NoMaD: **N**avigati**o**n with Goal-**Ma**sked **D**iffusion

Sehaj Ganjoo, Shobhnik Kriplani,
Abhishek Kumar Jha, Namashivayaa V
`Btech Mathematics and Computing`
Indian Institute of Science

April 14, 2025

Abstract

This report details our implementation and analysis of a visual navigation system based on the NoMaD framework. Our model combines an EfficientNet-Transformer perception backbone with a conditional diffusion-based planner, trained on SACSON and parts of RECON and Go_Stanford datasets. It generates multimodal waypoint predictions for goal-directed and exploratory navigation in diverse real-world settings. We present the model architecture, training methodology, evaluation metrics (logged via Weights & Biases), and provide a GitHub repository and project website. Deployment aspects are not covered.

1 Introduction

Robotic learning for navigation in unfamiliar environments requires the ability to perform both task-oriented navigation and task-agnostic exploration. Traditionally, these functionalities are tackled by separate systems.

What is NoMaD?

NoMaD is a transformer-based diffusion policy designed for long-horizon, memory-based navigation, that can:

- Explore unknown places on its own (goal-agnostic behavior).
- Go to a specific place or object when given a goal image (goal-directed behavior).

Our project implements the NoMaD Policy, leveraging its Transformer-based architecture and conditional diffusion decoder to learn from a rich, multimodal dataset (SACSON, Go Stanford, RECON). Unlike traditional models, NoMaD offers improved generalization and robustness with a compact design. This report focuses on its perception and training components, showing how a strong visual encoder and diffusion decoder enhance waypoint prediction. We present training dynamics, key metrics (cosine similarity, distance loss), and the model's generalization across diverse scenarios.

2 Methodology

Building on the Visual Navigation Transformer (ViNT) [7] and diffusion-based policy learning (see Appendix B), NoMaD extends these approaches with two key innovations: (1) attention-based goal masking for flexible behavior switching, and (2) a diffusion decoder for multimodal waypoint prediction.

Architecture Overview

- **Visual Encoder**: Processes RGB observations o_t using an EfficientNet-B0 backbone to extract feature embeddings.
- **Context Encoder**: Transformer-based module that maintains temporal context and handles goal conditioning
- **Diffusion Decoder**: Generates multimodal waypoint predictions through iterative denoising

Refer to Appendix C for a detailed breakdown of the architecture.

A. Attention based Goal Masking

- When $m = 0$, the policy **attends to the goal image** o_g to perform goal-conditioned navigation.
- When $m = 1$, the goal pathway is **masked out**, resulting in undirected exploration.

B. Diffusion Policy

1. **Forward Process:** Start with a real action a_t^0 from the dataset and add gaussian noise over multiple steps (K).
2. **Reverse Denoising(Learned Model):** starting from pure noise $a_t^k \sim \mathcal{N}(0, I)$, it denoises step by step to recover the final clean action a_t^0.
Each denoising step is :

$$a_t^{k-1} = \alpha(\alpha_t^k - \gamma_k.\epsilon_\theta(c_t, a_t^k, k)) + \mathcal{N}(0, \sigma^2.I)$$

C. Implementation Details

The github repository implementing NOMaD is as follows: RobotNavigation
The website hosting the project and the results is: NoMaD Refer to Appendix D for more details regarding the implementation.

2.1 Training Procedure

Training Objective: The loss combines MSE between predicted and actual noise with a temporal distance loss

$$\mathcal{L}_{NoMaD}(\phi, \psi, f, \theta, f_d) = MSE(\epsilon^k, \epsilon_\theta(c_t, a_t^0 + \epsilon^k, k)) + \lambda . MSE(d(o_t, o_g), f_d(c_t))$$

where: ψ, ϕ correspond to the visual encoders for the observation and goal images, f corresponds to the transformer layers, θ is a diffusion parameters and f_d corresponds to the temporal distance predictor.

2.2 Evaluation

We evaluate the model in two modes:
(i) **Unconditional (UC)**, where the policy predicts future waypoints based only on the observation history; and
(ii) **Goal-Conditioned (GC)**, where it also receives a goal image to guide planning. This distinction allows us to measure both general scene understanding and goal-directed behavior.

3 Experiments and Results

Refer to Appendix E for further results and details regarding the experiments.

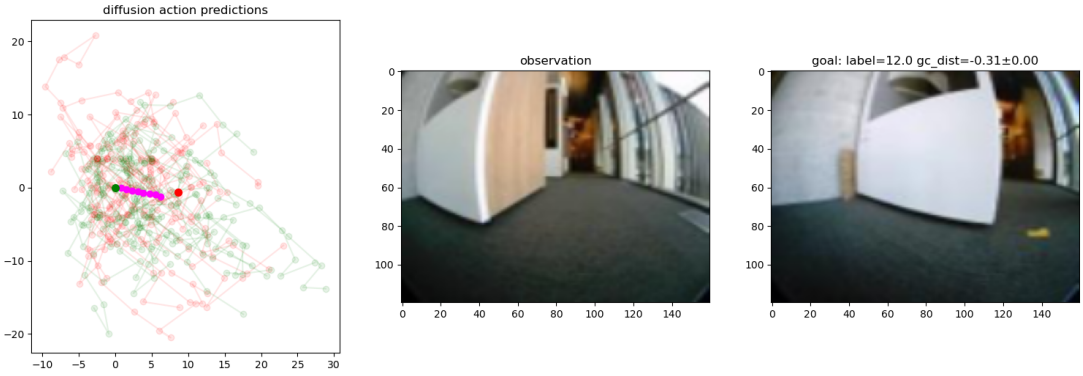
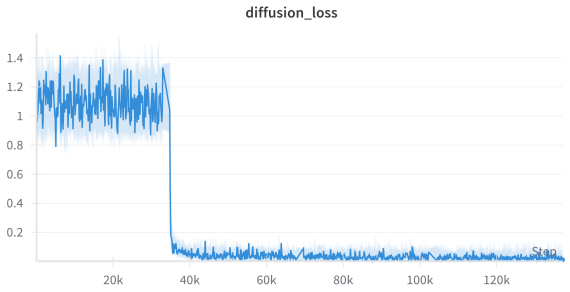
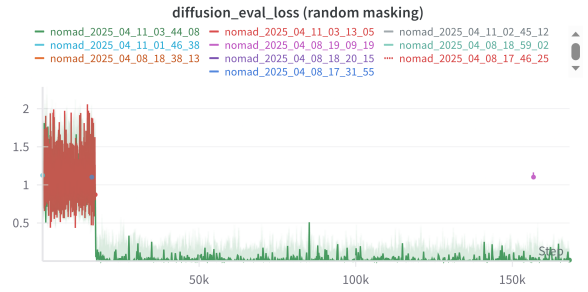


Figure 1: Illustrates a scenario where a robot is trying to navigate towards a specific goal (Train Action Sample)

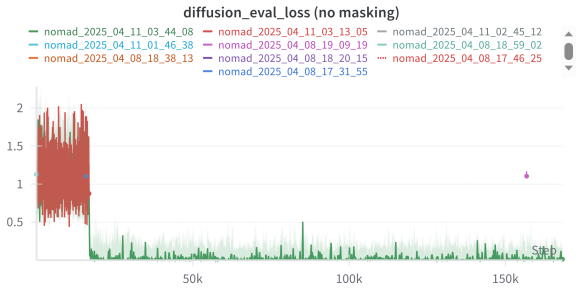
Diffusion Loss



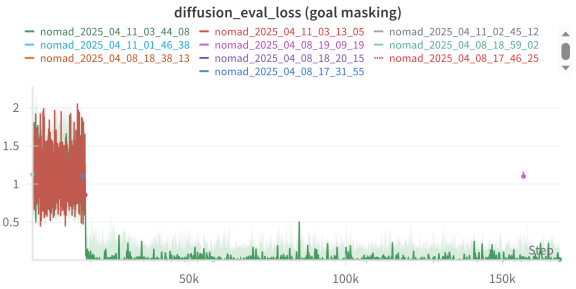
(a) Diffusion Loss on Training Set



(b) Diffusion evaluation loss with random masking



(c) Diffusion evaluation loss without masking



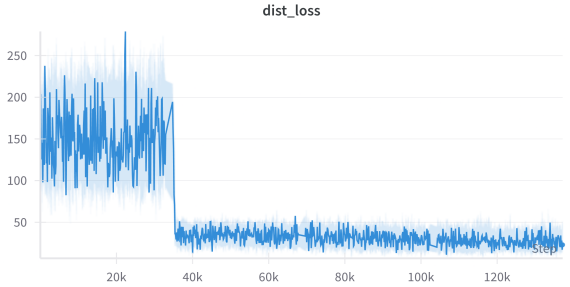
(d) Diffusion evaluation loss with goal masking

Figure **??** compares training and evaluation diffusion losses under random, no, and goal masking strategies.

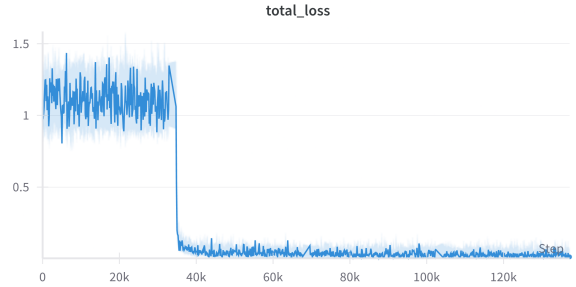
1. 2a: Training loss decreases sharply around 30k–40k steps and then stabilizes, indicating effective learning.

2. 2d, 2c, 2b: Evaluation losses are higher and more unstable, but all show an initial sharp decline.

