REMAP: RETRIEVAL-AUGMENTED MULTIMODAL PLANNING WITH UNSTRUCTURED ROBOT MEMORY

Anonymous authors
Paper under double-blind review

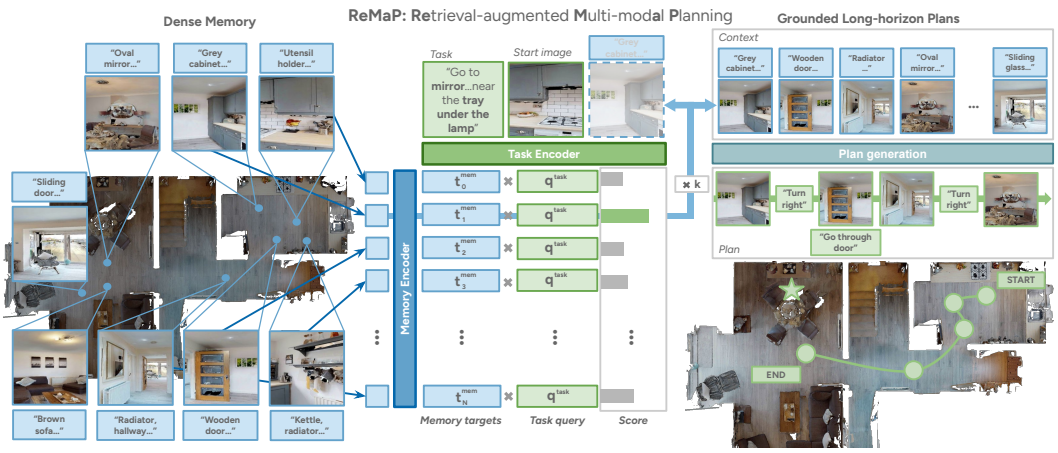


Figure 1: **We introduce ReMaP, a RAG-inspired memory and planning method for long-horizon navigation.** ReMaP can efficiently retrieve task-relevant context from large, dense memories and inform grounded plans for long-horizon navigation tasks.

ABSTRACT

General-purpose robotic systems must not only perform low-level skills, but also accomplish long-horizon goals. In partially observable settings, like navigation, this requires maintaining a memory or "mental map" that supports efficient, flexible retrieval of task-relevant prior experience. To this end, we present ReMaP, or **Re**trieval-Augmented **M**ultimod**a**l **P**lanning, a navigation memory and planning framework inspired by retrieval-augmented generation (RAG). ReMaP uses a learned embedding model that captures both semantic and geometric information to retrieve context from a dense memory of prior experience. This context is used to ground a plan of sub-goals, generated by a VLM, for a low-level navigation policy to execute. To assess ReMaP's capabilities for long-horizon navigation, we evaluate ReMaP as part of a complete robotic system on real-world robot navigation tasks, beating other navigation baselines by 20% in success rate and 27$\times$ in speed, and on GOAT-bench, where it outperforms prior RGB-based methods by 3% in success rate and 7.8% in SPL.

1 INTRODUCTION

Generalist robotic policies conditioned on language commands have progressed significantly in recent years and can now attain excellent results for a range of low-level skills in manipulation, driving, and navigation (Black et al., 2026; Gao et al., 2026; Renz et al., 2025; Hirose et al., 2025; 2024a). However, broadly capable and general-purpose robotic systems need to not only perform short-horizon behaviors, but carry out long-term goals in complex environments. In partially observed domains, such as vision-based navigation, this requires maintaining a memory of past experience and an ability to construct a "mental map" of the environment to plan through. This mental map must be flexible — based on the task and the experience collected by the robot, the robot must be able to dynamically retrieve relevant information, both semantic and geometric, that informs how

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

to complete the task. As the amount of experience grows, it becomes more challenging to quickly extract task-relevant information simply by feeding the raw unfiltered experience into the context of a policy. For example, a robot might traverse a floor of an office building with dozens of rooms, perhaps with overlapping trajectories. Then, if the robot is asked to locate a specific object in a specific room, this would require placing large amounts of redundant information in context.

Some prior methods have sought to address this problem by formatting the data into a graph structure, representing the entire environment as the root node, the intermediate nodes as regions or rooms, the leaf nodes as objects, and the edges between these nodes as their relationships (Loo et al., 2025; Werby et al., 2024; Ginting et al., 2024; Du et al., 2026; Rajvanshi et al., 2024; Rosinol et al., 2020). By traversing through this graph, the robot location and a target object location can be connected through their semantic relationships. For example, the "stove" node will be connected to the "kitchen" node by the relationship "in the", which will be "connected to" the "hallway" node and so on. However, this structure often comes at the cost of flexibility and generality, as the information from the environment is restricted to pre-determined relationships, and the downstream tasks that the robot can solve are then also limited by these relationships. As a result, relationships will be ignored that don't fit into the rigid structure (e.g., objects in hallways or connecting regions won't be recorded), and perceptual errors are hard to correct (e.g., objects missed in the labeling step will be impossible to reference downstream).

Alternatively, some prior works retain large memories and provide all the information as context for large language models (LLMs) or vision-language models (VLMs), relying on their semantic priors and large contexts to extract task-relevant information (Shah et al., 2022; Chiang et al., 2024; Wang et al., 2026). Not only is this expensive but, as experience scales, these models become more prone to temporal and spatial confusion. Therefore, this begs the question *is it possible to achieve general, flexible planning for robots with large, unstructured memory?*

A key observation is that for a given task, it is likely that a subset of memory will be relevant. For example, for the task "go down the hallway toward the kitchen", information in memory about bedrooms, bathrooms, or stairs can be ignored. Therefore, rather than searching over the entire memory to find relevant information for a task, it would be more effective to first rapidly determine this useful subset, then generate the trajectory or plan to execute the task. In LLM research, a similar problem has been encountered–a prompt may require sifting through a large number of documents to find the desired response (e.g., a law firm might to locate a specific clause for a case out of thousands of documents). Retrieval-augmented generation (RAG) (Lewis et al., 2021) has been used to make indexing into and then effectively using large amounts of data possible. Specifically, embeddings are associated with each chunk of data, and at inference-time, the embeddings with a high relevance to the prompt are pulled into context. Then, the generation step is conditioned on this considerably smaller subset of information to actually respond to the prompt. Importantly, these embeddings can be *learned*, which allows the relationship between prompts and context to be inferred from large amounts of data and makes the retrieval step a simple dot product, which can be rapidly performed across large amounts of data.

A RAG-like memory framework for navigation would therefore mitigate the two main challenges in prior work; by learning the relationship between tasks and context through embeddings, we need not pre-determine rigid relationships as graph methods do, and by using an embedding model to extract the memory subset relevant to the task, we can scale memory to be much larger without sacrificing inference speed.

To this end, we introduce ReMaP, a memory framework for robot policies inspired by RAG. As shown in Fig. 1, ReMaP consists of a *retrieval step* and a *generation step*. In the retrieval step, a learned embedding model efficiently pulls task-relevant information out of memory, stored as raw prior observations with generated language labels. The first query to the embedding model is the embedding of the task and initial observation. Then, the next query is constructed from the most recent piece of context, allowing the query to be localized in the task, and allowing for ordered retrieval and reduced duplication in the retrievals. In the generation step, the context and task specification is provided to a VLM, which constructs a grounded plan consisting of subgoals, specified by language, images, or their combination. The plan is then executed with a strong short-horizon navigation policies, which can be conditioned with language commands or image goals. Therefore, the generation step serves to both synthesize the plan from context and align it with the capabilities of the chosen navigation policy.

To our knowledge, ReMaP is the first robotic memory framework to train an embedding model for context retrieval, and therefore is also the first method to inherit the efficiency and flexibility of RAG for robotics. We evaluate ReMaP in the setting of iterative vision-language navigation (Krantz et al., 2023), which assumes access to an initial tour of the environment. We evaluate the performance of ReMaP on GOAT-Bench (Khanna* et al., 2024), finding that it outperforms the best RGB-based baseline by 3% in success rate and 7.8% in SPL on GOAT-Bench. We also evaluate the end-to-end performance of ReMaP on a set of challenging, real-world long-horizon tasks, finding that is outperforms the best baseline by over 20% in task success rate. Finally, we evaluate ReMaP on a retrieval speed benchmark, achieving a 27× speedup. We believe ReMaP is a step towards a more general, flexible, and efficient memory framework for robot platforms.

2 Related Work

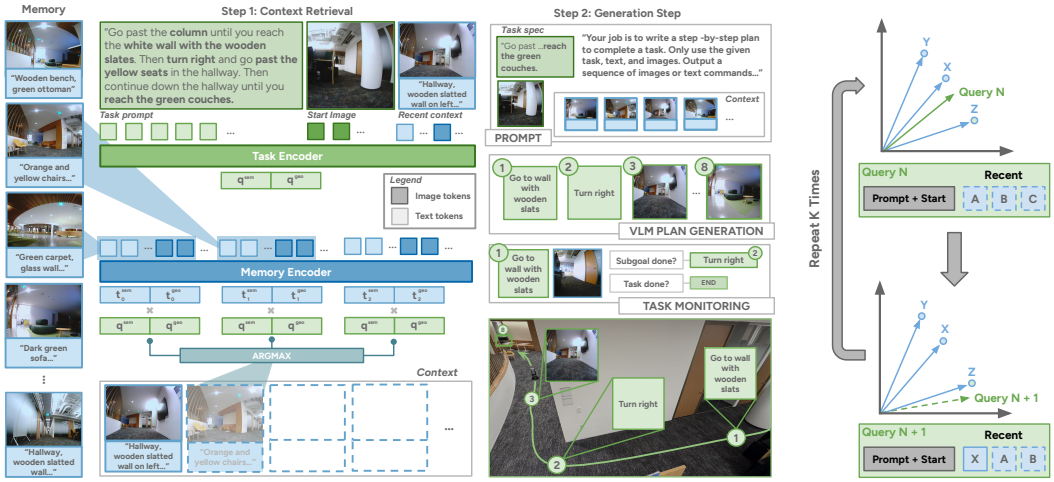
Language-conditioned navigation policies. As vision and language foundation models have become more powerful, language-conditioned navigation policies have become more prevalent. Several of these works focus on short-horizon navigation to observable goals (Hirose et al., 2025; Glossop et al., 2025; Hirose et al., 2024a; Zhang et al., 2026; Liu et al., 2026). Others aim to execute long-horizon tasks, either by accepting tasks that describe intermediate steps to complete the task (Cheng et al., 2025), or retaining a memory in the context of the policy (Chiang et al., 2024). However, these memories keep only a sliding window of the recent past (Zhang et al., 2025), or are limited to the current episode (Sridhar et al., 2025). However, even the models with the longest context store about 200 image observations (Chiang et al., 2024) at 3 FPS, while large environments can easily take several minutes, and grow the number of observations to over a thousand.

With ReMaP, we introduce a framework that can easily scale to memory to thousands of datapoints. By training an embedding model, retrieval consists of computing a dot product between the task and context embeddings, which is highly efficient and scales linearly with the size of memory.

Memory for long-horizon navigation. Long-horizon navigation tasks require retaining information about what the robot previously saw as it navigates through an environment, such that this information can later be used to inform how to execute the tasks. This information can be geometric, such as robot, object, or structure positions, visual, as images or videos, and semantic, as language that describes entities or relationships in the environment. Some prior work leaves this information in a raw format (Song et al., 2025; Wang et al., 2026; Goetting et al., 2024), leveraging LLMs or VLMs to sample from the memory. Other methods use metric maps, adding layers with object information to represent geometric and semantic relationships (Chaplot et al., 2020; Han et al., 2025; Chang et al., 2023; Zhu et al., 2025; Yang et al., 2025; Ziliotto et al., 2024) or topological graphs, where the edges represent reachability between nodes store poses (An et al., 2024; Chen et al., 2024) or images (Shah et al., 2022; Chiang et al., 2024). Many methods converges to the "scene graph" representation, a graph which represents relationships between objects and parts of the environment (Loo et al., 2025; Ginting et al., 2024; Rajvanshi et al., 2024; Werby et al., 2024; Du et al., 2026; Zhao et al., 2024; Yan et al., 2025; Honerkamp et al., 2024; Huang et al., 2026). Constructing these graphs requires perceptual pipelines that detect and localize objects while using them at inference time requires mapping tasks back to this representation. To make both these phases easier, many methods use LLMs or VLMs (Loo et al., 2025; Du et al., 2026; Rajvanshi et al., 2024; Werby et al., 2024; Zhao et al., 2024; Ji et al., 2025) to construct or traverse the graph, while some directly provide the graph as input to the VLM. However, all of these methods impose pre-determined relationships to construct the maps or graphs, pruning information that does not fit into them. As a result, the methods are limited downstream, often to behaviors like object-reaching. Moreover, traversing these graphs, especially when combined with VLMs, takes a considerable amount of time, limiting the scale to which these memories can grow.

ReMaP does not pre-suppose any kind of task specification and does not bake these into the memory representation. Rather, we store as much raw information in memory as possible in the form of images and language descriptions. We train our embedding model on diverse tasks across a broad set of environments, therefore learning embedding maps that can more generally represent the semantic and geometric relationships between tasks and context.

Embedding models for LLMs. Learning general embedding spaces has been valuable for providing good representations that align multiple modalities, such as images and language (Zhai et al.,



(a) Retrieval and generation steps for ReMaP. (b) Recursive retrieval.

Figure 2: **Overview of ReMaP.** We first recursively extract task-relevant context from memory. Then, the task specification and the context are provided to a VLM, which generates an executable plan for a robot navigation policy, informed by the information from memory. In Fig. 2b, we show how each query is constructed from the task prompt, start image, and the recent elements of context. This localizes the query in the task, meaning that ordered context can be retrieved, preventing duplicate retrievals and ensuring geometric consistency.

2023; Faysse et al., 2025; Radford et al., 2021). These embeddings provide initializations or frozen representations that are useful starting points for downstream tasks, including robotics (Gadre et al., 2022; Nguyen et al., 2024). However, these models are generally concerned with semantic relationships, whereas an embedding model for robot navigation must also be able to embed geometric information. With ReMaP, we tackle the unique problem of retrieval for downstream navigation. