Learning Temporal Action Chunking for Motor Control

by

Ruiyu Gou

B.Sc., University of British Columbia, 2022

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL
STUDIES
(Computer Science)

The University of British Columbia
(Vancouver)

August 2024

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Learning Temporal Action Chunking for Motor Control

submitted by **Ruiyu Gou** in partial fulfillment of the requirements for the degree of **Master of Science** in **Computer Science**.

Examining Committee:

Michiel van de Panne, Professor, Computer Science, UBC
Supervisor

Ian Mitchell, Professor, Computer Science, UBC
Supervisory Committee Member

Abstract

Deep reinforcement learning has had significant success at learning motor control tasks. Typically, these policies are fully closed loop or 'state indexed', implying a control policy that is queried at every control time step with the current state in order to estimate the best current action corresponding to that state. However, this approach ignores the inherent predictability of many systems, wherein the future states and actions are often quite predictable and can thus be controlled in an open-loop or 'time indexed' fashion. Chunking of action sequences is a well-established mechanism in cognitive systems to enhance learning capabilities and learning efficiency. By modeling actions in time-indexed chunks, one reduces the computational and perceptual demands required for control. Learning this type of temporal action abstraction remains under-explored. We present a method that learns a chunk-based state-and-time-indexed policy from any existing state-indexed reinforcement learning policy, with minimal added complexity. We show that with a straightforward multi-layer neural network, the chunk-based policy can decrease the required control frequency significantly. In particular, we show a reduction from 60Hz to 10Hz for the control of a 3D humanoid capable of robust and realistic movement across varying terrain. We further propose an adaptive runtime algorithm that can leverage long action chunks while reverting to single-step actions as needed in order to achieve robust behavior.

Lay Summary

We propose an alternative formulation for reinforcement learning policies for motor control based on action chunks instead of single action. We introduce a robust learning algorithm to learn such policies, and a runtime algorithm to enable responsive policy execution based on environment uncertainties. We show a decrease of control frequency from 60 Hz to less than 10 Hz on a challenging humanoid stepping stone task using chunk-based policies.

Preface

This thesis is unpublished independent work by the author, Ruiyu Gou, with guidance and suggestions from their supervisor Dr. Michiel van de Panne.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

BC behavior cloning

DAGGER Dataset Aggregation

DASS Deterministic Actions Stochastic States

DRL deep reinforcement learning

IRL inverse reinforcement learning

MDP Markov decision process

MLP multi-layer perceptron

MPC model predictive control

PPO Proximal Policy Optimization

RBC robust behavior cloning

RBCSS robust behavior cloning with scheduled sampling

RELU rectified linear unit

Acknowledgments

My deepest gratitude goes to my supervisor, Michiel van de Panne, for introducing me to this fascinating field during my undergraduate studies and for providing support, guidance, and keen insights throughout my master's program. This thesis would not have been possible without his mentorship, detailed comments and feedback over multiple iterations. Michiel's genuine curiosity has been inspiring and I hope to carry it forward in all my future work.

My thanks to the second reader of this thesis, Ian Mitchell, for providing constructive feedback on iterating the thesis from a control perspective.

I would like to thank my past collaborators and lab mates — Tianxin Tao, Niloofar Khoshsiya, Daniele Reda, Setareh Cohan, Nick Ioannidis, and Xindong Lin — for all the exciting idea exchanges and discussions. Most importantly, I am grateful to them for making the past few years so joyful. I will always cherish the memories of our all-nighter deadlines, ramen lunches on rainy days, and outings in the beautiful BC mountains, snow or shine.

Thanks to our lab alumni, Yuni Fuchioka and Nam Hee Kim, for discussions that broadened the initial exploration of this project. To Hung Yu Ling and Zhaoming Xie for their help and insights on the physics-based control environment. Thanks to X660 folks and my cohort — Chunjin Song, Zhijie Wu, Aaron Dharna, Jenny Zhang, and many others—for fostering a great lab atmosphere that cheered me up when my reward curves flatlined. Special thanks to Shengran Hu for providing valuable feedback during the thesis presentation making.

Thanks to my family, friends, and my dearly missed dog.

To Larian Studios for releasing *Baldur's Gate 3* early, leaving me enough time to focus on this thesis.

Chapter 1

Introduction

Creating an autonomous intelligent humanoid agent involves integrating many capabilities including reasoning, perception and control of movements, enabling interaction with and explorations of the environment. In this work, we focus on motion, which, contrary to intuition, might be the most challenging aspect. In the book *Mind Children* [25], Hans Moravec discusses this dichotomy, noting that it has become relatively easy for computers to exhibit adult-level performance in solving problems on intelligence tests or in games like checkers, but significantly more difficult to equip them with the sensorimotor skills of a one-year-old. Moravec suggests an explanation; human sensorimotor systems, having evolved over a longer period, are highly efficient. The more efficient a system is, the more autonomous and unconscious it appears, making "the difficult look easy". Inspired by this, this thesis would like to take a deliberate step back by questioning the basic formulation of state-indexed policies commonly used in deep reinforcement learning (DRL) for motor control. We draw insights from cognitive chunking and construct policies based on state-and-time-indexed action chunks. By learning and evaluating chunk-based policies in physics-based motor control tasks, we seek to improve our understanding of motion intelligence.

State-indexed vs. state-and-time-indexed

Deep learning based controllers have proven their effectiveness for the task of controlling movements, also known as motor control. The typical formulation is

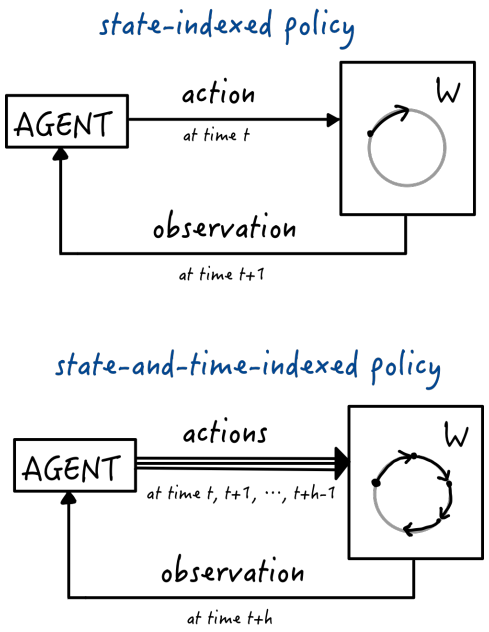


Figure 1.1: The feedback loop of a state-indexed policy and a state-and-time-indexed policy. For convenience, we will refer to it as a chunk-based policy.

as follows: the environment dynamics are modelled as a Markov decision process (MDP); an agent receives a current observation, and passes it through a policy network, to infer an action. The simulation or the world model applies the action, returns a reward together the next observation, which the policy immediately consumes and outputs next action. The reward signal is then used to steer and improves the policy. Figure 1.1 shows this feedback loop for a state-indexed policy.

However, this setup arguably does not directly model how humans or animals move. We do not always pause every few milliseconds to reprocess our perception, to reevaluate our surroundings and to generate one discrete-time action. We do not plan actions in single-step isolation. This process, if performed explicitly, is inefficient. Consider when we swing a racquet to hit a tennis ball that is travelling at 200 km/h, or bend our fingers to press down a piano key, we do not plan our actions in isolated muscle contraction and relaxation. Instead, the body executes through 'habitual response' or 'muscle memory' at a finer level, a sequence of actions that

are highly correlated over time. This sequential execution could be in an open-loop manner, as no intermediate state feedback is needed until the end the of sequence. This points to an intriguing question: can we learn such policies which operate on action chunks, as a more natural and efficient way to control movements?

With chunking, a minimalist state-and-time-indexed policy can be formulated as depicted in Figure 1.1. The agent receives an observation, then outputs a sequence (chunk) of actions. The world or a simulator executes this sequence as a whole, and then feeds the observations available at the end of the chunk back to the agent in order to predict the next action chunk.

Deep learning controllers as simplified models

By querying the policy less frequently, we reduce the computational demands on the system just as reducing cognitive load in a brain, thereby enhancing both compute-efficiency and tolerance against any latency, as within the sequence actions can be executed without further query and the delayed feedback can be gathered to enhance uncertainty estimation and to produce corrective actions later.

In seeking to better understand and replicate a human sensorimotor control policy, we are curious about various aspects of motion control such as the latency and frequency of response (from external sensor to perception) and resolution (from perception to motor control), and how a deep learning based controller can model this with simplifications that may or may not make sense.

First and foremost, it is important to acknowledge that, given the highly integrated nature of neuromuscular systems, the model is an analogy at best. For instance, human gait synergy requires different muscle groups to collaborate at different phases of the task. This system is by nature compositional on different temporal scales: each motor unit includes 3 to 2000 muscle fibers, and each muscle twitch contraction time varies between 20 to 100 ms for different motor units [46]. Response also takes time: human sprinter reaction time from hearing the gun to their first change in force averages around 130 to 180 ms [10] which is a significant chunk of time in highly dynamic motions. In contrast, we have deep learning based controllers with a simplified model, isolating each joint and outputting individual target angles or torques. We also assume zero latency between perception

Task name	Control Freq. (Hz)	Physics Δt (s)	# frame skip
Ant [36]	20	0.01	5
Hopper [11]	125	0.002	4
Swimmer [1]	25	0.01	4
Pusher [1]	20	0.01	5
Reacher [1]	50	0.01	2
Dexterous hand [30]	25	0.002	20
Walker 2D [1]	125	0.002	4
Humanoid locomotion [39]	67	0.003	5
Humanoid stand up [39]	125	0.002	4

Table 1.1: Control frequencies for physics-based motor control tasks in the gymnasium package [1]. Simulator physics time step and number of frame skipped (equivalently, action repeat) are included if applicable.

and motor action.

Different sensory inputs may also arrive at different frequencies. Consider a humanoid stepping stone controller [48] for example: the controller assumes observation space of (1) its own joint angles and velocities, i.e. proprioceptive information, (2) foot contact, i.e. a simplified version of tactile sensing , (3) the location and orientation of the next stepping stone, i.e. visual sensor input. Human visual sensing arguably can be deemed to be upper-bounded by the flicker fusion threshold of 40 - 60 Hz depending on ages, sometimes up to 100 Hz in response to different conditions [34] . Meanwhile, tactile sensing can detect vibrations up to 5400 Hz, and for a temporal duration for as low as 5 ms as opposed to 25 ms for vision, and 0.01 ms for aural (which is not typically modeled as part of the observation space) [45].

Nonetheless, a simplified controller treats both observation space and action space, of all components, as being constant fixed time step sequences flows. In the humanoid stepping stone task, we employ the policy with a frequency of 60 Hz. Existing works using conventional state-indexed reinforcement learning (RL) control policies are trained and evaluated on classic physics-based gym environments with a control frequency of 20 - 50 Hz for more static robot arm manipulation tasks and 67 - 125 Hz for more dynamic tasks such as hopper or humanoid locomotion

and get up (see Table 1.1 for details). These control frequencies have typically already taken into account a common action repeat (or 'frame skip') abstraction layer on top of simulator physics step. We further discuss this subtlety in the context of MDP formulation in § 3.1.