Distance Loss and Total Loss



(a) Distance Loss on Training Set



(b) Total Loss on Training Set

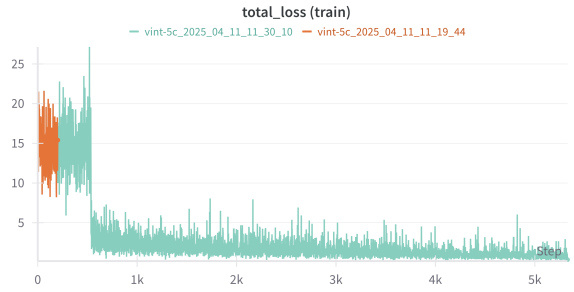
The plot agrees that the model is learning effectively, confirmed by the sharp drop in the loss around 35k steps. The losses later stabilizes, suggesting successful convergence and consistent learning performance.

Remarks:

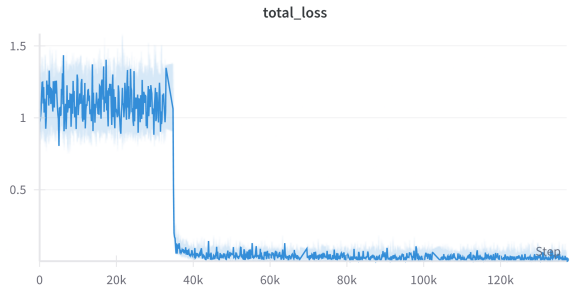
The plots in 3a, 2a and 3b correspond to another run of the experiment with similar datasets, conditions and parameters. Refer to Appendix J for more details regarding the plots of `distance_loss`, `diffusion_loss` and `total_loss`

Comparison with ViNT

Both models were evaluated under identical conditions and intrestingly in the **goal-conditioned (GC)** setting, both demonstrated comparable performances. Refer to Appendix F for further comparision plots.



(a) Total Loss on Training Set for ViNT



(b) Total Loss on Training Set for NoMaD

Appendices

A Related Work and Contextual Foundations of NoMaD

Exploration in unfamiliar enviroments is approached as the problem of efficient mapping, typically formulated around information maximization to guide the robot toward unexplored regions.
We factorize the classical exploration problem into two categories:

- Local exploration strategies that rely on current observations. Objective is to learn control policies that can take diverse,short-horizon actions
- Global exploration strategies that utilize a map of the environment. Basically a high-level planner based on a topological graph that uses the policy for long-horizon goal-seeking

Robots exploring a new area are essentially trying to map it efficiently—this means covering as much area as possible, ideally without wasting time.
However, building detailed geometric maps, can be difficult without accurate depth perception.

Several prior approaches have investigated learning-based exploration policies. Some approaches use simulation data (training in virtual environments).
Others learn from real-world data directly. These models may use:

- Intrinsic rewards: Encouraging the robot to explore new things.
- Semantic prediction: Going to interesting or informative places.
- Latent variable models: Abstract models of how actions affect the world.

Yet, policies trained in simulation frequently struggle to transfer to real-world environments. Even real-world-trained models can underperform in complex indoor and outdoor settings.
Enter NoMaD : A New Method

The work most closely related to NoMaD is ViNT (refer Appendix X for more details), which combines a goal-conditioned policy with a separate subgoal proposal module. The subgoal proposals are generated using an image diffusion model, condiitioned on robot's current view. NoMaD improves on this by:

- Not generating images.
- Directly predicting actions using diffusion models, which are typically used in image generation tasks but can model complex probabilities really well.
- This makes NoMaD more accurate and much lighter (needs 15x fewer parameters).

One of the core challenges in modeling robot exploration policies is the inherently multimodal nature of action sequences.
Observation-conditioned diffusion models have emerged as powerful tools because they can learn complex action distributions without needing without needing explicit state prediction. Nomad builds upon this adding **goal conditioning** to diffusion-based action generation, meaning it is capable of both:

- Goal-directed exploration
- Undirected exploration

B Technical Preliminaries

The primary objective is to develop a visual navigation policy, denoted by π, that enables a robot to navigate using only RGB images from its onboard camera.
The policy devised should operates as follows:

- It receives a sequence of past and current observations: $o_t := o_{t-P:t}$.
- It predicts a distribution over future actions: $a_t := a_{t:t+H}$.

- Optionally, it can also condition on a goal image o_g, representing the desired destination.

Depending on whether a goal is provided, the policy behaves differently:

- **Goal-directed navigation:** When a goal image o_g is available, π generates actions that guide the robot toward the goal.
- **Exploratory behavior:** When no goal is given (as in pure exploration settings), π must still generate safe and purposeful actions—avoiding obstacles and staying on traversable paths—while efficiently covering the environment.

To handle long-horizon planning and complex environments, the system is further augmented with:

- A topological memory graph $\mathcal{M}$, which maintains a structured map of past visual observations.
- A high-level planner that leverages this memory to decide on intermediate goals and broader exploration strategies.

Visual Goal-Conditioned Policies: ViNT as the Backbone

NoMaD builds on the ViNT (Visual Navigation Transformer) architecture, a Transformer-based model tailored for goal-conditioned navigation.

Key Components of ViNT:

- **Visual Encoding:** Each observation is processed using an EfficientNet-B0 encoder to extract feature embeddings.
- **Goal Fusion:** The current and goal image features are combined using a goal fusion encoder.
- **Transformer Attention:** These fused features (tokens) are passed through a Transformer model to generate a context vector c_t.
- **Predictions:** The context vector is used to predict:
  - A distribution over future actions: $a_t = f_a(c_t)$.
  - An estimate of temporal distance to the goal: $d(o_t, o_g) = f_d(c_t)$.

These outputs are learned via supervised training, where the model is shown expert trajectories and learns to imitate them.

> *However, ViNT is inherently goal-conditioned—it cannot operate in the absence of a goal image, limiting its ability to explore autonomously.*

Extending to Long-Horizon Planning with Topological Memory

To overcome this limitation, NoMaD incorporates a topological memory $\mathcal{M}$:

- Nodes represent previously encountered visual observations.
- Edges represent traversable paths, established using ViNT's predicted distances.

This memory graph enables both:

- **Subgoal Planning:** If a goal cannot be reached directly, the planner identifies a sequence of reachable subgoals via $\mathcal{M}$.
- **Structured Exploration:** Even without a goal, $\mathcal{M}$ helps guide exploration toward promising, yet-unvisited areas.

Frontier-Based Exploration with NoMaD

To evaluate NoMaD's ability to generalize to new environments, the authors implement a frontier-based exploration strategy:

- **Frontier:** The boundary between explored and unexplored areas in the environment.
- The high-level planner selects frontiers as subgoals, encouraging the robot to expand coverage.

This builds on the ViKiNG framework, but replaces its prior policy with NoMaD, which supports both:

- **Goal-seeking** behavior (when a goal image is provided), and
- **Autonomous exploration** (when no goal is available).

C Methodology: Ctd.

Architecture Overview

Figure 5 illustrates the NoMaD architecture, which consists of three main components:

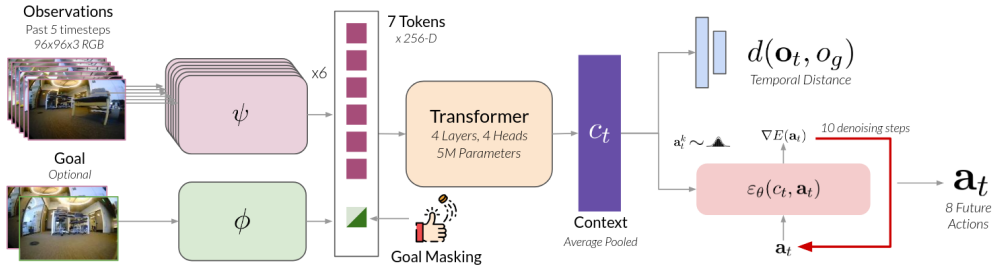


Figure 5: NoMaD architecture. The visual encoder processes RGB observations, the transformer maintains temporal context, and the diffusion decoder generates waypoint predictions. Goal masking occurs in the attention layers of the context encoder.

A. Attention based Goal Masking

To seamlessly switch between goal-directed behavior and exploration, we set a binary goal mask m in NoMaD. The model builds on the ViNT (Visual Navigation Transformer) framework, by modifying the the computation of the observation context vector c_t as follows:

$$c_t = f(\psi(o_i), \phi(o_t, o_g), m)$$

where:

- $\psi(o_i)$: Encodes the current observation.
- $\phi(o_t, o_g)$: Encodes the time-step and goal image.
- m: A binary mask that determines whether to attend to the goal image or not.
- When $m = 0$, the policy **attends to the goal image** o_g to perform goal-conditioned navigation.
- When $m = 1$, the goal pathway is **masked out**, resulting in undirected exploration. Masking is applied in the attention layers, so goal embeddings are ignored if m=1.

During training, the goal mask m is sampled from a Bernoulli distribution with probability $p_m = 0.5$, ensuring equal exposure to both behaviors.
At test time, m is explicitly set based on the task: $m = 0$ for goal-reaching, and $m = 1$ for exploration.

B. Diffusion Policy

Refer to Appendix I for more on Diffusion policies.
In general, diffusion models are generative models that work by gradually adding noise to data (a forward process), and then learning to remove that noise (a reverse denoising process) to generate realistic samples.
To model complex, multimodal action distributions, especially in unstructured or ambiguous environments, NoMaD employs a **diffusion model** to approximate the conditional distribution of the next action as $p(a_t|c_t)$.
The denoising process is as follows:
1. **Forward Process:** Start with a real action a_t^0 from the dataset and add gaussian noise over multiple steps (K).

$$a_t^k = \sqrt{\alpha_k} a_t^{k-1} + (\sqrt{1 - \alpha_k})\epsilon$$

where:

- $\epsilon \sim \mathcal{N}$(0,I) is a random noise
- α_k is a noise scheduler (eg square cosine)

- By step K, the action is almost pure noise.

