This appendix provides the following sections:

- documentation of the dataset (Appendix A),
- training details (Appendix B), and
- extra results (Appendix C).

A Dataset Documentation

A.1 Clip Snippet Experts

We signify a clip snippet expert by the snippet it is tracking. We denote a snippet by the clip ID, its start step, and its end step. For example, `CMU_006_12-151-336` is the snippet corresponding to the clip `CMU_006_12` with start step 151 and end step 336. Taking `CMU_006_12-151-336` as an example expert, the file hierarchy for the snippet expert is:

```
CMU_006_12-151-336
├── clip_info.json ..... Contains clip ID, start step, and end step.
└── eval_rsi/model
    ├── best_model.zip ..... Contains policy parameters and hyperparameters.
    └── vecnormalize.pkl ..... Used to get normalizer for observation and reward.
```

The expert policy can be loaded using Stable-Baselines3's functionality.

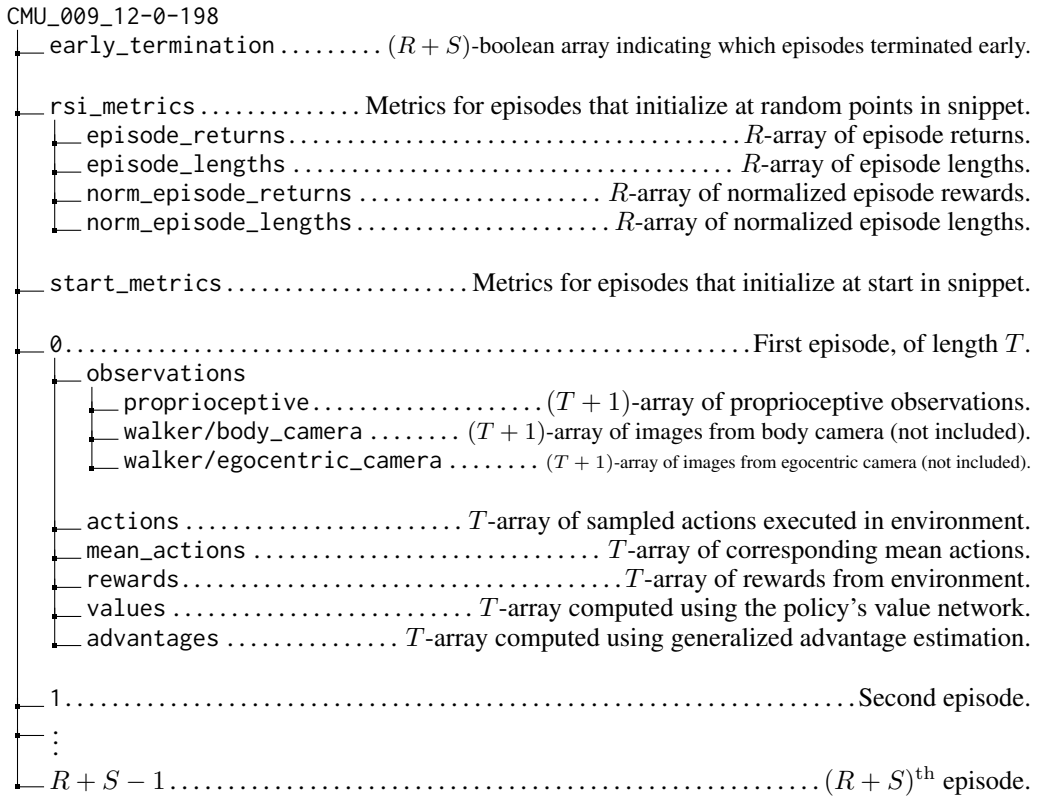
A.2 Expert Rollouts

The expert rollouts consist of a collection of HDF5 files, one file per clip. An HDF5 file contains expert rollouts for each constituent snippet as well as miscellaneous information and statistics. To facilitate efficient loading of the observations, we concatenate all the proprioceptive observations (joint angles, joint velocities, actuator activations, etc.) from an episode into a single numerical array and provide indices for the constituent observations in the `observable_indices` group.

Taking `CMU_009_12.hdf5` (which contains three snippets) as an example, we have the following HDF5 hierarchy:

```
CMU_009_12.hdf5
├── n_rsi_rollouts ..... R, number of rollouts from random time steps in snippet.
├── n_start_rollouts ..... S, number of rollouts from start of snippet.
├── ref_steps ..... Indices of MoCap reference relative to current time step. Here, (1, 2, 3, 4, 5).
├── observable_indices
│   └── walker
│       ├── actuator_activation ..... (0, 1, ..., 54, 55)
│       ├── appendages_pos ..... (56, 57, ..., 69, 70)
│       ├── body_height ..... (71)
│       ├── ⋮
│       └── world_zaxis ..... (2865, 2866, 2867)
│
├── stats ..... Statistics computed over the entire dataset.
│   ├── act_mean ..... Mean of the experts' sampled actions.
│   ├── act_var ..... Variance of the experts' sampled actions.
│   ├── mean_act_mean ..... Mean of the experts' mean actions.
│   ├── mean_act_var ..... Variance of the experts' mean actions.
│   ├── proprio_mean ..... Mean of the proprioceptive observations.
│   ├── proprio_var ..... Variance of the proprioceptive observations.
│   └── count ..... Number of observations in dataset.
│
├── CMU_009_12-0-198 ..... Rollouts for the snippet CMU_009_12-0-198.
├── CMU_009_12-165-363 ..... Rollouts for the snippet CMU_009_12-165-363.
└── CMU_009_12-330-529 ..... Rollouts for the snippet CMU_009_12-330-529.
```

Each snippet group contains $R + S$ rollouts. The first S episodes correspond to episodes intialized from the start of the snippet and the last R episodes to episodes initialized at random points in the snippet. We now uncollapse the `CMU_009_12-0-198` group within the HDF5 file to reveal the rollout structure:



To keep the dataset size manageable, we do *not* include image observations in the dataset. We do provide code to log them when rolling out the experts for generating the dataset.

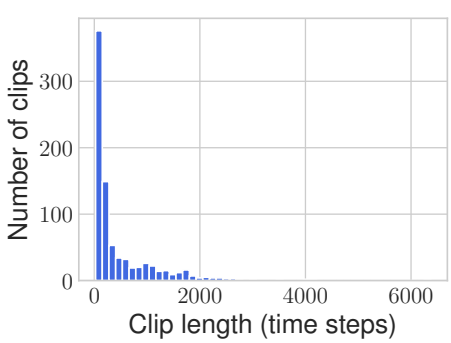
A.3 Hosting Plan

The link to the dataset can be found on the project website. We provide a "large" rollout dataset where $R = S = 100$ with size 600 GB and a "small" rollout dataset where $R = S = 10$ with size 50 GB. The dataset website also includes the policies we trained in Section 5, i.e., the multi-clip tracking policies, RL-trained task policies, and the GPT policy. We also provide a Python script to download individual experts and rollouts from the dataset.

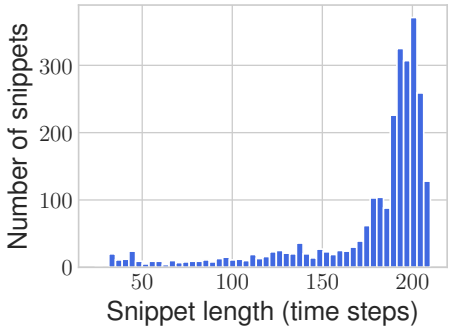
B Training Details

B.1 Clip Snippet Experts

B.1.1 MoCap Snippets



(a) Lengths of the MoCap clips.



(b) Lengths of the snippets generated from the clips.

Figure 1: Lengths of clips and snippets.

The MoCap dataset has a wide spread in clip length (Fig. 1a), with the longest clip being 6371 time steps (191 seconds). Training clip experts to track long clips is potentially slow and laborious, so we follow Merel et al. [2019] by dividing clips longer than 210 time steps (6.3 seconds) into short snippets. In particular, we divide the clip into uniformly-sized snippets with an overlap of 33 time steps (1 second) such that the longest snippet has at most 210 time steps. This yields a snippet dataset with a much tighter range of snippet lengths (Fig. 1b). We do *not* divide the clips from the "Get Up" subset of the MoCap dataset since they contain involved motions of getting up from the ground.

B.1.2 Expert Training Details

Table 1: Hyperparameters for clip snippet expert training.