Here, we aim to develop an additional layer of temporal abstraction by enabling time-indexed chunking of varying action outputs. This chunk-based policy approach allows for compositionality across different frequency scales, bringing the simplified controller model arguably closer to mimicking the real human sensorimotor system while reducing the overall control frequency.

Modeling uncertainties

We hypothesize that chunking is feasible given the deterministic evolution of most motions, which stands in contrast to MDP formulations which assume stochastic transitions. Sources of stochasticity include both environment dynamics and policy execution. We also note that even though the policy is queried deterministically during inference, i.e. the most probable action is chosen greedily, it could be learned in a sub-optimal fashion. To this end, we seek to integrate modeling of these uncertainties together with the chunk-based policy to enhance its robustness.

Contributions

In this work, we aim to enhance the efficiency and adaptability of motor control through temporal action chunking. We propose an algorithm robust behavior cloning with scheduled sampling (RBCSS) to learn a chunk-based policy given an expert state-indexed policy. We demonstrate that such policies can reduce the query frequency of continuous control tasks without hampering performance. We incorporate a chunked value function that captures the stochasticity arising from both environment and sub-optimal policy learning. An adaptive runtime algorithm conditioned on the value function allows the system to fall back to the one-step optimal policy when uncertainty is high in the coming chunk, thus increases policy robustness. We evaluate our chunk-based policies on the challenging physics-based motor control task of 3D humanoid locomotion across variable-height stepping stone task on varying terrains. The results demonstrate that leveraging temporal action

chunks achieves comparable performance to the state-indexed reinforcement learning approach, with a considerable reduction in control frequency.

Chapter 2

Related Work

Our work is inspired in part by the action chunking concept in cognitive psychology. This chapter reviews related work that explores temporal extension for motor control including reinforcement learning (RL) with temporally extended actions, model-based control with search over a future horizon, and recent developments in sequence-based controllers such as diffusion model or transformer policies.

2.1 Chunking In Cognitive Psychology

In cognitive science, "chunking" is studied under two different contexts; one refers to how animal's memory system stores, organizes and queries information in groups and the other relates to organizing motor programs which often assumes some primitive structure such as ordered sets or sequences [12]. Our work focuses on the latter and we refer to it specifically as "action chunking". In general, action chunking studies the grouping of primitive actions into higher-level units, namely chunks, that can be executed as a whole.

Evidence of action sequence execution has long been established through experimental observations in continuous sequence serial reaction time tasks. Increase in response speed suggests that participants are not just reacting to individual stimuli but are instead executing pre-learned and integrated action chunks [27]. In a study for primate movement sequence [22], monkeys are reinforced by reward to hit targets on a monitor. In a mode where the visual cues appear in repeated order

during training, the monkeys learn to predict the sequence of actions without re-evaluating the visual cues [41–43]. Moreover, clear neuronal activity differences between motor sequence initiation and the subsequent performance are observed in rodents during rapid reactions.[15, 16].

Learning mechanisms of such action chunks have been of interest. Studies, e.g. [41, 42], propose that humans learn new motor skills by selecting individual actions one by one in a slow stage, and then eventually learn to execute entire sequences, i.e., motor chunks. This learning hypothesis is supported by studies such as [26], where animals show aversion to immediately repeating prior actions or executing a sequence of actions in unfamiliar environments. It also suggests that the learning pattern of motor sequence depends on short-term visuospatial working memory capacity [5]. Conversely, some view action chunking as a decomposition of a long planning horizon. When observing how monkeys learn locally optimal movement chunks, research suggests chunking serves as a divide-and-conquer unit for more efficient optimal control [32].

Several studies seek to develop an analogy for sensorimotor learning using RL computational models. Habitual action sequence formation can be viewed as an open-loop policy where the reward incorporate factors such as potential performance loss and evaluation speedup [8]. This effectively models the performance-and-computation trade-off in a unified framework. Another interesting perspective interprets chunking as action policy compression [17] when cognitive resources are limited and temporal predictability is present. Figure 2.1 illustrates the two types of policy compression. First, if two different states S_0 and S_1 encode the same information Z_0, where Z is the latent space of task-relevant information, they can lead to the same action and therefore can be cognitively grouped. Such chunking can be implicitly done by the policy network. During training, the network or its encoder is usually thought to be able to learn the mapping from different states into the same latent code if they process the same information. Second, if two states are deterministically sequenced together, their actions a_0 and a_1 can also be chunked as an action sequence. Alternatively, if different possible next states produce the same optimal action, the policy can be agnostic and still chunk the actions into a sequence [8]. Following the terminology suggested by [12], we refer to the first as input chunking and the second as output or action chunking. We explicitly model

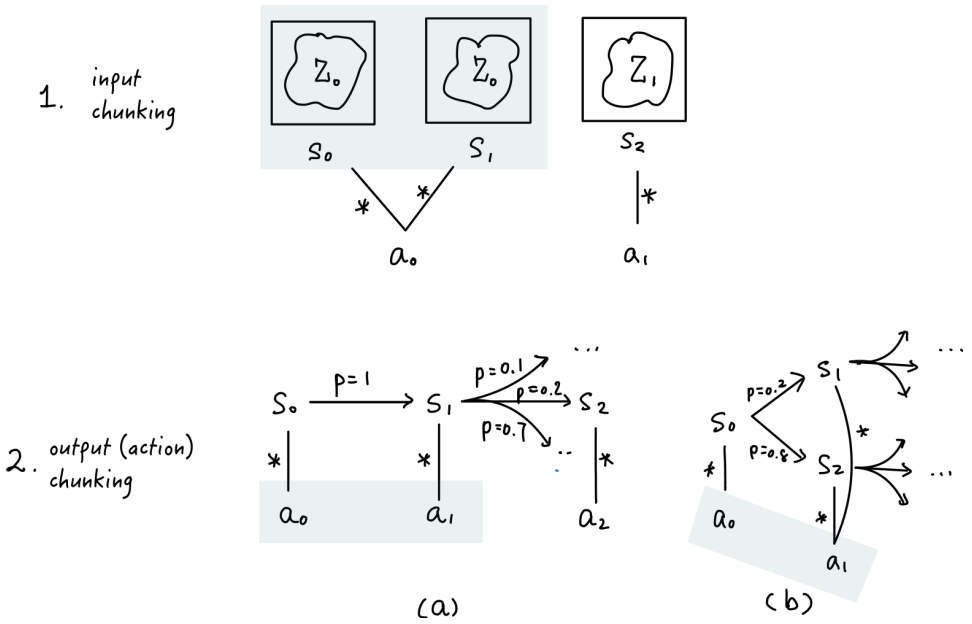


Figure 2.1: Two types of chunking as policy compression. This diagram is inspired by Figure 1 in [17], input vs. output chunks dichotomy in [12] and Figure 3 in [8].

the latter in our work.

2.2 Temporal Actions In Reinforcement Learning

While deep RL policies predominantly rely on the state-indexed approach, significant effort has been made to incorporate temporal extensions during both learning and execution stages. The 'options' framework proposed by [38] laid the mathematical foundation for semi-Markov decision processes, to model continuous-time discrete-event systems which consider temporally-extended actions through closed-loop policies. Hierarchical reinforcement learning in general extends similar concepts of multi-level systems that operate across various spatial and temporal scales simultaneously, building on single-step elementary actions at the lowest-level. Our work fits naturally with these frameworks, where we adopt a simple, clearly-defined unit of operation: small multi-step open-loop action chunks.

A substantial body of deep reinforcement learning work has explored tempo-

ral extension in the form of action repetition. Early deep RL work in the game domain such as deep Q-network [23] wraps a k-step frame skip around the underlying high frequency Atari game simulations. In this approach, the agent outputs an action to be repeated for k frames duration and re-evaluates the environment only every k frames. This technique was initially introduced to reduce the high frequency demands to run a policy in real-time [4], as well as to align the decision-making pace with human game-play. Repeating actions also helps the exploration stage, as a key press often needs to be held for extended duration (a lot more than a few milliseconds) to produce meaningful exploration in a video game. Subsequent studies demonstrate that the frame skip parameter k significantly affects the agents' peak performance, with its optimal value varying considerably across different Atari 2600 games [6], showing that control granularity could be a critical choice for different environments and tasks.

Action repeat duration can be dynamically learned, effectively exploring various frequencies and time scales during runtime ([3, 9, 18, 21, 37]). In our work, similar adaptation takes form of a runtime regret model that dynamically assesses the necessity for querying the policy again versus committing to long chunks of actions. While existing studies on temporally extended actions primarily focus on simple environments such as Gridworld, maze navigation or Atari games, our focus is on motor control where the action space is high dimensional and continuous. Additionally, we seek to model sequences of varying actions over time and therefore not being limited to simply repeating the last action.

2.3 Receding Horizon Control

Another popular approach to motor control is model predictive control (MPC) [24], which is typically executed in a receding horizon fashion. As depicted in Figure 2.2, at every time step during the control loop, the method predicts future path of the system and re-optimizes for a future control trajectory of a certain horizon (shown as green lines). Then it applies the first action in the trajectory (dotted green line), updates the state estimates. At next time step, this process is repeated with the horizon moved by one step forward (shown as blue lines), thus the "receding" horizon. The next trajectory is warm-started using the current one to speed

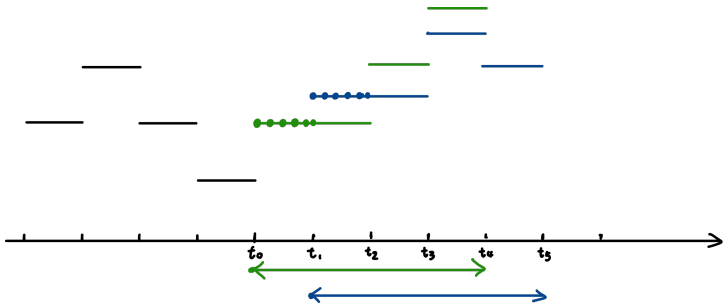


Figure 2.2: A discrete time receding horizon control approach.

up the process. Compared to deep reinforcement learning (DRL) approaches where a model-free policy is obtained, MPC methods often rely on a world model being available, such as having an accurate dynamics model, to compute improved future trajectories during this process. Nonetheless, MPC-based methods [13, 39] have demonstrated ability to solve humanoid control tasks in real-time without needing to train a policy.

Recent advancements in sequential or generative models have further expanded the possibilities of modeling the time-indexed future horizon trajectory. Transformer-based models formulate RL as a sequential decision-making problem. Our fixed-size action chunk policy formulation is similar in spirit to the Action Chunking Transformer [49], with the difference that we show that a multi-layer perceptron (MLP) with similar structure as the state-indexed policy is sufficient without explicit sequence modeling. Despite predicting a future chunk, ACT queries the policy at every time step during inference, adopting the receding horizon control approach. They note that sparse querying results in jerky motion, a smoothing window is applied to overlapping action predictions at the same time step. Our policy is intended to be queried only every k time steps and produces natural motion for a challenging 3D dynamic task which requires full-body balance.

