2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Crowd-FM: Learned Optimal Selection of Conditional Flow Matching-generated Trajectories for Crowd Navigation

Antareep Singha[2*], Laksh Nanwani[1*], Mathai Mathew P.[1], Samkit Jain[1], Phani Teja Singamaneni[4], Arun Kumar Singh[3], K. Madhava Krishna[1]

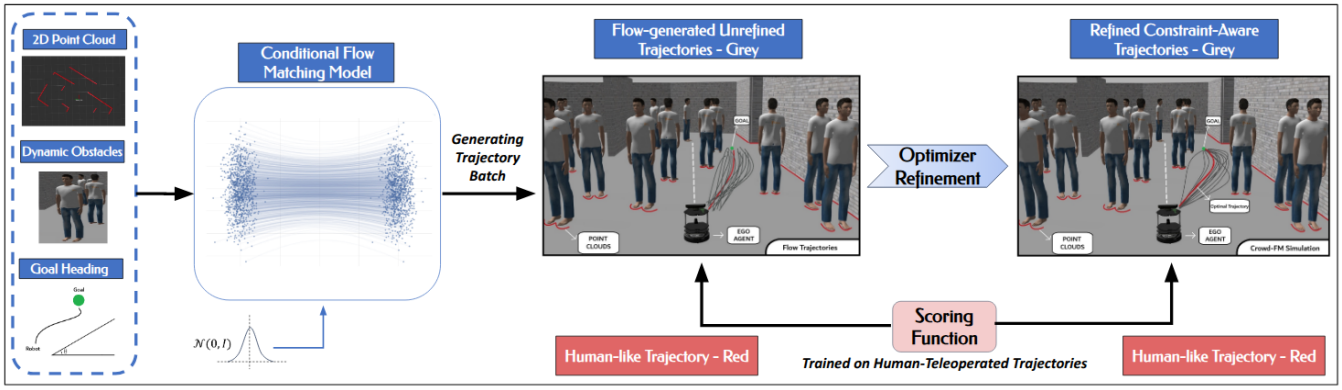

Fig. 1: Crowd-FM is a long-horizon local planner that is capable of rapidly generating collision-free trajectories in a batch. It takes in as input 2D point cloud data, dynamic obstacle positions and velocities, and the heading-to-goal angle. A Conditional Flow Matching model is trained to generate such trajectories conditioned on the input sensor data. Finally, the trajectories are refined using a Projection Optimizer [1] to meet kinodynamic constraints. A separate Scoring Function is trained on Human Expert Trajectories, to enable selection of human-like trajectories from the ones generated by CFM.

***Abstract*— Safe and computationally efficient local planning for mobile robots in dense, unstructured human crowds remains a fundamental challenge. Moreover, ensuring that robot trajectories are similar to how a human moves will increase the acceptance of the robot in human environments. In this paper, we present Crowd-FM, a learning-based approach to address both safety and human-likeness challenges. Our approach has two novel components. First, we train a Conditional Flow-Matching (CFM) policy over a dataset of optimally controlled trajectories to learn a set of collision-free primitives that a robot can choose at any given scenario. The chosen optimal control solver can generate multi-modal collision-free trajectories, allowing the CFM policy to learn a diverse set of maneuvers. Secondly, we learn a score function over a dataset of human demonstration trajectories that provides a human-likeness score for the flow primitives. At inference time, computing the optimal trajectory requires selecting the one with the highest score. Our approach improves the state-of-the-art by showing that our CFM policy alone can produce collision-free navigation with a higher success rate than existing learning-based baselines. Furthermore, when augmented with inference-time refinement, our approach can outperform even expensive optimisation-based planning approaches. Finally, we validate that our scoring network can select trajectories closer to the expert data than a manually designed cost function.**

I. INTRODUCTION

Mobile robots are still far from achieving effective trajectory modelling that enables them to operate in scenarios characterized by extreme dynamic complexity. Such scenarios are difficult to navigate even for humans, and in these situations, simply prioritizing safety above all else becomes the norm. Classical planners, with their mathematically rigorous models, are effective in ensuring safety but often fail in environments with high uncertainty and variability, as seen in velocity-obstacle–based approaches such as RVO [2]. In contrast, recent approaches based on generative modelling, such as VQ-VAEs [3], [4] or Reinforcement Learning [5]–[7], attempt to capture complex behaviors but suffer from limitations in scalability, sample efficiency, and generalization. Furthermore, much of the existing literature demonstrates success only in relatively simple, low-interaction settings [8], while performance degrades substantially in densely interactive environments common in real-world applications.

The core concept of Crowd-FM is to reframe the planning problem. Instead of searching for a single optimal path, it first generates a rich distribution of plausible future trajectories using Conditional Flow Matching [9], [10], which provides smoother and more expressive trajectory representations compared to discrete latent methods such as VQ-VAEs. A learned Scoring Function then selects the optimal one from the pool of Flow-generated trajectories, achieving a balance between robustness and efficiency.

While some works condition Flow Matching on 3D point clouds [11] or depth priors from RGB-D images [12], our approach uses 2D LiDAR data complemented by dynamic obstacle states. These states can be obtained from lightweight

* Equal contribution.
[1] Robotics Research Center, IIIT Hyderabad, India. {lakshanshul, mathewp8616}@gmail.com, {samkit.jain@students, mkrishna}@iiit.ac.in
[2] Nanyang Technological University, Singapore. antareep002@e.ntu.edu.sg
[3] University of Tartu, Estonia. aks1812@gmail.com
[4] Inria, Université de Lorraine, France. phaniteja.sp@gmail.com
Project Page: https://smart-wheelchair-rrc.github.io/crowdfm-webpage/
We acknowledge IHub-Data(Project:**M2-029**) for funding this work. It was also co-funded by the European Union and Estonian Research Council via Project:**TEM-TA101** and Grant:**PSG753** by Estonian Research Council.

trackers [13] [14] in the real world, avoiding the computational overhead of high-dimensional visual inputs. This design creates an efficient balance between representation richness and real-time feasibility, making Crowd-FM readily deployable on resource-constrained mobile platforms.