Total environment steps	150 million
Environment steps per policy update	8192
PPO epochs	10
PPO minibatch size	512
PPO clipping parameter ε	0.25
GAE parameter λ	0.95
Discount factor γ	0.95
ℓ_2 gradient norm clipping value	1
Adam step size	$1e{-}5$ for first 50M env. steps $6e{-}6$ for next 50M env. steps $3e{-}6$ for last 50M env. steps

We use the Stable-Baselines3 [Raffin et al., 2021] implementation of PPO [Schulman et al., 2017] to optimize each expert. Each expert is a neural network with three hidden layers, 1024 neurons in each hidden layer, and the $\tanh$ activation. At the start of each episode, we randomly select a time step from the corresponding clip snippet (excluding the last 10 time steps from the snippet) and initialize the humanoid to match the clip features at the corresponding step. We evaluate the policy every 1 million environment steps using 1000 episodes under the same initialization scheme (but now excluding the last 30 time steps of the snippet) and the same action noise of 0.1 as for training rollouts. We also end the training if the average normalized episode length is at least 0.98 and the average normalized episode reward does not improve by more than 1% from the current best reward after 10 million environment steps. We normalize the observation and reward using running statistics from the environment. We give the other relevant hyperparameters in Table 1. For details

of the reward function and early termination of an episode, we refer the reader to the appendix of Hasenclever et al. [2020].

We ran the training on a mix of Azure Standard_H8 (8 CPUs), Standard_H16 (16 CPUs), Standard_NC6s_v2 (6 CPUs and 1 P100 GPU), and Standard_ND6s (6 CPUs and 1 P40 GPU) VMs.

The observables for the clip expert are: `joints_pos`, `joints_vel`, `sensors_velocimeter`, `sensors_gyro`, `end_effectors_pos`, `world_zaxis`, `actuator_activation`, `sensors_touch`, `sensors_torque`, `time_in_clip`.

B.2 Multi-Clip Tracking Policy

Table 2: Hyperparameters for multi-clip tracking policy training.

Adam step size	$5e{-4}$
Minibatch sequence length T	30
Minibatch size	256
ℓ_2 gradient norm clipping value	1
KL divergence weight β	0.1
Autoregressive parameter α	0
Weighting temperature λ	CWR: 0.2 AWR: 8 RWR: 4

We train the multi-clip policy $\pi(a_t, z_t|s_t, s_t^{\text{ref}}, z_{t-1}) = \pi_{\text{enc}}(z_t|s_t, s_t^{\text{ref}}, z_{t-1})\pi_{\text{dec}}(a_t|s_t, z_t)$ by optimizing the following imitation objective:

$$\mathbb{E}_{\substack{(s_{1:T}, s_{1:T}^{\text{ref}}, \bar{a}_{1:T}, c)\sim\mathcal{D},\\ z_{0:T}\sim\pi_{\text{enc}}}}\left[\sum_{t=1}^{T}\left[w_c(s_t, \bar{a}_t)\log\pi_{\text{dec}}(\bar{a}_t|s_t, z_t) - \beta\,\text{KL}(\pi_{\text{enc}}(z_t|s_t, s_t^{\text{ref}}, z_{t-1})\,\|\,p(z_t|z_{t-1}))\right]\right],$$

where $p(z_t|z_{t-1}) = \mathcal{N}(z_t; \alpha z_{t-1}, (1-\alpha^2)I)$ for some $\alpha \in [0, 1]$. We do this (for each data point in a minibatch) by sampling $z_0 \sim \mathcal{N}(0, I)$, sampling a T-step data sequence (of humanoid states $s_{1:T}$, MoCap references $s_{1:T}^{\text{ref}}$, and expert's mean actions $\bar{a}_{1:T}$) from the dataset $\mathcal{D}$, unrolling the recurrent policy through the sampled sequence, performing backpropagation through time on the objective function, and finally updating the network using the Adam optimizer [Kingma and Ba, 2015]. To speed up training, we normalize the humanoid state s_t and MoCap reference s_t^{ref} using the corresponding mean and standard deviation computed over the entire dataset. For the weighted schemes, we multiply the weight w_c by a constant that ensures the average data weight is 1 so that the KL regularization term maintains the same relative weight. For all schemes, we also sample data from shorter clips at a higher rate to ensure the rollout data from the clips is uniformly even. This gives about 1% improvement in policy evaluation compared to vanilla sampling.

