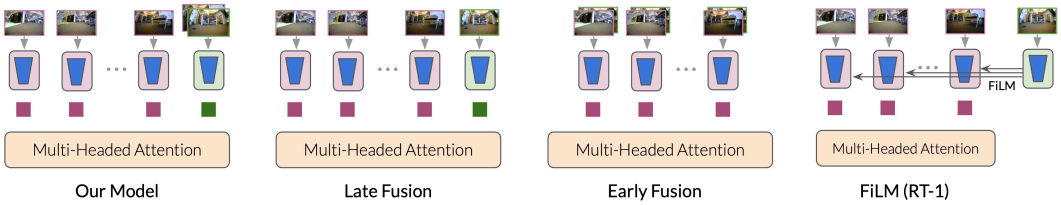

Figure 8: Different goal fusion architectures considered for ViNT.

Appendix

A ViNT Model Architecture

Table 5 shows the model architecture in detail. We use all 18 layers of the EfficientNet-B0 convolutional encoder [31], initialized from scratch, to tokenize the input and subgoal observations. For fusing the subgoal image with the current observations, we use *early fusion*: the two RGB images o_t and o_G are stacked along their channel dimension, yielding a $6 \times 85 \times 64$ tensor per training data point. The stacked subgoal tensor is encoded using the EfficientNet encoder and tokenized with a 512-dimensional embedding. To encode a stacked tensor, we modify the first convolution to accept the desired number of input channels.

Layer	Input [Dimensions]	Output [Dimensions]	Layer Details
1	o_t, o_g [64, 85, 3]	I_t^g [64, 85, 6]	Concatenate along channel dimension.
2	I_t^g [64, 85, 6]	E_t^g [1, 1000]	Fusion EfficientNet Encoder [31]
3	$o_{t:t-P}$ [P+1, 64, 85, 3]	$E_{t:t-P}$ [P+1, 1000]	Context EfficientNet Encoder [31]
4	E_t^g [1, 1000]	$E_t^{g'}$ [1, 512]	Fusion Encoding Compression MLP
5	$E_{t:t-P}$ [P+1, 1000]	$E'_{t:t-P}$ [P+1, 512]	Context Encoding Compression MLP
6	$E'_{t:t-P}$ [P+1, 512], $E_t^{g'}$ [1, 512]	S [P+2, 512]	Concatenate into sequence
7	S [P+2, 512]	$\tilde{S}$ [1, 32]	Feed into Transformer Decoder
8	$\tilde{S}$ [1, 32]	D	Predict distance with MLP head
9	$\tilde{S}$ [1, 32]	A, [1, T, 4]	Predict odometry and angles with MLP head

Table 5: Architectural Details of ViNT The inputs to the model are RGB images $o_{t:t-P} \in [0,1]^{P \times 3 \times 85 \times 64}$ and $o_g \in [0,1]^{3 \times 85 \times 64}$, representing the current, past, and goal images. We seek to predict a T timestep horizon of positions (x, y), angles (sin, cos) and a distance.

Goal fusion architectures: There are many considerations for when and where to combine the input data for effective feature extraction and task performance as shown in Table 6. While FiLM works well for language, we find that training was unstable for image-navigation. As a result, we consider directly encoding image information and passing them to a Transformer. Fusing the goal with observation information in the transformer layers by concatenating separately encoded tokens is known as "late fusion". Late fusion would work effectively for adaptation as there would be a single goal token that's independently encoded. As a result, any input data could become that token. But, late fusion tends to perform poorly as the model requires joint features between current state and goal state to be passed to the Transformer. If independently encoded, the Transformer will only be able to attend to independently extracted features. "Early fusion", on the other hand, will fuse the goal and observation input before the EfficientNet encoding, allowing the learning of joint features between the goal image and current state. This heavily increases performance, but at the cost of making adaptation not possible. This is because the goal information is now forced to become an image to be concatenated to the observation for creating the final token. With ViNT, this can be bypassed though via the adaptation process described in B.4. Specifically, we can perform task adaptation by predicting the final token conditioned on input data and task goal information and directly cutting out the goal encoder.

