Appendix

A Prompt Engineering

To use large language models for a particular task, as opposed to a general text completion, one needs to encode the task as a part of the text input to the model. There exist many ways to create such encoding and the process of the representation optimization is sometimes referred to as *prompt engineering* [13]. In this section, we discuss the prompts we used for **LLM** and **VLM**.

A.1 LLM Prompt Engineering

All our experiments use GPT-3 [12] as the **LLM**, accessible via OpenAI's API: `https://openai.com/api/`. We used this model to extract a list of landmarks from free-form instructions. The model outputs were very reliable and robust to small changes in the input prompts. For parsing simple queries, GPT-3 was surprisingly effective with a single, zero-shot prompt. See the example below, where the model output is highlighted:

```
First, you need to find a stop sign.  Then take left and
right and continue until you reach a square with a tree.
Continue first straight, then right, until you find a white
truck.  The final destination is a white building.
Landmarks:
1. Stop sign
2. Square with a tree
3. White truck
4. White building
```

While this prompt is sufficient for simple instructions, more complex instructions require the model to reason about occurrences such as re-orderings, e.g. *Look for a glass building after after you pass by a white car*. We leverage GPT-3 ability to perform *in-context learning* [69] by adding three examples in the prompt:

```
Look for a library, after taking a right turn next to a
statue.
Landmarks:
1.  a statue
2.  a library

Look for a statue.  Then look for a library.  Then go towards
a pink house.
Landmarks:
1.  a statue
2.  a library
3.  a pink house

[Instructions]
Landmarks:
1.
```

We use the above prompt in all our experiments (Section 5.1, 5.2), and GPT-3 was successfully able to extract all landmarks. The comparison to other extraction methods is described in Section 5.3 and Appendix D.2.

A.2 VLM Prompt Engineering

In the case of our **VLM**— CLIP [13] — we use a simple family of prompts: *This is a photo of* ___, appended with the landmark description. This simple prompt was sufficient to detect over 95% of the landmarks encountered in our experiments. While our experiments did not require more careful prompt engineering, Radford et al. [13] and Zeng et al. [43] report improved robustness by using an ensemble of slightly varying prompts.

B Building the Topological Graph with VNM

This section outlines finer details regarding how the topological graph is constructed using **VNM**. LM-Nav assumes access to observations from a prior traversal in the environment — for the experiments in our paper, we use a single human traversal followed by the graph generation process described below. Empirically, we found the system to be robust to the mechanism in which the traversal was collected (e.g. random, lawnmower, bee-lining), as long as the robot observes relevant landmarks at some point in the traversal.

We use a combination of learned distance estimates (from **VNM**), spatial proximity (from GPS), and temporal proximity (during data collection), to deduce edge connectivity. If the corresponding timestamps of two nodes are close ($< 2s$), suggesting that they were captured in quick succession, then the corresponding nodes are connected — adding edges that were physically traversed. If the **VNM** estimates of the images at two nodes are close, suggesting that they are *reachable*, then the corresponding nodes are also connected — adding edges between distant nodes along the same route and giving us a mechanism to connect nodes that were collected in different trajectories or at different times of day but correspond to the nearby locations. To avoid cases of underestimated distances by the model due to aliased observations, e.g. green open fields or a white wall, we filter out prospective edges that are significantly further away as per their GPS estimates — thus, if two nodes are nearby as per their GPS, e.g. nodes on different sides of a wall, they may not be disconnected if the **VNM** does not estimate a small distance; but two similar-looking nodes 100s of meters away, that may be facing a white wall, may have a small **VNM** estimate but are not added to the graph to avoid *wormholes*. Algorithm 2 summarizes this process — the timestamp threshold ϵ is 1 second, the learned distance threshold τ is 80 time steps (corresponding to $\sim$ 20 meters), and the spatial threshold η is 100 meters.