We use PyTorch Lightning [Falcon, 2019] to train the multi-clip policy. The encoder and decoder are both neural networks with 1024 neurons per hidden layer and use layer norm and the ELU activation. The encoder has two hidden layers, while the decoder has three hidden layers. We ran the training on Azure Standard_ND24s VMs, each equipped with 24 CPUs and 4 P40 GPUs. We periodically evaluate the multi-clip policy by running 1000 episodes on the set of MoCap snippets following the same reference state initialization scheme as in the rest of the paper. We found we only need to train the policy for about $50\,000$ steps (about 10% of an epoch) before plateauing on policy evaluation (Appendix C.2.1). We give the other relevant hyperparameters in Table 2.

The observables for the policy are:

- Encoder: `joints_pos`, `joints_vel`, `sensors_velocimeter`, `sensors_gyro`, `end_effectors_pos`, `world_zaxis`, `actuator_activation`, `sensors_touch`, `sensors_torque`, `body_height`, `reference_rel_bodies_pos_local`, `reference_rel_bodies_quats`
- Decoder: `joints_pos`, `joints_vel`, `sensors_velocimeter`, `sensors_gyro`, `end_effectors_pos`, `world_zaxis`, `actuator_activation`, `sensors_touch`, `sensors_torque`

B.3 Transfer for Reinforcement Learning

B.3.1 Go-to-Target Task

This task matches that of Hasenclever et al. [2020], which we refer the reader to for details.

B.3.2 Velocity Control

In this task, a target speed $s^* \in [0, 4.5]$ and direction $\psi^* \in [0, 2\pi)$ are randomly sampled every 10 seconds. Defining the target velocity as $v_t^* = (s^* \cos \psi^*, s^* \sin \psi^*)$ and the humanoid's current velocity as v_t, the reward is defined as the product of a speed factor and direction factor:

$$r_t = \exp\left(-\left(\frac{\|v_t\| - \|v_t^*\|}{\eta}\right)^2\right)\left(\frac{1 + \text{score}(v_t, v_t^*)}{2}\right)^k,$$

where $\text{score}(v_t, v_t^*) = v_t \cdot v_t^* / \|v_t\| \|v_t^*\|$ gives the cosine of the angle between the two velocity vectors. In our experiments, we set $\eta = 0.75$ and $k = 7$. We also experimented with the velocity error reward used by Bohez et al. [2022] but found that our reward was easier to optimize. We terminate the episode either after 2000 time steps (60 seconds) or if any body part other than the feet touches the ground.

B.3.3 Hyperparameters

Table 3: Hyperparameters for RL transfer tasks.

Total environment steps	150 million
Environment steps per policy update	16 384
PPO epochs	10
PPO minibatch size	1024
PPO clipping parameter ε	0.2
KL divergence threshold for early stopping	0.3
Entropy bonus coefficient	General low-level policy: 1e−4 Locomotion low-level policy: 1e−3 No low-level policy: 1e−4
GAE parameter λ	0.95
Discount factor γ	0.99
ℓ_2 gradient norm clipping value	1
Adam step size	5e−5
Number of actors	32
Initial standard deviation for task policy	With low-level policy: 2.5 Without low-level policy: 0.5
Maximum per-element action magnitude for task policy	With low-level policy: 3 Without low-level policy: 1

Like the snippet experts, we train the task policies using the Stable-Baselines3 implementation of PPO. Each task policy is a neural network with three hidden layers, 1024 neurons per hidden layer, and the $\tanh$ activation. We ran the training on Azure Standard_ND6s (6 CPUs and 1 NVIDIA P40 GPU) VMs. We give other hyperparameters in Table 3.