Method	Performance	Adaptation
Late Fusion	✗	✓
Early Fusion	✓	✗
FiLM (RT-1) [27]	✗	✓
ViNT	✓	✓

Table 6: Results of different goal fusion architectures.

B Implementation Details

B.1 Training ViNT

See Table 7 for a detailed list of hyperparameters for training the ViNT foundation model.[1]

Training objective:

Hyperparameter	Value
ViNT Model	
# Parameters	31M
Resolution	85×64
Encoder	EfficientNet-B0
Token dim.	512
Attn. hidden dim.	2048
# Attn. layers n_L	4
# Attn. heads n_H	4
FF dim	??
Temporal context P	5
Prediction horizon H	5
MLP layer sizes	(256, 128, 64, 32)
ViNT Training	
# Epochs n_{ep}	30
Batch size	300 [1]
Learning rate	5×10^{-4}
Optimizer	AdamW
Warmup epochs	4
Scheduler	Cosine
Scheduler Period	10
Compute Resources	8×V100
Training Time	1 - 2 days
Diffusion Model	
# Parameters	318M
Resolution	128×128
# Up/Down Blocks	4
Attn. Resolutions	32, 16, 8
Layers per Block	2
Attn. Head Dim	8
Channels	128, 128, 256, 512, 640
Diffusion Type	continuous time
Noise Schedule	linear

Hyperparameter	Value
Diffusion Training	
Dropout	0.1
Batch Size	128
Optimizer	AdamW
Warmup Steps	1000
Learning Rate	1e-4
LR Schedule	cosine decay
Adam β_1	0.95
Adam β_2	0.999
Adam ϵ	1e-8
Weight Decay	0.001
EMA Inv. Gamma	1.0
EMA Power	0.75
EMA Max Decay	0.9999
CFG Mask Proportion	0.2
Train Steps	250,000
Training Time	30 hours
Compute Resources	1×v4-8 TPU board
Diffusion Sampling	
Sampler	DDIM [42]
DDIM η	0.0
Sampling Steps	200
Guidance Weight	1.0
Other	
Maximum distance	20
Distance tradeoff λ	0.01

Table 7: Hyperparameters for core components of ViNT

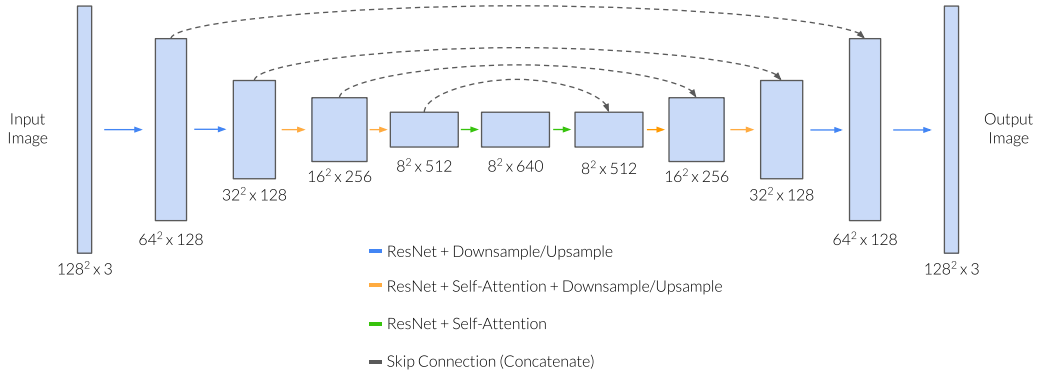

Figure 9: Subgoal diffusion model U-Net architecture. Each ResNet consists of 2 residual blocks. Downsampling and upsampling is done with strided convolutions.