Recent work also seeks to model robotics policy actions using a denoising diffusion process [7]. At every time step, a noise prediction model keeps predicting single diffusion step noise to guide a random sample towards a fixed horizon of action sequence conditioned on a window of past observations. The prediction is then used for control in a receding horizon manner. Diffusion-based policy or planners

are mostly used in manipulation tasks and use end effectors' trajectory as action space as it is a smoother sequence [7, 14].

In general, our work also explicitly models a sequence of actions for a set future horizon, but differs in that we commit to executing the entire action chunk generated instead of adopting a receding horizon control approach.We demonstrate that with a robust learning algorithm, optionally with a runtime adaptive component, the compounding uncertainty can be manageable. Another issue with generative methods, such as diffusion planners, is that they are particularly slow for inference, making them challenging to apply to dynamics control tasks in real-time. Our work models the chunk policy with a simple MLP network of similar structure as the pure state-indexed RL policy, adding little overhead for querying.

Chapter 3

Preliminaries

In this section we review the fundamental problem formulation in RL, and the one-step Markov decision process (MDP), including its optimal solution and the optimization algorithm used to train expert policies for subsequent experiments. Additionally, we provide a brief summary of imitation learning approaches, which our learning algorithm builds upon.

3.1 Reinforcement Learning

An agent learns to make decisions by interacting with an environment and receive reward signals to positively or negatively reinforce its decision-making policy. The goal of RL is for this policy to maximize the cumulative reward over time.

This process is typically formally defined using the framework of finite, discrete-time MDPs. At every discrete time step $t = t_0, t_1, t_2, \ldots$, the environment state is denoted $s_t \in S$. We assume a fully-observable MDP. An agent with a state-indexed policy $\pi(s)$ observes the current state s_t, and uses this to choose the current action $a_t \in A_S$. Governed by stochastic dynamics $P(s_{t+1}|s_t, a_t)$, the environment takes in one step of action, transitions from state s to state s_{t+1} as well as giving a reward r_t. The target solution is a stochastic policy $\pi : S \times A \to [0, 1]$ that specifies the probabilities of actions to take in each state to maximize the expected cumulative discounted reward, defined as:

$$
\begin{aligned}
V^{\pi}(s) &= \mathbf{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... | s_t = s, \pi] \\
&= \sum_{A_s} \pi(s,a)[r(s,a,s') + \gamma \sum_{s' \in S} p(s'|s,a) V^{\pi}(s')]
\end{aligned} \tag{3.1}
$$

where $\gamma \in [0,1]$ is the discount factor. This function is called the value function of state s following policy π. An optimal policy is defined as the policy that maximizes this value function, that is:

$$
\pi^* = \arg\max_{\pi} V^{\pi}(s)
$$

This discrete-time, finite setup extends to continuous motor control tasks with optional additional transformations between the policy output action and the low-level action seen by the environment dynamics. In practice, such layers of abstraction are often overlooked. For example, when the output action space is defined using target joint angles, an underlying PD controller tracks the target angle for a time step Δt. When the output action is defined as joint torques, the torque is repeated for multiple simulator physics steps. From the agent's perspective, this setup appears as a discrete-time process identical to an MDP with each tick lasting Δt. In this way, many control systems are already, perhaps unknowingly, engaging with a form of temporally extended actions in motor control RL problems.

Proximal Policy Optimization

We train the state-indexed policies used in this work using Proximal Policy Optimization (PPO) [35] with a clipped objective. PPO is an on policy policy gradient algorithm. Its key idea follows the vanilla policy gradient algorithm, where actions that lead to higher return are encouraged and vice versa. Additionally, it uses a simple clipping trick on the advantage estimate to prevent the policy update from moving too far away from the current policy. PPO, known for its stability, has been widely used and demonstrated good performance in various DRL tasks ranging from large-scale self-play agents for games [44], physics-based character control [29] to tuning large language models [50].

3.2 Imitation Learning

Imitation learning follows the same problem formulation, with a policy $\pi(s)$ interacting with an MDP. The difference is that instead of maximizing a reward function known a priori, it assumes access to demonstrations of an expert π^*.

If only the state trajectories of expert demonstrations are available, a reward function can be learned. The learned reward function can potentially be conditioned on certain environment dynamics or varying task capabilities. Such approach is known as *inverse reinforcement learning (*IRL*)* [26], as it attempts to infer the optimization goal from an optimized policy.

Alternatively, if the both state and action trajectories are available, the agent's policy can be trained to replicate the expert's decisions using supervised learning. Such approach is referred to as *behavior cloning (*BC*)* [31]. The agent learns a mapping from states to actions, using a dataset of state-action pairs collected from expert demonstrations. Given a dataset D consists of expert state-action pairs $\{(s_i^*, a_i^*)\}_{i=1}^N$, a continuous action policy π parameterized by θ can be optimized using the mean squared error loss over the dataset:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \|\pi_\theta(s_i^*) - a_i^*\|^2$$

As we deal with tasks that can be solved with state-indexed policies, we adopt an imitation learning approach. We start with training expert policies using RL which can be later queried on the fly, hence we learn the chunk-based policies by BC. However, naive BC has the limitation that the training dataset states do not span the possible state space visited by the learner policy. In the next chapter, we will discuss possible solutions to this issue and propose a remedy that is suitable for learning action chunks.

Chapter 4

Learning Action Chunks

We now introduce our formulation of action chunks and chunk-based policies (§4.1) and describe the related learning algorithm robust behavior cloning with scheduled sampling (RBCSS) (§4.2).

4.1 Action Chunk

A Markov decision process (MDP) model often allows for stochasticity in the observations, the policy itself, and the dynamics. In practice, real world scenarios involving motion control tend to be deterministic over short horizons. Simulators, which aim to model the physical systems relevant to the task, are largely deterministic except for possible numerical instabilities and the possibility of chaotic dynamics for some dynamical systems.

As motivated earlier, we propose a formulation that uses the state-indexed policy to learn action chunks. With minimal additional policy complexity, the formulation aims to enable the policy to predict and execute multiple actions consecutively for a fixed chunk size H. The standard one-step MDP formulation remains the same, but the policy now outputs H sequential actions based on the current state, rather than a single action. The value function and the optimization goal now account for the cumulative reward of executing a sequence of actions.

$$\pi^H(s_t) = [a_t^0, a_t^1, \ldots, a_t^{t+H-1}]$$

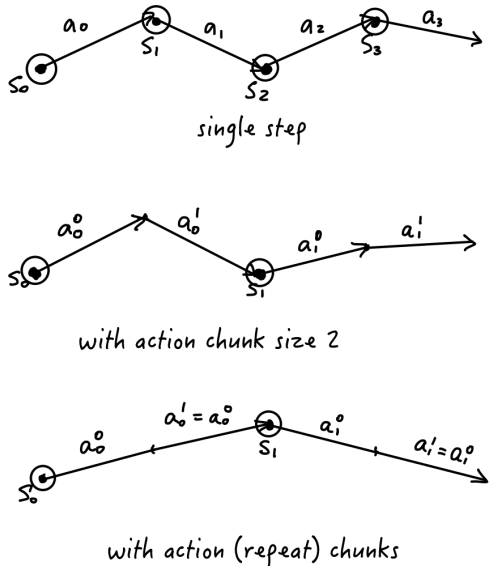


Figure 4.1: Example trajectories consisting of single-step actions and with action chunks of size 2. Circled arrow start indicates a state feedback is acquired.

where s_t is the current state, a_t^h is the action to be executed at time $t+h$. Deep reinforcement learning (RL) parameterizes the policy using a neural network with output dimension $H \times d_a$, where d_a is the dimensionality of the action space. The corresponding value function V^H for a state s is defined as:

$$V^H(s_t) = \mathbf{E}\left[(\sum_{t=0}^{H-1} \gamma^t r_t) + \gamma^H V^H(s_{t+H}) \mid s_t = s, \pi^H \right] \tag{4.1}$$

As illustrated in Figure 4.1, one of the simplest examples of action chunking is *action repeat*. An action repeat policy repeats a particular action H times, as discussed previously (§ 2.2).

4.2 Robust Behavior Cloning for Chunk-based Policies

Given a task that can be solved by a state-indexed expert policy, how can we learn a chunk-based policy, π^H? A straight-forward answer is to extend behavior cloning (BC) to accommodate sequences of actions rather than individual actions. The

dataset $\mathscr{D}$ in this case consists of tuples where each state is associated with the subsequent action sequence. This can be collected by applying a sliding window over the trajectory:

$$\mathscr{D} = \left\{ \left(s_t^*, \left[a_t^*, a_{t+1}^*, \ldots, a_{t+H-1}^* \right] \right) \right\}_{t=1}^{N-H+1} \tag{4.2}$$

as the student policy, π_θ^H, is optimized by minimizing the mean squared loss over $\mathscr{D}$:

$$L(\theta) = \frac{1}{N} \sum_{t=1}^{N} \sum_{i=1}^{H} \| \pi_\theta^H(s_t^*)_i - a_{t,i}^* \|^2$$

In practice, we observe failures of this approach even for $H = 1$, i.e. basic BC. This is a well-known issue [31], due to the student policy drifting out of distribution of its training data. This distribution shift problem can be attributed to how the dataset is collected. By default, BC data is collected from the expert policy through deterministic roll-outs. The expert policy only visits "expert states", and therefore the dataset is centred around these states.

Consider a policy which maps a 2D state space to a 1D action as in Figure 4.2. BC collects roll-outs using the optimal policy and gives a $\mathscr{D}$ that consists of optimal states. A student policy trained on this optimal trajectory has no information on the neighbouring state-action mapping. The resulting policy, visualized as the surface, can thus be completely different even in the immediate neighborhood of the expert trajectory. During inference, when the student policy encounters a perturbation and begins to deviate into out-of-distribution states, the error quickly compounds and the trajectory thus drifts further away into non-recoverable states, as is shown in Figure 4.3.

A common solution to this issue is to collect expert action data from states that a non-expert might find themselves in by adding some (potentially informed) noise during roll-outs. This general principle is simple and often proves effective in practice [20, 33, 40, 47]. We refer to algorithms that follow this principle as robust behavior cloning (RBC). We now discuss two dimensions of the RBC design space: (1) how to inject noise to visit non-expert states and; (2) when to inject the noise.