2. **Reverse Denoising(Learned Model):** starting from pure noise $a_t^k \sim \mathcal{N}(0, I)$, it denoises step by step to recover the final clean action a_t^0.
Each denoising step is :

$$a_t^{k-1} = \alpha(\alpha_t^k - \gamma_k.\epsilon_\theta(c_t, a_t^k, k)) + \mathcal{N}(0, \sigma^2.I)$$

Where:

- Here, ϵ_θ is the noise prediction network conditioned on the context c_t, which may or may not include the goal depending on m.
  - It is a 1D conditional U-Net with 15 CNN layers.
  - Input:the noisy action a_t^k, the context vector c_t, and the diffusion step k.
  - the predicted noise vector $\hat{\epsilon}_k$,During training, it is compared to the true noise added earlier.
- γ, α, σ are scheduler constants.

D Implementation Details

D.1 Environment Setup

All experiments were conducted on a system equipped with an NVIDIA GeForce GTX 1660 SUPER GPU and a 12$^{\text{th}}$ Gen Intel® Core™ i9-12900K (24-core) processor.
The code was implemented in Python 3.12.9 in a conda environment. All the packages and libraries used are listed in the requirements.txt file in the github repo.
The codebase was adapted from an open-source repository[3], and experiment tracking was performed using `Weights & Biases (WandB)` for logging loss curves, evaluation metrics, and debugging errors. Refer to Appendix J for more details and results.

D.2 Data Pipeline

We trained the NoMaD model using the **SACSON** and parts of **RECON** dataset along with **go_stanford**, which contains diverse real-world trajectories across various environments and robot platforms. The SACSON and go_stanford dataset were already processed and we it split into training and validation sets using 80-20 split ratio. The RECON dataset consisted of bag files, which needed to be preprocessed to extract RGB images, actions, and ground-truth waypoints.

Training Configuration:

- We trained the model with batch size of 47 for 10 epochs.

- The weighting factor λ for the auxiliary distance loss was set to 10^{-4}.
- The model was trained using the **AdamW** optimizer with a learning rate of $1e^{-4}$ and a weight decay of $1e^{-2}$.

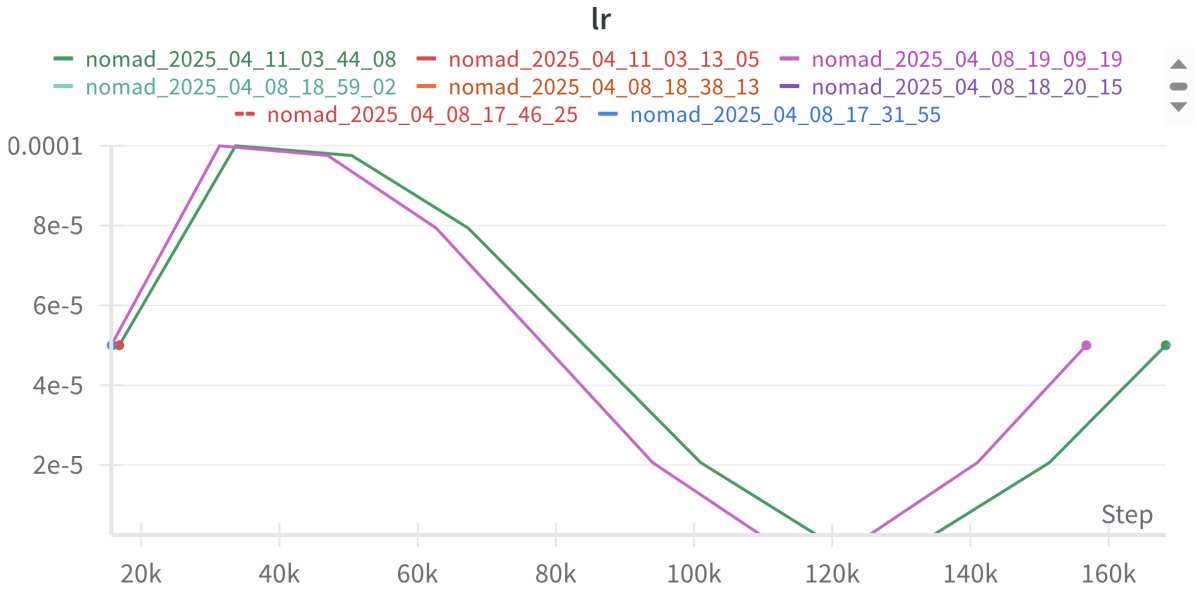


Figure 6: Learned rate schedule for the NoMaD model.

- We applied **cosine annealing** to decay the learning rate over time.
- We used **goal masking** with probability $p_m = 0.5$, encouraging the model to generalize across goal-visible and goal-agnostic contexts.
- The diffusion process used a **square cosine schedule** with $K = 10$ denoising steps.
- The ViNT observation encoder was an EfficientNet-B0, mapping 96×96 RGB images into a 256-dimensional latent embedding.
- The transformer used for context encoding had 4 layers and 4 attention heads.
- Training checkpoints and EMA snapshots were saved at the end of each epoch for model tracking and potential evaluation.
- Training and validation loss curves were logged using `Wandb` for detailed inspection.

Evaluation:
Evaluation was performed on the validation set using the following metrics:

- distance loss
- diffusion loss

- cosine similarity
- action loss

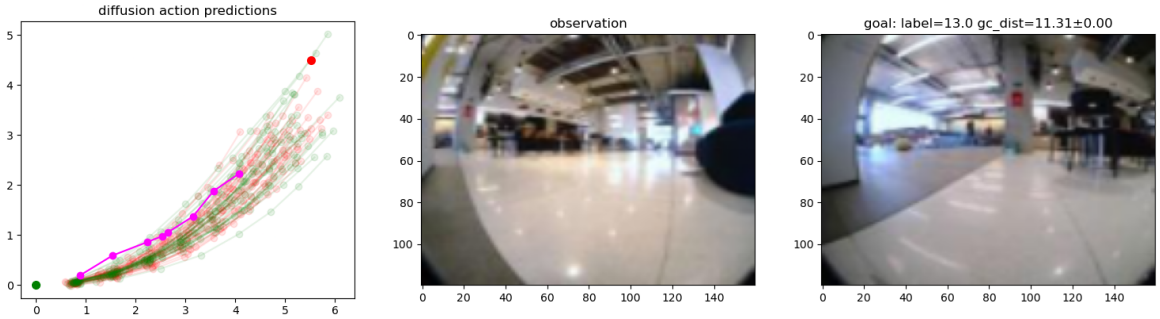
D.3 Using WandB

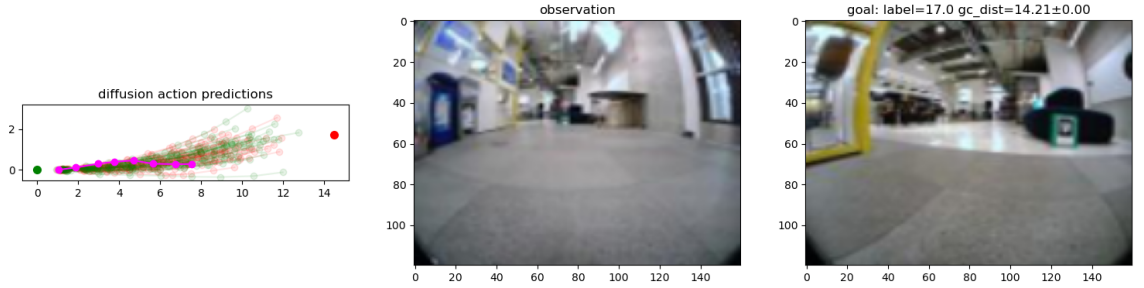
We used wandb to log training progress, visualize losses, and monitor both model behavior and system resources (e.g., GPU/CPU utilization) throughout experimentation.
Metrics such as action loss, goal prediction error, and the learning rate schedule were automatically tracked and visualized, which helped with debugging and plotting out results. Refer to Appendix F for other resulting plots and visualizations.

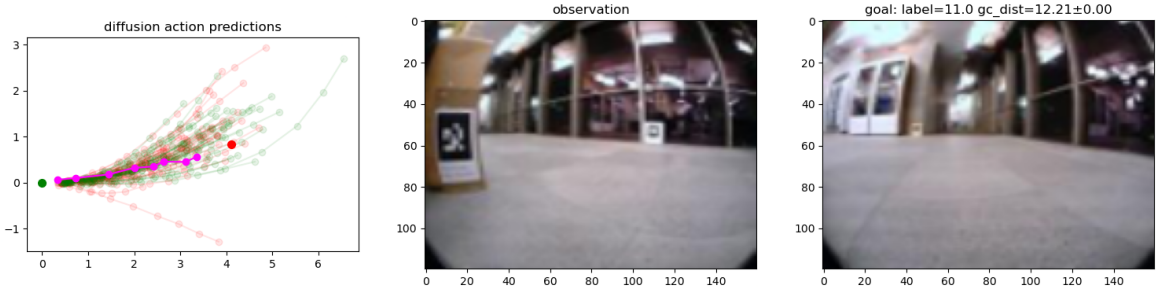
E Results and Experiments

The figures below illustrates scenarios where a robot is trying to navigate towards a specific goal. The left plot shows the diffusion model's sampled action trajectories in a 2D projection space. Green and red lines denote various sampled action sequences, coressponding to different samples or different stages in the diffusion process (e.g., noisy vs. denoised predictions), while the pink trajectory represents the final action sequence selected by the model. This trajectory leads from the robot's current position toward the predicted goal location. The spread of the predicted actions reflects the model's uncertainty about the optimal path.

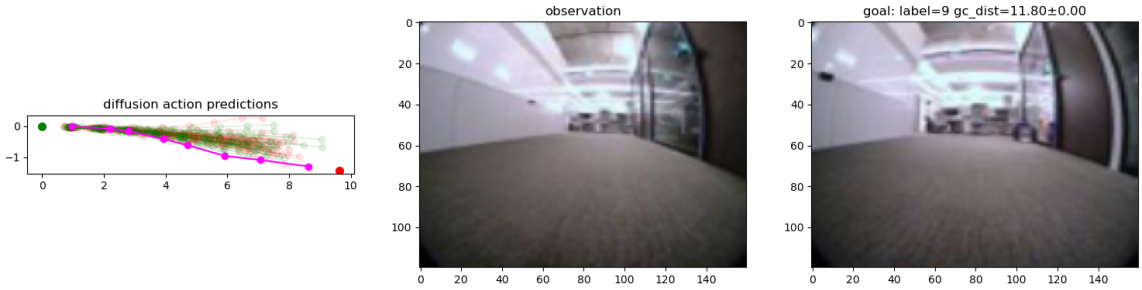
Train action samples:

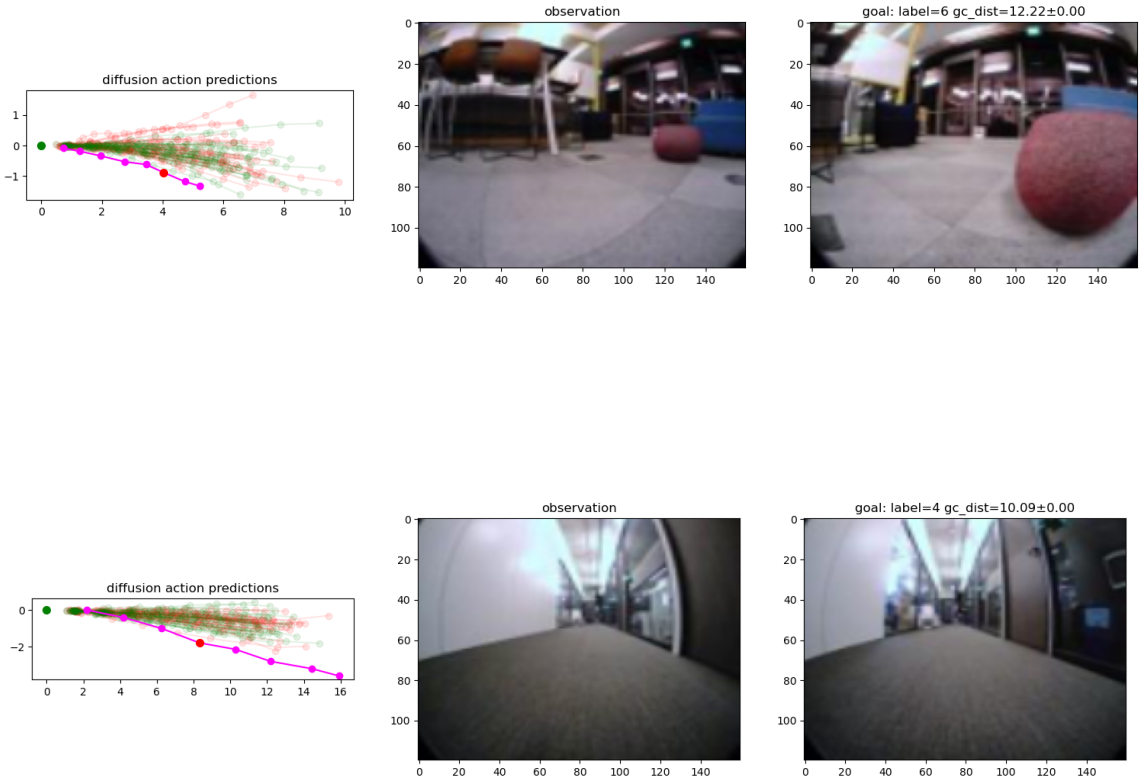






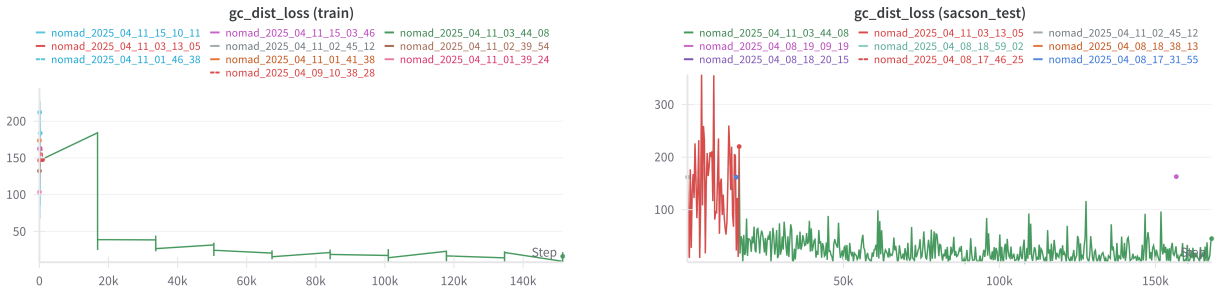
Sacson test action samples:





Goal-Conditioned (GC) Evaluation

We set the goal mask $m = 0$ to evaluate the model's performance in goal-directed navigation.
Distance Loss:



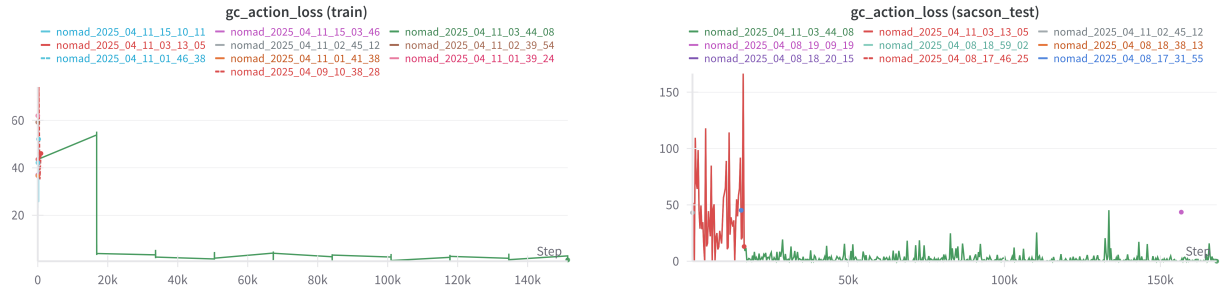
(a) Distance Loss on Training Set

(b) Distance Loss on Validation Set

Figure 7: Distance loss comparison between training and validation sets under goal-conditioned evaluation.

We see that the loss has plateaued after the first epoch, both in test and training sets, indicating that the model quickly learns a strong initial estimate of the distance to the goal.

Action loss:

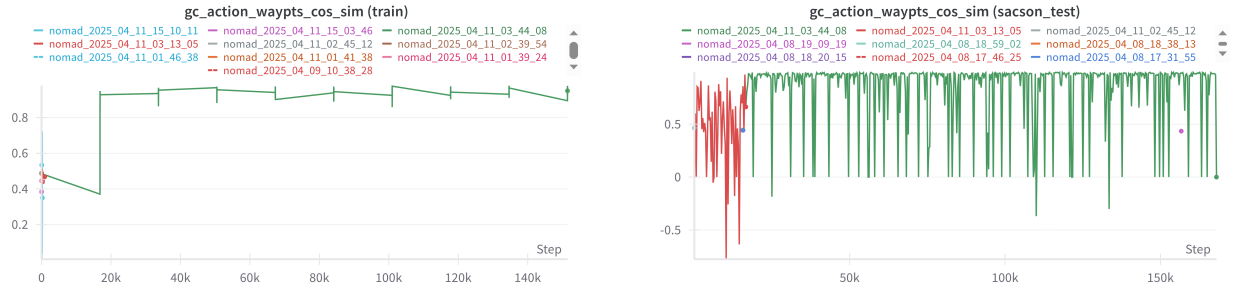


(a) Action Loss on Training Set (b) Action Loss on Validation Set

Figure 8: Action loss comparison between training and validation sets under goal-conditioned evaluation.

The action loss steadily decreases across epochs in both training and validation datasets, suggesting that the diffusion-based policy learns to generate feasible future actions conditioned on the visual goal and current context.

gc_action_waypts_cos_sim:



(a) Cosine Similarity on Training Set (b) Cosine Similarity on Validation Set

Figure 9: Cosine similarity between predicted and ground-truth waypoints on the training and validation sets under goal-conditioned evaluation.

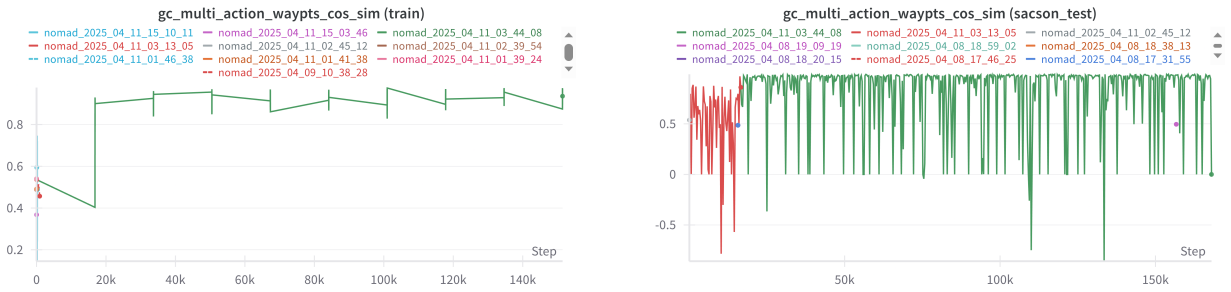
The Cosine Similarity metric evaluates how well the predicted action direction aligns with the ground-truth direction by measuring the angle between the two vectors. A value closer to 1 indicates better directional alignment.

- During training, the model achieves a high and stable cosine similarity on the training set, indicating that it effectively learns to predict actions aligned with the ground-truth directions in seen environments.

- However, the validation set shows noticeable variance in cosine similarity across epochs. This could be attributed to the fact that the dataset configuration had `learn_angle = False`, meaning the model was not explicitly trained to predict the robot's orientation or angular displacement.

- As a result, while the model can align its predictions in direction (to some extent), the lack of angular supervision might reduce its generalization capacity in unseen environments, leading to inconsistency in cosine similarity.

gc_multi_action_waypts_cos_sim:



(a) Cosine Similarity on Training Set

(b) Cosine Similarity on Validation Set

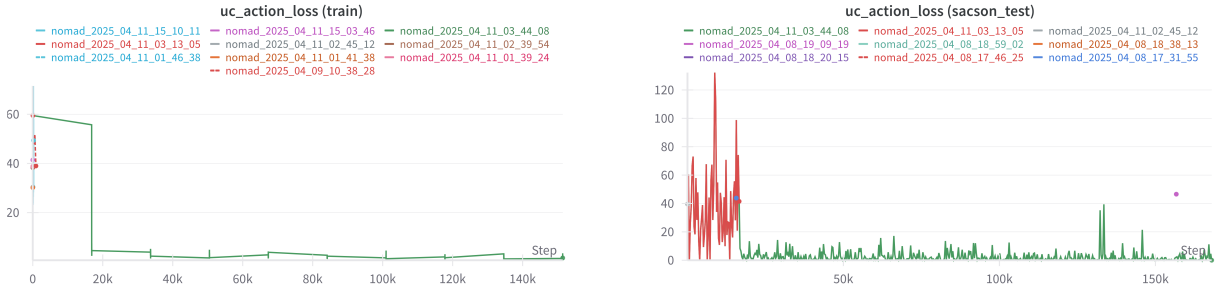
Figure 10: Cosine similarity between predicted and ground-truth multi-waypoints on the training and validation sets under goal-conditioned evaluation.

Multi-action cosine similarity compares the overall alignment of full predicted trajectory vectors with ground truth, rather than frame-by-frame. This provides a more holistic measure of long-horizon trajectory quality. We see a similar trend in multi-action cosine similarity as action waypts cosine similarity. On the training set, the plot consistently improves and stabilizes at a high value, indicating that the model effectively learns to align its predicted trajectories with the desired paths. Similarly we notice the lower and more volatile cosine similarity on the validation set.

Unconditional (UC) Evaluation

We set the goal mask $m = 1$ to evaluate the model's performance in open-ended exploration. Interestingly, the model's behavior in this unconditioned setting closely mirrors that observed during goal-conditioned evaluations. This indicates that the model encounters similar generalization challenges and follows comparable learning dynamics, even in the absence of explicit goal cues.

Action loss:

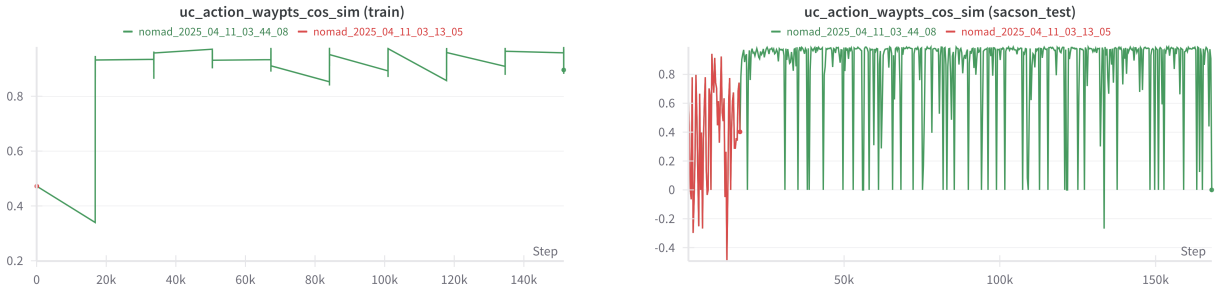


(a) Action Loss on Training Set

(b) Action Loss on Validation Set

Figure 11: Action loss comparison between training and validation sets under goal-agnostic evaluation.

uc_action_waypts_cos_sim:

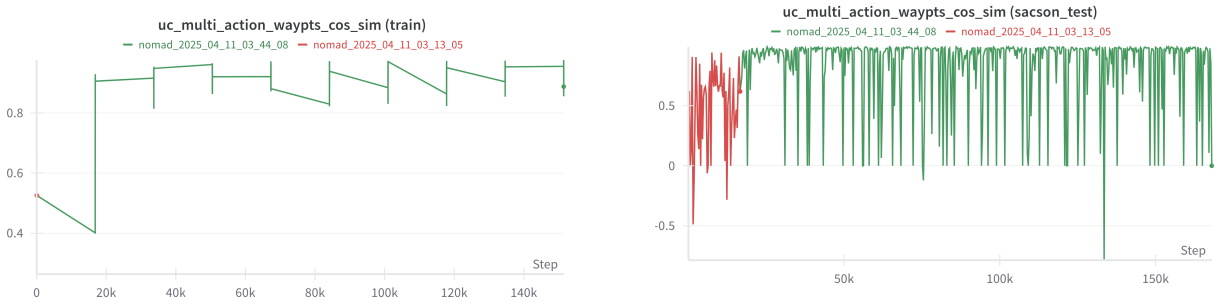


(a) Cosine Similarity on Training Set

(b) Cosine Similarity on Validation Set

Figure 12: Cosine similarity between predicted and ground-truth waypoints on the training and validation sets under goal-agnostic evalutation

uc_multi_action_waypts_cos_sim:



(a) Cosine Similarity on Validation Set

Figure 13: Cosine similarity between predicted and ground-truth multi-waypoints on the training and validation sets under goal-agnostic evalutation

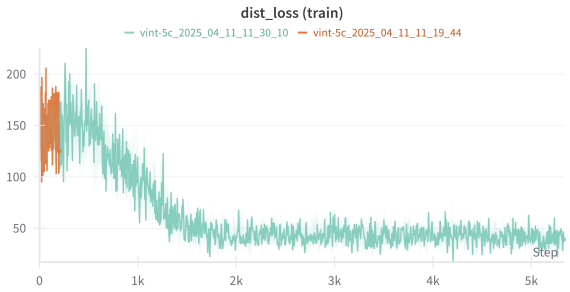
F Comparison with ViNT

To establish a baseline, we trained the original ViNT architecture using the same dataset and training hyperparameters as used for NoMaD. The model configuration included the same EfficientNet-B0 encoder and transformer-based architecture, but excluded the diffusion decoder.

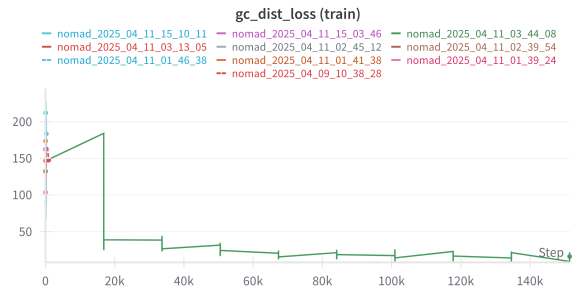
Both models were evaluated under identical conditions on the validation set, using metrics such as Mean Squared Error (MSE) for actions and cosine similarity between predicted and ground-truth waypoints. The ViNT model was trained for 10 epochs with a batch size of 47.

Interestingly, in the **goal-conditioned (GC)** setting, both ViNT and NoMaD demonstrated comparable performance across the key evaluation metrics. This suggests that the integration of a diffusion-based decoder in NoMaD does not negatively impact performance on goal-directed planning tasks. Instead, it enables a more expressive generative modeling capability while preserving task effectiveness.

Here are some plots from WandB comparing the two models on the basis of `distance_loss` and `action_loss`:

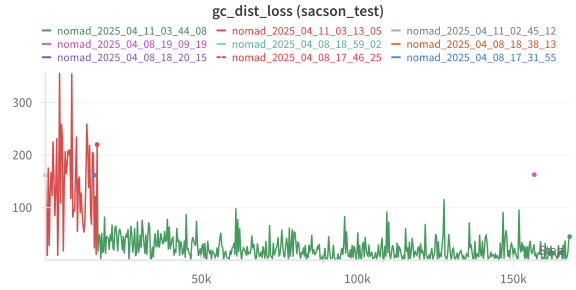


(a) Distance Loss on Training Set for ViNT



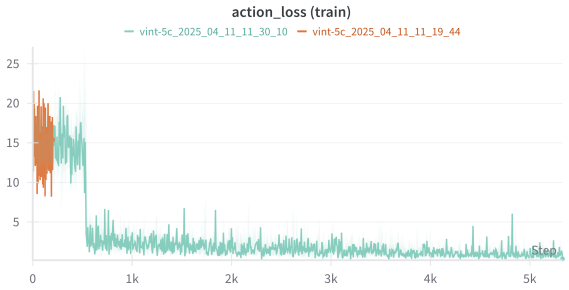
(b) Distance Loss on Training Set for NoMaD

(c) Distance Loss on Validation Set for ViNT

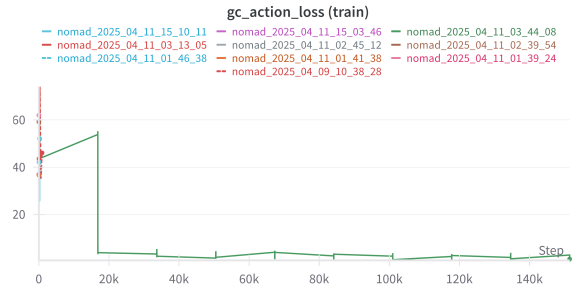


(d) Distance Loss on Validation Set for ViNT

Figure 14: Distance loss comparison between ViNT and NoMaD on training and validation sets.

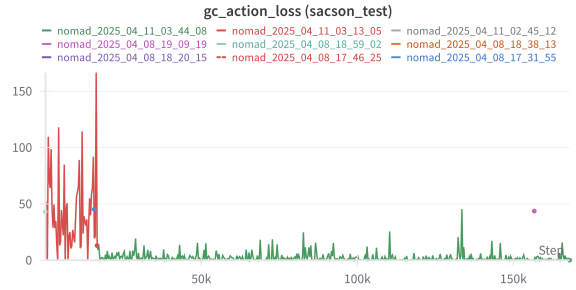


(a) Action Loss on Training Set for ViNT



(b) Action Loss on Training Set for NoMaD

(c) Action Loss on Validation Set for ViNT

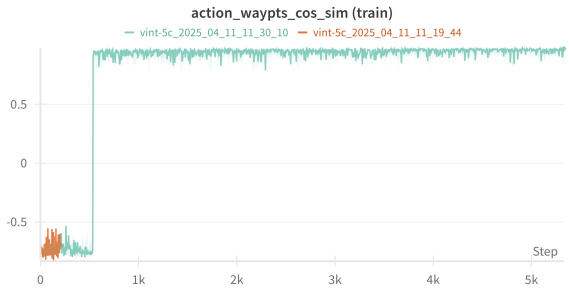


(d) Action Loss on Validation Set for NoMaD

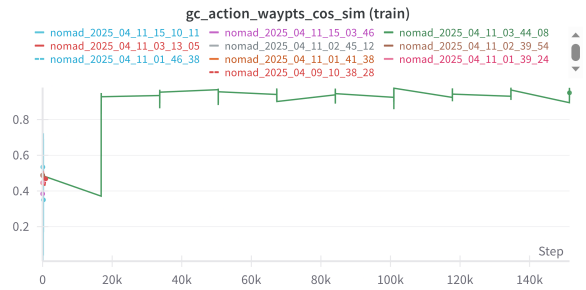
Figure 15: Action loss comparison between ViNT and NoMaD on training and validation sets.

Here are some plots from WandB comparing the two models(NoMaD and ViNT)

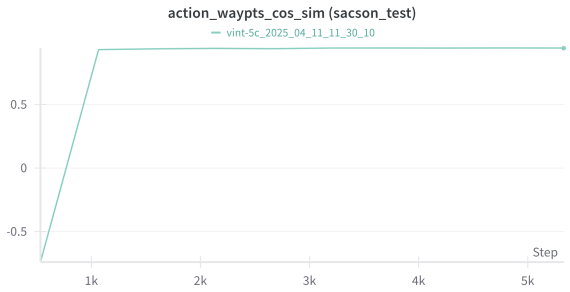
Action Cosine Similarity



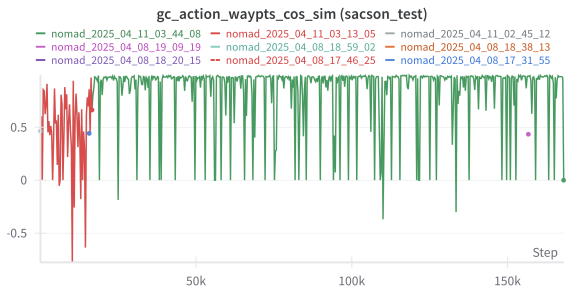
(a) ViNT Action Cosine Similarity on Training Set



(b) NoMaD Action Cosine Similarity on Training Set

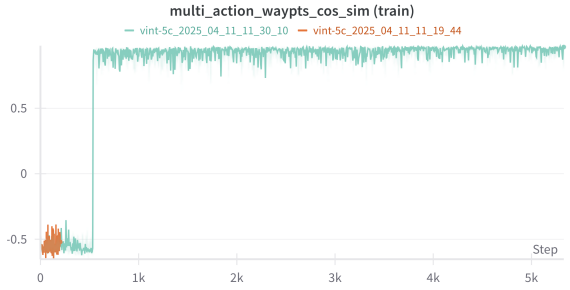


(a) ViNT Action Cosine Similarity on Validation Set

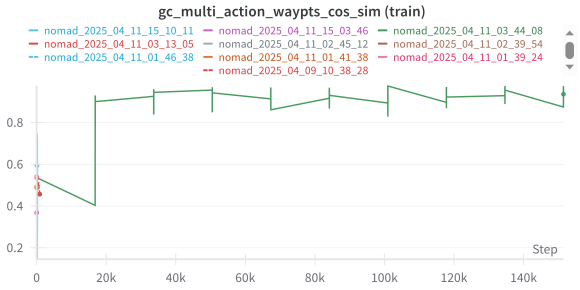


(b) NoMaD Action Cosine Similarity on Validation Set

Multi Action Cosine Similarity

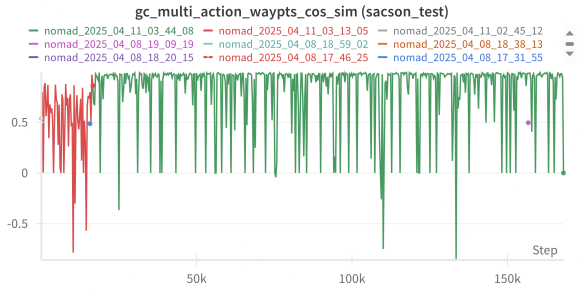


(a) ViNT Multi Action Cosine Similarity on Training Set



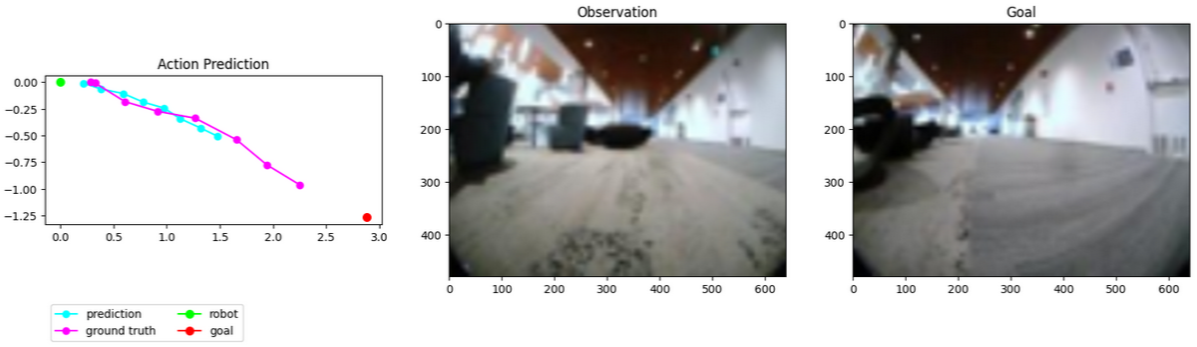
(b) NoMaD Multi Action Cosine Similarity on Training Set

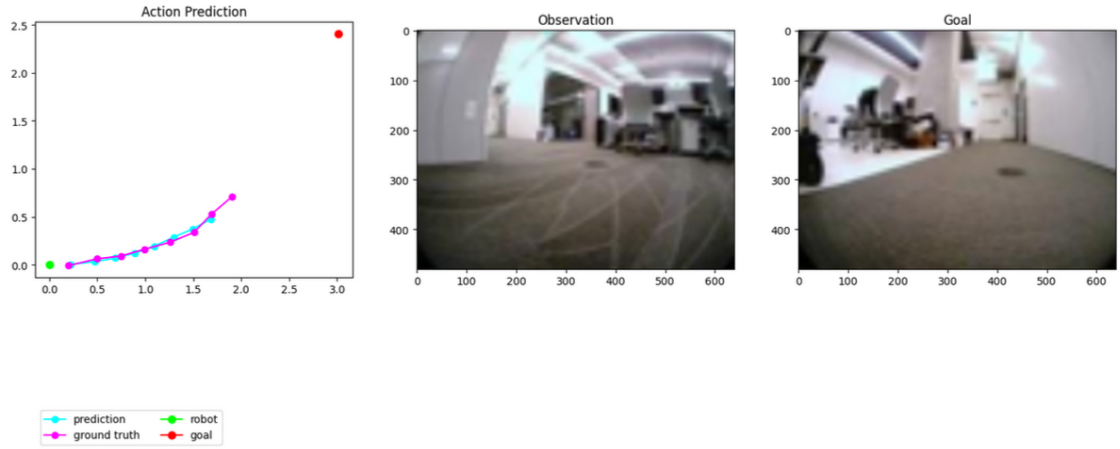
(a) ViNT Multi Action Cosine Similarity on Validation Set



(b) NoMaD Multi Action Cosine Similarity on Validation Set

Following are some of the visualizations of **ViNT** model's predictions on the validation set.





