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 13, 2025

Abstract

This report presents our work on implementing and analyzing the training and perception components of a visual navigation system based on diffusion policies, as adapted from the NOMAD (Navigation with Goal Masked Diffusion) framework. Our approach combines a visual perception backbone based on EfficientNet and Transformer, with a trajectory diffusion model for motion planning to support both goal-directed and exploratory navigation. We trained our model on the SACSon,RECON and go_stanford dataset, which features diverse real-world trajectories across various environments and robot platforms. We leverage a conditional diffusion model to generate multimodal waypoint predictions, enabling the agent to reason about complex, uncertain navigation scenarios. Our contributions include a detailed breakdown of the model architecture, training methodology, and evaluation metrics—particularly focusing on waypoint alignment through cosine similarity. We have left out the deployment aspects.

1 Introduction

Robotic learning for navigation in unfamiliar environments requires the ability to perform both task-oriented navigation (i.e., reaching a known goal) and task-agnostic exploration (i.e., searching for a goal in a novel environment). Traditionally, these functionalities are tackled by separate systems — for example, using subgoal proposals, explicit planning modules, or distinct navigation strategies for exploration and goal-reaching.

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 involves implementing the NoMaD Policy adapting its Transformer-based architecture and conditional diffusion decoder to learn from a rich, multimodal dataset (SACSoN,SCAND,go_stanford,RECON) composed of real-world trajectories. Unlike traditional latent-variable models or methods that rely on separate generative components for subgoal planning, the unified diffusion policy exhibits superior generalization and robustness in unseen environments, while maintaining a compact model size. In this report, we focus on the perception and training components of this policy, emphasizing how a strong visual encoder combined with a diffusion-based decoder leads to improved alignment of predicted and ground-truth waypoints. We analyze the training dynamics, present key quantitative metrics such as cosine similarity and distance loss, and highlight the model's ability to generalize across diverse scenarios.

2 Methodology

The NoMaD (Navigation with Goal-Masked Diffusion) framework introduces a unified visual navigation policy capable of both goal-conditioned navigation and open-ended exploration within a single architecture. Building on the Visual Navigation Transformer (ViNT) [**?**] and diffusion-based policy learning (see Appendix **??**), 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

Figure 1 illustrates the NoMaD architecture, which consists of three main components:

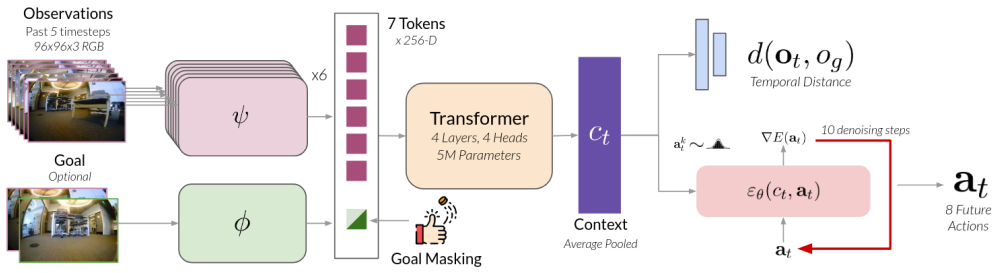


Figure 1: NoMaD architecture. The visual encoder processes RGB observations, the transformer maintains temporal context, and the diffusion decoder generates waypoint predictions. Goal masking occurs in the attention layers of the context encoder.

- **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

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 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.

C. Implementation Details

2.1 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.
The codebase was adapted from an open-source repository, and experiment tracking was performed using `Weights & Biases (WandB)` for logging loss curves, evaluation metrics, and hyperparameter sweeps.

2.2 Data Pipeline

We trained the NoMaD model using the **SACSoN** and parts of **RECON** dataset, which contains diverse real-world trajectories across various environments and robot platforms. The SACSoN dataset was 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.

2.3 Training Procedure

Training was done on a single NVIDIA GPU using a batch size of 32 for 10 epochs.
Training Objective:
We calculate the Mean Squares Error(MSE) between the predicted and the actual noise. For overall loss function, we add the MSE temporal distance loss to the objective function.

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

where:

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

Training Configuration:

- 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 $1e^{-2}$.
- 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.