B.4 Motion Completion with GPT

We train a variant of minGPT [Karpathy, 2020] that we adapted to accept continuous inputs and output continuous actions. This particular model has 57 million parameters and was trained with a context length of 32 time steps, corresponding to roughly one second of motion. Similar to the multi-clip policy (Appendix B.2), we sample state $s_{(t-31):t}$ and mean-action $\bar{a}_{(t-31):t}$ sequences of length 32 from the MoCapAct dataset $\mathcal{D}$. To speed up training, we normalize the humanoid state s_t using the corresponding mean and standard deviation computed over the entire dataset. We use the mean squared error loss on the sequence of predicted actions from the GPT. We trained GPT using PyTorch Lightning [Falcon, 2019] on Azure Standard_NC24s_v3, each equipped with 24 CPUs and

Table 4: Hyperparameters for GPT training.

Adam step size	$3\mathrm{e}{-6}$
Minibatch size	256
ℓ_2 gradient norm clipping value	1
Attention dropout probability	0.1
Embedding dropout probability	0.1
Residual dropout probability	0.1
Block size	32
Embedding size	768
Attention heads	8
Number of layers	8
Weight decay	0.1

4 V100 GPUs, for 2 million steps, corresponding to one week of wall-clock time. We give the other relevant hyperparameters in Table 4.

The observables for the GPT policy are: `joints_pos`, `joints_vel`, `sensors_velocimeter`, `sensors_gyro`, `end_effectors_pos`, `world_zaxis`, `actuator_activation`, `sensors_touch`, `sensors_torque`, `body_height`. Importantly, GPT is not given any reference data from the MoCap clip, so any motion generation was done only on the basis of the historical context provided.

C More Results

C.1 Clip Snippet Experts

C.1.1 Training Curves

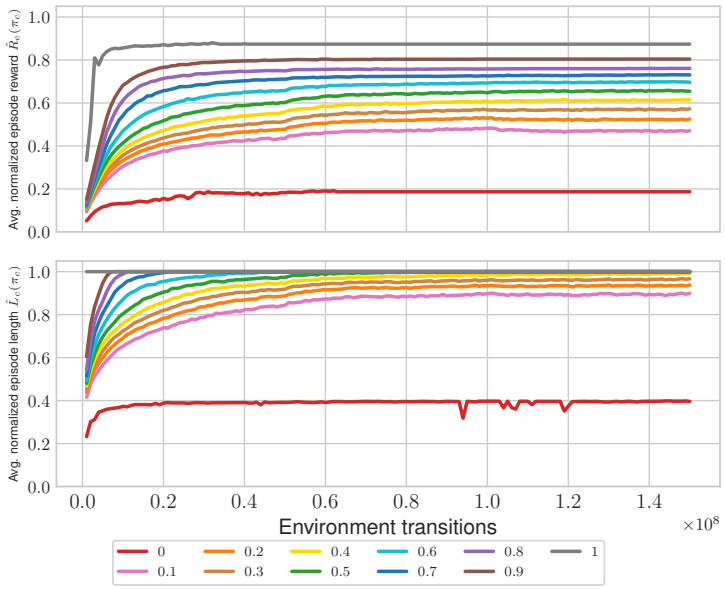


Figure 2: Snippet expert training curves on MoCap dataset.

We give the learning curves for the experts in Fig. 2. In particular, we plot the quantiles $0, 0.1, \ldots, 0.9, 1$ to visualize how the *distribution* of experts improves over the course of training. Overall, we see reliable improvement of the experts with convergence at about 100 million environment transitions.