II. RELATED WORKS

Classical and Optimization-Based Navigation: Early approaches to robot navigation in crowded environments were grounded in geometric formulations such as Velocity Obstacles and Reciprocal Velocity Obstacles (RVO) [2], which provide mathematically rigorous methods for collision avoidance. Time-Elastic-Band-based [15] planners like CoHAN [16], [17] have also proven to be human-aware in moderately crowded settings. Sampling-based optimization has been explored, with methods such as PRIEST [1] augmenting trajectory sampling with projection-based feasibility checking. While effective at ensuring robot safety, these approaches often become overly conservative in highly dynamic, interactive environments.

Reinforcement Learning for Navigation: Deep reinforcement learning (DRL) has been extensively studied for crowd navigation. Works such as CrowdNav [6] and its graph-based extensions [8] introduced policies that account for human motion and interactions. DRL-VO [5] integrated velocity obstacle constraints into the DRL framework, while DenseCAvoid [7] demonstrated anticipatory behaviors in dense crowds. Although these methods show promise, their performance degrades in highly dynamic scenes, and training often requires large-scale data with careful reward shaping.

Generative Models for Trajectory Planning: Generative modelling has recently emerged as a promising alternative to reinforcement and classical methods. Diffusion-based approaches such as NoMaD [18] learn goal-conditioned distributions for long-horizon planning, while VQ-VAE–based methods like CrowdSurfer [3] and variations with differentiable safety filters [4] have achieved strong results in dense navigation tasks. However, these models suffer from limitations such as computational overhead (diffusion) or due to latent bottlenecks which quantize away detail (VQ-VAE).

Flow-Based Generative Models: Flow-based methods provide an alternative for distribution learning. Lipman et al. [9] introduced Flow Matching as a stable and efficient training paradigm, with subsequent work improving generalization via minibatch optimal transport [10]. Applications of Conditional Flow Matching have been demonstrated in robotic manipulation from pointclouds [11] and navigation with depth priors [12]. In this work, **we extend these ideas to dense crowd navigation**, where Flow Matching enables robust trajectory distribution learning, combined with a learned scoring function to boost success.

III. METHODOLOGY

In this section, we set up the crowd navigation problem and its input space representation, subsequently describing how we solve it. We describe Conditional Flow Matching(CFM), the core generative part of Crowd-FM, and how our CFM model architecture is set up. We then explain how we design a learned Scoring Function to choose the optimal trajectory from a batch of Flow-generated candidates.

A. Trajectory Parameterization

The primary objective is to generate a continuous and dynamically feasible trajectory defined by its coordinates $x(t), y(t)$ over a time horizon. The trajectory generation process is conditioned on a context vector, C, which encapsulates high-dimensional sensory inputs (environmental context and goal). Directly learning a mapping from the context C to the infinite-dimensional space of continuous functions is intractable. To achieve this, we parameterize the continuous robot trajectories using a Bernstein polynomial of order $n(10)$, uniquely defined by a set of $n+1$ control points, over the time horizon $t \in [t_1, t_n]$. This approach effectively transcribes the infinite-dimensional optimal control problem into a finite-dimensional nonlinear programming problem, which is more amenable to machine learning algorithms.

The parameterization is done in the following manner:

$$\begin{bmatrix} x(t_1) \\ \vdots \\ x(t_n) \end{bmatrix}^T = \mathbf{P}\ \mathbf{c}_x, \begin{bmatrix} y(t_1) \\ \vdots \\ y(t_n) \end{bmatrix}^T = \mathbf{P}\mathbf{c}_y, \mathbf{W} = \begin{bmatrix} \mathbf{P} & \mathbf{0} \\ \mathbf{0} & \mathbf{P} \end{bmatrix} \quad (1)$$

where the matrix $\mathbf{P}$ is a matrix formed by time-dependent polynomial basis functions. The vectors $\mathbf{c}_x, \mathbf{c}_y$ are the coefficients attached to the individual basis functions. We can represent the derivatives like $\dot{x}(t), \ddot{x}(t), \dot{y}(t), \ddot{y}(t)$ in a similar manner as (1) using $\dot{\mathbf{P}}$ and $\ddot{\mathbf{P}}$. The combined trajectory representation is then given as $\boldsymbol{\xi} = \begin{bmatrix} \mathbf{c}_x & \mathbf{c}_y \end{bmatrix}^T$.

Our trajectory representation follows previous motion planning works like [3], [4], and [1]. By choosing to generate Bernstein control points rather than a sequence of waypoints, the generative model is not burdened with learning the concepts of smoothness or continuity from data since the mathematical structure of the Bernstein representation guarantees these priors. However, ensuring the resulting trajectories adhere to the robot's dynamic constraints requires further refinement, which is addressed in III-D.2. Although our choice of Bernstein polynomials is motivated by the same advantages demonstrated in recent diffusion-based works such as GPD [19], the way these representations are used differs substantially. GPD leverages diffusion over Bernstein coefficients with a guided denoising process and a trajectory stitching procedure to ensure feasibility. In contrast, Crowd-FM uses CFM to directly learn the transformation to Bernstein coefficients via ODE integration, enabling faster and more stable inference.

B. Flow Matching Objective

CFM learns a time-dependent vector field, $u_\tau(\boldsymbol{\xi})$, that transports samples from a simple, known prior distribution, $q_0(\boldsymbol{\xi})$ (a standard normal distribution $\mathcal{N}(0, I)$), to a complex target data distribution, $q_1(\boldsymbol{\xi})$. This transport is achieved by integrating the ODE

$$d\boldsymbol{\xi} = u_\tau(\boldsymbol{\xi})d\tau$$

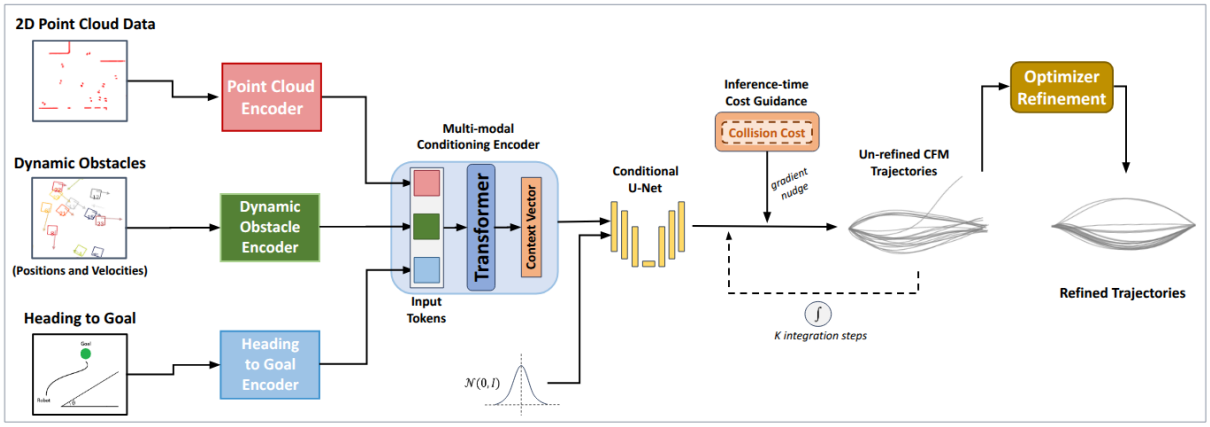

Fig. 2: The Conditional Flow Matching model is used to learn a multi-modal distribution of collision-free trajectories in terms of Bernstein coefficients to ensure that the reconstructed trajectories have higher continuity and differentiability. The model is conditioned on a Transformer-based input space encoder that takes into account the environmental context at every timestep. The inference-time integration is implicitly guided by a collision cost term encouraging collision-free generations. Finally, the generated trajectories are refined using a single optimization step [1] to satisfy kinodynamic constraints.

from an **"artificial time"** $\tau = 0$ to $\tau = 1$. It is worth noting that this "artificial time" is distinct from the time t used in III-A. A neural network, $v_\theta(\boldsymbol{\xi}, \tau, C)$, with C as the condition(context vector), is trained to approximate this true vector field. A key advantage of CFM is its use of a regression-based objective that is tractable and avoids the need for costly ODE simulation during the training phase.

As discussed in subsection III-A, we train the CFM model directly in the space of Bernstein control points, represented by the vector $\boldsymbol{\xi} \in \mathbb{R}^{2(n+1)}$. The prior distribution, $q_0(\boldsymbol{\xi})$, is a standard multivariate Gaussian, while the target distribution, $q_1(\boldsymbol{\xi})$, is the empirical distribution of control points extracted by fitting Bernstein polynomials to ground truth trajectories in the training dataset.

Following the independent coupling strategy employed in recent works like [12], a conditional probability path $p_\tau(\boldsymbol{\xi}|\boldsymbol{\xi_0}, \boldsymbol{\xi_1})$ and a corresponding vector field $u_\tau(\boldsymbol{\xi}|\boldsymbol{\xi_0}, \boldsymbol{\xi_1})$ is defined for any pair of samples $\boldsymbol{\xi_0} \sim q_0(\boldsymbol{\xi})$ and $\boldsymbol{\xi_1} \sim q_1(\boldsymbol{\xi})$. This path is a simple linear interpolation in the high-dimensional space of control points:

$$\boldsymbol{\xi}_\tau = (1 - \tau)\boldsymbol{\xi_0} + \tau\boldsymbol{\xi_1}$$

The vector field that generates this path is constant with respect to both time and position along the path, pointing directly from the noise sample to the data sample:

$$u_\tau(\boldsymbol{\xi}|\boldsymbol{\xi_0}, \boldsymbol{\xi_1}) = \boldsymbol{\xi_1} - \boldsymbol{\xi_0}$$

Consequently, the neural network v_θ is trained on a manifold of physically valid and meaningful trajectory representations at every intermediate flow time τ.

For our problem, the neural network, $v_\theta(\boldsymbol{\xi_\tau}, \tau, C)$, is designed to predict the target vector field $u_\tau = \boldsymbol{\xi_1} - \boldsymbol{\xi_0}$. The model is trained by minimizing the Conditional Flow Matching loss, which takes the form of a simple mean squared error objective:

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{\tau \sim U(0,1), \boldsymbol{\xi_0} \sim q_0, \boldsymbol{\xi_1} \sim q_1, C}$$
$$\left[\|v_\theta((1 - \tau)\boldsymbol{\xi_0} + \tau\boldsymbol{\xi_1}, \tau, C) - (\boldsymbol{\xi_1} - \boldsymbol{\xi_0})\|^2\right]$$

This objective directly regresses the output of the learned vector field onto the constant-velocity field that connects a random noise sample to a target data sample, conditioned on the environmental context.

C. Flow Matching Model Architecture

This subsection details the Conditional Flow Matching architecture (Fig. 2) adapted to learn the complex multi-modal distribution of Bernstein control points that define expert trajectories.

The architecture comprises two primary components: a multi-modal input space encoder that processes the environmental and goal information into a context vector C, and a conditional U-Net that approximates the vector field v_θ. Subsequently, Inference-time Cost Guidance and Optimizer refinement are added to the pipeline, as discussed in subsections III-D.1 and III-D.2 respectively, to ensure safety and smoothness at all times.

1) Multi-modal Conditioning Encoder: The multi-modal conditioning encoder processes static obstacles, dynamic obstacles, and heading to goal angles at every timestep through specialized encoders before fusing them using a self-attention mechanism.

a) Point Cloud Encoder: For a point cloud input $X_{\text{pcd}} \in \mathbb{R}^{N_{\text{pts}} \times 2}$, where N_{pts} is the maximum number of point-cloud points, a feature extractor f_{pcd} composed of a series of 1D convolutions and ReLU activations maps each point to a high-dimensional feature vector. A global max-pooling operation is then applied across all points to yield a permutation-invariant global feature vector, $c_{\text{pcd}} = \max_{i=1\ldots N_{\text{pts}}} f_{\text{pcd}}(X_{\text{pcd}})$. This vector is processed by a multi-layer perceptron (MLP), g_{pcd}, to produce the final static embedding, $C_{\text{static}} = g_{\text{pcd}}(c_{\text{pcd}})$.

b) Dynamic Obstacle Encoder: The Dynamic Obstacle information is compiled into a specialized tensor for the model to work with. For an input dynamic obstacle tensor $X_{\text{dyn}} \in \mathbb{R}^{N_{\text{obs}} \times 4}$, representing N_{obs}, each with a 4D kinematic state(position and velocity), a dynamic obstacle encoder f_{dyn} processes them into a high-dimensional feature vector. An MLP converts obstacle features into high-

dimensional embeddings, which are augmented with positional embeddings. Subsequently, a minimal Transformer Encoder with 4 heads and w layers employs self-attention to model interactions among the obstacles. Finally, max-pooling aggregates these features into a single vector, which a final MLP processes to produce the dynamic context embedding($D = 512$), C_{dynamic}.

c) Goal Encoder: The navigation goal is provided as a 2D heading vector, $X_{\text{goal}} \in \mathbb{R}^2$. This vector is processed by a dedicated MLP, g_{goal}, to produce a goal embedding: $C_{\text{goal}} = g_{\text{goal}}(\text{X}_{\text{goal}})$.

d) Multi-modal Fusion: The individual embeddings, C_{static}, C_{dyn}, and C_{goal}, are treated as tokens for a fusion module. The tokens are concatenated to form an input sequence, which is then processed by F_{fusion}, a Transformer Encoder with 8 heads and 3 layers. The Transformer Encoder uses self-attention to produce the final context, a sequence of fused tokens $C = F_{\text{fusion}}(S)$. This sequence captures the complex interplay between the robot's environment, other moving agents, and its objective. This context is directly used to condition the 1D U-Net as explained below.

2) Conditional U-Net: The core of Crowd-FM is the flow prediction network that is implemented as a 1D U-Net. This network is designed to predict the vector field that transports the control points from the prior to the data distribution.

The U-Net takes three inputs:

1) The interpolated control point vector $\boldsymbol{\xi_\tau} \in \mathbb{R}^{2(n+1)}$.
2) The scalar flow time $\tau \in [0, 1]$.
3) The sequence of context tokens C from the conditioning encoder.

The U-Net architecture consists of a symmetric encoder-decoder structure with skip connections. The encoder progressively downsamples the 1D representation of the trajectory control points, and the decoder upsamples this representation, integrating information from the corresponding encoder layers via skip connections. The flow time τ is transformed into a sinusoidal time embedding and added to the intermediate features at each resolution level. The context C is injected into the network within the U-Net's residual blocks, allowing the model to condition the predicted flow on the specific environmental and goal context. The final output of the network is the predicted vector field $v_\theta(\boldsymbol{\xi_\tau}, \tau, C)$, which has the same dimension as the input $\boldsymbol{\xi_\tau}$.

D. Inference-time Corrections

We perform two kinds of inference-time modifications to the flow predicted trajectories. First, we embed a guidance step into the flow integration process. Second, we use a more powerful projection optimizer on the flow output to further push it towards feasible regions.

1) Guided Integration Process: To encourage the integration process to generate collision-free trajectories at run-time in dense crowds, we apply an inference-time collision cost guidance, quite similar to how [20] implements gradient-based control for image generation. The cost is formulated as:

$$\mathcal{L}_{\text{collision}} = \frac{1}{N} \sum_{k=1}^{N_{\text{waypoints}}} \max\left(0, d_{\text{safe}}^2 - \min_{m=1}^{N_{\text{points}}} \|\mathbf{p}_k - \mathbf{o}_m\|_2^2\right)$$

where:

- $\mathbf{p}_k(x(t), y(t))$ is the k-th waypoint along the trajectory(transformed from coefficients)
- $\mathbf{o}_m$ is the m-th obstacle point
- d_{safe} is the safety margin
- N_{points} is the number of obstacle points

It is important to note that $\mathbf{p}_k$ is a function of $\boldsymbol{\xi_\tau}$, and is implied that:

$$\mathcal{L}_{\text{collision}} = f(\mathbf{p}_k), \quad \text{and}$$

$$\mathbf{p}_k = g(\boldsymbol{\xi_\tau}) \implies \mathcal{L}_{\text{collision}} = f(g(\boldsymbol{\xi_\tau}))$$

The gradient of this function, $\nabla\mathcal{L}_{\text{collision}}$, is a vector that points in the direction of the steepest ascent of the cost. Consequently, the negative gradient, $-\nabla\mathcal{L}_{\text{collision}}$, points towards more desirable, lower-cost states.

We recall from the Flow Matching Objective that the ODE is formulated as:

$$\frac{d\boldsymbol{\xi_\tau}}{d\tau} = v(\boldsymbol{\xi_\tau}, \tau, C)$$

To encourage collision-free generation, we modify the original ODE by adding this negative gradient term to the learned vector field. The new, guided ODE is defined as:

$$\frac{d\boldsymbol{\xi_\tau}}{d\tau} = v(\boldsymbol{\xi_\tau}, \tau, C) - \lambda \cdot \nabla_{\boldsymbol{\xi_\tau}} \mathcal{L}_{collision}(\boldsymbol{\xi}_\tau)$$

where λ is a scalar guidance scale hyperparameter that controls the strength of the corrective guidance. By solving this modified ODE, the state $\boldsymbol{\xi_\tau}$ evolves according to a composite vector field. It simultaneously follows the learned data manifold (via $v(\boldsymbol{\xi_\tau}, \tau)$) and descends the gradient of the cost function (via $-\lambda \cdot \nabla_{\boldsymbol{\xi_\tau}} \mathcal{L}_{\text{collision}}(\boldsymbol{\xi}_\tau)$).

2) Optimizer Refinement (Projection Optimizer): The final flow trajectories are further refined by the projection optimizer of [1], which produces the closest trajectories to the flow prediction that is also collision-free.

E. Learned Scoring Function

The flow model is designed to output a roll-out of several trajectories, but one of them must be selected for traversal. This can be done in multiple ways, including a pre-defined cost-based selection where the trajectory with the lowest overall cost is selected. However, more often than not, this yields trajectories that deviate from human behavior. To account for human-like behavior in Crowd-FM, we move towards a Trajectory Selector that inherently learns how to choose trajectories based on expert human demonstrations. In this subsection, we explain how we train a learnable scoring function based on the dynamic environmental context to imitate expert demonstrations. The scoring function learns a mapping $S(\boldsymbol{\xi}, C)$ of a candidate trajectory $\boldsymbol{\xi}$ and the current context vector C to a scalar score, enabling the selection of the best trajectory.

