

## Appendix

### A.1 Additional Details on Training with DAgger

Our expert policy is derived from that of [10], where we additionally train on a stairs terrain. The expert takes in the past thirty steps of proprioceptive inputs, along with a local heightmap observation from the current timestep. We include all parameters for expert training and behavior cloning in Table. 3, 4, 5, 6 in Section (Section A.6). We refer the readers to [10] for further details.

We save intermediate checkpoints during training across multiple seeds, and use them as sampling policies to collect data for the student before the initial dagger step.

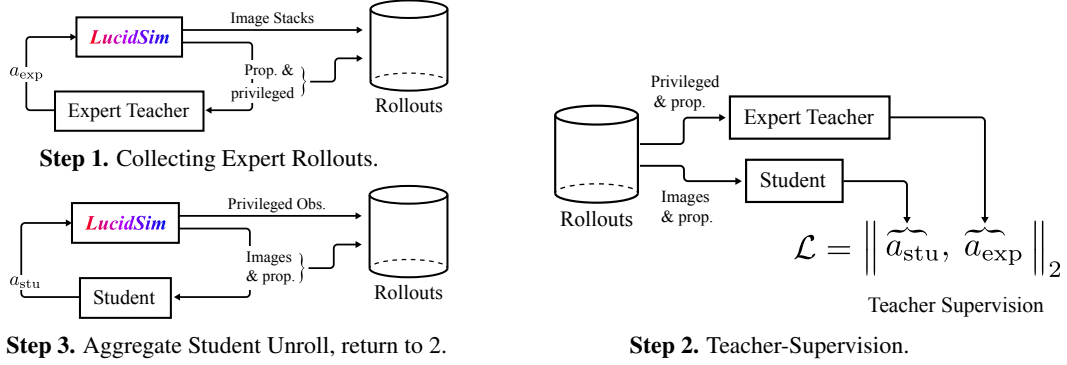


Figure A15: **Learning Procedure.** Our DAgger learning procedure has three steps. **Step 1:** collect expert rollouts from the teacher from LucidSim. **Step 2:** Run behavior cloning. **Step 3:** Using the improved student to sample on-policy data from LucidSim. Aggregate the datasets together and return to step 2, supervised learning.

### A.2 Student Policy Architecture Details

We present a schematic of the policy architecture in figure A16. The backbone is a parallel transformer used by the Pathway Language Model (PaLM [23]).

Past work on quadruped parkour used a composite architecture that processes the input images via a vision network into a compact latent vector that is then fed into a recurrent backbone [10]. We wanted to minimize the number of components, and chose instead to use a transformer architecture borrowed from the Pathway Language Model (PaLM [23]) built off a parallel transformer. This greatly simplifies working with multi-modal inputs. To process the input camera feed, we simply dice each image frame into small patches, all processed in parallel by a shallow fully convolution network. We then stack these patch-tokens with an embedding of the proprioceptive observation of the same timestep, followed by adding a learned embedding to each token position. We then concatenate all time frames into a linear array, and feed the resulting latent vectors as input tokens into the transformer backbone. We found that for RGB input, it is helpful to also include a batch normalization layer before the FCNN. To compute the action output, we stack an additional class (cls) token at the end of the input sequence. The corresponding output token is processed by a shallow action head.

We also adopt the multi-query attention (MQA [24]), which uses a single query head with multiple keys to reduce the inference cost. Our five-layer transformer policy is able to run at  $> 50\text{Hz}$  while

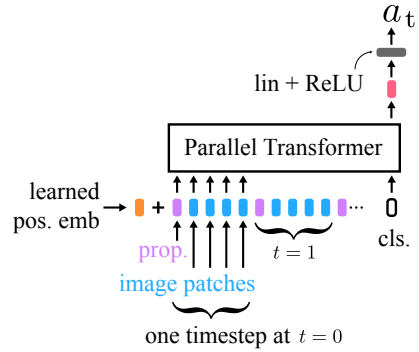


Figure A16: **Policy Architecture.** We treat both the proprioceptive observation and patches of the image frames as tokens in a sequence. The action is computed via an additional class (cls) token.



389 Data generation contribute to the bulk of the wall-clock time of each experiment. We accelerate  
 390 data generation by distributing the rendering requests across a large number of image generation  
 391 GPU workers using a task queue. Figure A18(a) presents the system architecture we use for the  
 392 initial round of sampling with the teacher policy. The unrolls are done on unroll workers that runs  
 393 asynchronously from the image generation workers. Image warping are separately as a batch offline.

