Supplementary Material for "Trace and Pace: Controllable Pedestrian Animation via Guided Trajectory Diffusion"

Davis Rempe*,[1,2] Zhengyi Luo*,[1,3] Xue Bin Peng[1,4] Ye Yuan[1] Kris Kitani[3]
Karsten Kreis[1] Sanja Fidler[1,5,6] Or Litany[1]

[1]NVIDIA [2]Stanford University [3]Carnegie Mellon University [4]Simon Fraser University
[5]University of Toronto [6]Vector Institute

A. Introduction

In this document, we include additional details and results omitted from the main paper due to page limits. Sec. B and Sec. C cover details of the TRACE and PACER models, respectively. Sec. D provides additional details of the experiments presented in the main paper, while Sec. E gives additional results to supplement those in the main paper. Sec. F discusses limitations and future work in more detail.

Extensive video results are included **on the provided `project page`**. We highly encourage readers to view them to better understand our method's capabilities.

B. TRACE Details

In this section, we provide details on the **TRA**jectory Diffusion Model for **C**ontrollable P**E**destrians (TRACE) presented in Sec 3.1 of the main paper.

B.1. Model Details

B.1.1 Denoising-Diffusion Formulation

Input Representations. In practice, the history trajectories of the ego pedestrian $\mathbf{x}^{\text{ego}} = [\mathbf{s}_{t-T_p} \quad \dots \quad \mathbf{s}_t]$ and N neighboring pedestrians $X^{\text{neigh}} = \{\mathbf{x}^{\text{i}}\}_{i=1}^{N}$ given as input to the diffusion model include more than just positions, heading, and speed. In particular, each past state is

$$\mathbf{s} = [x \quad y \quad h_x \quad h_y \quad v \quad l \quad w \quad p] \in \mathbb{R}^8$$

where (x, y) is the 2D position, (h_x, h_y) is the 2D heading vector computed from the heading angle θ, v is the speed, (l, w) is the 2D bounding box dimensions of the person, and $p \in \{0, 1\}$ indicates whether the person is present (visible) at that timestep or not (*e.g.*, due to occlusions in real-world data). If a person is not visible at some step, *i.e.*, $p = 0$, then the full state vector is zeroed out before being given to the diffusion model. All trajectories are transformed into the local frame of the ego pedestrian at the current time step.

The rasterized map input $\mathcal{M} \in \mathbb{R}^{H \times W \times C}$ is in bird's-eye view, and is cropped around the ego pedestrian and transformed into their local frame. For all experiments $H = W = 224$ px at a resolution of 12 px/m. The map is cropped such that 14 m to the front, left, and right of the ego are visible, and $\sim$4.6 m behind. Each channel of $\mathcal{M}$ is a binary map with 1 indicating the presence of some semantic property. For example, in the *ORCA* dataset, there are only two layers – one for walkable area and one for obstacles. In nuScenes, there are seven layers representing lane, road segment, drivable area, road divider, lane divider, crosswalk, and sidewalk. Notice that the map for TRACE does not contain fine-grained height information as in the map for PACER. As such, TRACE is in charge of high-level obstacle avoidance while PACER factors in both obstacles and terrain.

Denoising with Dynamics. As discussed in the main paper, during denoising the future state trajectory is always a result of actions, *i.e.* diffusion/denoising are on $\boldsymbol{\tau}_a$, similar to [21]. In detail, given an input noisy action sequence $\boldsymbol{\tau}_a^k$ the denoising process is as follows: (**1**) compute the input state sequence $\boldsymbol{\tau}_s^k = f(\mathbf{s}_t, \boldsymbol{\tau}_a^k)$ using the dynamics model f, (**2**) pass the full input trajectory $\boldsymbol{\tau}^k = [\boldsymbol{\tau}_s^k; \boldsymbol{\tau}_a^k]$ to the denoising model to predict the clean action trajectory $\hat{\boldsymbol{\tau}}_a^0$, (**3**) compute the output state trajectory $\hat{\boldsymbol{\tau}}_s^0 = f(\mathbf{s}_t, \hat{\boldsymbol{\tau}}_a^0)$, (**4**) if training, compute the loss in Eqn 3 of the main paper on the full output clean trajectory $\hat{\boldsymbol{\tau}}^0 = [\hat{\boldsymbol{\tau}}_s^0; \hat{\boldsymbol{\tau}}_a^0]$.

We use a unicycle dynamics model for f [1]. Though humans are in theory more agile than the unicycle model, we find it regularizes predictions to be generally smooth, which is how pedestrians usually move and is amenable to being followed by an animation model. Since our model requires actions as input, we compute these from the state-only input data through a simple inverse dynamics procedure.

Parameterization. At each denoising step k, TRACE must predict the mean of the distribution used to sample the

*equal contribution

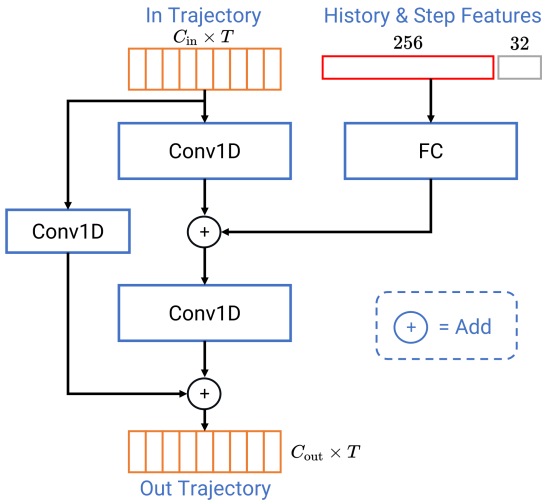

Figure 1. Architecture of a single layer of denoising 1D U-Net.

slightly less noisy trajectory for step $k-1$:

$$p_\phi(\boldsymbol{\tau}^{k-1} \mid \boldsymbol{\tau}^k, C) := \mathcal{N}(\boldsymbol{\tau}^{k-1}; \boldsymbol{\mu}_\phi(\boldsymbol{\tau}^k, k, C), \boldsymbol{\Sigma}_k). \quad (1)$$

There are three common ways to parameterize this prediction (we recommend [9] for a full background on these formulations): (1) directly output $\boldsymbol{\mu}$ from the network, (2) output the denoised clean trajectory $\boldsymbol{\tau}^0$, or (3) output the noise $\boldsymbol{\epsilon}$ used to corrupt the clean trajectory. TRACE uses (2), but the formulations are equivalent. In particular, we can compute $\boldsymbol{\mu}$ from $\boldsymbol{\tau}^k$ and $\boldsymbol{\tau}^0$ using

$$\boldsymbol{\mu}(\boldsymbol{\tau}^0, \boldsymbol{\tau}^k) := \frac{\sqrt{\bar{\alpha}_{k-1}}\beta_k}{1-\bar{\alpha}_k}\boldsymbol{\tau}^0 + \frac{\sqrt{\alpha_k}\,(1-\bar{\alpha}_{k-1})}{1-\bar{\alpha}_k}\boldsymbol{\tau}^k \quad (2)$$

where β_k is the variance from the schedule (we follow [4, 9] and use a cosine schedule), $\alpha_k := 1-\beta_k$, and $\bar{\alpha}_k := \prod_{j=0}^{k} \alpha_j$. Therefore, we can plug the output from TRACE $\hat{\boldsymbol{\tau}}^0$ along with the noisy input $\boldsymbol{\tau}^k$ into Eq. (2) to get the desired next step mean $\boldsymbol{\mu}_\phi$. We can also use the fact that $\boldsymbol{\tau}^0$ is corrupted by

$$\boldsymbol{\tau}^k = \sqrt{\bar{\alpha}_k}\boldsymbol{\tau}^0 + \sqrt{1-\bar{\alpha}_k}\boldsymbol{\epsilon} \quad (3)$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ to compute $\boldsymbol{\epsilon}$ from the output of TRACE:

$$\boldsymbol{\epsilon} = \frac{\boldsymbol{\tau}^k - \sqrt{\bar{\alpha}_k}\boldsymbol{\tau}^0}{\sqrt{1-\bar{\alpha}_k}}. \quad (4)$$

This allows the use of the classifier-free sampling strategy defined in Eqn 4 of the main paper, which requires mixing $\boldsymbol{\epsilon}$ outputs from the conditional and unconditional models.