C.1.2 Expert Performance vs. Snippet Length

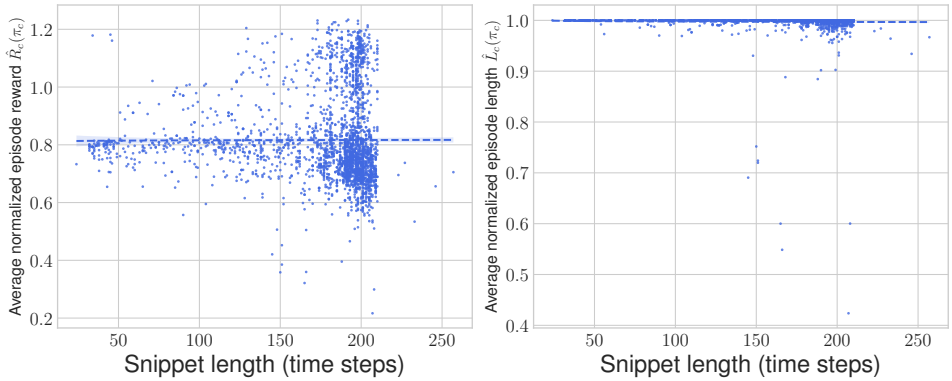


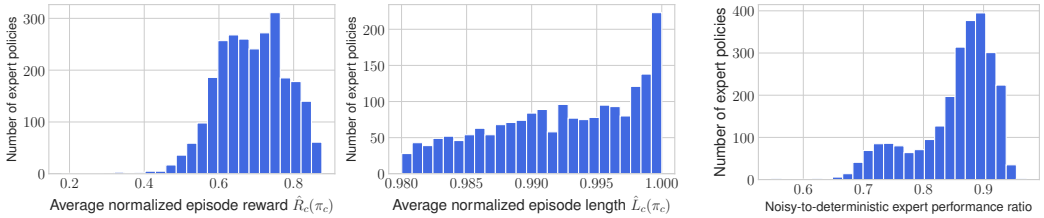
Figure 3: Scatter plot of experts' performance versus the snippet length. Here, the Gaussian noise of the experts is disabled. The performance appears to be independent of snippet length.

Here, we study whether longer snippets are "harder" to track by the expert. Fig. 3 shows scatter plots of the experts' normalized episode reward and length as a function of the snippet length. Overall, the snippet length does not appear to affect the experts' performance as indicated by the fitted curves being relatively flat.

C.1.3 Noisy Expert Evaluations

Table 5: Clip expert results on the MoCap snippets within `dm_control` using the stochastic π_c.

	Mean	Standard deviation	Median	Minimum	Maximum
Average normalized episode reward	0.689	0.092	0.690	0.179	0.876
Average normalized episode length	0.984	0.029	0.990	0.403	1.000



(a) Episode rewards and lengths of the noisy experts.

(b) Performance ratio of noisy expert to deterministic expert.

Figure 4: Noisy expert results on the MoCap snippets within `dm_control`. The noisy experts incur a small performance drop from their deterministic counterparts.

Because the MoCapAct dataset is formed from noisy rollouts of the experts, it is sensible to assess the performance of the experts when rolled out with noise. Table 5 and Fig. 4a show that the experts still have strong performance. We point out the noisy experts on average attain 85% of the performance of the deterministic experts (Fig. 4b).

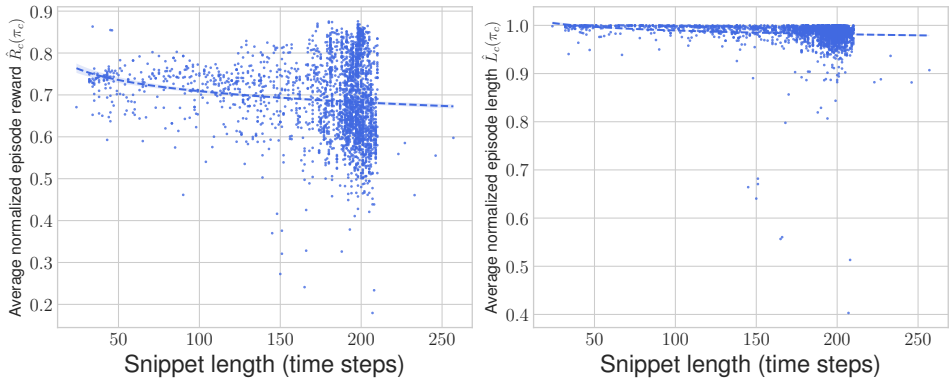


Figure 5: Scatter plot of noisy experts' performance versus the snippet length. There is a minor decrease in performance as the snippet length increases.

From the scatter plot of the noisy experts (Fig. 5), we see a minor decrease in reward and episode length as the snippet gets longer. This is probably due to longer snippets giving more time steps for the noise to destabilize the humanoid.

C.2 Multi-Clip Tracking Policy

C.2.1 Training Curves

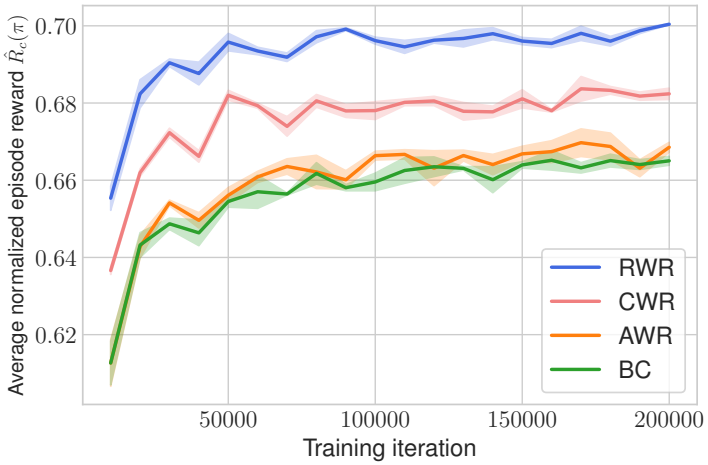


Figure 6: Multi-clip policy training curves on MoCap snippets.

Fig. 6 shows the reward curves for the four weighting schemes. Overall, the reward plateaus after about 50 000 iterations for each scheme, and reward-weighted regression performs markedly better than the other three schemes.

C.2.2 Autoregressive Parameter α

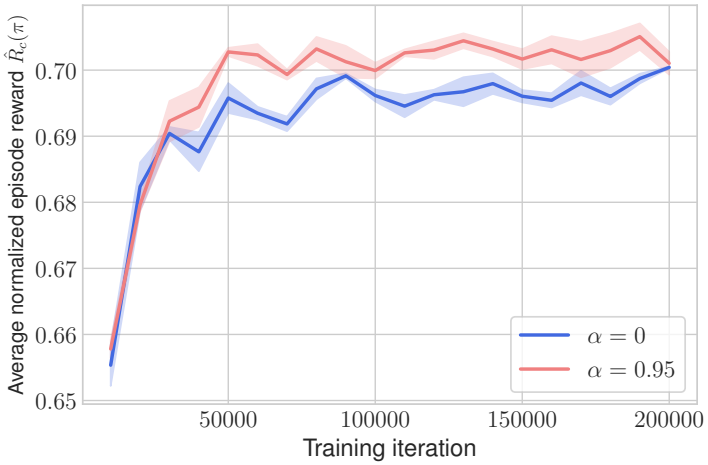


Figure 7: Comparison of multi-clip policy's performance when varying the autoregressive parameter α for the prior distribution $p(z_t|z_{t-1})$. Here, we use the RWR-weighting scheme. Performance is broadly similar for both values of α.

Merel et al. [2019] found that using an autoregressive parameter of $\alpha = 0.95$ gave 50% improvement in policy performance over $\alpha = 0$. Interestingly, in our experiments we found that the performance gap is much smaller (Fig. 7), with $\alpha = 0.95$ only giving 3% improvement. Accordingly, we set $\alpha = 0$ for our experiments (corresponding to a temporally independent prior of $p(z_t|z_{t-1}) = \mathcal{N}(z_t; 0, I)$) so that we could better control the size of the intentions z_t generated by our reference encoder.

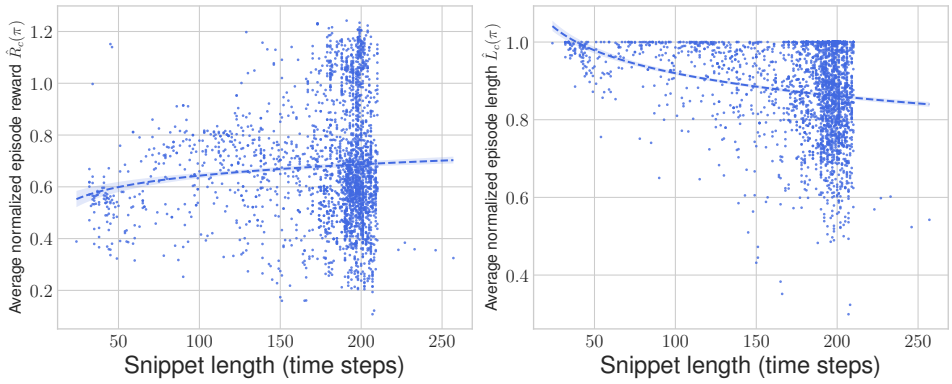


Figure 8: Scatter plot of the multi-clip policy's performance versus the snippet length. Here, the Gaussian noise of the policy is disabled. Longer snippets tend to result in lower episode lengths.