394 However, sampling the environment with the student policy is more challenging because future steps  
 395 depend on the the action from the student policy, which requires the previous render as input. We  
 396 implement remote procedural call (RPC) in our system stack to support this requirement. We present  
 397 the on-policy sampling setup in Figure A18(b). At the start of each flow stack (of seven frames), we  
 398 send out a generation request for the first frame. Once this frame received, it is warped using the  
 399 optical flow while the environment is stepped to provide the subsequent  $T - 1$  frames, where  $T = 7$   
 400 in practice. We then send out another rendering request, and the process repeats according to the  
 401 warping interval  $T$ .

## 402 A.5 Details on The Domain Randomization Baseline

403 We randomize the appearance of the terrain by sampling textures (solid, checker, noisy, gradient),  
 404 material properties (reflectance, shininess, specular), colors by geometry group, similar to [14]. We  
 405 also randomize the lighting parameters of each light in the scene. We adapt the implementation from  
 406 Robosuite [26] to accomplish this. Just as with LucidSim, a new appearance is sampled for every  
 407 frame stack (every 7 frames in practice).

408 We present image samples from the domain randomization baseline on all four domains in Fig-  
 409 ure A19.

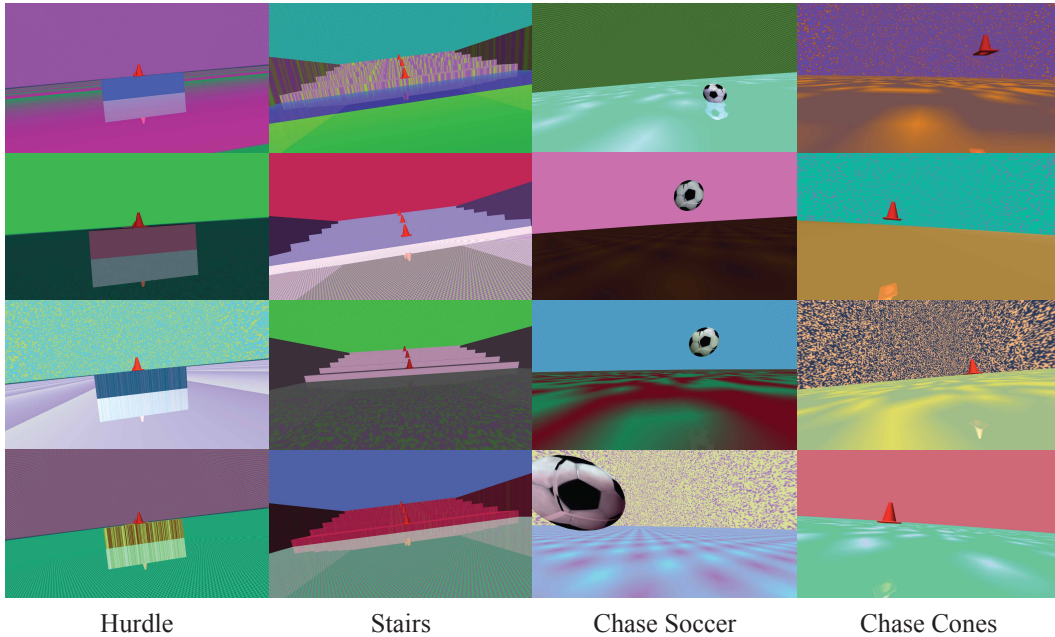


Figure A19: **Domain Randomization Baseline.** Textures, colors, material properties, and lights are randomized every 7 steps. We do not randomize the cone so that the policy can learn to use it as a landmark. This makes it a fair comparison as LucidSim.

## 410 A.6 Training and Model Parameter Tables

411 This section includes Table. 3, 4, 5, 6.

Table 3: Behavior Cloning Parameters

Hyperparameter	Value
max. timesteps per rollout	600
rollouts per DAgger Iteration	1000
learning rate	5e-4
timesteps	70
optimizer	Adam
weight decay	5e-4
momentum	0.9
dropout	0.1

Table 4: Expert Training Parameters

Hyperparameter	Value
value loss coefficient	1.0
clip range	0.2
entropy coef	0.01
learning rate	2e-4
# minibatches per epoch	4
# epochs per rollout	5
# timesteps per rollout	24
discount factor	0.99
GAE parameter	0.95
max grad norm	1.0
optimizer	Adam
joint stiffness	20
joint damping	0.5

Table 5: Expert Randomization Parameters

Term	Min	Max	Unit
friction range	0.6	2.0	-
added mass	0.0	3.0	kg
Body Center of Mass	-0.20	0.20	m
push velocity ( $v_x, v_y$ )	0.0	0.5	m/s
Motor Strength	80	120	%
Forward Velocity Command ( $v_x$ )	0.3	0.8	m/s

Table 6: Expert Reward Terms

Term	Symbol	Scale
parkour velocity tracking [10]	$\min(\langle \mathbf{v}, \hat{\mathbf{d}}_w \rangle, v_{cmd})$	1.5
yaw tracking	$\exp\{- \omega_z - \omega_z^{cmd} \}$	0.5
z velocity	$v_z^2$	-1.0
roll-pitch velocity	$ \omega_{xy} ^2$	-0.05
base orientation	$\mathbb{I}_{flat}  \mathbf{g}_{xy}^{proj} ^2$	-1.0
hip position	$ \mathbf{q}_{hip} - \mathbf{q}_{hip}^0 ^2$	-0.5
collision	$\mathbb{I}_{collision}$	-10.0
action rate	$ \mathbf{a}_t - \mathbf{a}_{t-1} ^2$	-0.1
joint accelerations	$ \ddot{\mathbf{q}} ^2$	-2.5e-7
delta joint torques	$ \boldsymbol{\tau}_t - \boldsymbol{\tau}_{t-1} ^2$	-1.0e-7
joint torques	$ \boldsymbol{\tau} ^2$	-1e-05
joint error	$ \mathbf{q} - \mathbf{q}^0 ^2$	-0.04
foot vertical contact [10]	$\sum_i \mathbb{I}_{vertical\ contact}^{(i)}$	-1.0
foot clearance [10]	$\sum_i \mathbb{I}_{edge\ contact}^{(i)}$	-1.0

## 412 A.7 Real-to-Sim Evaluation Environments

413 Figure A20 provides an overview of our process for constructing the evaluation environments. For  
414 each task, we select a few scenes that differ in appearance (e.g. red bricks, pavement, grass, indoors).  
415 We report the results from three different scenes on each task in Tables 1 and 2. We capture  $\approx 500$   
416 images for each scene, and extract the resulting collision mesh from Polycam. For appearance, we  
417 run COLMAP [27, 28] to obtain camera pose estimates, and reconstruct the scene using 3D Gaussian  
418 Splatting (3DGS) [29, 30, 31].



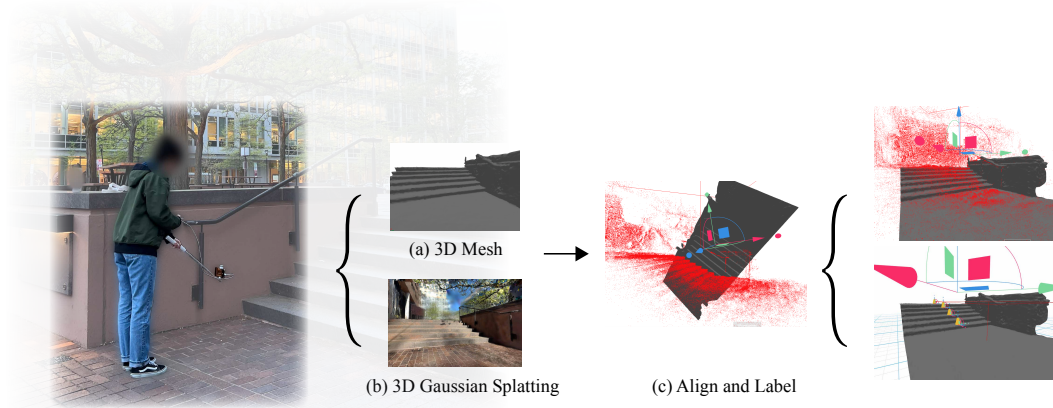


Figure A20: **Constructing Benchmark Environments** The 3D mesh (a) and 3D Gaussian Splat (b) are initially unaligned. (c) we manually scale and align them, and add markers for orange cones that appear in the evaluation environment.

419 We use our custom viewer to align the collision mesh with the gaussian splat. For the hurdle and  
 420 stairs scenes, we manually label 3-5 waypoints along the course that appear as orange traffic cones.  
 421 We use the collision mesh as the terrain, and the 3DGS render as visual observation to the robot.  
 422 For objects that are not present in the initial scan (i.e., soccer ball, traffic cones), we apply the mask  
 423 rendered by the physics engine to insert them into the robot's ego view.