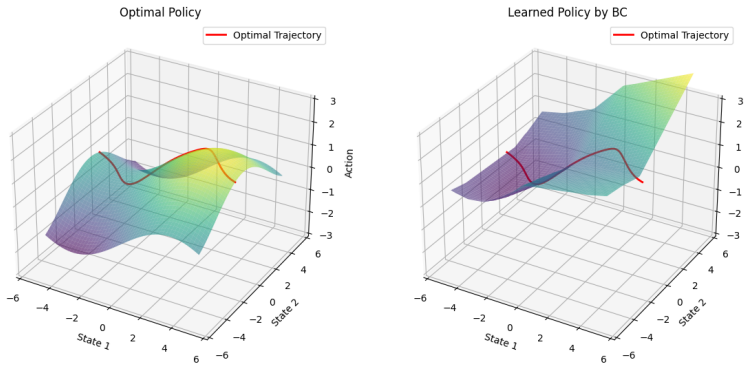


Figure 4.2: An expert policy (left) that maps a 2D state space (x-z plane) to a 1D action (y axis), and a student policy (right) that perfectly reproduces the expert trajectory, but that differs elsewhere.

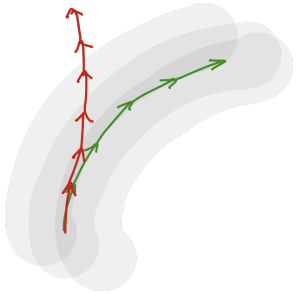


Figure 4.3: Student policy (red) drift out of training distribution (grey shade) collected by expert policy (green).

How to inject noise

There are several possible noise sources. One can directly perturb the actions, resulting in a noised state space visited during roll-outs. This approach demands tuning the noise scale that is used to perturb high-dimensional actions. The noise can come from a stochastic policy. Deterministic Actions Stochastic States (DASS) uses a stochastic expert policy to collect trajectories where recording deterministic optimal actions for training [40, 47]. The perturbations can also come from the non-expert nature of student policy. Dataset Aggregation (DAGGER) [33] iter-

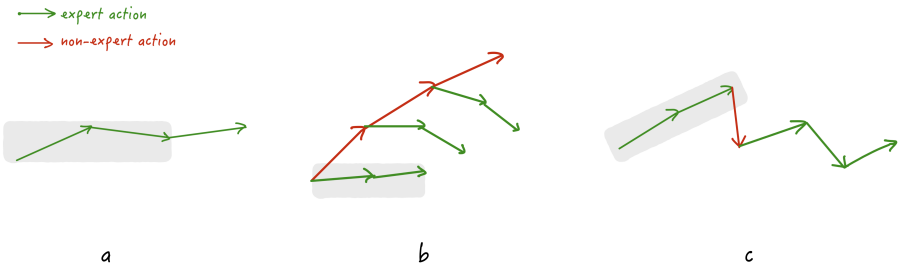


Figure 4.4: Chunked state-actions tuple $(s, a_0, a_1, ...)$ highlighted in grey for training. Valid chunked actions for training are shown for (a) behavior cloning (BC), (b) Dataset Aggregation (DAGGER) or Deterministic Actions Stochastic States (DASS), (c) robust behavior cloning with scheduled sampling (RBCSS).

atively collects state trajectories using the current student policy [1], and queries the expert policy to get the corresponding optimal actions. Then the newly collected (student states, expert actions) are aggregated into $\mathscr{D}$ to optimize the student policy before next iteration. Intuitively, this gradually closes the state distribution shift between the student policy and the expert policy and makes sure the student policy is eventually well-trained on its own state coverage. Figure 4.4 illustrates this roll-out approach (b) in comparison with BC (a).

In practice, directly using the student policy can be challenging, as the initial student policy can create a state distribution that is also out-of-distribution for the expert policy. We propose a simple form of perturbation using action repeat, which repeats the last action for one or more steps. As the repeated action is produced by the expert policy, there are no further perturbation tuning considerations.

Scheduling of noise

A second key consideration here concerns when to add the noise or perturbation. Both DASS and DAGGER collect a perturbed state trajectory $\{(s_i, a_i)\}_{i=1}^{N}$, then query

[1] The DAGGER [33] algorithm has the states collected by a mixture of the student policy and the expert policy, i.e. the student policy is used to collect the states with an increasing fraction of time as iteration increases. However, in the experiments they use a specific simple schedule where all iterations, except the first one, use pure student policy to collect states. The latter is what is usually referred to when we talk about DAGGER.

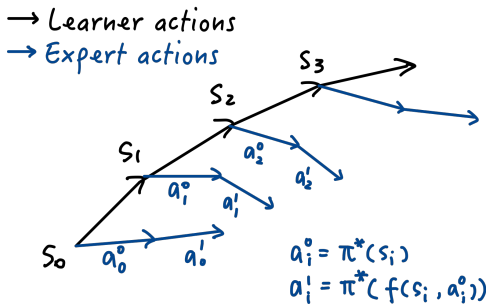


Figure 4.5: A example 2-step chunk-based extension of DAGGER or DASS formulation. Stepping $f(s_i, a_i^0)$ is not always possible, as in practice f often has hidden state information that is not fully captured by s_i.

the expert policy for the optimal action a_i^* for each s_i visited, forming a dataset $\mathscr{D}_D$ of $\{(s_i, a_i^*)\}_{i=1}^N$ for training the student policy as shown in Figure 4.4.b. To extend this single-step version to time-indexed action chunks, we require a dataset of $(s_i, a_i^*, a_{i+1}^*, \ldots, a_{i+H-1}^*)$ tuples to train on, as shown in Figure 4.5. Querying the optimal policy for multi-step actions is in principle possible. However, it is awkward in practice, as this requires resetting the simulator to a given state which can be difficult due to hidden internal state information, and is impossible in the real physics world when working directly with a physical system.

We propose a different version of noise injection inspired by scheduled sampling that alternates between the noise policy and the expert policy to collect the trajectories. Consider the $H = 2$ case illustrated in the last row of Figure 4.6 as an example. At the first state s_0, the expert policy is queried for an action a_0^*. We step the environment with this expert action a_0^*, and query the expert policy again for a_1 with the new state s_1. The next state transition from s_2 introduces a perturbed state by adopting a non-expert action. In practice, this could be as simple as repeating the last expert action, i.e. $a_2 = a_1^*$. The expert policy is always queried for more than H steps for training a π^H policy to ensure sufficient numbers of chunked action sequences such as (s_0, a_0, a_1) in the grey highlight are available. We then filter out continuous expert action chunks $(a_i^*, a_{i+1}^*, \ldots, a_{i+H-1}^*)$ from the trajectories with the chunk's starting state s_i. At the end of each iteration, the student policy

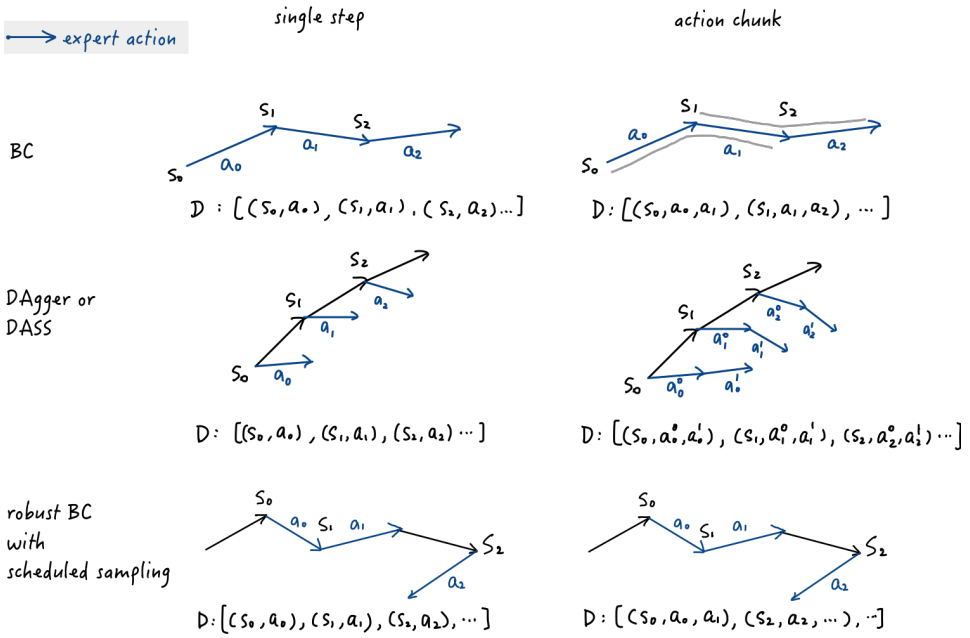


Figure 4.6: An overall comparison of training dataset $\mathscr{D}$ composition for (in rows) BC, Dataset Aggregation (DAGGER) or Deterministic Actions Stochastic States (DASS), robust behavior cloning with scheduled sampling (RBCSS) with (in columns) single-step action and action chunk. The expert actions are indicated as blue arrows and non-expert actions are black. Labeled transitions are the training data.

π_θ^H is optimized for the loss:

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{H} \|\pi_\theta^H(s_i)_j - a^*_{i,j}\|^2$$

i.e., for the perturbed state distribution and expert action chunks queried with each state. We give a comprehensive comparison of the training dataset $\mathscr{D}$ in Figure 4.6. Algorithm 1 gives the pseudo-code for our robust behavior cloning with scheduled sampling (RBCSS) method.

Algorithm 1 robust behavior cloning with scheduled sampling (RBCSS)

With mixture probability p, chunk size H, per iteration number of environment steps n, total number of iterations N, maximum buffer size B
Initialize dataset $\mathscr{D} \leftarrow$ expert demonstrations
Initialize chunked policy π_θ^H
    optionally, bootstrapped from expert policy π^* adding $H-1$ heads
Choose non-expert $\tilde{\pi}$ to be one of the following:
    (1) π_θ^H (2) stochastic π^* (3) action repeater
for each iteration $i = 1$ to N **do** ▷ learning iteration
    Initialize T_i trajectory buffers
    for $j = 1$ to n **do**
        if $\mathbf{U}(0,1) < \frac{1-p^i}{H}$ **then** ▷ schedule
            for $h = j$ to $j + \mathbf{U}(1,H)$ **do** ▷ collect perturbed state-actions
                $a_h = \tilde{\pi}(s_h)$ ▷ apply non-expert (noisy) action(s)
            end for
        else
            for $h = j$ to $j + H$ **do** ▷ collect expert state-actions
                $a_h = \pi^*(s_h)$
            end for
        end if
    end for
    Filter T_i for contiguous expert chunks $(a_i^*, a_{i+1}^*, \ldots, a_{i+H-1}^*)$
    $\mathscr{D} \leftarrow \{\mathscr{D} + T_i\}$[:-B]
    Compute $L(\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{H} \|\pi_\theta^H(s_i^*)_j - a_{i,j}^*\|^2$
    Update policy π_θ^H
    if π^H reaches desired performance **then**
        early stop
    end if
end for

Bootstrapped initialization of the network

If we assume that the first few layers of the networks are extracting features from the environment observation, such abstract information is universal and can be reused when adapting to a chunked policy. In our experiments, if we start by training an one-step expert using Proximal Policy Optimization (PPO), the expert weights are available. Therefore, the chunked policy model can optionally be initialized using the same weights except for the new connections leading to future

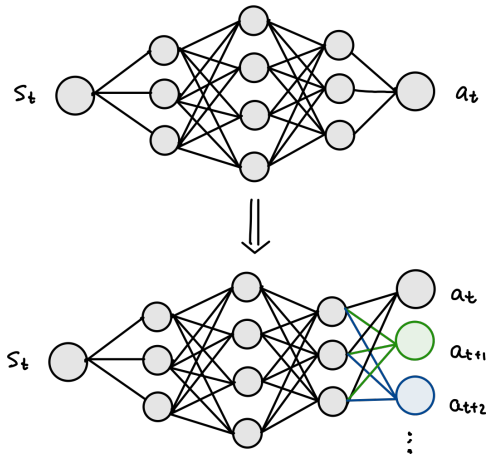


Figure 4.7: A chunk-based policy initialized from a state-indexed policy. The colored neurons and connections are newly instantiated.

actions as in Figure 4.7. This boosts initial learning and achieves faster and more stable convergence in our case. We also freeze updates to the pre-trained connections for the first few iterations to allow newly initialized connections to warm up.