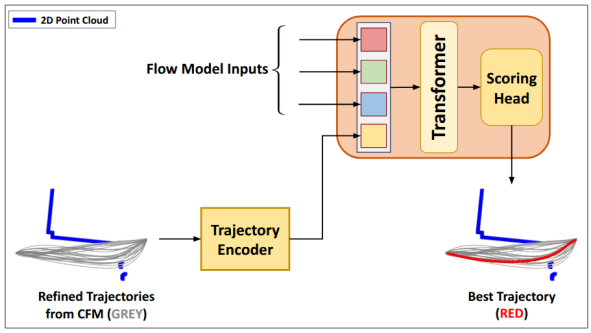

Fig. 3: The learned scoring function has a similar input space representation as the Flow model. It borrows the input encoders from the Flow model architecture, namely the Point Cloud, Dynamic Obstacles, and the Goal Encoders. Additionally, the Flow-generated trajectories are encoded and tokenized as the input to a Transformer Encoder. The transformer output is passed to the Scoring Head to output raw scores for each trajectory generated.

1) Scoring Function Architecture: The scorer is a Transformer-based architecture that jointly reasons over the environmental context and a set of K candidate trajectories generated at run-time.

- *Input Representation:* The scorer takes as input the same static obstacle, dynamic obstacle, and goal heading information as the flow model. Additionally, it takes a batch of K flow-generated candidate trajectories, $\{\boldsymbol{\xi}_1, ..., \boldsymbol{\xi}_K\}$(each trajectory is represented by Bernstein control points).
- *Input Space Encoders:* The scorer uses the same encoder modules as the flow model to produce context embeddings $C_{\text{static}}, C_{\text{dyn}}$, and C_{goal}. A separate Trajectory Encoder is introduced to process the candidate trajectories. For each candidate $\boldsymbol{\xi}_k$, its control points are passed through a 1D convolutional network followed by mean pooling and an MLP to produce a trajectory embedding $C_{\text{traj},k}$.
- *Multi-modal Fusion and Scoring:* To differentiate between the sources of information, learnable modality-type embeddings ($E_{\text{static}}, E_{\text{dyn}}, E_{\text{goal}}, E_{\text{traj}}$) are added to their respective context embeddings. The context embeddings are then concatenated with the set of K trajectory embeddings to form a single sequence of tokens:

$$S_{\text{in}} = [C_{\text{traj},1} + E_{\text{traj}}, ..., C_{\text{traj},K} + E_{\text{traj}}, C_{\text{static}}$$

$$+E_{\text{static}}, C_{\text{dyn}} + E_{\text{dyn}}, C_{\text{goal}} + E_{\text{goal}}]$$

This sequence is processed by a Transformer Encoder with 8 heads and 4 layers, allowing each trajectory token to attend to all other trajectory tokens and the full environmental context. The output tokens corresponding to the trajectories, $\{\hat{C}_{\text{traj},1}, \ldots, \hat{C}_{\text{traj},K}\}$, are then passed through a final MLP scoring head, which outputs a single scalar logit for each trajectory, producing the raw scores $\{\text{score}_1, \ldots, \text{score}_K\}$.

2) Training Objective: The scorer is trained to identify which of the generated candidate trajectories is **"best"** with respect to an expert demonstration. Training the Scoring Function follows a similar trajectory as the final inference. A frozen, pre-trained flow model generates a set of K candidate trajectory coefficients that are optimizer refined and then converted to trajectories $\{P_k\}_{k=1}^{K}$.
Given an expert trajectory P_{expert} from our custom dataset of the ego-agent navigating in dense crowds, the ground truth label is determined by finding the candidate trajectory that is closest to the expert in Euclidean space. Let j be the index of this closest candidate:

$$j = \arg \min_{k \in \{1,\ldots,K\}} \|P_k(t) - P_{\text{expert}}(t)\|_2$$

The Scoring Function is trained using a cross-entropy loss, treating the problem as a K-class classification task, where the target labels are dynamically generated for each training instance. This objective trains the scorer to assign the highest logit to the candidate that best imitates the expert's path. During inference, it is sufficient to select the trajectory index with the highest score.
The loss is regularized by the cost from the optimizer discussed in section III-D.2, encouraging the model to favor trajectories that are not only close to the expert but also have low optimization costs. The final loss is:

$$\mathcal{L}_{\text{scorer}} = \text{CrossEntropy}(\text{scores}, j) + \lambda \cdot \text{mean}(\text{cost}_{\text{optimizer}})$$

where λ is a hyperparameter balancing the two loss components.

For every set of K candidate trajectories, we define the ground truth on the fly by identifying the candidate that has the minimum Euclidean distance to the expert demonstration. The index of this "closest" candidate becomes the target label for the cross-entropy function.

Therefore, while the absolute identity of any given index changes between training steps, the relative correspondence between the scores and the dynamically generated target is preserved, ensuring that the cross-entropy loss correctly penalizes the model for failing to assign the highest score to the geometrically optimal trajectory, as seen in Fig. 3.

F. Data Collection

A critical factor in the performance of Crowd-FM is the quality of expert demonstrations used for training. However, existing public datasets for crowd navigation are typically limited to sparse interactions, simple layouts, or reactive behaviors, and therefore do not capture the dense, high-interaction settings needed for effective deployment. Moreover, they rarely emphasize smooth, non-freezing trajectories or provide data in a form readily compatible with Bernstein polynomial representations. To overcome these limitations, we curated a dataset specifically tailored to the requirements of Crowd-FM, combining both simulation and real-world scenarios.