G Vision Transformer (ViT)

1. Core Architecture

ViT adapts the **Transformer** architecture, originally designed for NLP, to **image recognition** by treating images as sequences of patches.

(a) Patch Embedding

An input image $\mathbf{X} \in \mathbf{R}^{H \times W \times C}$ is split into N non-overlapping patches of size $P \times P$. Each patch is flattened into a 1D vector and linearly projected:

$$\mathbf{z}_i = \mathbf{E}\mathbf{x}_i + \mathbf{b}$$

where:

- $\mathbf{x}_i \in \mathbf{R}^{P^2 \times C}$ is the flattened patch
- $\mathbf{E} \in \mathbf{R}^{D \times P^2 C}$ is the embedding matrix
- D is the embedding dimension (e.g., 768)

For an image of size 224×224 and patch size 16×16:

$$N = \left(\frac{224}{16}\right)^2 = 196 \text{ patches}$$

(b) Class Token (CLS)

A learnable classification token $\mathbf{z}_{cls} \in \mathbf{R}^D$ is prepended:

$$\mathbf{Z} = [\mathbf{z}_{cls}; \mathbf{z}_1; \mathbf{z}_2; \ldots; \mathbf{z}_N]$$

(c) Positional Embeddings

Learned positional embeddings are added to retain spatial information:

$$\mathbf{Z} = \mathbf{Z} + \mathbf{E}_{pos}, \quad \mathbf{E}_{pos} \in \mathbf{R}^{(N+1) \times D}$$

2. Transformer Encoder

The ViT encoder consists of **L identical layers**, each containing:

- Multi-Head Self-Attention (MHSA)
- Layer Normalization (LN)
- Feed-Forward Network (FFN)
- Residual Connections

(a) Multi-Head Self-Attention (MHSA)

Input embeddings are split into h heads (e.g., 12 heads for ViT-Base). For each head:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

where $d_k = D/h$.

(b) Feed-Forward Network (FFN)

A 2-layer MLP with GELU activation:

$$\text{FFN}(\mathbf{x}) = \mathbf{W}_2\text{GELU}(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$$

where:

- $\mathbf{W}_1 \in \mathbf{R}^{4D\times D}$
- $\mathbf{W}_2 \in \mathbf{R}^{D\times 4D}$

(c) Layer Normalization & Residual Connections

Each sub-layer is wrapped with:

$$\mathbf{x}_{\text{out}} = \text{LayerNorm}(\mathbf{x} + \text{SubLayer}(\mathbf{x}))$$

3. Training & Fine-Tuning

- **Pre-training**: On large datasets (e.g., JFT-300M)
- **Fine-tuning**: Adapt to downstream tasks by replacing classification head

H EfficientNet

1. Compound Scaling

EfficientNet introduces **compound scaling**:

$$d = \alpha^\phi, \quad w = \beta^\phi, \quad r = \gamma^\phi$$

where:

- ϕ is a scaling coefficient
- α, β, γ are constants ($\alpha = 1.2, \beta = 1.1, \gamma = 1.15$)

Constraint:

$$\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

2. MBConv Block

The core building block consists of:

- 1×1 Expansion Conv (expansion factor $t = 6$)
- Depthwise Conv
- Squeeze-and-Excitation (SE) Block
- 1×1 Projection Conv

(a) Depthwise Separable Convolution

- **Depthwise Conv**:

$$\mathbf{y}_{i,j,k} = \sum_{m,n} \mathbf{K}_{m,n,k} \mathbf{x}_{i+m,j+n,k}$$

- **Pointwise Conv**:

$$\mathbf{z}_{i,j,l} = \sum_{k} \mathbf{W}_{k,l} \mathbf{y}_{i,j,k}$$

(b) Squeeze-and-Excitation (SE)

- **Squeeze**: Global average pooling $\rightarrow \mathbf{s} \in \mathbf{R}^C$
- **Excitation**:

$$\mathbf{s}_{excite} = \sigma(\mathbf{W}_2 \text{ReLU}(\mathbf{W}_1 \mathbf{s}))$$

- **Rescale**:

$$\mathbf{x}_{out} = \mathbf{s}_{excite} \odot \mathbf{x}$$

3. Neural Architecture Search (NAS)

EfficientNet-B0 was discovered using:

$$\text{maximize ACC}(m) \times \left[\frac{\text{FLOPS}(m)}{T}\right]^w$$

where:

- $w = -0.07$
- $T = 400M$ (target FLOPS for B0)

I Diffusion Policy: Visuomotor Policy Learning via Action Diffusion

[4] **Diffusion Policy**, a novel approach for generating robot behavior by representing a robot's visuomotor policy as a conditional denoising diffusion process. This method leverages the strengths of diffusion models to address challenges in robot policy learning, such as handling multimodal action distributions, high-dimensional action spaces, and training stability. The authors benchmark Diffusion Policy across 15 tasks from 4 different robot manipulation benchmarks, demonstrating an average improvement of 46.9% over existing state-of-the-art methods.
Traditional policy learning from demonstrations faces challenges due to:

- **Multimodal action distributions**: Multiple valid actions for the same observation.
- **Sequential correlation**: Actions are temporally dependent.

- **High precision requirements**: Robot tasks often require precise control.

Prior work has explored various action representations (e.g., mixtures of Gaussians, categorical representations) and policy representations (e.g., explicit vs. implicit policies). However, these methods often struggle with stability, scalability, or expressiveness.

Diffusion Policy Formulation

Diffusion Policy is based on Denoising Diffusion Probabilistic Models (DDPMs), adapted for robot visuomotor control. The key modifications include:

- **Action-space diffusion**: The output $\mathbf{x}$ represents robot actions instead of images.
- **Conditional denoising**: The denoising process is conditioned on input observations $\mathbf{O}_t$.

The policy predicts action sequences using a receding horizon control strategy:

- At time t, the policy takes the latest T_o steps of observations and predicts T_p steps of actions.
- Only T_a steps are executed before re-planning, balancing temporal consistency and responsiveness.

Training

The training process involves:

1. Sampling an action $\mathbf{A}_t^0$ from the dataset.
2. Adding noise $\boldsymbol{\varepsilon}^k$ to create a noisy action $\mathbf{A}_t^k$.
3. Training the noise prediction network ϵ_θ to estimate $\boldsymbol{\varepsilon}^k$ given $\mathbf{O}_t$ and $\mathbf{A}_t^k$.

The loss function is:

$$\mathcal{L} = \text{MSE}(\boldsymbol{\varepsilon}^k, \epsilon_\theta(\mathbf{O}_t, \mathbf{A}_t^0 + \boldsymbol{\varepsilon}^k, k)).$$

Network Architectures

Two architectures are explored:

- **CNN-based**: Uses 1D temporal CNNs with FiLM conditioning for observation features. Suitable for most tasks but struggles with high-frequency action changes.
- **Transformer-based**: Introduces a time-series diffusion transformer to reduce oversmoothing. Performs better for complex tasks but requires more tuning.

Visual Conditioning

Visual observations are encoded independently for each timestep and concatenated into a latent embedding $\mathbf{O}_t$. A ResNet-18 encoder is used with modifications:

- Spatial softmax pooling replaces global average pooling.
- GroupNorm replaces BatchNorm for stable training.

Noise Schedule

The square cosine schedule from iDDPM is used to control the noise level during denoising, balancing high- and low-frequency action characteristics.

Inference Acceleration

Denoising Diffusion Implicit Models (DDIM) reduce inference iterations from 100 to 16, enabling real-time control with low latency (0.1s on an NVIDIA 3080 GPU).

Multimodal Action Distributions

Diffusion Policy naturally handles multimodal action distributions by:

- Stochastic initialization: Starting from random noise allows exploration of different modes.
- Stochastic optimization: Gaussian perturbations during denoising enable transitions between modes.

The method scales well to high-dimensional action spaces, enabling joint prediction of action sequences for long-horizon planning. Unlike implicit policies that require negative sampling, Diffusion Policy learns the score function directly, avoiding instability issues. Diffusion Policy performs better with position control than velocity control, contrasting with prior work that favors velocity control.
The authors evaluate Diffusion Policy on 15 tasks across 4 benchmarks:

- **Robomimic**: 5 manipulation tasks with human demonstrations.
- **Push-T**: A planar pushing task with RGB or keypoint observations.
- **Multimodal Block Pushing**: Tests long-horizon multimodality.
- **Franka Kitchen**: A multi-task environment with 7 objects.

Key Findings

- Diffusion Policy outperforms baselines (LSTM-GMM, IBC, BET) by 46.9% on average.
- Handles both short- and long-horizon multimodality effectively.
- Stable training with consistent hyperparameters across tasks.

J WandB

These are our initial plots of `distance_loss`, `diffusion_loss` and `total_loss`.

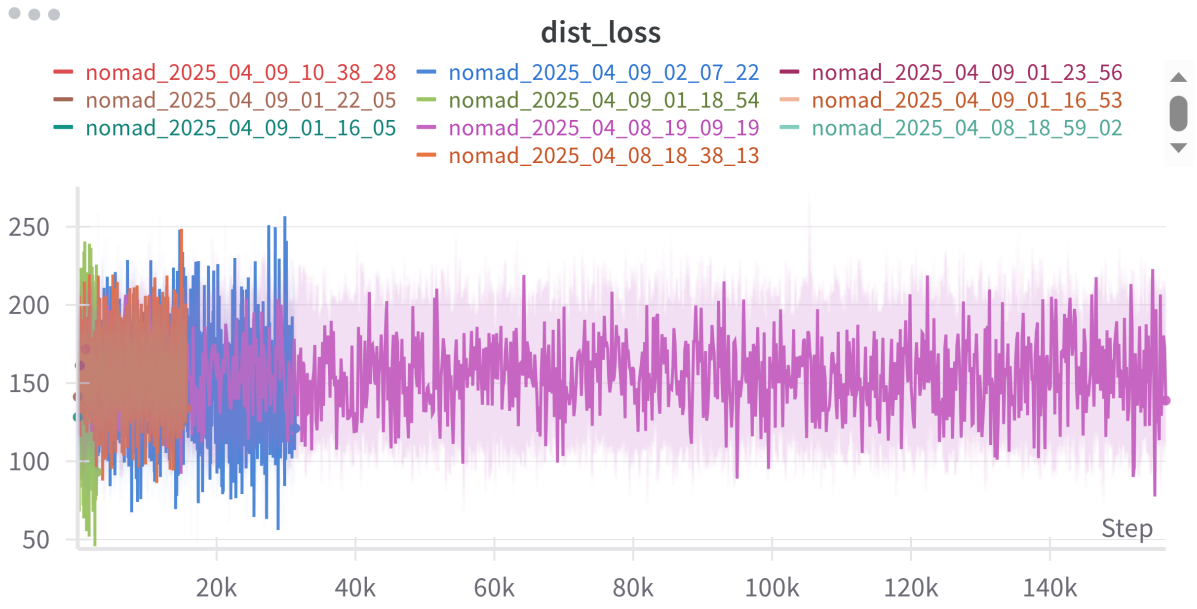


Figure 20: Initial plot of distance Loss on Training Set

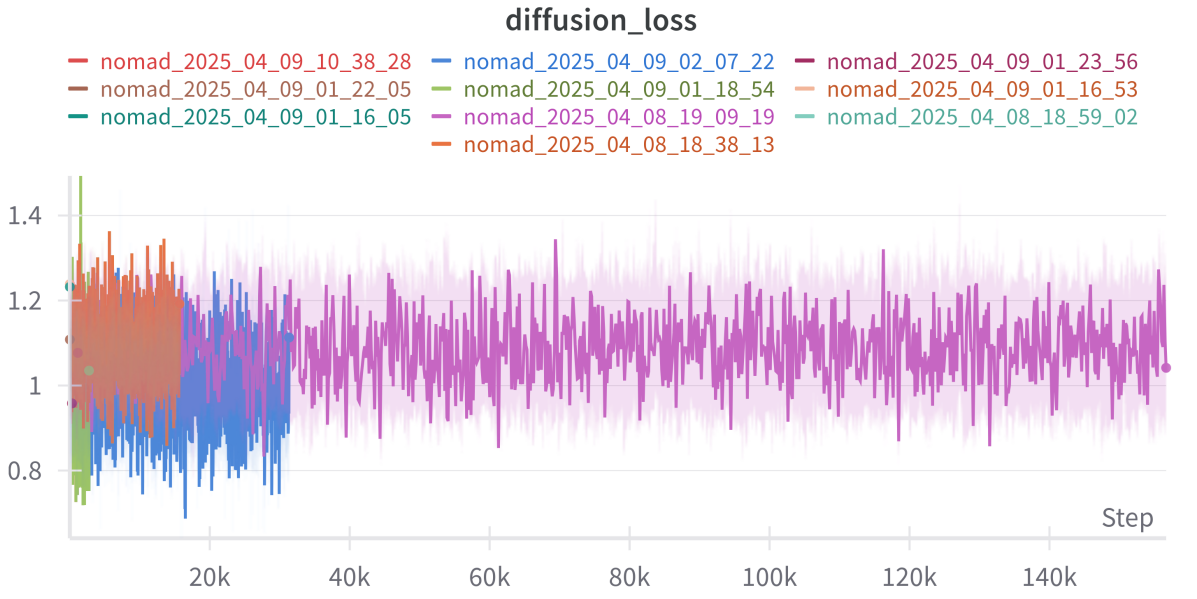


Figure 21: Initial plot of diffusion Loss on Training Set

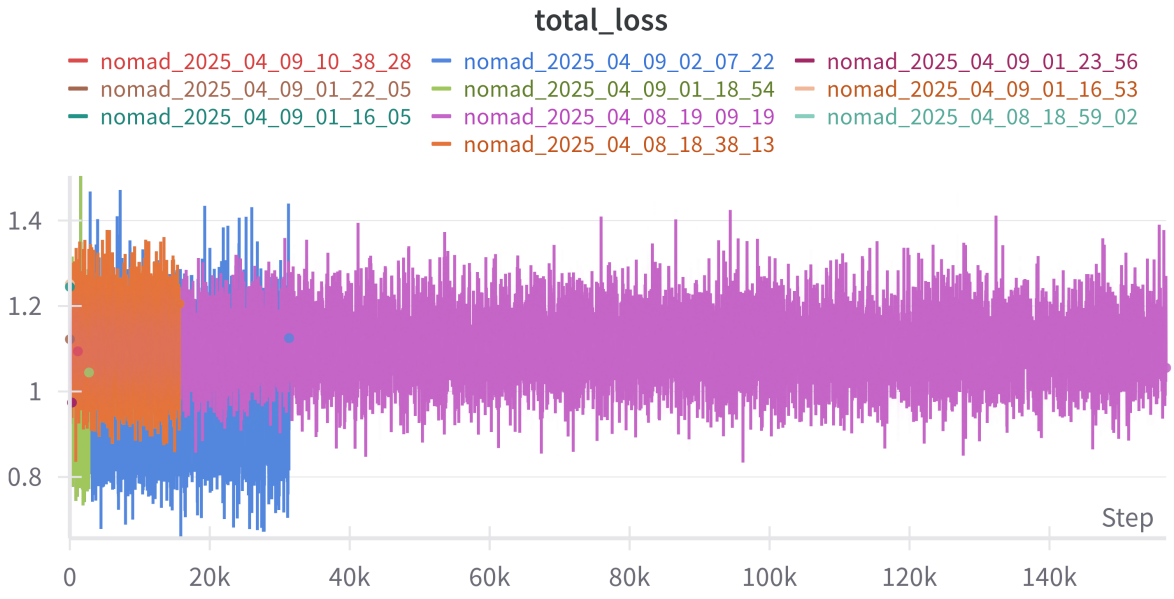


Figure 22: Initial plot of total Loss on Training Set

It is somewhat surprising that the total distance20, diffusion losses 21 as well as total loss 22 do not show the expected downward trend. However, other loss curves clearly demonstrate a consistent decrease, indicating that the model was indeed able to learn and generalize effectively. Upon investigation, we discovered that the overall loss trend was skewed due to a specific run that encountered bugs in gradient propagation. As a result, the model in that particular run failed to learn properly. This anomaly in the plot was crucial in helping us identify the need for debugging within the training pipeline.
While the total loss from correctly functioning runs was not plotted separately on Weights & Biases (W&B), the consistent downward trends in individual components,such as distance and action losses,strongly indicate effective learning in those cases. To confirm this, we reran the training with a fresh model using the same parameters and dataset. The final plots included in this report are from this corrected run. We retained the original auxiliary plots, as their results were consistent with the corrected run.

K Team Contributions

Abhishek Kumar Jha:

- Implemented the Nomad Architecture and helped in the Training and Experiments
- Made the **README** file for the repository.
- Assisted in creating the report and Presentation.
- Contributed towards making the code demo video.
- Helped in integrating and debugging the diffusion_policy repository

Namashivayaa V:

- Explained the workflow of the suggested papers to the entire team.
- Contributed in report formation, by adding important formulas and results in the appendix section.
- Extracted the .bag files which were around 300GB.

Sehaj Ganjoo:

- Implemented the ViNT architecture and trained it
- Helped in debuggeding the errors while training the NoMaD model
- Assisted in conducting experiments and analysis for the project
- Contributed towards the Presentation and project report

Shobhnik Kriplani:

- Made a website for the project
- Implemented the NoMaD architecture
- Developed the training pipeline and contributed towards data extraction
- Conducted experiments and analysis

L Challenges and Debugging

- CUDA out of memory errors were common during training, especially with larger batch sizes. We mitigated this by reducing the batch size and using gradient accumulation.
- The model was initially not being able to learn effectively, leading to high loss values. We debugged this by checking the data pipeline and ensuring that the input images were correctly preprocessed and normalized.
- The system in UG computational labs didnot have ROS installed, which was required to process the datasets. With the help of our TA, we used a docker file to set up a container with ROS and the required dependencies.
- The original codebase has several data-type related bugs which had to be fixed. For example, the original code was using torch.float32 for some tensors, while the model expected them to be in torch.float64. This caused several errors during training.
- The initial implementation caused several system crashes due to heavy computational load. We had to reduce evalutation batchsize and reduce the number of epochs to avoid this.

References

[1] Dhruv Shah et al. *NoMaD: A Diffusion Policy for Visual Navigation.* 2023 arXiv:2310.07896

[2] Diffusion Policy GitHub Repository:

[3] NoMaD GitHub Repository

[4] Cheng Chi et al. *Diffusion Policy: Visuomotor Policy Learning via Action Diffusion.* 2023 arXiv:2303.04137

[5] Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.* 2021. arXiv:2010.11929

[6] Tan & Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.* 2019. arXiv:1905.11946

[7] Dhruv Shah et al., *ViNT: A Foundation Model for Visual Navigation* 2023 arXiv:2306.14846