Chapter 5

Adaptive Chunking

Imagine a high-speed racing car driving along a track depicted in 5.1. The track is divided into different regions, each presenting unique challenges:

- **Region A**: This part is open and straightforward, with minimal obstacles or turns. Here, the car can safely operate using larger chunks of actions without frequent re-evaluations, i.e. without state feedback. One can rely more on predictive action sequences, similar to how humans might navigate a familiar task space with minimal conscious attention and instead relying on muscle memory.
- **Region B**: This part has increased task difficulty, such as for narrow turns or obstacles. This region requires an accurate policy with little tolerance for error. The need for precise control makes committing to long action chunks less desirable.
- **Region C**: With unpredictable dynamics such as sudden wind gusts or slippery patches, this section of task demands a highly responsive control strategy due to its stochastic nature. This type of uncertainty is aleatoric, as in it is inherent and cannot be explained by the model. Frequent re-evaluation of the policy is needed to allow for corrective behaviors. This mirrors human responses to sudden disturbances.

Similar analogies can be found in various sensorimotor control tasks. Depend-

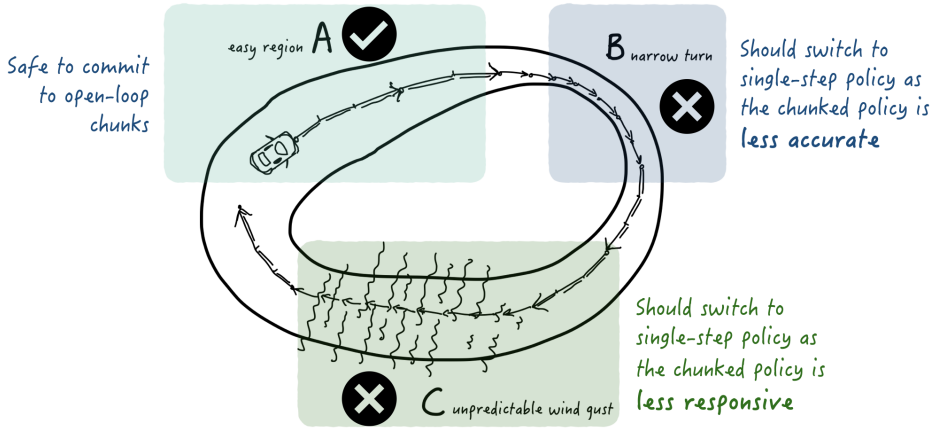


Figure 5.1: Example car-track.

ing on the current situation, we can imagine switching between open-loop muscle memory (region A) versus highly conscious and deliberate feedback-based control (region B and C). Inspired by this, we propose a runtime adaptive chunking algorithm that intelligently switches between large chunks and single-step policies. By fitting a mixed value function V_c measuring the future expected rewards from committing to an immediate chunk, we derive a regret estimate that measures how much the agent may lose by committing to the action chunk. The agent dynamically switches between chunked action and single-step actions based on this regret estimate. This runtime algorithm also provides a flexible trade-off between computational load and performance robustness.

5.1 Regret Modeling

We define a chunked value function $V_c(s)$ as the expected cumulative future discounted reward if, starting from state s, the agent first commits to the chunk sequence of actions $\pi^H(s)$, and then continues with an optimal one-step policy π^*. Note that under this definition, V_c is a special value function that assumes a mixed policy. $V_c(s)$ is computed using a sampled discounted reward estimate with a terminal value bootstrap. It differs from a typical value function estimate as it does not bootstrap its own chunk-only value, but instead uses the optimal state-indexed

policy value estimator V^o from the original single-step expert policy.

$$V_c^{\pi^H}(s_t) = \sum_{i=0}^{H-1} \gamma^i r_{t+i} + \gamma^H V_o(s_{t+H}) \tag{5.1}$$

We define the regret of committing to an immediate chunk:

$$R(s_t, \pi^H, \pi^*) = V_o(s_t) - V_c^{\pi^H}(s_t) \tag{5.2}$$

and use it as an online proxy to decide whether to switch between different chunk lengths, in this case, H or 1.

The regret function incorporates two sources of uncertainty to be an effective indicator for both region B and region C scenarios. First, since $V_c^{\pi^H}$ is estimated based on the actual state distribution visited by π^H, it captures the epistemic uncertainty as which is presented in a potentially sub-optimal chunked policy, thus can be used to signal a need for using shorter-chunk frequency in region B. Second, $V_c^{\pi^H}$ effectively models the aspect of aleatoric state-dependent uncertainty in region C that the agent may care about.

5.2 Adaptive Chunking

At runtime, we would like an agent to be able to decide whether it is in a state where it can safely go open-loop using action chunks like region A, or in a state where it should be cautious and query at a higher frequency like region B. Our criteria is simple: when the regret exceeds a certain threshold as measured as a fraction of the expert policy expected cumulative rewards, the agent commits to a more frequent evaluation.

We include pseudo-code for runtime adaptive chunking in Algorithm 2. Given a single-step and an H-step policy, at every state feedback s_t, the regret is estimated. If the regret is larger than the threshold portion of maximum value, it commits to a single-step policy for one step. If the regret is lower, it commits to an action chunk of length H, and returns a state feedback at the end of the chunk.

An alternative solution might be to have a separate model that predicts the future state and to make the chunking decision based on that deviation. The predictive brain hypothesis [19] suggests this type of approach used by animal's sensorimo-

Algorithm 2 Adaptive Chunking (with a one-step policy and an H-step policy)

Given one-step policy π^* with value function V_o, chunked policy π^H and the chunked value function V_c, horizon H, threshold ratio ρ between (0,1)
Initialize observation s_t
while not terminated or truncated **do**
    if $V_o(s_t) - V_c(s_t) > \rho \cdot \max(V_o(s_t), 0)$ **then**
        $\{a_t\} = \pi^*(s_t)$
    else
        $\{a_t, a_{t+1}, ..., a_{t+H-1}\} = \pi^H(s_t)$
    end if
    Execute selected action(s)
    Update s
end while

tor system, where we inherently keep a predictive model of the states we expect to see, and respond to deviations from that prediction. Curiosity-driven methods [28] adopt the similar idea to encourage exploration. However, the key insight here that drives us away from such explicit model is that we do not need all information in a state prediction and state prediction by itself is not task-aware, as opposed to the more concise regret function.

Chapter 6

Results

We conduct experiments on three different motor control tasks which follow physics-based dynamics and have continuous action spaces, as shown in Figure 6.1. The complexity of these environments ranges from the simplest, an inverted pendulum swing-up problem, to a second-order point mass navigating a landscape with upthrust forces, and the most complex, a physics-based 3D humanoid navigating stepping stones. We focus the bulk of our evaluations on the humanoid locomotion environment as it is the most difficult and interesting task.

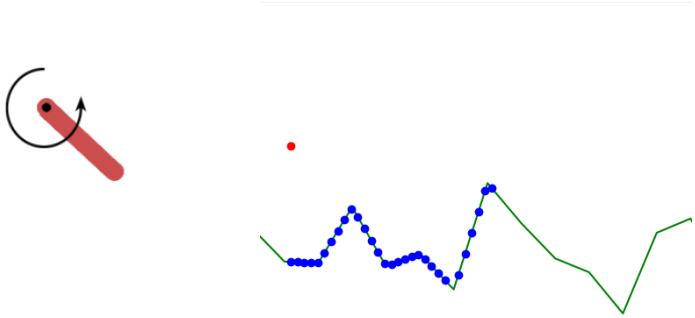
All experiments use expert single-step RL policies trained via Proximal Policy Optimization. A chunk-based policy is parameterized by a neural network of the exact same depth and width except changing the output from the original d_a dimensions to $H \times d_a$ dimensions.

We aim to answer the following questions:

Q1 How does a chunk-based policy perform compared to a state-indexed policy?

Q2 For a given task, what is a reasonable fixed-length chunk size?

Q3 Can adaptive chunking help with performance for longer chunk sizes?



(a) Pendulum swing up

(b) Point mass

(c) Humanoid stepping stone

Figure 6.1: Tasks.

6.1 Task Descriptions

Pendulum

A simple test case is provided by a pendulum environment. The agent attempts to swing and stabilize a pendulum in an upright position by applying a torque. The state space consists of the pendulum's angle and angular velocity. The action space is a single continuous value representing the torque applied. During the episode, the agent is penalized for deviating from upright and also for the torque applied for every time step. Thus, the maximum possible reward is zero for any given time step.

We train an optimal policy using PPO. The policy is parameterized by a 3-layer multi-layer perceptron (MLP) consisting of 128, 128 and 64 neurons per layer with rectified linear unit (RELU) activations. The resulting optimal policy achieves episodic return of average -142 with standard deviation 100, as evaluated over 100 episodes with random initial angle and angular velocity. This is consistent with the documented PPO stable baseline result [2].

Point Mass

In this environment, the task is to control a point mass to navigate flying over a landscape with varying heights using minimal energy. The agent is a point mass. The agent advances forward at a constant speed $20m/s$ and can apply an upwards force with acceleration a ranging from 0 to $1m/s^2$ to avoid excessive elevation or collisions with the landscape, at a time step $\Delta t = 0.1s$. The gravity is set to be $0.5m/s^2$. The observation space consists of the position and velocity of the point mass, as well as the heights of the landscape as 30 scan-dots ahead of the point mass. The point mass is constrained to fly between $h = 0$ to $1m$, and the terrain is $x = 1000m$ long generated by linearly connecting 100 randomly sampled heights following a uniform distribution between [0, 0.5]. There also exist disturbances that vary depending on the proximity of the point mass to the ground, which models a state-dependent environmental stochasticity. This is implemented as a random acceleration noise $n \sim \mathcal{N}(0, 0.1 * \frac{1}{h})$ given the point mass elevation h above current terrain. The point mass is given a step reward r to stay alive, keep close to the terrain, apply a minimized upthrust acceleration for energy minimization while being penalized for crashing:

$$r = \exp(-10a - h + 1 - 1000 * t)$$

where t is a binary number indicating whether the point mass has crashed into the terrain.