Data collection was performed using the Barn environment to capture challenging static obstacle layouts and PEDSIM to model dense dynamic crowds. The dataset consists of

World	Number of runs	DRL-VO		CoHAN 2.0		Crowdsurfer		Crowd-FM(Ours)	
		Succ. runs	Rate	Succ. runs	Rate	Succ. runs	Rate	Succ. runs	Rate
Cumberland	15	6	0.40	10	0.67	11	0.73	**13**	**0.87**
Lobby World	18	11	0.61	14	0.78	**17**	**0.94**	16	0.89
Freiburg	15	8	0.53	9	0.60	11	0.73	**12**	**0.80**

TABLE I: Comparison of navigation success rates across different planners and environments with **35 dynamic agents**. *Note: CoHAN 2.0 success rates could be a little higher with more extensive parameter tuning.*

around five hours of navigation data using both the Jackal and Turtlebot2 robots. Two sources of trajectories were included:

1) *Manually teleoperated trajectories*: Collected in both simulation and real-world deployments, these are used to train the Scoring Function.
2) *Post-processed trajectories*: Refined with [1] to ensure dynamics and consistency with the Bernstein polynomial representations employed for training the CFM.

For training, the expert trajectories are parameterized with Bernstein polynomials, converting continuous paths into compact sets of control points. During inference, Crowd-FM is conditioned on three inputs: the 2D LiDAR point cloud, the heading angle to the global goal, and a tensor of dynamic obstacle states (x, y, v_x, v_y).

IV. VALIDATION AND BENCHMARKING

In this section, we detail our extensive evaluation of Crowd-FM. We show qualitative results, conduct quantitative benchmarking, and perform ablation studies in challenging simulated environments, comparing our method against three leading baselines: CrowdSurfer [3], DRL-VO [5], and CoHAN2.0 [16]. We then validate the practical feasibility of our pipeline in real-world settings.

A. Qualitative Analysis

This sub-section deals with the performance of the entire Crowd-FM pipeline across different PEDSIM environments:

- *Smooth and dynamically feasible trajectories:* Bernstein polynomial parameterization, coupled with Flow Matching, enforces continuity. This results in smooth trajectories, avoiding the jittery behavior observed in baseline planners.
- *Robust reactivity to dynamic obstacles:* Flow-generated trajectories adapt quickly to the motion of surrounding agents, allowing the robot to maneuver without freezing, a common failure mode for VQ-VAE, optimization, and DRL-based methods.
- *Diversity of options:* Unlike VQ-VAE, which tends to collapse onto a limited set of discrete maneuver templates, Crowd-FM generates a rich spectrum of candidate paths. As illustrated in Fig. 4, Crowd-FM proposes multiple feasible routes around obstacles, while CrowdSurfer's trajectories show a collapse to a unimodal distribution, highlighting the lack of diversity from its VQ-VAE priors when compared to the expressive representations learned by Flow Matching.
- *Effective trajectory selection:* The learned Scoring Function leverages this diversity by consistently choosing the safest and most efficient option among Flow candidates. This selection mechanism allows Crowd-FM to combine exploration (via Flow) with refinement (via scoring), improving both safety and efficiency.

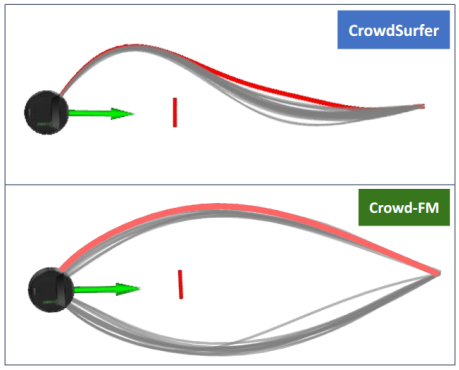

Fig. 4: *Diversity of Trajectories:* The small red point cloud in front of the robot is the obstacle to be avoided. The upper image shows the trajectories generated by CrowdSurfer. The lower image shows the trajectories generated by Crowd-FM. The trajectories generated using Crowd-FM have more diverse outputs and can generate safe paths in multiple directions. The red trajectory is the best trajectory selected by both planners.

B. Quantitative Analysis

Crowd-FM sets a new standard for performance in crowd navigation, outperforming all baselines in nearly every metric across standard PEDSIM [21] environments (Table I). While the performance of optimization-based planners like CoHAN 2.0 could potentially be improved through more exhaustive, environment-specific parameter tuning, Crowd-FM remains fundamentally superior in real-time inference efficiency, overall time-to-goal efficiency, and its inherent ability to prevent freezing behaviors in dense scenarios. This particularly improves upon CrowdSurfer, despite their shared use of generative priors. We stress-test both these pipelines in more challenging environments and share our results in Table II.

This improvement stems from two pillars: (1) our CFM model's near-perfect learning of collision avoidance from data, and (2) exceptional computational speed (refer Sec. IV-B.4). The following subsections are dedicated to ablation studies, especially focusing on the effectiveness of the vanilla CFM-generated trajectories, followed by the runtime analysis of our pipeline.

*1) **Ablation Study without Optimizer Refinement**:* Here, we test Crowd-FM without the Optimizer refinement step by comparing the trajectories generated by the Vanilla-CFM with another baseline (DRL-VO) and see if the Conditional Flow Matching models the input data distribution well. As is evident from Table III, vanilla CFM alone is able to beat a leading RL-based baseline, DRL-VO, by a fair margin. This shows that CFM alone can model collision avoidance, especially in dense crowds. Qualitative examples of vanilla-CFM trajectories can be found in Fig. 5.

World	Number of Runs	Number of Agents	CrowdSurfer				Crowd-FM			
			Trajectory Length (m) ↓	Trajectory Time (s) ↓	Velocity (m/s) ↑	Success Rate ↑	Trajectory Length (m) ↓	Trajectory Time (s) ↓	Velocity (m/s) ↑	Success Rate ↑
Cumberland	15	35	18.82	57.13	0.52	0.73	**16.18**	**36.49**	**0.71**	**0.87**
Lobby	18		13.87	32.85	0.61	**0.94**	**13.76**	**25.42**	**0.72**	0.89
Freiburg	15		15.18	44.71	0.53	0.73	**14.12**	**28.7**	**0.66**	**0.8**
Cumberland	15	45	17.46	85.81	0.44	0.67	**17.38**	**55.13**	**0.55**	**0.8**
Freiburg	15		14.24	63.43	**0.56**	0.6	**12.17**	**59.41**	0.46	**0.73**

TABLE II: Performance comparison between Crowd-FM and Crowdsurfer. The metrics in the table are an average over the successful runs.

World	Number of Runs	DRL-VO				Vanilla CFM			
		Trajectory Length (m) ↓	Trajectory Time (s) ↓	Velocity (m/s) ↑	Success Rate ↑	Trajectory Length (m) ↑	Trajectory Time (s) ↓	Velocity (m/s) ↑	Success Rate ↑
Cumberland	15	18.47	55.95	0.40	0.40	**16.58**	**54.42**	**0.48**	**0.67**
Lobby	18	**13.15**	39.89	0.39	**0.61**	13.61	**36.99**	**0.48**	**0.61**
Freiburg	15	14.18	44.50	0.41	0.53	**13.43**	**34.45**	**0.51**	**0.6**

TABLE III: Performance comparison between the Vanilla CFM(with Cost Guidance) and DRL-VO planner with 35 agents. Metrics for Trajectory Length, Time, and Velocity are averaged over all successful runs. The results demonstrate that CFM models collision avoidance more effectively than DRL-VO, achieving a higher success rate, shorter paths, higher average velocity, and lower trajectory times.

Vanilla CFM	Success Rates ↑
w/o Cost Guidance	0.57
w/ Cost Guidance	**0.67**

TABLE IV: We test the impact of Implicit Cost-Guidance on the performance of Vanilla-CFM over 10 runs with K=5 integration steps. The increase in the success rate indicates the usefulness of the guidance.

Method	Success Rate ↑		
	Cumberland	Lobby	Freiburg
Cost Function	0.80	0.83	**0.80**
Score Function (Ours)	**0.87**	**0.89**	**0.80**

TABLE V: Comparison of predefined cost-based planning and our learned Scoring Function across different environments.

*2) **Effect of Inference-time Cost Guidance**:* One of the two ways we employ inference-time refinement of CFM-generated trajectories is the Inference-time Cost Guidance. We test this property of Crowd-FM by disabling Optimizer refinement on the Cumberland and Freiburg environments in Pedsim with 25 agents for 10 trials. Table IV shows a clear improvement in Success rates for the collision-cost guided Flow objective compared to the un-guided one. Additionally, we present qualitative results for the same in Fig. 5.

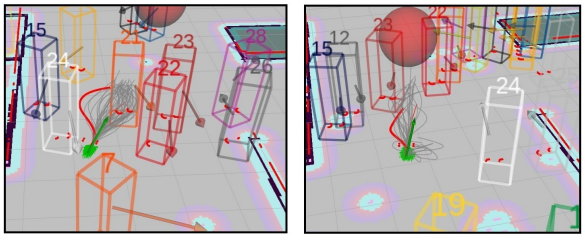

Fig. 5: *Effect of Inference-time Cost Guidance*: Left image shows the trajectories generated by Crowd-FM *with* Cost Guidance. The right image shows the trajectories generated by Crowd-FM *without* Cost Guidance. Trajectories generated using collision cost guidance are found to be more controlled near obstacles than the trajectories without it. (Tests done without Optimizer Refinement)

*3) **Effect of the Learned Scoring Function**:* To validate that the proposed Scoring Function improves the human-likeness of robot trajectories, we introduce a quantitative metric termed the *Human-Likeness Points(HLP)*. For a candidate trajectory P_c and the corresponding expert (human) trajectory P_h, we define HLP as the average point-wise deviation over a time horizon T:

$$\text{HLP}(P_c, P_h) = \frac{1}{T}\sum_{t=1}^{T} \|P_c(t) - P_h(t)\|_2.$$

Lower HLP values indicate that the chosen trajectory is closer to the expert demonstration and thus more human-like. We compare trajectories selected by our learned Scoring Function against those chosen by predefined, hand-tuned cost functions that primarily optimize for collision avoidance and smoothness. These tests were done in an open-loop setting to take into account the expert human demonstrations. A custom wheelchair prototype was teleoperated in indoor environments amid dense crowds to create the test dataset. We evaluated HLPs for both methods on 10 such scenes and provided our results as a Grouped Bar Chart in Fig. 6.

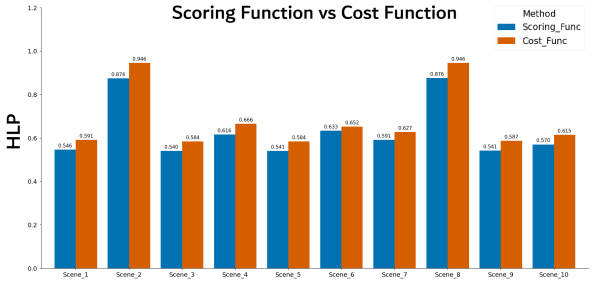

Fig. 6: Grouped Bar Plot showing the HLPs for Scoring Function selected Trajectories(BLUE) and Cost-Function selected Trajectories(ORANGE).

Evidently, our Scoring Function consistently reduces the HLP across all environments, indicating that the selected trajectories are closer to expert data. At the same time, success rates in Table V improve over different closed-loop environments. This demonstrates that the scoring mechanism not only preserves safety but also aligns robot motion more closely with human navigation patterns.

*4) **Runtime Analysis**:* To evaluate real-time feasibility, we benchmarked the Crowd-FM pipeline on a laptop with an NVIDIA RTX 3060 GPU. The entire planning loop, including data processing and trajectory generation, takes ∼75ms. A detailed breakdown shows that the CFM model generates the initial trajectory batch in ∼45ms, and the

Learned Scoring Function selects the optimal candidate in ∼5ms. The remaining time is utilized by the Optimizer Refinement step (∼25ms), which ensures dynamic feasibility. Performances are naturally expected to improve with better GPUs. Crowd-FM's high computational efficiency maintains a consistent control rate suitable for dense crowd navigation.

C. Real-World Experiments

We validated Crowd-FM in real-world scenarios on a Pioneer 3-DX mobile robot. The robot was equipped with an RPLiDAR 2D scanner and utilized a LegTracker [13] module to estimate dynamic obstacle velocities and positions. The experiments were conducted in our lab and in the academic block, where our method consistently demonstrated reasonable success rates in densely populated environments. In these complex settings, the system achieved an overall success rate of 17 runs out of 20 total trials. Fig. 7 shows stills from two such representative testing scenarios.