2.4 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.

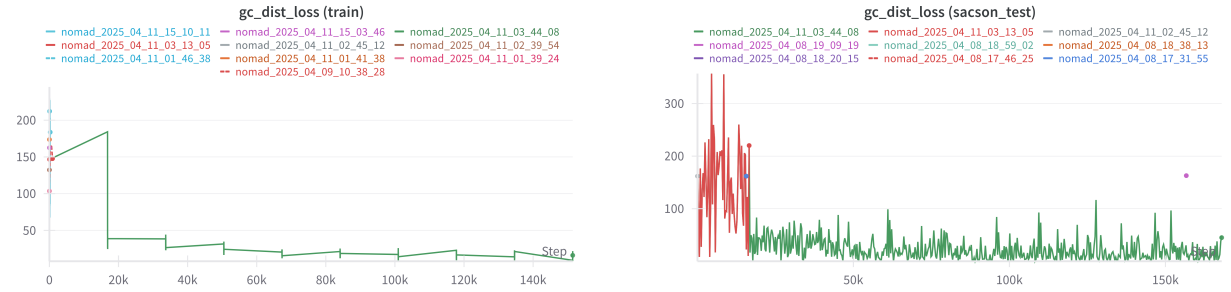
Evaluation was performed on the validation set using the following metrics:

- distance loss
- diffusion loss
- cosine similarity
- action loss

3 Experiments

Goal-Conditioned (GC) Evaluation

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



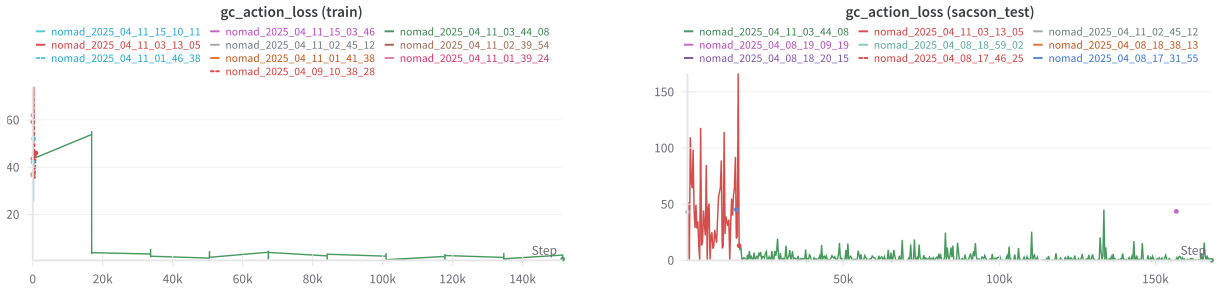
(a) Distance Loss on Training Set

(b) Distance Loss on Validation Set

Figure 2: 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 the model has learned to predict the distance to the goal.

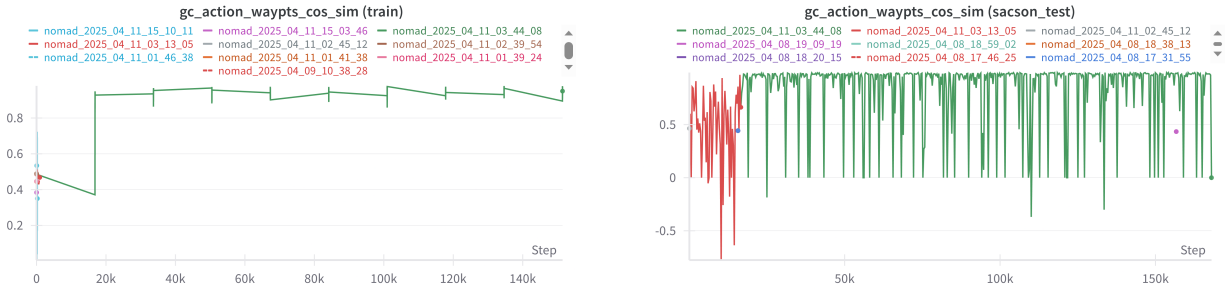
Action loss:



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

Figure 3: Action loss comparison between training and validation sets under goal-conditioned evaluation.

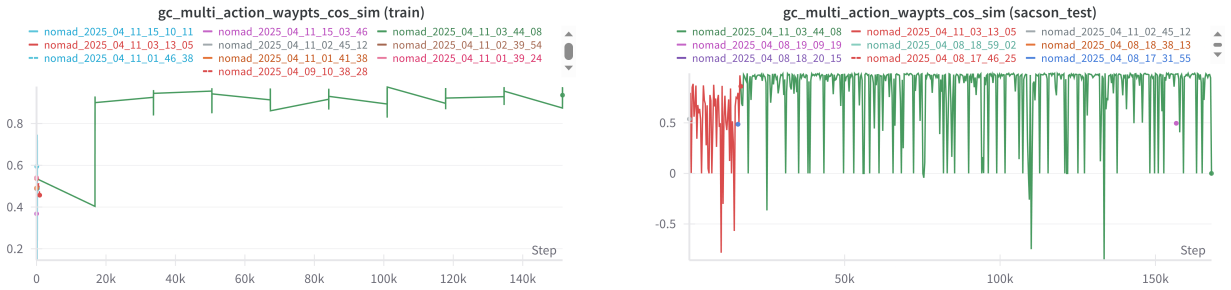
gc_action_waypts_cos_sim:



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

Figure 4: Cosine similarity between predicted and ground-truth waypoints on the training and validation sets under goal-conditioned evaluation.

gc_multi_action_waypts_cos_sim:



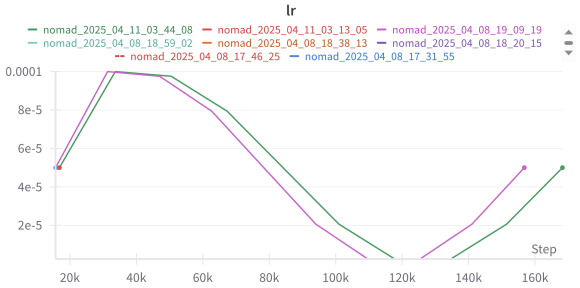
(a) Cosine Similarity on Training Set (b) Cosine Similarity on Validation Set

Figure 5: Cosine similarity between predicted and ground-truth multi-waypoints on the training and validation sets under goal-conditioned evaluation.

Distance Loss:

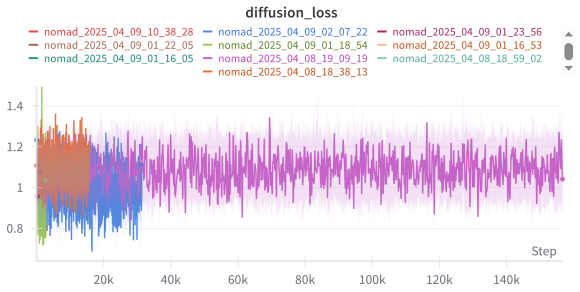
(a) Distance Loss on Training Set

Learning Rate:

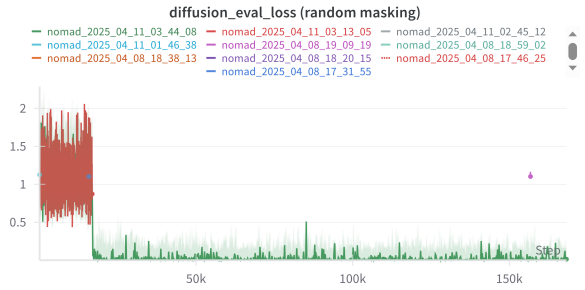


(a) Learning Rate on Training Set

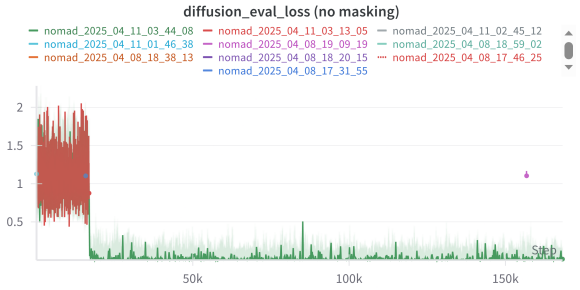
Diffusion Loss:



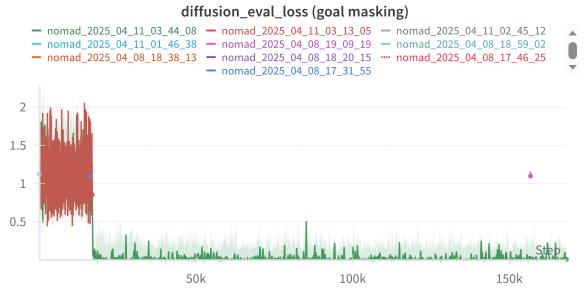
(a) Diffusion Loss on Training Set



(b) Diffusion evaluation loss with random masking



(c) Diffusion evaluation loss without masking



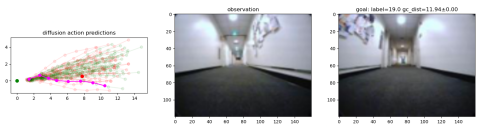
(d) Diffusion evaluation loss with goal masking

Figure 8: Diffusion loss on training set and evaluation loss with different masking strategies

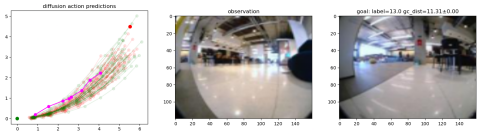
Total Loss:

(a) Total Loss on Training Set

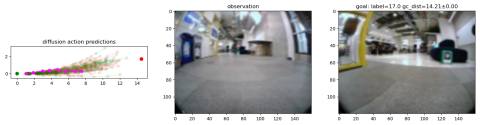
3.1 Train action samples:



(a) Sample actions generated by the model during training.



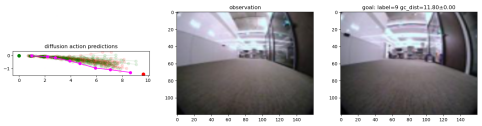
(b) Sample actions generated by the model during training.



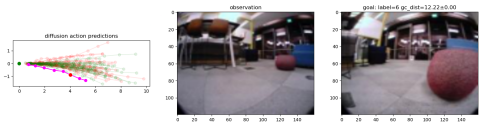
(c) Sample actions generated by the model during training.

(d) Sample actions generated by the model during training.

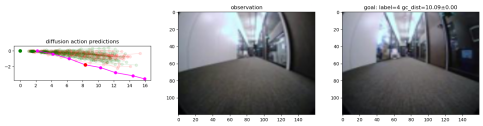
3.2 Sacson test action samples:



(a) Sample actions generated by the model during testing.



(b) Sample actions generated by the model during testing.



(c) Sample actions generated by the model during testing.

Comparison with ViNT

To establish a baseline, we trained the original ViNT architecture using the same dataset and training hyperparameters as used for NoMaD.
We used the same EfficientNet-B0 encoder and transformer architecture, but without the diffusion decoder.
We evaluated both models 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.
Interestingly, we observed that in the **goal-conditioned (GC)** setting, both ViNT and NoMaD achieved comparable performance across key metrics. This suggests that the introduction of a diffusion-based decoder in NoMaD does not degrade performance in goal-directed planning tasks. Instead, it provides a more expressive generative mechanism while maintaining task performance.
This supports the hypothesis that **diffusion-based modeling can match deterministic approaches in accuracy while offering better generative flexibility**, particularly for multi-modal or long-horizon planning scenarios.

3.3 Training Metrics

- Final training loss: 0.003
- Cosine similarity: 0.47 (multi-action waypoints)
- Distance loss: 10.2

3.4 Observations

Loss plateaued after around 5,000 batches. Training logs show improvement in cosine similarity and reduction in loss. Action losses remained stable across UC and GC branches.

4 Challenges and Debugging

5 Conclusion and Future Work

We successfully trained the NOMAD policy and analyzed the perception module. Future work could involve domain randomization, hyperparameter tuning, and evaluating transfer to real-world or simulated environments.

References

1. H. Janner et al., "NOMAD: Planning with Diffusion for Visual Navigation," 2022.
2. Diffusion Policy GitHub Repository: `https://github.com/wayveai/diffusion-policy`

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 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

D 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)

References

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

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