B.1.2 Architecture

The denoising architecture is shown in Fig. 2 of the main paper. At each denoising step k, the step index is processed with a positional embedding followed by a small MLP that gives a 32-dim feature. The map feature extractor uses the convolutional backbone of ResNet-18 [3] as the encoder followed by a 2D U-Net [14] decoder that leverages skip connections to layers in the encoder. For all experiments, the output feature grid $\boldsymbol{\Psi}$ is then $56 \times 56 \times 32$.

The ego $\mathbf{x}^{\text{ego}}$ and neighbor X^{neigh} history encoders operate on past trajectories $\in \mathbb{R}^{T_p \times 8}$ that are flattened to be a single input vector. The ego and all neighbor trajectories are processed by an MLP with 4 hidden layers giving a feature vector of size 128. A different MLP is learned for ego and neighbor trajectories (*i.e.* all neighbors are processed by the same MLP, which is different from the ego MLP). Neighbor trajectory features are max-pooled to get a single *interaction* feature. The resulting ego and interaction features are finally jointly processed by another MLP with 4 hidden layers, giving a 256-dim feature summarizing the past trajectory context.

Note that the processing of input conditioning described thus far is only necessary to do *once* before starting the denoising process. Only the denoising 1D U-Net needs to be run at every step. At step k of denoising, the 2D position at each timestep $t+i$ of the current noisy input trajectory $\boldsymbol{\tau}^k$ is queried in the map feature grid to obtain a feature $\mathbf{g}_{t+i} = \boldsymbol{\Psi}(x_{t+i}, y_{t+i}) \in \mathbb{R}^{32}$. This query is done through bilinear interpolation of map features at the corresponding point. Over all timesteps, these form a feature trajectory $\mathbf{G} = [\mathbf{g}_{t+1} \dots \mathbf{g}_{t+T_f}]$ that is concatenated along the channel dimension with $\boldsymbol{\tau}^k \in \mathbb{R}^{T_f \times 6}$ (containing both actions and states) to get the full trajectory input to the denoising U-Net $[\boldsymbol{\tau}^k; \mathbf{G}] \in \mathbb{R}^{T_f \times 38}$. Each layer of the U-Net also receives the concatenation of the past trajectory context and denoising step feature.

The architecture of a single U-Net layer is shown in Fig. 1. The input trajectory at each layer is first processed by a 1D convolution. The input trajectory history and step index feature are projected to the same feature size, broadcast over the temporal dimension, and then added to the intermediate trajectory features. Another convolution is performed before adding to the input trajectory in a residual fashion. In the encoding part of the U-Net shown in Fig. 2 of the main paper, a $2\times$ downsampling over the temporal dimension is performed between layers, while a $2\times$ upsampling is done in the decoding part. The encoder is three layers with output channels being 64, 128, and 256-dim.

B.1.3 Training Details

TRACE uses $K = 100$ denoising steps in both training and testing. Training uses a fixed learning rate of 2e-4 with the Adam optimizer [5] and runs for 40k iterations. TRACE trains on a 32 GB NVIDIA V100 GPU and takes $\sim$2 days on the ORCA dataset and $\sim$3 days on the mixed

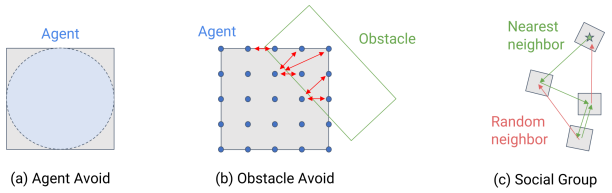

Figure 2. Illustrations of various guidance objectives. See text for details.

nuScenes+ETH/UCY data.

During training, the neighbor history and map conditioning are randomly dropped out with a 10% probability. Note that these are dropped independently and that the ego history is never dropped. In practice, to drop map conditioning, all pixels are filled with a 0.5 value; this means that the model is aware that it "does not know" about the map context, it *is not* simply fed an empty map with no obstacles (all zeros). To drop neighbor conditioning, the neighbor history feature is zeroed out. The same mechanism is used to train on "mixed" data with varying annotations, *e.g.*, some data samples have no maps. In this case, a map is still given to the model but filled with 0.5 value pixels.

B.2. Guidance Details

B.2.1 Scene-Level Guidance

Some guidance objectives are based on multi-agent interactions, *e.g.*, agent collision avoidance and social groups. In this case, we assume that all pedestrians in a scene can be denoised simultaneously in a batched fashion. At each denoising step, the loss function is evaluated at the current trajectory prediction of all pedestrians and gradients are propagated back to each one for guidance. This can be seen as sampling a scene-level future rather than a single agent future. If we want to sample M possible scene futures, we can draw M samples from each agent and assume that the mth sample from each agent corresponds to the same scene sample. In other words, we compute the scene-level guidance by considering only the mth sample from each agent.

This multi-agent guidance slightly complicates the *filtering* procedure described in Sec. 4 of the main paper whereby trajectory samples are strategically chosen to minimize the guidance loss. In the case of a multi-agent objective, the trajectory that minimizes the guidance loss for one agent may not be globally optimal, so it is undesirable to filter the agents independently. We instead do filtering at the scene level, similar to how guidance is computed: we compute the summed guidance loss across the mth sample from all agents and choose the *scene-level sample* that minimizes this aggregate loss.

B.2.2 Guidance Objectives

Next, we describe the different test-time guidance objectives (losses) that we have implemented for TRACE. Objectives operate on the future state trajectory $\boldsymbol{\tau}_s$ that starts at timestep $t+1$ and contains state $\mathbf{s}_j$ at time j.

Agent Avoid and Social Distance. We use the same agent collision penalty as in TrafficSim [17] and STRIVE [13] which approximates each agent with disks to efficiently and differentiably compute a collision loss. For pedestrians, it is sufficient to use a single disk for each agent (see Fig. 2(a)). With this approximation, collision detection is fast and the collision loss is computed based on the extent of disk overlap between agents. It is easy to artificially inflate the size of the disk in order to implement a desired social distance between pedestrians. Note that this is a multi-agent objective, so guidance is enforced at the scene level, as previously discussed.

Obstacle Avoid. We extend the differentiable environment collision penalty introduced in STRIVE [13] to more robustly handle collision avoidance and provide more useful gradients. The core idea is illustrated in Fig. 2(b); for each timestep where the pedestrian's bounding box is overlapping with an obstacle, we query a grid of points on the agent (in our experiments, this is 10×10) and define a loss with respect to points that are embedded in the obstacle. For each embedded point, we compute the minimum distance d_{min} to a non-embedded point on the agent and define the loss at that point as $\mathcal{L} = 1 - (d_{\text{min}}/b)$ where b is the bounding box diagonal of the agent. Summing the loss at all embedded points gives the total loss.