B.2 Subgoal Diffusion

For generating subgoals, we use an image-to-image diffusion model. It takes an image o_t as input and produces samples from $g(s_i \mid o_t)$, where s_i are candidate subgoal images reachable from o_t. To produce training pairs for the diffusion model, we first select o_t uniformly at random from the training data and then select s_i to fall between 5 and 20 timesteps in the future from o_t.

Following Saharia et al. [34], we implement image conditioning as simple channel-wise concatenation to the U-Net input. We use the Flax U-Net implementation from the diffusers library [43] with textual cross-attention removed since we do not condition on text inputs.

We use the continuous-time diffusion formulation from Kingma et al. [44] with a fixed linear noise schedule rather than a learned one. Also unlike Kingma et al. [44], we use the unweighted training objective, called L_{simple} in Ho et al. [33, Equation 14] and Kingma et al. [44, Appendix K]. We employ classifier-free guidance [45] and find that it helps produce subgoals with better visual fidelity, which is consistent with prior work [46].

B.3 Long-Horizon Physical Search via Topological Graphs

As in Shah and Levine [26], we implement physical search similarly to a standard A* algorithm, by keeping track of an open set Ω of possible unvisited subgoals (generated by our diffusion model) and following Alg. 1.

Algorithm 1: Long-Horizon Navigation via Topological Graph

while goal G not reached **do**
  $s \leftarrow \min_f(\Omega)$;
  $P \leftarrow$ ShortestPath($\mathcal{M}$, o_t, s^-)
  for (s, s') in P **do**
    ViNT.GoToGoal(s');
  end for
  ViNT.GoToGoal(s)
  $o_t \leftarrow$ Observe();
  AddNode($\mathcal{M}$, o_t, parent: s^-);
  Sample $s_i \sim g(s_i|o_t)$;
  Add(Ω, s_i);
end while

[1] Over the course of the project, we used a variety of workstations equipped with different GPU configurations, including 3×1080Ti, 2×4090, 4×P100, 8×V100, and 8×A100. With the model architecture fixed, the batch size and training time varies significantly across these devices, and the entry in Table 7 is representative of our most common training configuration.

Nodes are visited according to a costing function $f(s)$ that depends on the distance from the current state o_t to the parent node s^- (measured along the graph), the predicted distance from s^- to s, and a heuristic function h (similar to that of A*) providing long-horizon navigation hints:

$$f(s) = d_{\mathcal{M}}(o_t, s^-) + d_{\text{pred}}(s^-, s) + h(s, G, C)$$

In general, the heuristic can be any function providing a notion of distance between a subgoal s and the long-horizon goal G, optionally with some context C. For our experiments, we considered three heuristics to demonstrate the flexibility of our approach:

- **Coverage maximization:** For undirected exploration we have no long-horizon goal and thus use $h(s) = 0$.
- **Position-guided:** For long-horizon GPS goals (outdoors) and 2D position goals (indoors), we use Euclidean distance $h(s) = \|s - G\|$.
- **Satellite-guided:** In the context-guided experiments, we train a learned heuristic function that uses the satellite image as an input to learn a a heuristic for "good" subgoals. We train a convolutional neural network on the overhead image to predict the probability that the subgoal s is included on a trajectory from o_t to G, trained using a contrastive objective [47]. Additional information can be found in Shah and Levine [26].

B.4 Fine-tuning ViNT

Image Fine-tuning:

- **Architecture:** We utilize the exact same architecture as ViNT with no changes.
- **Training:** For fine-tuning the image-goal directed model, we utilize the same training process for ViNT with a learning rate of 0.0001, AdamW optimizer, but no warmup or cosine scheduler. We train with a batch size of 200 across 8 GeForce 1080 Ti GPUs for all fine-tuning and adaptation experiments and we do not mix any prior data for fine-tuned training.

