the non-mobile case, this computer is a desktop workstation with an AMD Ryzen 9 7950X processor. In both cases, the model runs on a separate workstation with an NVIDIA RTX 4090 GPU; the robot computer and the inference workstation are both connected to the same LAN via a wired Ethernet connection, and communication happens via the WebSocket protocol. Model inference uses bfloat16 precision and $n = 5$ denoising steps. Measurements are taken from 50 inference calls during a real episode rollout, and ± one standard deviation is shown.

A.4 Hyperparameters

Hyperparameter	Description	Simulation	Real-world
n	Denoising steps	5	5
H	Prediction horizon	8	50
$s_{\min}$	Minimum execution horizon	-	25
β	Guidance weight clipping	5	5
b	Delay buffer size	-	10

Table 3: Hyperparameters used for RTC (Algorithm 1). In simulation, d is held constant for each experiment, so $s_{\min}$ and b are not needed. Additional hyperparameters for the simulated experiments can be found in the code.

A.5 Code Release

The code for the simulated experiments is available at `https://github.com/Physical-Intelligence/real-time-chunking-kinetix`.

A.6 Compute Resources

All the experiments in this work use no more than 8 NVIDIA H100 GPUs (one NVIDIA DGX server) at a time. H100s are used via a cloud provider.

Simulated experiments. Training expert policies with RPO [49] with 6 seeds $\times$ 12 environments takes approximately 4 hours on 4xH100s. Generating data from those policies takes approximately 20 minutes on 6xH100s. Training imitation learning policies with flow matching for each environment takes approximately 1.5 hours on 2xH100s. Evaluating the policies for 2048 trials per environment takes approximately 5 minutes on 6xH100s.

Real-world experiments. We use policies fine-tuned from the $\pi_{0.5}$ [23] base model. Each fine-tuning run takes approximately 24 hours on 8xH100s. All of our real-world inference is done on a single NVIDIA RTX 4090 GPU in a workstation in the same building as the robots.