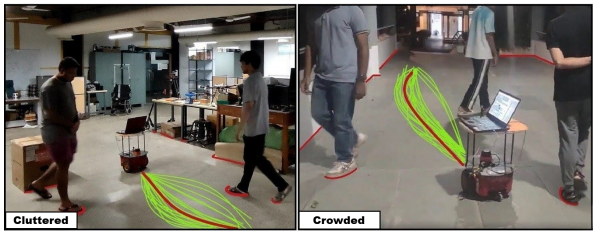

Fig. 7: Crowd-FM in action on a **P3-DX robot** in two distinct scenarios - a cluttered environment with high static obstacle density(lab), and a crowded environment with high dynamic obstacle density(academic block).

V. Conclusion and Future Work

This work introduces Crowd-FM, a trajectory-planning framework that utilizes Conditional Flow Matching to learn expressive distributions over Bernstein polynomial control points. Unlike prior generative approaches, which have discrete latent bottlenecks(VQ-VAEs) or are computationally expensive(Diffusion), Crowd-FM demonstrates that flow-based transport can capture diverse and smooth navigation, while remaining efficient at inference. A lightweight projection optimization step further refines the candidate trajectories, boosting success by ∼20%, without resorting to a slower complete trajectory optimization. We validate the approach both in simulation and real-world settings, highlighting that smooth, non-freezing, and human-like trajectories can be rapidly generated in dense crowds.

An essential future direction is to expand the dataset with more heterogeneous scenarios, allowing us to capture richer behaviors and test the scoring function under different crowd dynamics. We also see CFM's potential in learning social cues and scene semantics through its integration with complementary planning modules, such as human motion prediction models, moving towards social navigation.

VI. Acknowledgment

The authors would like to acknowledge the use of large language models (Gemini, ChatGPT) and writing enhancement tools (Grammarly) solely for refining the grammatical structure and the linguistic flow of this manuscript.

References

[1] F. Rastgar, H. Masnavi, B. Sharma, A. Aabloo, J. Swevers, and A. K. Singh, "Priest: Projection guided sampling-based optimization for autonomous navigation," *IEEE Robotics and Automation Letters*, vol. 9, no. 3, pp. 2630–2637, 2024. 1, 2, 3, 4, 6

[2] J. van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *2008 IEEE International Conference on Robotics and Automation*, pp. 1928–1935, 2008. 1, 2

[3] N. Kumar, A. Singha, L. Nanwani, D. Potdar, T. R, F. Rastgar, S. Idoko, A. K. Singh, and K. M. Krishna, "Crowdsurfer: Sampling optimization augmented with vector-quantized variational autoencoder for dense crowd navigation," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 16854–16860, 2025. 1, 2, 6

[4] S. Idoko, B. Sharma, and A. K. Singh, "Learning sampling distribution and safety filter for autonomous driving with vq-vae and differentiable optimization," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3260–3267, 2024. 1, 2

[5] Z. Xie and P. Dames, "Drl-vo: Learning to navigate through crowded dynamic scenes using velocity obstacles," *IEEE Transactions on Robotics*, vol. 39, p. 2700–2719, Aug. 2023. 1, 2, 6

[6] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning," in *ICRA*, 2019. 1, 2

[7] A. J. Sathyamoorthy, J. Liang, U. Patel, T. Guan, R. Chandra, and D. Manocha, "Densecavoid: Real-time navigation in dense crowds using anticipatory behaviors," *CoRR*, vol. abs/2002.03038, 2020. 1, 2

[8] C. Chen, S. Hu, P. Nikdel, G. Mori, and M. Savva, "Relational graph learning for crowd navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020. 1, 2

[9] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le, "Flow matching for generative modeling," in *The Eleventh International Conference on Learning Representations*, 2023. 1, 2

[10] A. Tong, K. Fatras, N. Malkin, G. Huguet, Y. Zhang, J. Rector-Brooks, G. Wolf, and Y. Bengio, "Improving and generalizing flow-based generative models with minibatch optimal transport," *Transactions on Machine Learning Research*, 2024. 1, 2

[11] E. Chisari, N. Heppert, M. Argus, T. Welschehold, T. Brox, and A. Valada, "Learning robotic manipulation policies from point clouds with conditional flow matching," *Conference on Robot Learning (CoRL)*, 2024. 1, 2

[12] S. Gode, A. Nayak, D. N. Oliveira, , M. Krawez, C. Schmid, and W. Burgard, "Flownav: Combining flow matching and depth priors for efficient navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2025. 1, 2, 3

[13] A. Leigh, J. Pineau, and H. Zhang, "Person tracking and following with 2d laser scanners," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2015, pp. 726–733, 06 2015. 2, 8

[14] Z. Xu, H. Shen, X. Han, H. Jin, K. Ye, and K. Shimada, "Lv-dot: Lidar-visual dynamic obstacle detection and tracking for autonomous robot navigation," *arXiv preprint arXiv:2502.20607*, 2025. 2

[15] P. T. Singamaneni and R. Alami, "Hateb-2: Reactive planning and decision making in human-robot co-navigation," in *International Conference on Robot & Human Interactive Communication*, 2020. 2

[16] P. T. Singamaneni, A. Favier, and R. Alami, "Human-aware navigation planner for diverse human-robot ineraction contexts," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021. 2, 6

[17] P.-T. Singamaneni, A. Favier, and R. Alami, "Watch out! there may be a human. addressing invisible humans in social navigation," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 11344–11351, IEEE, 2022. 2

[18] A. Sridhar, D. Shah, C. Glossop, and S. Levine, "Nomad: Goal masked diffusion policies for navigation and exploration," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 63–70, 2024. 2

[19] A. Srikanth, P. Mahanjan, K. Saha, V. Mandadi, P. Paul, P. Wadhwani, B. Bhowmick, A. Singh, and M. Krishna, "Gpd: Guided polynomial diffusion for motion planning," 2025. 2

[20] X. Liu, L. Wu, S. Zhang, C. Gong, W. Ping, and Q. Liu, "Flowgrad: Controlling the output of generative odes with gradients," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 24335–24344, June 2023. 4

[21] C. Gloor, "PEDSIM: Pedestrian crowd simulation." http://pedsim.silmaril.org, 2016. 6