GPS-Adaptation:

- **Architecture:** To adapt to GPS-style goals, we cut off the goal encoder block from ViNT. We then learn a fixed tensor of size 3000 and concatenate it to the GPS-command goal in ego-centric coordinates. We then pass this into a 2-layer MLP which outputs the prediction of the final token for the transformer. The architecture is shown in Figure 10.
- **Training:** During training, instead of randomly sampling future images and passing them into the goal encoder, we sample future odometry information. Once we have a future goal coordinate for self-supervision, we convert to local coordinates and pass into our architecture and optimize the same objective as ViNT. We use a cosine scheduler with a learning rate warmup to 0.0001 for 4 epochs. We also sample goal points from between 1.25s and 1.75s rather than from 0.5s to 2.5s.

Command-Adaptation:

- **Architecture:** For discrete command goals, we adopt a similar approach for GPS-style goals. We learn a fixed tensor for each discrete command and use the command index to select the corresponding latent to pass into a 2-layer MLP for predicting the final token. In this way, we learn a dictionary of latents, each corresponding to a distinct command. This architecture is illustrated in Figure 10 as well.
- **Training:** If training data is labelled with command, we just supply the command and run standard training. We assume training data is not labelled with the discrete command, so we label the trajectories with commands on-the-fly. For our experiments, we use "left", "right", and "straight" as our discrete commands. We sample a future odometry point just like the GPS Adaptation, but rather than using the point itself, we use the lateral coordinates to determine what command that training example corresponds to. For our experiments we bin each trajectory that has a normalized lateral coordinate greater than 0.05, otherwise it is labelled as "straight". We use a cosine scheduler with a learning rate warmup to 0.0001 for 4 epochs.

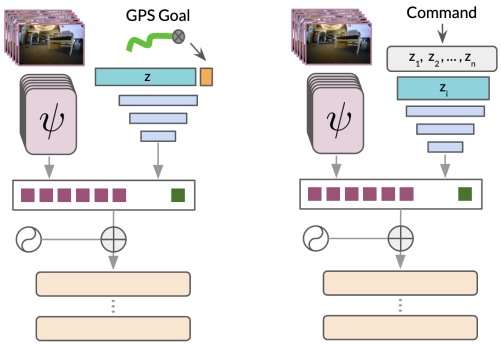

Figure 10: Different adaptation architectures for ViNT. Shown on the left is the GPS-adaptation architecture. As seen, the local coordinates of the goal are concatenated to the fixed latent, z. On the right is the command-adaptation architecture where we assume command index i is given, hence sending z_i to the MLP.

	Dataset	Platform	Speed	Total Hrs.	Hrs. Used	Environment
1	GoStanford [48]	TurtleBot2	0.5m/s	17h	14h	office
2	RECON [38]	Jackal	1m/s	25h	25h	off-road
3	CoryHall [37]	RC Car	1.2m/s	2h	2h	hallways
4	Berkeley [26]	Jackal	2m/s	4h	4h	suburban
5	SCAND-S [49]	Spot	1.5m/s	8h	4h	sidewalks
6	SCAND-J [49]	Jackal	2m/s	1h	1h	sidewalks
7	Seattle [50]	Warthog	5m/s	1h	1h	off-road
8	TartanDrive [51]	ATV	10m/s	7h	5h	off-road
9	NeBula [52]	ATV	10m/s	10h	10h	off-road
10	SACSoN [53]	TurtleBot2	0.5m/s	75h	10h	office
11	BDD [13]	Car(s)	20m/s	10h	4h	on-road
	Ours			160h	80h	

Table 8: The ViNT training dataset contains over 100 hours of navigation data in diverse environments across 8 different robots.

C Training Dataset

The ViNT training dataset contains over 100 hours of real-world navigation trajectories, sourced entirely from existing datasets. The dataset consists of a combination of tele-operated and autonomous navigation behaviors collected across 8 distinct robotic platforms, including 4 commercially available platforms (TurtleBot, Clearpath Jackal, Warthog and Spot) and several custom platforms (Yamaha Viking ATV, RC Car, passenger automobiles). The trajectories contain widely varying robot dynamics and top speeds, ranging between 0.2 and 10m/s, operating in a diverse set of environments (e.g., office buildings, hallways, suburban, off-road trails, university campuses, etc.). All data is either publicly available, or collected by *other* researchers for past projects; *no additional training data was collected specifically for training ViNT*.

Remember to mention: total size, number of robots, conversion to number of frames and so on.

D Robotic Platforms for Evaluating ViNT

Vizbot: A custom-built robot platform inspired by the design of Niwa et al. [54], based on a Roomba. It is equipped with an off-the-shelf PCB-mounted fisheye camera.

Unitree Go 1: A commercially available quadruped robot equipped with the original forward-facing camera. *There is no training data from a Go1 in the training dataset*, although SCAND has some data from a Boston Dynamics Spot, which is a very different platform.

Clearpath Jackal UGV: A commercially available off-road platform equipped with an off-the-shelf PCB-mounted fisheye camera. *This system resembles the data collection platform used for the RECON, Berkeley, and SCAND-J datasets*, but has a different camera and mounting height.

LoCoBot: A popular open-source platform based on a Kobuki, equipped with an off-the-shelf PCB-mounted fisheye camera. *There is no training data from a LoCoBot*, although GS was collected on a similar TurtleBot2, albeit with a different spherical camera at a lower height.

E Evaluation Setup and Details

E.1 Real-World Setup

E.1.1 Indoor Experiments

For setting up the indoor coverage exploration experiments on the LoCoBot, we choose a random starting point on the building's floor, and keep the starting point consistent across all baselines we test. For large scale floors, we test a variety of starting points on the floor, since most baselines are not able to cover the entire floor in the 10 minute time limit we enforce on the experiments.

For setting up the indoor guidance exploration experiments on the LoCoBot, we mark the starting point of the LoCoBot we drive the LoCoBot to the desired goal point (around 50m to 100m away), and use the in-built odometry tracking system on the LoCoBot to specify an odometry goal point to the exploration algorithm when we evaluate it.

E.2 Multi-robot Generalization Experiments

The setup for the multi-robot generalization experiment is very similar to the coverage exploration experiments. The only differences are the baselines we evaluate.

E.2.1 Baselines

For generating Table 1, we test three baseline low-level policies on each robot. Each baseline uses the graph-based exploration scheme described in Section 4.2. We use the following baselines:

1. **In-Domain:** We train a single-dataset policy model (ViNT architecture) and diffusion model on the RECON and SACSoN datasets, the largest outdoor and indoor datasets in our training set, and evaluate them on each of our robots to identify the best single-dataset model for each robot.
2. **GNM:** We use the pre-trained model checkpoint from the authors of GNM [18] coupled with *our* diffusion model (since GNM is not compatible with the exploration task) to evaluate each robot.
3. **ViNT:** We use our pre-trained ViNT policy and image diffusion model (no fine-tuning) to evaluate each robot.

E.3 CARLA Setup

This section describes the setup and implementation details for ViNT fine-tuning and adaptation experiments in the CARLA autonomous driving simulator, as presented in Sections 6.3 and 6.4.

E.3.1 CARLA Data Collection

We collect expert trajectories with an oracle rule-based self-driving agent and gather odometry and RGB information across the trajectory at 4 Hz. These trajectories have random spawn points and random destination points up to to 900 meters in length. We collect 52 trajectories in the CARLA Town 02 for held-out testing, and collect 181 trajectories in Town 01 for training. This makes for a dataset size of 5 hours for the autopilot control data. Inspired by [21], we also collect short trajectories of the agent correcting back onto the right lane after drifting off course in Town 01 and Town 02 for training and testing, respectively. This data is 4 hours long, and we add it to the autopilot data for training.

E.3.2 Fine-tuning Experiments

For testing the fine-tuning system which trains ViNT to do the same task in new domains, we utilize the collected test trajectories as expert maps to follow. Each Town 02 test trajectory creates a graph where every node is a timestamped odometry point corresponding to an image. To evaluate a model on a test trajectory, we spawn it at the same start point and localize it on the trajectory's map. We then query the image for the goal node which corresponds to node 1.5s after the current node. This goal image is sent to ViNT along with the 4Hz image context to compute a short-range trajectory. This is tracked by a simple PID controller. The average progress towards the goal before collision is collected and reported across all trials. Table 4 summarizes the results of these experiments with multiple baselines and data sizes.

E.3.3 Adaptation Experiments

To test the new tasks, we adopt a similar evaluation setup to the fine-tuning experiments. But we query the odometry for the goal node which corresponds to node 1.5s after the current node rather than the image. For positional-adaptation, we move the goal coordinates into a local frame and send it to ViNT. For routing-adaptation, we determine the difference in lateral coordinates between the current node and the goal node. We choose the current node as reference to ensure an open-loop experiment and to allow for pre-computation of the command signals to be sent. We then apply the same binning strategy during training using a 0.05 normalized distance as the boundary between "left", "right", and "straight". The control system downstream of this is identical to image fine-tuning and the experiment terminates when at the goal or when colliding. The progress towards the goal before collision is collected and averaged across all trials in Table 4.

E.3.4 Baselines

We have the following baselines for the CARLA experiments:

1. **Scratch**: ViNT trained from scratch on the CARLA on-task dataset.
2. **Pre-trained Visual Representations**
   (a) **ImageNet**: ViNT initialized with the EfficientNet-B0 weights pre-trained on ImageNet, other parameters initialized from scratch, and fine-tuned with the CARLA on-task dataset.
   (b) **SimCLR**: ViNT initialized with the EfficientNet-B0 weights pre-trained with SimCLR [7] on the training data described in Section C, other parameters initialized from scratch, and fine-tuned with the CARLA on-task dataset.
   (c) **VC-1**: ViNT initialized with a pre-trained ViT-B model checkpoint from the authors of VC-1 [40] *and frozen*, other parameters initialized from scratch, and fine-tuned with the CARLA on-task dataset. The VC-1 encoder is pre-trained on a combination of Ego4D, manipulation, navigation, and ImageNet images using Masked Auto-Encoding [39, 55].
3. **GNM**: The pre-trained embodiment-agnostic model checkpoint from the authors of GNM [18], fine-tuned with the CARLA on-task dataset. Note that GNM has 8.7M trainable parameters, compared to ViNT's 31M.

We note that the VC-1 baseline's weak performance in Section 6.4 may be explained by the fact that it is *frozen*, while all other visual encoders were free to fine-tune. This is representative of typical downstream usage [40]. Despite training on multiple, diverse datasets, the visual representation's general-purpose features are not optimized for the navigation task, hampering zero-shot transfer to out-of-domain tasks. To provide a fair comparison of the quality of pre-trained visual features, we compare this performance to ViNT-FE (a pre-trained ViNT model that has it's visual encoder frozen). ViNT-FE has an equal number of trainable parameters to the VC-1 baseline, and frozen visual representations (see Table 9).

Method	Images	Positions
VC-1 [40]	0.19	0.49
ViNT-FE	0.32	0.78
ViNT	0.82	0.89

Table 9: Evaluation of ViNT fine-tuning with and without a frozen encoder, as compared to a general-purpose visual encoder. Even when frozen, ViNT's navigation-relevant features appear to transfer more readily to out-of-distribution inputs than general-purpose features.

F Code and Video Release

We are providing the code for implementation and deployment of ViNT in the supplemental material.

We are also sharing videos of ViNT deployed on diverse robots and in challenging terrains on our project page (we were not able to upload it due to size limitations). Please check our project page for these videos: sites.google.com/view/vint-anon/