A subtle, but very important, implementation detail here is that the embedded points must be detached (*i.e.* stop grad) before computing the loss. Intuitively, embedded points are treated as points on the obstacle, *not* the agent. So when the loss is computed with respect to these points, it gives a meaningful gradient back to the non-embedded agent points which propagates back to the agent position and heading.

Local Waypoint at Specific Time. This loss encourages an agent to be at a specific 2D goal waypoint $\mathbf{p}_g = (x, y)$ at a specific time step j that falls *within* the planning horizon T_f of the model. It simply encourages the trajectory to be at that location at the timestep with the loss $\mathcal{L} = ||\mathbf{s}_j - \mathbf{p}_g||_2$.

Local Waypoint at Any Time. This loss encourages an agent to be at a specific 2D goal waypoint $\mathbf{p}_g = (x, y)$ at *any* timestep that is *within* the planning horizon T_f of the model. The loss is defined as

$$\mathcal{L} = \sum_{j=t+1}^{t+T_f} \delta_j \cdot ||\mathbf{s}_j - \mathbf{p}_g||_2^2 \tag{5}$$

$$\delta_j = \frac{\exp(-||\mathbf{s}_j - \mathbf{p}_g||)}{\sum_j \exp(-||\mathbf{s}_j - \mathbf{p}_g||)}. \tag{6}$$

Intuitively, it tries to minimize the distance from all points in the trajectory to the target location, but each timestep is weighted by δ_j which is the *softmin* over the distances of each step from the waypoint. The δ_j form a distribution over trajectory timesteps where states close to the waypoint will have higher probability and therefore be weighted more in the loss.

Global Waypoint at Specific Time. This loss encourages an agent to be at a specific 2D goal waypoint $\mathbf{p}_g = (x, y)$ at a specific timestep j that falls *beyond* the planning horizon T_f of the model. This is useful during closed-loop operation in which the agent should eventually reach the point, but at the current step t is not within the planning horizon. Intuitively, the loss encourages making enough progress toward the waypoint such that when it becomes in range, we can revert to the Local Waypoint loss and hit the target exactly at the desired time.

To do this, we would like to ensure that the future trajectory ends in a location such that the pedestrian can travel in a straight line at a "preferred" speed v_{pref} and get to the waypoint on time. Formally, the trajectory should be within a target distance defined as:

$$d_{\text{goal}} = (j - t) \cdot dt \cdot v_{\text{pref}} \tag{7}$$

where dt is the step size of TRACE output (0.1 sec in our experiments). Since the pedestrian may not be able to actually travel a straight line path (*e.g.* in environments with obstacles), we incorporate an *urgency* parameter $u \in [0, 1]$ that encourages getting there earlier and modifies the goal distance as

$$\tilde{d}_{\text{goal}} = d_{\text{goal}} \cdot (1 - u). \tag{8}$$

Then the loss with respect to this target distance is defined as

$$\mathcal{L} = \text{ReLU}(||\mathbf{s}_{T_f} - \mathbf{p}_g||_2 - \tilde{d}_{\text{goal}}) \tag{9}$$

which penalizes the trajectory if the final state is not within the goal distance.

Global Waypoint at Any Time. This loss encourages an agent to be at a specific 2D goal waypoint $\mathbf{p}_g = (x, y)$ at *any* timestep *beyond* the planning horizon T_f of the model. To determine whether the goal waypoint is outside the current horizon, we check if the agent could progress along a straight line to the goal at a preferred speed v_{pref} and reach the waypoint within the planning horizon. If so, the loss reverts to the Local Waypoint loss.

If the waypoint is indeed beyond the planning horizon, the loss attempts to progress according to some urgency $u \in [0, 1]$. To do this, we first compute the maximum distance that could be covered in the current horizon:

$$d_{\text{max}} = T_f \cdot dt \cdot v_{\text{pref}} \tag{10}$$

and use the urgency to get the goal distance we wish to cover over the horizon

$$d_{\text{goal}} = u \cdot d_{\text{max}}. \tag{11}$$

The loss is computed based on how much progress is made over the horizon:

$$\mathcal{L} = \text{ReLU}(d_{\text{goal}} - d_{\text{progress}}) \tag{12}$$

$$d_{\text{progress}} = ||\mathbf{s}_t - \mathbf{p}_g||_2 - ||\mathbf{s}_{t+T_f} - \mathbf{p}_g||_2. \tag{13}$$

Social Groups. This loss encourages groups of agents to travel together. A social group is based on one *leader* pedestrian that is not affected by the social group loss (via detach/stop grad); others in the group will tend to move with the leader. Intuitively, we want each agent in the social group to maintain a specified social distance d_{soc} to the closest agent also in the same social group. Let ψ be a map from one agent index to another, *e.g.* $i = \psi(k)$ means agent k in the social group is mapped to agent i. Then the social group loss for agent i at timestep j in the future trajectory is

$$\mathcal{L} = \left(||\mathbf{s}_j^i - \mathbf{s}_j^{\psi(i)}||_2 - d_{\text{soc}}\right)^2. \tag{14}$$

As shown in Fig. 2, most of the time ψ maps each agent to the closest agent in the group, but with some probability based on a cohesion parameter $c \in [0, 1]$ the mapping will be to a random agent in the group. So with a larger cohesion, agents in the group are all encouraged to be equidistant from each other, while with low cohesion agents will not closely follow the leader and connected components in the social group graph may break off and ignore others.

Learned Value Function. Given a learned value function $V(\boldsymbol{\tau}_s)$ that predicts the future rewards over a given trajectory, in general this guidance loss is simply $\mathcal{L} = \exp(-V(\boldsymbol{\tau}_s))$. When TRACE is used with PACER, the value function is $V(v_t|\boldsymbol{\tau}_s, \boldsymbol{h}_t, \boldsymbol{o}_t, \boldsymbol{\beta})$; it takes in the current humanoid state $\boldsymbol{h}_t$, environmental feature $\boldsymbol{o}_t$, and humanoid shape $\boldsymbol{\beta}$, which are fixed throughout denoising.

C. PACER Details

In this section, we give details on the **P**edestrian **A**nimation **C**ontroll**ER** (PACER) presented in Sec 3.2.

C.1. Implementation Details

Humanoid State. The state $\boldsymbol{h}_t$ holds joint positions $\boldsymbol{j^t} \in \mathbb{R}^{24\times3}$, rotations $\boldsymbol{q^t} \in \mathbb{R}^{24\times6}$, linear velocities $\boldsymbol{v^t} \in \mathbb{R}^{24\times3}$, and angular velocities $\boldsymbol{\omega^t} \in \mathbb{R}^{24\times3}$ all normalized w.r.t. the agent's heading and root position. The rotation is represented in the 6-degree-of-freedom rotation representation. SMPL has 24 body joints with the root (pelvis) as the first joint, which is not actuated, resulting in an action dimension of $\boldsymbol{a}_t \in \mathbb{R}^{23\times3}$. No special root forces/torques are used.

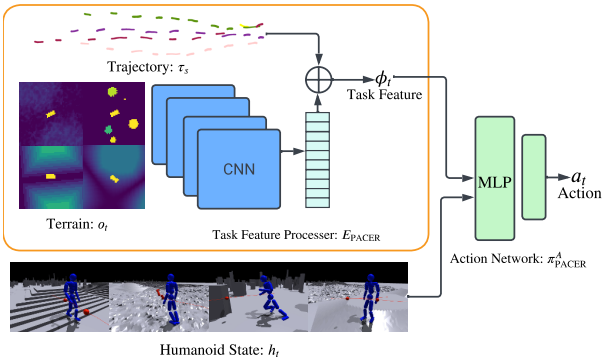

Figure 3. The PACER policy network π_{PACER} consists of a task feature processer $E_{\text{PACER}}(\boldsymbol{\phi}_t|\boldsymbol{o}_t, \boldsymbol{\tau}_s)$ and an action policy network $\pi^{\text{A}}_{\text{PACER}}(\phi_t, \boldsymbol{h}_t)$.

Network Architecture. As mentioned in the main paper, the environmental feature $\boldsymbol{o}_t$ is a rasterized local height and velocity map of size $\boldsymbol{o}_t \in R^{64\times64\times3}$. The first channel is the terrain height map relative to the current humanoid root height, and the second & third channels are the 2D linear velocities (x and y directions) in the egocentric coordinate system. The map corresponds to a 4m $\times$ 4m square area centered at the humanoid root, sampled on an evenly spaced grid. The trajectory $\boldsymbol{\tau}_s \in R^{10\times2}$ consists of the 2D waypoints for the next 5 seconds sampled at 0.5 s intervals.

The architecture of π_{PACER} can be found in Fig. 3. Due to the high dimensionality of the environmental features $\boldsymbol{o}_t$, we separate the policy network into a task feature processer $E_{\text{PACER}}(\boldsymbol{\phi}_t|\boldsymbol{o}_t, \boldsymbol{\tau}_s)$ and an action network $\pi^A_{\text{PACER}}(\boldsymbol{a}_t|\boldsymbol{\phi}_t, \boldsymbol{h}_t, \boldsymbol{\beta})$. The task feature processor transforms task-related features, such as environmental features $\boldsymbol{o}_t$ and trajectory $\boldsymbol{\tau}_s$ into a latent vector $\boldsymbol{\phi}_t \in \mathbb{R}^{256}$. Then, π^A_{PACER} computes the action $\boldsymbol{a}_t$ based on the humanoid state $\boldsymbol{h}_t$, body shape $\boldsymbol{\beta}$, and $\boldsymbol{\phi}_t$. The overall policy network is then $\pi_{\text{PACER}}(\boldsymbol{a}_t|\boldsymbol{o}_t, \boldsymbol{h}_t, \boldsymbol{\beta}, \boldsymbol{\tau}_s) \triangleq \pi^A_{\text{PACER}}(E_{\text{PACER}}(\boldsymbol{o}_t, \boldsymbol{\tau}_s), \boldsymbol{h}_t, \boldsymbol{\beta})$. E_{PACER} is a four-level convolutional neural network with a stride of 2, 16 filters, and a kernel size of 4. π^A_{PACER} is a standard MLP with ReLU activations. It has two layers, each with 2048 and 1024 units. The policy maps to the Gaussian distribution over actions $\pi_{\text{PACER}}(\boldsymbol{a}_t|\boldsymbol{o}_t, \boldsymbol{h}_t, \boldsymbol{\beta}, \boldsymbol{\tau}_s) = \mathcal{N}(\mu(\boldsymbol{o}_t, \boldsymbol{h}_t, \boldsymbol{\beta}, \boldsymbol{\tau}_s), \Sigma)$ with a fixed covariance matrix Σ. Each action vector $\boldsymbol{a}_t \in \mathbb{R}^{23\times3}$ corresponds to the PD targets for the 23 actuated joints on the SMPL human body. The discriminator $\boldsymbol{D}(\boldsymbol{h}_{t-10:t}, \boldsymbol{a}_t)$ shares the same architecture as π^A_{PACER}, while the value function $V(v_t|\boldsymbol{o}_t, \boldsymbol{h}_t, \boldsymbol{\beta}, \boldsymbol{\tau}_s)$ shares the same architecture as the policy π_{PACER}.

C.2. Reward and Loss

Reward. Following AMP [12], our policy π_{PACER} is learned through goal-conditioned RL where the reward contains a task reward $r^{\boldsymbol{\tau}}_t$, a style reward r^{amp}_t, and an energy penalty r^{energy}_t. The style reward is computed by the discriminator $\boldsymbol{D}(\boldsymbol{h}_{t-10:t}, \boldsymbol{a}_t)$ based on 10 steps of aggregated humanoid state. We use the same set of observations, loss formulation, and gradient penalty as in AMP [12] to train our discriminator. Task reward $r^{\boldsymbol{\tau}}_t$ is a trajectory-following reward that measures how far away the humanoid's center $\boldsymbol{c}_t$ on the xy plane is from the 2D trajectory: $\exp(-2\times\|\boldsymbol{c}_t - \boldsymbol{\tau}_t\|^2)$. The energy penalty is expressed as $-0.0005 \cdot \sum_{j\in \text{ joints}} |\boldsymbol{\mu_j}\dot{\boldsymbol{q}}_{\boldsymbol{j}}|^2$ where $\boldsymbol{\mu_j}$ and $\dot{\boldsymbol{q}}_{\boldsymbol{j}}$ correspond to the joint torque and the joint angular velocity, respectively.

Motion Symmetry Loss. During our experiments, we noticed that asymmetric gaits emerge as training progresses. It manifests itself as "limping" where the humanoid produces asymmetric motion, especially at a lower speed. This could be due to the small temporal window used in AMP (10 frames), which is not sufficient to generate symmetrical motion. Compared to AMP, we use a humanoid with more than double the degrees of freedom (69 vs 28), and the complexity of the control problem grows exponentially. This could also contribute to limping behavior as it becomes harder for the discriminator to discern asymmetric gaits. Thus, we utilize the motion symmetry loss proposed in [19] to ensure symmetric gaits. Specifically, we first design two functions Φ_s and Φ_a that can mirror the humanoid state and action along the character's sagittal plane. Symmetry is then enforced by ensuring that the mirrored states lead to mirrored actions:

$$L_{\text{sym}}(\theta) = \|\pi_{\text{PACER}}(\boldsymbol{h}_t, \boldsymbol{o}_t, \boldsymbol{\beta}, \boldsymbol{\tau}_s) - \Phi_a(\pi_{\text{PACER}}(\Phi_s(\boldsymbol{h}_t, \boldsymbol{o}_t, \boldsymbol{\beta}, \boldsymbol{\tau}_s)))\|^2, \tag{15}$$

Notice that the motion symmetry loss is not a reward and is directly defined on the policy output. As the loss can be computed in an end-to-end differentiable fashion, we directly optimize this loss through SGD.

C.3. Training

Our training procedures closely follow AMP [12], with notable distinctions in the motion dataset, initialization, termination condition, terrain, and humanoids used. Training takes $\sim$3 days to converge on one NVIDIA RTX 3090.

Dataset. We use a small subset of motion sequences from the AMASS dataset [7] to train our humanoid controller. Specifically, we hand-picked $\sim$200 locomotion sequences consisting of walking and turning at various speeds, as well as walking up and down stairs. These motions form the reference motion database and provide our AMP Discriminator $D(\boldsymbol{h}_t, \boldsymbol{a}_t)$ with "real" samples.

Initialization. To initialize our humanoids during training, we use reference state initialization [11] to randomly sample a body state $\boldsymbol{h}_0$. The initial root positions are randomly sampled from a "walkable map" that corresponds to all locations that can be used as a valid starting point (*e.g.* not on top of obstacles). As we use NVIDIA's Isaac Gym, we

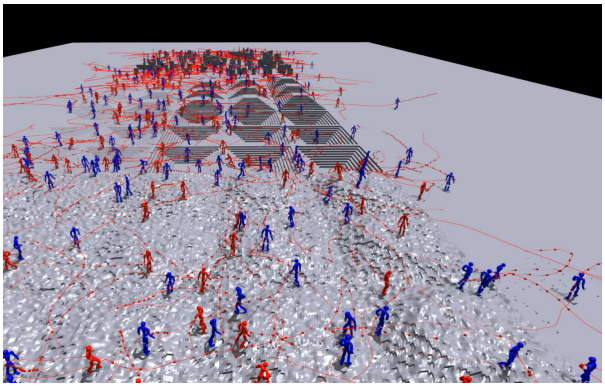

Figure 4. During training, 2048 humanoids are simulated in parallel on our synthetic terrain.

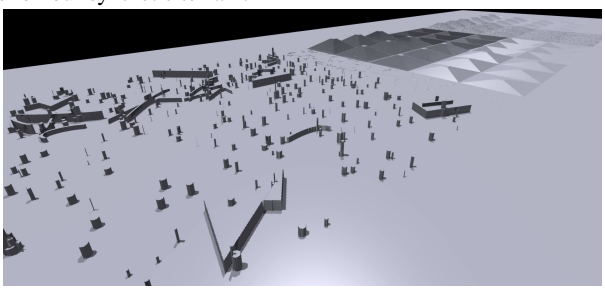

Figure 5. Synthetic terrains used for training PACER. From left to right: obstacles, discrete terrains, stairs (up), stairs (down), uneven terrains, and slopes.

create 2048 humanoids that are simulated simultaneously in parallel during training: see Fig. 4.

Random terrain, trajectory, and body shape sampling. To learn a model that can traverse diverse types of terrain that pedestrians may encounter in real life, we train our trajectory-following controller on a variety of different environments. Specifically, we follow ANYmal [15] to create terrain curricula with varying difficulties to train our agents. Six types of terrain are created: slopes, uneven terrain, stairs (down), stairs (up), discrete, and obstacles. The terrains follow a gradual increase in difficulty, where we vary the slope angle, terrain unevenness, slope angle of stairs, and obstacle density, as shown in Fig. 5.

Trajectory samples for training are generated procedurally: τ_s is randomly sampled by generating velocities and turn angles. We limit the velocity to be between [0, 3] m/s and the acceleration to be between [0, 2] m/s^2.

To train with different body shapes, we extract all unique human body shapes from the AMASS dataset, which amount to 476 shapes (273 male and 200 female). We randomly sample (with replacement) 2048 body shapes to create humanoids at the beginning of the training process. To create reference humanoid states $\hat{\boldsymbol{h}}_{\boldsymbol{t}}$ for the discriminator, we perform forward kinematics based on the sampled pose and the humanoids' kinematic tree. At the beginning of every 250 episodes, we randomly sample a new batch of pose sequences from the motion dataset and create new reference humanoid states. In this way, we obtain reference states of diverse body types and motions.

Termination condition. To speed up training, we employ early termination [11] and terminate the episode if there is a collision force greater than 50 N on the humanoid body, with either the scene or other humanoids. The ankles and foot joints are exceptions to this rule, as they are in contact with the ground. This condition also serves as a fall detection mechanism, as falling will involve a collision force from the ground. Notice that this termination condition encourages the humanoid to avoid obstacles and other humanoids since a collision will trigger an early termination.

D. Experimental Details

In this section, we include details of the experiments presented in Sec 4 of the main paper.

D.1. Dataset Details

The *ORCA* dataset contains two distinct subsets, *ORCA-Maps* and *ORCA-Interact*. ORCA-Maps is generated with up to 10 pedestrians and 20 obstacles in each scene. This contains many obstacle interactions, but fewer agent-agent interactions. ORCA-Interact has up to 20 pedestrians, but no obstacles, and therefore has no map annotations. Each data subset contains 1000 scenes that are $10s$ long, and we split them 0.8/0.1/0.1 into train/val/test splits. The map annotations in the ORCA dataset contain two channels, one representing the walkable area and one representing obstacles. The bounding box diameter for every agent is fixed to $0.8m$.

In nuScenes [2], there are seven map layers representing the lane, road segment, drivable area, road divider, lane divider, crosswalk, and sidewalk. The bounding box diameters are given by the dataset. We follow the official trajectory forecasting benchmark for scenes in the train/val/test splits. For ETH/UCY [6, 10], we use the official training splits of each contained dataset for training.

All trajectory data in all datasets is re-sampled to 10 Hz for training and evaluation of TRACE.

D.2. Guidance Metrics

Here, we define the *Guidance Error* for each of the objectives evaluated in Sec 4.1 and 4.2 of the main paper.

Obstacle Avoid. This is the obstacle collision rate as defined below.

Agent Avoid. This is the agent collision rate as defined below.

Waypoint and Perturbed Waypoint. If the objective is to reach a waypoint at a specific timestep, this is simply the distance of the agent from the target waypoint at that specified timestep (in meters). Otherwise, if the objective is to reach at *any* timestep, the error is the minimum distance

between the agent and the goal waypoint across the entire trajectory.

Social Groups. For each pedestrian in the group, we measure the mean absolute difference between the specified social distance d_{soc} (see Sec. B.2.2) and the distance to the closest neighbor in the same social group.

Multi-Objective. For the multi-objective guidance presented in Tab 1 of the main paper (Waypoint + Avoidance), the reported guidance error is the waypoint error since obstacle and agent collision rates are already reported in other columns.

D.3. Other Metrics

Next, we describe in more detail the metrics used to evaluate the standalone TRACE model in Sec 4.1 and 4.2 of the main paper.

Obstacle Collision Rate. Measures the average fraction of time that an agent (as represented by a single disk) is overlapping with an obstacle on the map within a rollout. Note that a disk is used to represent each agent because this is how they are represented in the ORCA simulator, so the ground truth data contains no collisions using this representation.

Agent Collision Rate. Measures the average fraction of agents involved in an agent-agent collision within each scene rollout. This again uses the disk representation of each agent.

Realism (EMD). Compares the histogram of statistics over the entire test set between generated and ground truth trajectories. This is done for velocity, longitudinal acceleration, and lateral acceleration. In particular, the statistics at each timestep of the test set are aggregated together into a histogram. The histogram is then normalized such that it sums to 1. The earth mover's distance (EMD) between the ground truth and generated histograms is then computed[1] and reported. Note that this metric is computed wrt *the dataset being evaluated on*. For example, in Sec 4.1 of the main paper, even though TRACE is trained on both ORCA-Maps and ORCA-Interact, the metric is only computed for ORCA-Maps since this is the test data.

Realism (Mean). Measures the average longitudinal and lateral acceleration within a generated trajectory in m/s^2. Similar metrics are commonly used in the vehicle planning literature [18] as a proxy for how *comfortable* a ride is. In our case of pedestrian motion, this is still relevant since people tend to move in smooth motions without sudden changes in speed or direction.

[1] using pyemd

D.4. VAE Baseline Details

We adapt a conditional VAE model similar to the idea of STRIVE [13] for controlling trajectories through latent space optimization. We adapt the VAE design to our setting.

Architecture. The architecture operates in an agent-centric manner as in TRACE. It is a fairly standard conditional VAE (CVAE) where the conditioning (map and past trajectories) is processed into a single conditioning vector $\mathbf{c}$ that is given to the decoder. At training time, the decoder also takes in a latent vector $\mathbf{z}$ from the encoder (posterior), while at test time the latent vector is sampled from the prior $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$. To make the methods comparable, the map conditioning is encoded with the same ResNet-18 backbone that TRACE uses; ego and neighbor past trajectories also use the same architecture as TRACE. Since the model is agent-centric rather than scene-centric, the decoder D is simply an MLP that maps the conditioning and sampled latent to an output action trajectory (instead of a graph network as in STRIVE) as $\boldsymbol{\tau}_a = D(\mathbf{z}, \mathbf{c})$. In all experiments, the latent dimension is 64, while the conditioning feature vector is 256-dim.

Training. Training is done using a standard VAE loss consisting of a reconstruction and a KL divergence term. The KL term is weighted by 1e-4. The model is trained with the same batch size as TRACE (400) and for the same number of iterations (40k) with a learning rate of 2e-4.

Test-Time Optimization. The idea of test-time optimization is to search for a latent vector that is likely under the prior (*i.e.* represents a plausible future trajectory) but also meets the desired guidance objective. Concretely, the optimization objective is

$$\min_{z} \alpha \mathcal{J}(D(\mathbf{z}, \mathbf{c})) - \log p(\mathbf{z}) \tag{16}$$

where $\mathcal{J}$ is a guidance loss as described in the main paper and α balances the prior term with the guidance loss. Optimization is performed with Adam [5] using a learning rate of 0.02. For experiments in Sec 4.1 of the main paper, optimization uses 100 iterations (same as the number of diffusion steps K). For Sec 4.2, the iteration budget is increased to 200 to accommodate more difficult out-of-distribution objectives.

Discussion on VAE Comparison. The VAE with test-time optimization is generally a very strong baseline. Given a large enough compute budget, the optimization can usually faithfully meet the desired objective. However, the number of optimization iterations needed to meet an objective can be large; *e.g.* in Sec 4.2 it requires twice the number of diffusion steps, making it slower than TRACE. Moreover, when optimizing for a long time to closely meet objectives, the diversity of optimized samples becomes low as they converge to similar minima (Fig. 4 in the main paper). This is

due to the prior term in Eq. (16), which always drives the trajectory towards the mean.

D.5. Additional Experiment Details

Finally, we include miscellaneous details of the setup for each experiment in Sec. 4 of the main paper.

Augmenting Crowd Simulation (Sec 4.1). For the no guidance rows in Tab 1, we actually run the evaluation three times and report the averaged metrics. This is because when there is no guidance, no filtering is performed, so a random sample is chosen. Running with several random samples gives a more faithful evaluation of performance. In this experiment, TRACE uses $w = 0.0$ for classifier-free sampling. 20 samples are drawn and guided from the model for each pedestrian before filtering. The weighting α for each guide (in Eq. 6 of the main paper) is tuned manually to meet objectives while maintaining realistic trajectories. The *Waypoint* guidance used in Tab 1 is the *Local Waypoint at Any Time* introduced in Sec. B.2.2. The agent avoidance guidance uses an additional social distance buffer of $0.2m$.

Real-world Data Evaluation (Sec 4.2). In this experiment, 10 samples are drawn from the model before filtering and the waypoint guidance is *Global Waypoint at Any Time* since the model operates in a closed loop for longer than the planning horizon. This waypoint guidance uses an urgency of $u = 0.7$ and a preferred speed of $v_{\text{pref}} = 1.25m/s$. The perturbed waypoint objective randomly perturbs the target ground truth waypoint with Gaussian noise with a standard deviation of $2m$. The social group guidance uses $d_{\text{soc}} = 1.5$ and cohesion $c = 0.3$. In each nuScenes scene, social groups are determined heuristically by forming a scene graph where edges are present if two pedestrians are within $3m$ of each other and moving in a similar direction (velocities have a positive dot product): the connected components of this graph with more than one agent form the social groups.

Controllable Pedestrian Animation (Sec 4.3). In this experiment, 10 samples are drawn from the model before filtering. Waypoint guidance uses *Global Waypoint at Specific Time* with the waypoint randomly placed at a reasonable distance ($[7, 12]$ meters in front of the user and up to 5 meters to either side) 9 sec in the future. Agents are initialized in a standing pose with a uniform random initial root velocity in $[1.2, 2.0]m/s$.

E. Supplementary Results

In this section, we include additional experimental results omitted from the main paper due to page limits.

E.1. Qualitative Results

Extensive qualitative video results for both TRACE and PACER are provided **on the supplementary webpage**.

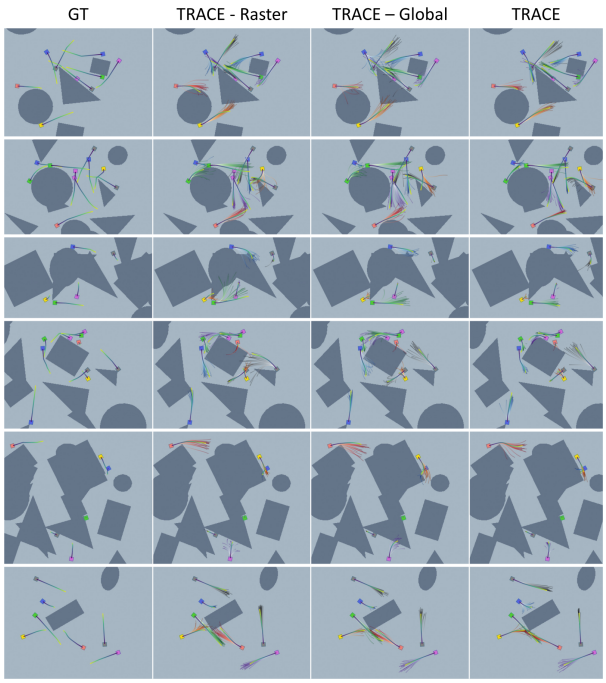

Figure 6. Random sampling with no guidance from different TRACE architecture ablations. Using the learned feature map has apparent benefits in subtle interaction with obstacles. 20 random samples are visualized for each pedestrians with the (arbitrarily) chosen plan in bold.

E.2. TRACE– No Guidance Ablation Study

The focus of our work is on controllability using guidance. However, it is still desired that the model performs well even when guidance is not used. In particular, random samples from the model should be robust (avoid collisions), realistic (similar to the training data distribution), and accurate (capture the ground truth future trajectory). To this end, we evaluate our architecture and training approach compared to baselines and ablations while using no guidance. We evaluate in the open-loop $5s$ rollout setting on ORCA-Maps as used in Sec. 4.1 of the main paper. Two additional metrics common in trajectory forecasting are evaluated: the average displacement error (ADE) and the final displacement error (FDE) [20]. For a trajectory sample defined from timesteps 1 to T, these are defined as $\text{ADE} = \frac{1}{T}\sum_{t=1}^{T} ||\hat{\mathbf{y}}_t - \mathbf{y}_t||_2$ and $\text{FDE} = ||\hat{\mathbf{y}}_T - \mathbf{y}_T||_2$ with $\hat{\mathbf{y}}$ the sample from the model and $\mathbf{y}$ the ground truth.

First, we evaluate the TRACE architecture, which uses a feature grid to condition denoising on the map input. An alternative way to condition trajectory generation is to encode the map into a single ("global") feature vector using a convolutional backbone. This global feature can then be given in the same way as the past trajectory features. The *VAE* baseline and *TRACE-Global* ablation do this using a ResNet-18 backbone. The *TRACE-Raster* ablation is similar to CTG [21], which rasterizes both the map *and* agent

Method	History Input	Map Feature	Accuracy ADE	Accuracy FDE	Collision Rate Obstacle	Collision Rate Agent	Realism (EMD) Vel	Realism (EMD) Lon Acc	Realism (EMD) Lat Acc
VAE [13]	States	Global	0.340	0.774	0.062	0.115	0.041	0.038	0.039
TRACE-Raster	Raster	Global	0.337	0.808	0.052	0.100	0.027	**0.013**	**0.014**
TRACE-Global	States	Global	**0.280**	**0.686**	0.056	**0.094**	**0.022**	**0.013**	0.016
TRACE	States	Grid	0.318	0.757	**0.046**	0.110	0.028	0.020	0.020

Table 1. No guidance evaluation on *ORCA-Maps* dataset. Ablation on architecture design choices.

Train Data	Drop Rate	Accuracy ADE	Accuracy FDE	Collision Rate Obstacle	Collision Rate Agent	Realism (EMD) Vel	Realism (EMD) Lon Acc	Realism (EMD) Lat Acc
ORCA-Maps	10%	0.351	0.819	**0.040**	0.112	0.030	0.023	0.024
Mixed	0%	**0.303**	0.719	0.042	0.123	0.028	**0.019**	**0.020**
Mixed	5%	0.307	**0.712**	**0.040**	**0.108**	**0.024**	0.020	0.023
Mixed	10%	0.318	0.757	0.046	0.110	0.028	0.020	**0.020**

Table 2. No guidance evaluation on *ORCA-Maps* dataset. Ablation on training routine.

histories and encodes them into a single global feature instead of encoding the trajectory states separately. Tab. 1 shows the results comparing these methods. In this experiment, we take 20 samples from each model and evaluate the one that is closest to the ground truth wrt the ADE. We see that using the feature grid map provides the lowest obstacle collision rate while maintaining competitive accuracy and realism. As qualitatively shown in Fig. 6, the use of the feature grid gives local cues to the model to inform subtle obstacle interactions and avoid collisions. Though agent collisions are slightly worse with the grid map, no model does particularly well, and exploring improved agent-agent interactions is an important direction for future work.

In the main paper, we discuss how mixed training data and classifier-free sampling (*i.e*. training with random dropping on conditioning) are important to enable flexibility for guidance. To ensure this training approach does not negatively affect base model performance without guidance, we compare to (1) an ablation that uses the ORCA-Maps data only to train (rather than a mix of ORCA-Maps+ORCA-Interact) and (2) ablations that use varying levels of dropping. Tab. 2 shows results, where again the sample closest to ground truth is evaluated. Interestingly, training with mixed data allows for increased accuracy compared to training only on the ORCA-Maps dataset. Increasing the drop probability past 5% has very little effect on performance and comes with the added benefit of using classifier-free sampling to get flexible guidance at test time.

E.3. TRACE– Effect of Classifier-Free Sampling

Next, we examine how weight w affects model performance when using classifier-free sampling both with and without guidance. First, we analyze the effect when evaluating on the ORCA-Maps dataset with no guidance in the open-loop setting (like Sec 4.1 of the main paper). In this case, we evaluate $w \geq 0$ which increases emphasis on the input conditioning to the model. Quantitative results are shown in Tab. 3: for each value of w, the evaluation is run 3 times with different random samples and the metrics are averaged. For $w \in [0, 1]$, the collision rates and acceleration realism remain similar or slightly improve, which we expect since input conditioning such as the obstacle map is emphasized. For w>1, the guidance tends to be too strong and the trajectory samples are almost deterministic. Though the quantitative difference is not large as w increases, in Fig. 7 we see that increasing does have a considerable qualitative effect.

w	Collision Rate Obstacle	Collision Rate Agent	Realism (EMD) Vel	Realism (EMD) Lon Acc	Realism (EMD) Lat Acc
0.0	0.051	0.131	0.019	0.012	0.014
0.3	0.051	0.130	0.027	0.008	0.010
0.5	0.050	0.132	0.029	0.008	0.009
0.7	0.050	0.132	0.033	0.009	0.008
1.0	0.049	0.130	0.040	0.010	0.009
2.0	0.051	0.132	0.063	0.017	0.015
3.0	0.052	0.138	0.087	0.025	0.022
4.0	0.051	0.145	0.102	0.033	0.028

Table 3. Classifier-free sampling analysis on *ORCA-Maps* dataset with no guidance.

w	Waypoint Error	Realism (Mean) Lon Acc	Realism (Mean) Lat Acc
0.0	1.129	0.233	0.218
-0.3	0.972	0.213	0.199
-0.5	0.802	0.212	0.204
-0.7	0.670	0.240	0.233
-1.0	0.546	0.345	0.348

Table 4. Classifier-free sampling analysis on nuScenes dataset using perturbed waypoint guidance.

Second, we look at results on the nuScenes dataset using perturbed waypoint guidance in the closed-loop setting (the same as in Sec. 4.2 of the main paper). In Sec. 4.2 of the main paper, we saw that using $w < 0$ improves susceptibility to guidance and allows the model to achieve out-of-

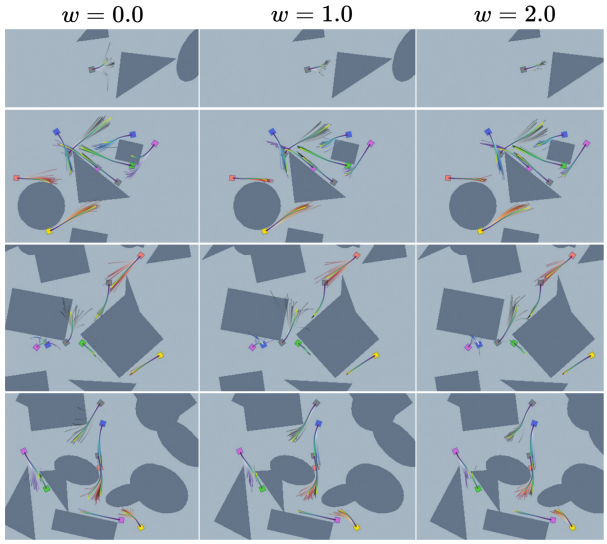

Figure 7. Sampling using increasing classifier-free weights w. 20 samples are visualized for each pedestrian. Larger w tends to emphasize collision avoidance and reduces the variance of the sampled trajectory distribution, especially for pedestrians near obstacles where conditioning has a large effect on motion.

Terrain	Model	Guide	Fail Rate	Traj Follow Error
Flat (Crowd)	Agent Unaware	None	0.252	0.102
	Agent Aware	None	0.087	0.082
	Agent Unaware	Agt Avoid	0.060	**0.067**
	Agent Aware	Agt Avoid	**0.013**	0.071
Random	Body Unaware	None	0.125	0.105
	Body Aware	None	**0.093**	**0.104**
	Body Unaware	Waypoint	**0.103**	**0.102**
	Body Aware	Waypoint	0.107	0.111

Table 5. PACER ablation study while using TRACE as the trajectory planner.

distribution objectives. This is further confirmed in Tab. 4. We see that the smaller the w, the better the waypoint reaching error. However, for $w < -0.5$ the mean accelerations of pedestrians start to deviate more from those observed in the ground truth nuScenes data, as the model is capable of producing more extreme trajectories to reach waypoints.

E.4. PACER– Ablation Study

In this experiment, we demonstrate the importance of multiple design decisions in the PACER model. First, we choose to make the animation controller agent-aware by including neighboring pedestrians in the heightmap given to the model. For comparison, we train a model that is agent *unaware*, *i.e.*, the input height map only contains obstacles. As seen in the top half of Tab. 5, even though TRACE is already agent-aware, having PACER endowed with awareness is highly beneficial. Both with and without agent avoidance guidance on TRACE, the agent-aware

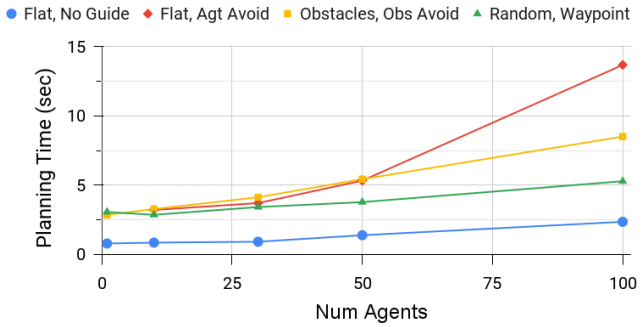

Figure 8. TRACE planning time within the end-to-end pedestrian animation system for varying terrains, guidance, and number of simulated agents.

model greatly improves the collision rate.

Second, we evaluate whether making PACER body-aware is necessary, *i.e.* if it needs to take the parameters of the SMPL body β shape as input, as it is already training in simulation with a variety of body shapes. The bottom half of Tab. 5 shows that while traversing random terrains, body awareness helps to improve the failure rate when no guidance is used. When waypoint guidance is added, performance is essentially unchanged.

As motion is best seen in videos, we also include videos of how the symmetry loss and body shape conditioning affect motion quality; please see the supplementary webpage.

E.5. Runtime Analysis

Fig. 8 shows an analysis of the average runtime for one planning step of TRACE within the end-to-end animation system (on an NVIDIA TITAN RTX). Varying numbers of simulated humanoids are tested using the terrains and guidance introduced in Sec 4.3 of the main paper. With $\leq$50 agents, TRACE planning takes $\leq$5 sec, but becomes more costly with 100 agents, especially using agent avoidance guidance. Since collision avoidance requires pairwise comparisons between many agents, it can be costly.

The standalone PACER model is real time, running at $\sim$30 fps for 1 humanoid and $\sim$25 fps for 100.

F. Discussions and Limitations

TRACE Efficiency. The main limitation of using our system in a real-time setting is the speed of the denoising process. This is a well-known issue with diffusion models, and the community is actively working to address it. For example, recent work on distilling diffusion models [8] could be applied here to greatly speed up sampling.

Multi-Objective Guidance. One challenge with using several objectives simultaneously to guide TRACE is balancing the weight α for each. Though it is not difficult to tune each weight individually, we found that when combined, the guidance strength can be too much depending on

the scene. Intuitively, if two guidance objectives are pushing a trajectory in the same direction (*e.g.* avoiding obstacle collision and going to a waypoint), the combined guidance will have compounded strength that may push the trajectory to diverge off-manifold. Work in image generation has noticed similar effects when using strong guidance, which manifests itself as saturated images. To avoid this, various forms of dynamic clipping during sampling have been introduced [16]. While this makes sense for images that have been normalized in a fixed range, it is not trivial for trajectories and we think this is an interesting problem for future work.

PACER Motions. Though PACER is robust and traverses diverse terrains while driving humanoids with different body shapes, it struggles with large obstacles when there is no way around them. The motion generated at low speed can also be unnatural as our motion database contains few samples where the humanoid is traveling at extremely low speed. Our humanoids also lack motion diversity, since most body types will have similar walking gaits and will not manifest common pedestrian behaviors such as talking on the phone or with each other. More research is needed to improve the quality and diversity of the motion.

References

[1] Principles of robot autonomy i, lecture 1 course notes: Mobile robot kinematics. https://stanfordasl.github.io/aa274a/pdfs/notes/lecture1.pdf. Accessed: 2022-11-15. 1

[2] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020. 6

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2

[4] Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *International Conference on Machine Learning (ICML)*, 2022. 2

[5] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2, 7

[6] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. Crowds by example. In *Computer graphics forum*, volume 26, pages 655–664. Wiley Online Library, 2007. 6

[7] Naureen Mahmood, N. Ghorbani, N. Troje, Gerard Pons-Moll, and Michael J. Black. Amass: Archive of motion capture as surface shapes. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5441–5450, 2019. 5

[8] Chenlin Meng, Ruiqi Gao, Diederik P Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. *arXiv preprint arXiv:2210.03142*, 2022. 10

[9] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021. 2

[10] Stefano Pellegrini, Andreas Ess, Konrad Schindler, and Luc Van Gool. You'll never walk alone: Modeling social behavior for multi-target tracking. In *2009 IEEE 12th international conference on computer vision*, pages 261–268. IEEE, 2009. 6

[11] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Trans. Graph.*, 37(4):143:1–143:14, July 2018. 5, 6

[12] Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. Amp: Adversarial motion priors for stylized physics-based character control. *ACM Trans. Graph.*, 40(4), July 2021. 5

[13] Davis Rempe, Jonah Philion, Leonidas J. Guibas, Sanja Fidler, and Or Litany. Generating useful accident-prone driving scenarios via a learned traffic prior. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 3, 7, 9

[14] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 2

[15] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning, 2021. 6

[16] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S Sara Mahdavi, Rapha Gontijo Lopes, et al. Photorealistic text-to-image diffusion models with deep language understanding. *arXiv preprint arXiv:2205.11487*, 2022. 11

[17] Simon Suo, Sebastian Regalado, Sergio Casas, and Raquel Urtasun. Trafficsim: Learning to simulate realistic multi-agent behaviors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10400–10409, 2021. 3

[18] Jingkang Wang, Ava Pun, James Tu, Sivabalan Manivasagam, Abbas Sadat, Sergio Casas, Mengye Ren, and Raquel Urtasun. Advsim: Generating safety-critical scenarios for self-driving vehicles. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9909–9918, 2021. 7

[19] Wenhao Yu, Greg Turk, and C. Karen Liu. Learning symmetric and low-energy locomotion. *ACM Transactions on Graphics (TOG)*, 37:1 – 12, 2018. 5

[20] Ye Yuan, Xinshuo Weng, Yanglan Ou, and Kris M Kitani. Agentformer: Agent-aware transformers for socio-temporal multi-agent forecasting. In *Proceedings of the IEEE/CVF*

International Conference on Computer Vision, pages 9813–9823, 2021. 8

[21] Ziyuan Zhong, Davis Rempe, Danfei Xu, Yuxiao Chen, Sushant Veer, Tong Che, Baishakhi Ray, and Marco Pavone. Guided conditional diffusion for controllable traffic simulation. *International Conference on Robotics and Automation (ICRA)*, 2023. 1, 8