Algorithm 2: Graph Building

```
1: Input: Nodes $n_i, n_j \in \mathcal{G}$ containing robot observations; VNM distance function $f_d$;
   hyperparameters $\{\tau, \epsilon, \eta\}$
2: Output: Boolean $e_{ij}$ corresponding to the existence of edge in $\mathcal{G}$, and its weight
3: learned distance $D_{ij} = f_d(n_i[\text{'image'}], n_j[\text{'image'}])$
4: timestamp distance $T_{ij} = |n_i[\text{'timestamp'}] - n_j[\text{'timestamp'}]|$
5: spatial distance $X_{ij} = \|n_i[\text{'GPS'}] - n_j[\text{'GPS'}])\|$
6: if ( $T_{ij} < \epsilon$) then return $\{True,\ D_{ij}\}$
7: else if $(D_{ij} < \tau)$ AND $(X_{ij} < \eta)$ then return $\{True,\ D_{ij}\}$
8: else return False
```

Since a graph obtained by such an analysis may be quite dense, we perform a *transitive reduction* operation on the graph to remove redundant edges.

C Mobile Robot Platform

We implement LM-Nav on a Clearpath Jackal UGV platform (see Fig. 1(right)). The sensor suite consists of a 6-DoF IMU, a GPS unit for approximate localization, wheel encoders for local odometry, and front- and rear-facing RGB cameras with a 170° field-of-view for capturing visual observations and localization in the topological graph. The **LLM** and **VLM** queries are pre-computed on a remote workstation and the computed path is commanded to the robot wirelessly. The **VNM** runs on-board and only uses forward RGB images and unfiltered GPS measurements.

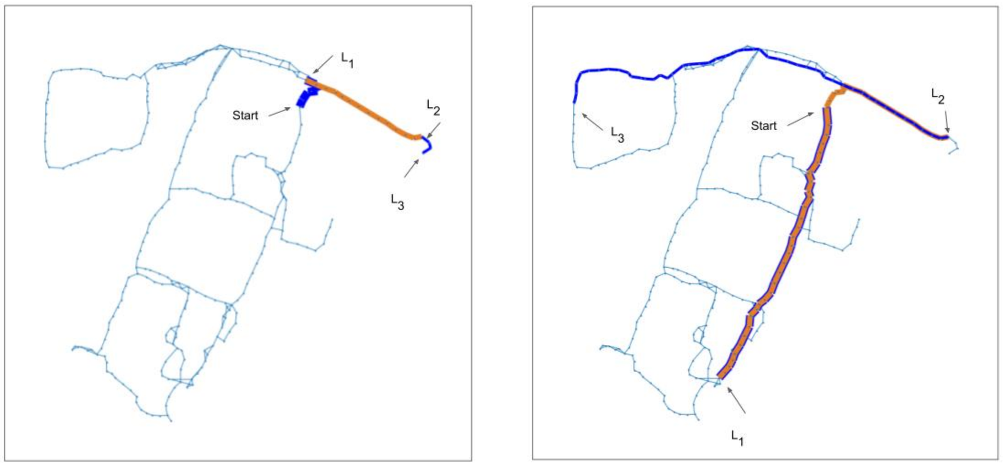

Figure 7: Examples of path planned by LM-Nav (left) and maximum likelihood planning (right). The start nodes and detected nodes are indicated with black arrows. In order to represent overlapping paths, we use colors interchangeably (start $\rightarrow L_1$: blue, $L_1 \rightarrow L_2$: orange, $L_2 \rightarrow L_3$: blue). The path taken by LM-Nav is significantly shorter, resulting in a $5\times$ more efficient plan.

D Miscellaneous Ablation Experiments

D.1 Ablating the Search Objective

The graph search objective described in Section 4.4 can be factored into two components: visiting the required landmarks (denoted by $P_l(\bar{v}|\bar{l})$) and minimizing distance traveled (denoted by $P_t(\bar{v})$). To analyze the importance of these two components, we ran a set of experiments where the nodes to be visited are selected based only on P_l. This corresponds to a *Max Likelihood* planner, which only picks the most likely node for each landmark, without reasoning about their relative topological positions and traversability. This approach leads to a simpler algorithm: for each of the landmark descriptions, the algorithm selects the node with the highest CLIP score and connects it via the shortest path to the current node. The shortest path between each pair of nodes is computed using the Floyd–Warshall algorithm.