The policy is trained using PPO, parameterized by a 3 layer MLP with 256, 256 and 256 neurons per layer with RELU, achieving average return of 500 ± 0.5 per episode over 10 episodes with randomly initialized initial conditions.

3D Humanoid Locomotion

The task goal of the humanoid stepping stone environment is bipedal humanoid locomotion across a sequence of stepping stones having random variation in their vertical placement, as shown in Figure 6.1c. The task specification and training of one-step policy fully follows the original work [48].

The 27 degrees-of-freedom humanoid is designed with human-like torque limits, following the configuration from [48], standing 1.60 meters tall and weighing 59 kilograms. At every time step, it receives proprioceptive information, including joint angles and velocities, root orientation, pelvis height and binary contact indicators for each foot, and generates direct torque outputs for 21-hinge joints. The agent optimizes for a composite reward that takes into account the stepping target distance, humanoid posture, joint limit and penalty for energy use [48]

The single-step expert RL policy and its value function are each parameterized by a 5-layer MLP with 256 hidden neurons each. Both networks are trained using PPO with a learning curriculum that progresses from easy to difficult stepping stone distributions [48]. The policy is queried with a control frequency of 60Hz. With episode time limit of 1000 time steps, the deterministic optimal policy achieves a reward of 3458.6 ± 145.2 across 10 episodes evaluation.

6.2 Chunk-based Policy Learning

Our results show that a chunk-based policy trained by RBCSS can perform comparably to the state-indexed policy across all tasks. Figure 6.2 shows the learning curve of different chunk sizes for the humanoid locomotion task normalized by the optimal episodic return. Each iteration consists of 10,000 environment steps. The learning process automatically terminates when the chunk-based student policy produces average returns higher or equal to the optimal policy π^* with a lower or equal standard error. We also observe qualitatively the performance of resulting π^H as verification. Every student policy is evaluated for 10 episodes. Here, we evaluate π^H without use of adaptive chunking. The $H = 6$ policy here shows that one can achieve near-optimal performance querying the policy only at 10 Hz.

Figure 6.3 shows that RBCSS is able to learn a chunk-based policy up to $H = 6$ while BC fails. Using simple action repeat from the last action as a noise injection,

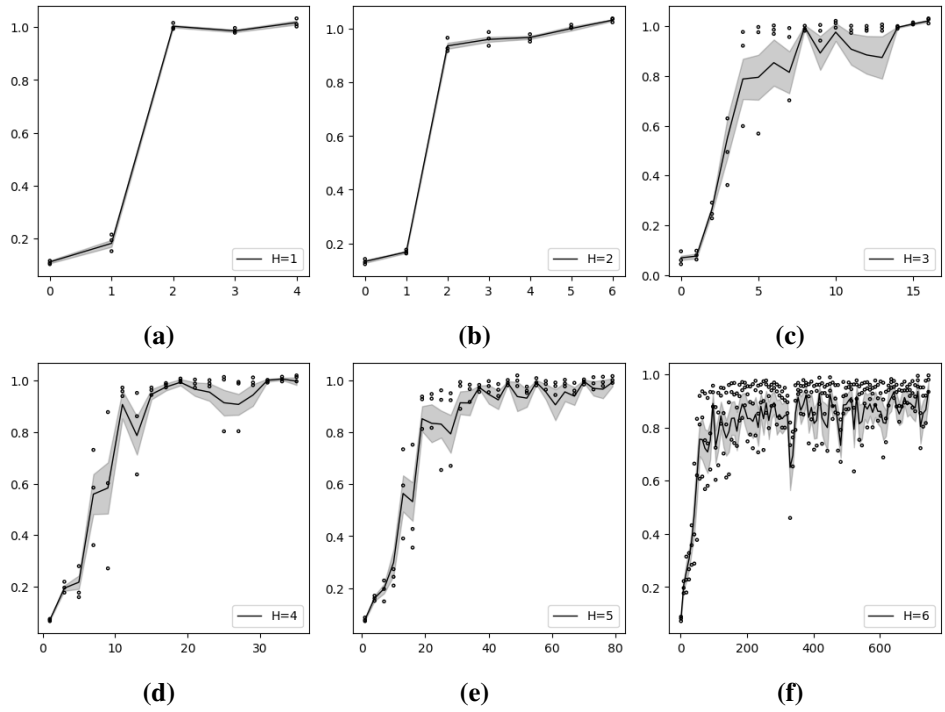


Figure 6.2: Learning curves for chunk sizes $H = 1,2,3,4,5,6$ for the humanoid stepping stone task. The plots show mean (line) and standard error (shaded region), as well as the 25, 50 and 75-percentile of normalized episodic returns (vertical axis) against RBC iterations (horizontal axis)

the learning results also show a much better sample efficiency and resulting performance compared to BC. Note that due to the mixed trajectory nature of RBCSS roll-outs, not all environment state-action(s) tuples are used in the training dataset, thus chunked RBCSS is actually trained on strictly fewer example as compared to BC.

Chunk-based policies π^H with a maximum $H = 20$ for the pendulum task, $H = 8$ for the point mass task, and $H = 6$ for the humanoid stepping stone task can be trained to achieve performance equivalent to the optimal single-step policy while running inference in fixed-size open-loop chunks, measure by both average episodic returns and their variance. For example, the $H = 6$ for humanoid stepping

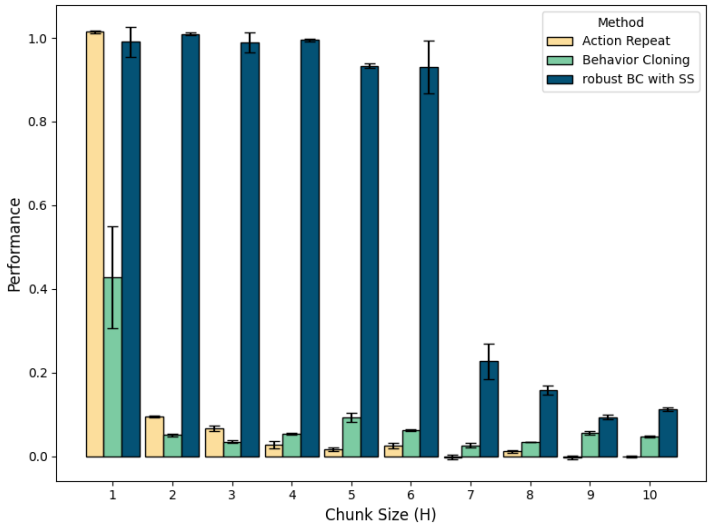
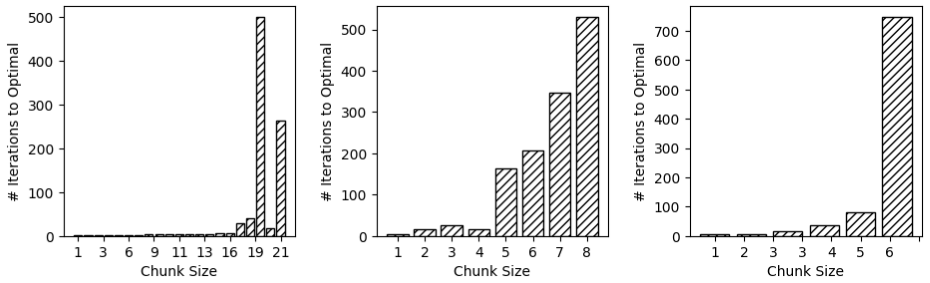


Figure 6.3: Comparison of normalized π^H performance on the humanoid stepping stone task across (1) learned by robust behavior cloning (RBC), (2) learned by behavior cloning (BC) and (3) action repeat, given 1 million steps of environment interactions for fair comparison. The action repeat run with chunk size 1 is the state-indexed optimal policy. The error bars depict one standard deviation of variation, as computed over 10 evaluation episodes.

stone task means it is a fully functional policy running at 10 Hz instead of the original 60 Hz, i.e. with 100 ms control Δt each predicting a window of 6 successive actions lasting 16.7 ms each. In practice, the number of RBC iterations required to reach this performance increases drastically with H, as shown in Figure 6.4. We show in Figure 6.5 the rendered frames for each control step. For chunk sizes beyond these limits, a pure chunk-based policy did not approach that of the original state-indexed expert policy. Figure 6.6 captures the learning progression of a $\pi^{H=7}$ for the stepping stone task. The resulting policy often loses balance when transitioning from a higher to a lower stepping stone. We use this sub-optimal $\pi^{H=7}$ to evaluate the adaptive chunking algorithm in the subsequent section.

Bootstrapping π^H provides faster learning as demonstrated in Figure 6.7. However, as can be seen, the non-bootstrapped policy eventually matches the performance.



(a) Pendulum **(b)** Point Mass **(c)** Humanoid stepping stone

Figure 6.4: Number of iterations that takes to train π^H for optimal non-adaptive performance for different chunk sizes.

Table 6.1: Training Hyper-parameters

Hyperparameter	Value
P	0.8
# of env steps per iteration	32
Bootstrap initialization	True
Maximum buffer size B	100000
Training epochs	50
Learning rate	0.001
batch size	128

Table 6.2: RBC and training hyper-parameters. The value has not been extensively tuned for optimal performance.

We discussed why the variant of using the student policy as a noise injector could result in poor performance when the student policy is not properly warmed up, resulting in limited state overlap with the expert policy distribution. We qualitatively observe this in our experiments. Using the student policy as the noise injector is sufficient for learning the pendulum swing-up chunking policies, as pendulum swing-up does not have a "crash" or "fall over" early termination. However, for the point mass and humanoid stepping stone task, this variant significantly underperforms the action repeat variant which we use to report our performance figures.

Figure 6.5: 24 consecutive environment control steps at 60 Hz (16.7 ms between frames) for the humanoid stepping stone task. Each row corresponds to one chunk of actions using an $H = 6$ chunk-only policy, lasting 100ms.

6.3 Adaptive Chunking

We next test adaptive chunking for long chunks, e.g. $H = 7$ and $H = 10$, where chunk-only policies fail to achieve good performance. We parametrize V_c as a lightweight neural network, similar to a critic network, fitted using an offline process as described earlier in Equation 5.1 . We use the critic network output V^* trained with the actor network π^* of a PPO agent as the single-step optimal value function V_o. Alternatively, the single-step policy value function can be estimated offline with any π^* .

Figure 6.8 shows the example environment state with their corresponding V_c and V_o together with the regret value to show which the algorithm deems a risky

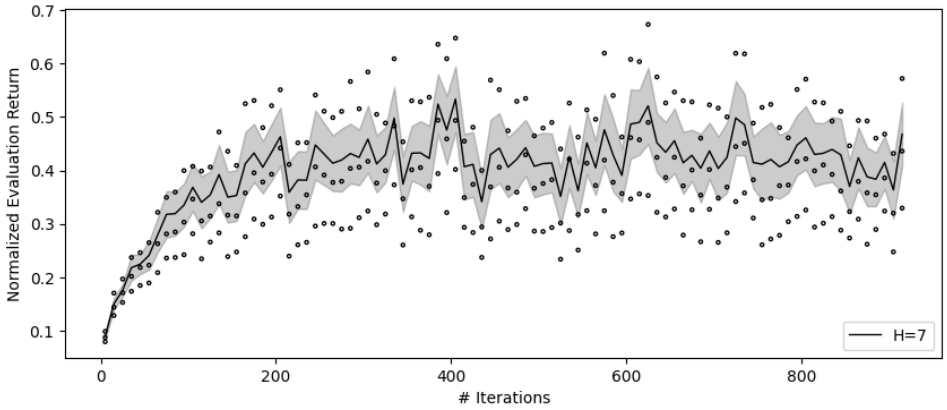


Figure 6.6: Learning curve for locomotion $\pi^{H=7}$

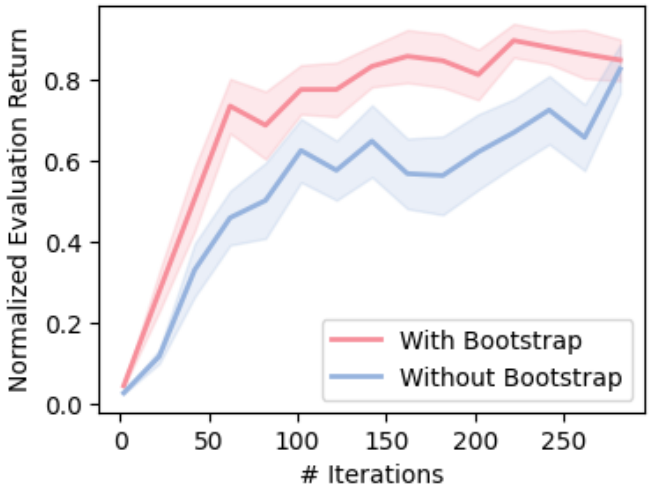
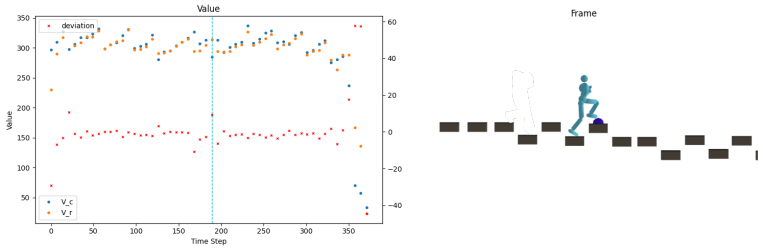


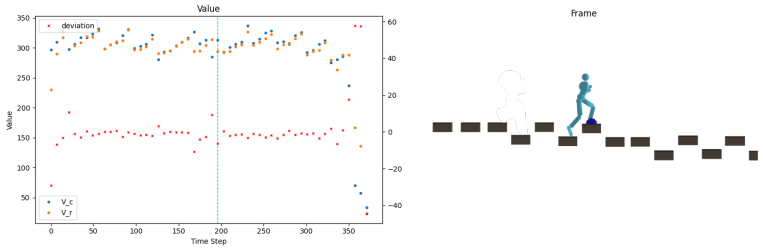
Figure 6.7: Learning progress comparison with and without bootstrap initialization of π^H weights from the original state-indexed single-step expert.

state within a trajectory. We also observe that V_c tends to overestimate compared to the one-step value function V_o. Compounding optimism bias of value estimation alongside aleatoric uncertainty is a possible explanation. Before contacting a lower stepping stone or experiencing a fall, the regret is relatively higher. Conversely, solid contact with a stepping stone during an ascending step is identified as a low-regret state encouraging committing to a chunked action sequence.

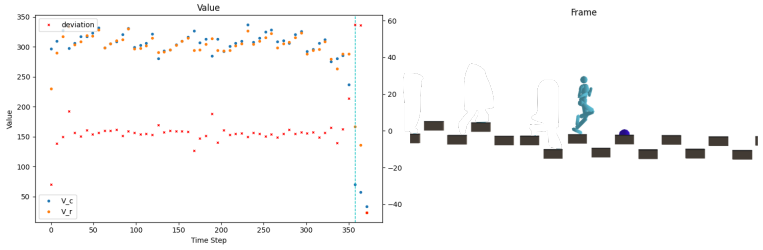
We show the runtime performance of adaptive chunking and their chunk action usage as a function of regret tolerance threshold ρ in Figure 6.9. First, we observe a strict performance increase above chunk-only execution for all ρs. As ρ increases, the chunk action usage increases, ranging 12 - 46% for $H = 10$ and 65 - 95% for $H = 7$. This means a significant portion of the resulting trajectory is resulting from committing to action chunks. The runtime performance decreases as the regret tolerance gets higher. At a low tolerance level, the $H = 10$ policy chooses to commit to action chunks for 30% of the time, but displays virtually no performance loss compared to using only the single-step policy. For $H = 7$, the policy achieves good performance even when chunks are used up to 85% of the time. Note that, the stepping stone task includes stepping across a dozen stones consecutively and the reward accumulates, the final measure is thus sensitive to policy accuracy because a single fall produces a significant decrease in cumulative reward. Qualitatively, a policy with 0.5 normalized performance can already perform good stepping between moderate height variations. We circle the value of ρ one might choose to use to achieve a balance between task reward and chunk action usage.



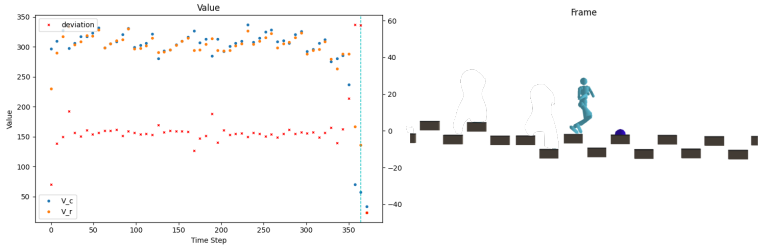
(a)



(b)

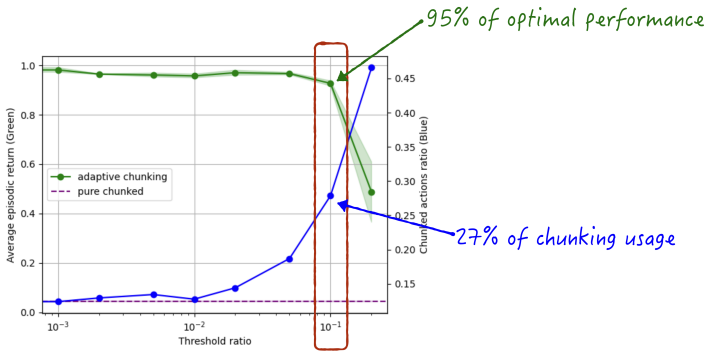


(c)

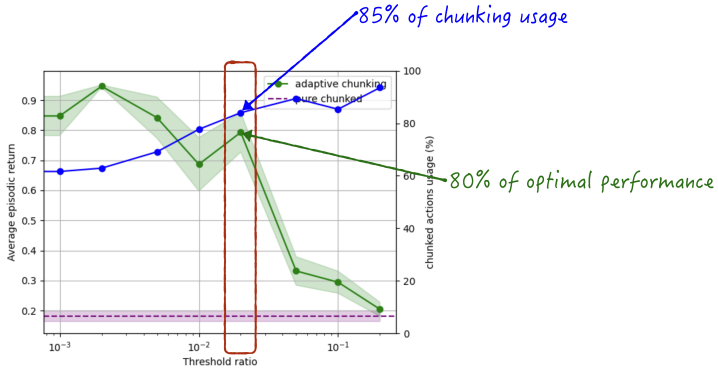


(d)

Figure 6.8: Predicted regret values for $H = 7$, plotted as the red crosses (right y-axis), and its corresponding frame.



(a) $H = 10$



(b) $H = 7$

Figure 6.9: Adaptive chunking normalized performance (green) and corresponding trajectory chunked action percentage (blue) over different threshold ρ, with shaded region ± 1 standard error. Chunk-only performance of the same policy is shown as purple dashed line.

Chapter 7

Conclusion

We extend standard state-indexed RL policies to policies based on multi-step action chunks. The chunks are executed open-loop with temporal action variation beyond commonly used action repeat. We provide an overview for robust behavior cloning (RBC) approaches, and propose an algorithm of RBC with scheduled sampling to learn chunk-based policies. Furthermore, we show that adaptive chunking can be done during execution to allow for accurate and responsive policy query as needed. Chunk-based policies are evaluated in environments such as a complex physics-based 3D humanoid stepping stone task, where we obtain a control frequency decrease from 60 Hz for single-step policy to 10 Hz for fixed chunk size, and as low as 6 Hz with adaptive chunking.

Action chunking draws inspiration from human and animals motor control and attempting to understand the system. There remain many interesting questions. Our current adaptive chunking strategy uses two fixed chunk sizes, for example, switching between $H = 7$ and $H = 1$. Can we use action chunks of any fixed duration $1 \leq H \leq H_{\max}$? Can we learn chunk-based policies directly using RL without having a single-step policy to imitate? Is it feasible to locally optimize and re-combine action chunks, taking advantage of existing sequence generation such as diffusion models? Knowing that optimal control frequencies vary with state and task, can the motor control policy switch between a 5Hz frequency for simple locomotion on flat terrain and a 60Hz frequency for complex dynamic movements like parkour? Arguably, the existing humanoid results already show some of this

when it switches between short and long chunks adaptively between stepping up (long chunks) and down (short chunks). Can we further generalize this behavior and enable a chunk-based policy to vary its control frequency across a more general task space with different granularity? With insights on the output chunking, can we adopt strategy to deal with perception, where proprioceptive information and visual state estimation also come in varying frequencies?

Current large language models and multi-modal sequence models use increasingly large and complex backbones, where chunk-based policies with a lower computational demand could prove practical in such a multi-component system that requires real-time application. With the temporal abstraction, an action chunk provides opportunities to model motion speed and important semantics such as pauses. Action chunks can be viewed as a way to tokenize motions and eventually construct a ”language of motion”. With this and future work, we hope to break down ”the difficult that looks easy” and provide insights towards understanding and replicating human-level motor control in autonomous agents.