C.2.3 Scatter Plots on Snippets and Clips

Fig. 8 shows the scatter plot of the multi-clip policy on all of the MoCap snippets. Compared to the noisy experts (Appendix C.1.3), we see a more noticeable decline in episode length on long snippets. Intuitively, this is because longer snippets allow for more opportunities for the multi-clip policy to make an episode-ending mistake. The normalized reward, on the other hand, does not give any meaningful trends.

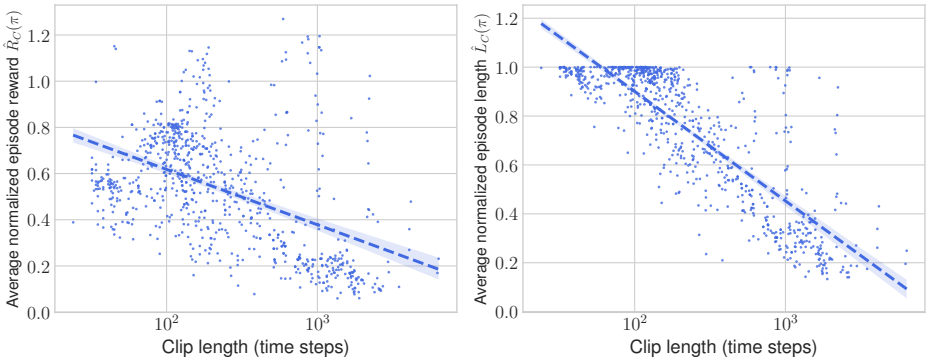


Figure 9: Scatter plot of the multi-clip policy's performance versus the clip length. Here, the Gaussian noise of the low-level policy is disabled. Longer clips tend to result in lower episode rewards and lengths.

One appealing feature of the multi-clip policy is the ability to roll out the policy on entire clips. This also allows us to discover whether the multi-clip policy has learned to "stitch" together the overlapping snippets from the dataset. Fig. 9 shows that while there are long clips that the policy can reliably track, the overall trend is that longer clips result in lower reward and episode length. Intuitively, many clips in the MoCap dataset correspond to locomotion behaviors, which gives many opportunities for the multi-clip policy to make episode-terminating mistakes. Usually, these mistakes correspond to the humanoid legs colliding or one of the feet making bad contact with the ground, both of which cause the humanoid to fall over. The fragility on longer clips points to a shortcoming of MoCapAct: the rollouts only cover (at most) a 6-second window. Because of this, the multi-clip policy is not trained on states that would be encountered deep into a rollout (e.g., 30 seconds into a rollout), which limits the multi-clip policy's performance on many longer clips. Long clips that have high rewards and episode lengths usually have the humanoid standing for long periods of time while doing various arm motions. Here, the motions are much simpler since the humanoid merely needs to maintain balance while standing still.

References

S. Bohez, S. Tunyasuvunakool, P. Brakel, F. Sadeghi, L. Hasenclever, Y. Tassa, E. Parisotto, J. Humplik, T. Haarnoja, R. Hafner, M. Wulfmeier, M. Neunert, B. Moran, N. Siegel, A. Huber, F. Romano, N. Batchelor, F. Casarini, J. Merel, R. Hadsell, and N. Heess. Imitate and Repurpose: Learning Reusable Robot Movement Skills From Human and Animal Behaviors. *arXiv preprint arXiv:2203.17138*, 2022. 5

W. Falcon. PyTorch Lightning. https://github.com/PyTorchLightning/pytorch-lightning, 2019. 4, 5

L. Hasenclever, F. Pardo, R. Hadsell, N. Heess, and J. Merel. CoMic: Complementary Task Learning & Mimicry for Reusable Skills. In *International Conference on Machine Learning*, pages 4105–4115. PMLR, 2020. 4, 5

A. Karpathy. minGPT. https://github.com/karpathy/minGPT, 2020. 5

D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015. 4

J. Merel, L. Hasenclever, A. Galashov, A. Ahuja, V. Pham, G. Wayne, Y. W. Teh, and N. Heess. Neural Probabilistic Motor Primitives for Humanoid Control. In *International Conference on Learning Representations*, 2019. 3, 9

A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 2021. 3

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 3