Table 4 summarizes the performance metrics for the two planners. Unsurprisingly, the max likelihood planner suffers greatly in the form of efficiency, because it does not incentivize shorter paths (see Figure 7 for an example). Interestingly, the planning success suffers as well, especially in complex environments. Further analysis of these failure modes reveals cases where **VLM** returns erroneous detections for some landmarks, likely due to the contrastive objective struggling with variable binding (see Figure 8 for an example). While LM-Nav suffers from these failures as well, the second factor in the search objective $P_t(\bar{v})$ imposes a *soft constraint* on the search space of the landmarks, eliminating most of these cases and resulting in a significantly higher planning success rate.

D.2 Ablating the LLM

As described in Section 5.3 we run experiments comparing performance of different methods on extracting landmarks. Here we provide more details on the experiments. The source code to run this experiments is available in the file `ablation_text_to_landmark.ipynb` in the repository (see Appendix E).

As the **metric of performance** we used average extraction success. For a query with a ground truth list of landmarks L_{gt}, where a method extracts list L_m, we define the methods extraction success as:

$$\frac{|\text{LCS}(L_m, L_{gt})|}{|L_{gt}|},$$

where LCS is longest common subsequence and $|\cdot|$ denotes a length of a sequence or a list. This metric is measuring not only if correct landmarks were extracted, but also whether they are in the

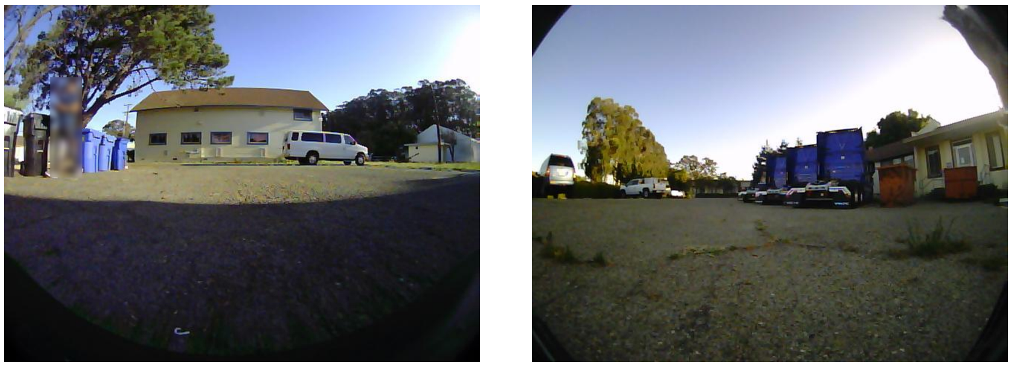

Figure 8: An example of failure to pick the correct image by maximum likelihood planning. Both images were selected for a prompt *A photo of a blue dumpster*. The left one was selected as a part of the LM-Nav's graph search and the right was selected by maximum likelihood planning. In the latter case, the selected image contains a blue semi-truck and an orange trailer, but no blue dumpsters. This might be an example of an issue with the variable binding. The left image was edited to maintain anonymity.

same order as in the ground truth sequence. When comparing landmarks we ignore articles, as we don't expect them to have impact on the downstream tasks.

All the experiments were run using APIs serving models. We used OpenAI's API (`https://beta.openai.com/`) for GPT-3 and GooseAI (`https://goose.ai`) for the other open-source models. Both providers conveniently share the same API. We used the same, default parameters, apart from setting `temperature` to 0: we don't expect that landmark extractions to require creativity and model's determinism improves reputability. For all the reported experiments, we used the same prompt as described in Appendix A. Please check out the released code for the exact prompts used.

E Code Release

We released the code corresponding to the **LLM** interface, **VLM** scoring, and graph search algorithm — along with a user-friendly Colab notebook capable of running quantitative experiments from Section 5.2. The links to the code and pickled graph objects can be found at our project page: `sites.google.com/view/lmnav`.

F Experiment Videos

We are sharing experiment videos of LM-Nav deployed on a Clearpath Jackal mobile robotic platform — please see `sites.google.com/view/lmnav`. The videos highlight the behavior learned by LM-Nav for the task of following free-form textual instructions and its ability to navigate complex environments and disambiguate between fine-grained commands.