Bibliography

[1] Gymnasium Documentation, . → pages viii, 4

[2] rl-baselines-zoo/benchmark.md at 028b1cff225c7da679bacac6f2888251bc383be4 · araffin/rl-baselines-zoo · GitHub, . → page 31

[3] Alexander, Vezhnevets, V. Mnih, J. Agapiou, S. Osindero, A. Graves, O. Vinyals, and K. Kavukcuoglu. Strategic Attentive Writer for Learning Macro-Actions. June 2016. doi:10.48550/arXiv.1606.04695. → page 10

[4] M. Bellemare, J. Veness, and M. Bowling. Investigating Contingency Awareness Using Atari 2600 Games. *Proceedings of the AAAI Conference on Artificial Intelligence*, 26(1):864–871, 2012. ISSN 2374-3468, 2159-5399. doi:10.1609/aaai.v26i1.8321. → page 10

[5] J. Bo, V. Borza, and R. D. Seidler. Age-Related Declines in Visuospatial Working Memory Correlate With Deficits in Explicit Motor Sequence Learning. *Journal of Neurophysiology*, 102(5):2744–2754, Nov. 2009. ISSN 0022-3077, 1522-1598. doi:10.1152/jn.00393.2009. → page 8

[6] A. Braylan, M. Hollenbeck, E. Meyerson, and R. Miikkulainen. Frame skip is a powerful parameter for learning to play atari. In *AAAI Workshop: Learning for General Competency in Video Games*, 2015. URL https://api.semanticscholar.org/CorpusID:194604. → page 10

[7] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Robotics: Science and Systems XIX*. Robotics: Science and Systems Foundation, July 2023. ISBN 978-0-9923747-9-2. doi:10.15607/RSS.2023.XIX.026. → pages 11, 12

[8] A. Dezfouli and B. W. Balleine. Habits, action sequences, and reinforcement learning. *The European Journal of Neuroscience*, 35(7):1036–1051, Apr.

2012. ISSN 0953-816X. doi:10.1111/j.1460-9568.2012.08050.x. → pages ix, 8, 9

[9] I. P. Durugkar, C. Rosenbaum, S. Dernbach, and S. Mahadevan. Deep reinforcement learning with macro-actions. *CoRR*, abs/1606.04615, 2016. URL http://arxiv.org/abs/1606.04615. → page 10

[10] A. Eikenberry, J. McAuliffe, T. N. Welsh, C. Zerpa, M. McPherson, and I. Newhouse. Starting with the "right" foot minimizes sprint start time. *Acta Psychologica*, 127(2):495–500, Feb. 2008. ISSN 0001-6918. doi:10.1016/j.actpsy.2007.09.002. → page 3

[11] T. Erez, Y. Tassa, and E. Todorov. Infinite-horizon model predictive control for periodic tasks with contacts. 06 2011. doi:10.15607/RSS.2011.VII.010. → page 4

[12] Herbert Terrace. Chunking & Serially Organized Behavior in Pigeons, Monkeys and Humans, 2001. URL https://pigeon.psy.tufts.edu/avc/terrace/default.htm. → pages ix, 7, 8, 9

[13] T. Howell, N. Gileadi, S. Tunyasuvunakool, K. Zakka, T. Erez, and Y. Tassa. Predictive Sampling: Real-time Behaviour Synthesis with MuJoCo. 2022. doi:10.48550/arXiv.2212.00541. → page 11

[14] M. Janner, Y. Du, J. B. Tenenbaum, and S. Levine. Planning with diffusion for flexible behavior synthesis, 2022. URL https://arxiv.org/abs/2205.09991. → page 12

[15] X. Jin and R. M. Costa. Start/stop signals emerge in nigrostriatal circuits during sequence learning. *Nature*, 466(7305):457–462, July 2010. ISSN 1476-4687. doi:10.1038/nature09263. → page 8

[16] X. Jin, F. Tecuapetla, and R. M. Costa. Basal Ganglia Subcircuits Distinctively Encode the Parsing and Concatenation of Action Sequences. *Nature neuroscience*, 17(3):423–430, Mar. 2014. ISSN 1097-6256. doi:10.1038/nn.3632. → page 8

[17] L. Lai, A. Z. Huang, and S. J. Gershman. Action chunking as policy compression. Sept. 2022. doi:10.31234/osf.io/z8yrv. → pages ix, 8, 9

[18] A. Lakshminarayanan, S. Sharma, and B. Ravindran. Dynamic Action Repetition for Deep Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), Feb. 2017. ISSN 2374-3468. doi:10.1609/aaai.v31i1.10918. → page 10

[19] O. Lappi. Egocentric Chunking in the Predictive Brain: A Cognitive Basis of Expert Performance in High-Speed Sports. *Frontiers in Human Neuroscience*, 16, Apr. 2022. ISSN 1662-5161. doi:10.3389/fnhum.2022.822887. → page 27

[20] M. Laskey, J. Lee, W. Y. Hsieh, R. Liaw, J. Mahler, R. Fox, and K. Goldberg. Iterative noise injection for scalable imitation learning, 2017. URL http://arxiv.org/abs/1703.09327. → page 18

[21] J. Lee, S. J. Park, Y. Tang, and M. hwan Oh. Learning uncertainty-aware temporally-extended actions. 2024. URL https://arxiv.org/abs/2402.05439. → page 10

[22] Y. Matsuzaka, N. Picard, and P. L. Strick. Skill Representation in the Primary Motor Cortex After Long-Term Practice. *Journal of Neurophysiology*, 97(2):1819–1832, Feb. 2007. ISSN 0022-3077. doi:10.1152/jn.00784.2006. → page 7

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with Deep Reinforcement Learning. 2013. doi:10.48550/arXiv.1312.5602. → page 10

[24] M. Morari and J. H. Lee. Model predictive control: Past, present and future. *Computers & Chemical Engineering*, 23(4-5):667–682, May 1999. ISSN 00981354. doi:10.1016/S0098-1354(98)00301-9. → page 10

[25] H. P. Moravec. *Mind children: the future of robot and human intelligence*. Harvard University Press, Cambridge, Mass, 1988. ISBN 978-0-674-57616-2 978-0-674-57618-6. → page 1

[26] A. Neuringer. Reinforced variability in animals and people: implications for adaptive action. *The American Psychologist*, 59(9):891–906, Dec. 2004. ISSN 0003-066X. doi:10.1037/0003-066X.59.9.891. → pages 8, 15

[27] M. J. Nissen and P. Bullemer. Attentional requirements of learning: Evidence from performance measures. *Cognitive Psychology*, 19(1):1–32, Jan. 1987. ISSN 0010-0285. doi:10.1016/0010-0285(87)90002-8. → page 7

[28] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-Driven Exploration by Self-Supervised Prediction. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 488–489, Honolulu, HI, USA, July 2017. IEEE. ISBN 978-1-5386-0733-6. doi:10.1109/CVPRW.2017.70. → page 28

[29] X. B. Peng, P. Abbeel, S. Levine, and M. Van De Panne. DeepMimic: example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics*, 37(4):1–14, Aug. 2018. ISSN 0730-0301, 1557-7368. doi:10.1145/3197517.3201311. → page 14

[30] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, and W. Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research, 2018. URL http://arxiv.org/abs/1802.09464. → page 4

[31] D. A. Pomerleau. Efficient Training of Artificial Neural Networks for Autonomous Navigation. *Neural Computation*, 3(1):88–97, 1991. → pages 15, 18

[32] P. Ramkumar, D. E. Acuna, M. Berniker, S. T. Grafton, R. S. Turner, and K. P. Kording. Chunking as the result of an efficiency computation trade-off. *Nature Communications*, 7(1):12176, July 2016. ISSN 2041-1723. doi:10.1038/ncomms12176. → page 8

[33] S. Ross, G. J. Gordon, and J. A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning, 2011. URL https://arxiv.org/abs/1011.0686. → pages 18, 19, 20

[34] K. Różanowski, J. Lewandowski, and T. Sondej. Mobile device for the measurement of threshold perception frequency of the flickering source of visible light. *Biocybernetics and Biomedical Engineering*, 35(3):147–156, Jan. 2015. ISSN 0208-5216. doi:10.1016/j.bbe.2014.11.002. → page 4

[35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. 2017. URL https://arxiv.org/abs/1707.06347. → page 14

[36] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. Oct. 2018. doi:10.48550/arXiv.1506.02438. → page 4

[37] S. Sharma, A. Srinivas, and B. Ravindran. Learning to Repeat: Fine Grained Action Repetition for Deep Reinforcement Learning. Sept. 2020. doi:10.48550/arXiv.1702.06054. → page 10

[38] R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, Aug. 1999. ISSN 0004-3702. doi:10.1016/S0004-3702(99)00052-1. → page 9

[39] Y. Tassa, T. Erez, and E. Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913, Vilamoura-Algarve, Portugal, Oct. 2012. IEEE. ISBN 978-1-4673-1736-8 978-1-4673-1737-5 978-1-4673-1735-1. doi:10.1109/IROS.2012.6386025. → pages 4, 11

[40] T. E. Truong, M. Piseno, Z. Xie, and C. K. Liu. PDP: Physics-Based Character Animation via Diffusion Policy, June 2024. URL https://arxiv.org/abs/2406.00960. → pages 18, 19

[41] W. B. Verwey. Concatenating familiar movement sequences: the versatile cognitive processor. *Acta Psychologica*, 106(1-2):69–95, Jan. 2001. ISSN 00016918. doi:10.1016/S0001-6918(00)00027-5. → page 8

[42] W. B. Verwey and D. L. Wright. Learning a keying sequence you never executed: Evidence for independent associative and motor chunk learning. *Acta Psychologica*, 151:24–31, Sept. 2014. ISSN 0001-6918. doi:10.1016/j.actpsy.2014.05.017. → page 8

[43] W. B. Verwey, E. L. Abrahamse, and E. De Kleine. Cognitive Processing in New and Practiced Discrete Keying Sequences. *Frontiers in Psychology*, 1, July 2010. ISSN 1664-1078. doi:10.3389/fpsyg.2010.00032. → page 8

[44] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, Nov. 2019. ISSN 1476-4687. doi:10.1038/s41586-019-1724-z. → page 14

[45] G. Wersényi. Perception Accuracy of a Multi-Channel Tactile Feedback System for Assistive Technology. *Sensors (Basel, Switzerland)*, 22(22): 8962, Nov. 2022. ISSN 1424-8220. doi:10.3390/s22228962. → page 4

[46] D. A. Winter. *Biomechanics and motor control of human movement*. Wiley, Hoboken, N.J, 4th ed edition, 2009. ISBN 978-0-470-39818-0. → page 3

[47] Z. Xie, P. Clary, J. Dao, P. Morais, J. Hurst, and M. van de Panne. Learning locomotion skills for cassie: Iterative design and sim-to-real. In L. P. Kaelbling, D. Kragic, and K. Sugiura, editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 317–329. PMLR, 30 Oct–01 Nov 2020. URL https://proceedings.mlr.press/v100/xie20a.html. → pages 18, 19

[48] Z. Xie, H. Y. Ling, N. H. Kim, and M. Van De Panne. ALLSTEPS: Curriculum-driven Learning of Stepping Stone Skills. *Computer Graphics Forum*, 39(8):213–224, Dec. 2020. ISSN 0167-7055, 1467-8659. doi:10.1111/cgf.14115. → pages 4, 32

[49] T. Zhao, V. Kumar, S. Levine, and C. Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Robotics: Science and Systems XIX*. Robotics: Science and Systems Foundation, July 2023. ISBN 978-0-9923747-9-2. doi:10.15607/RSS.2023.XIX.016. → page 11

[50] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving. Fine-tuning language models from human preferences, 2020. URL https://arxiv.org/abs/1909.08593. → page 14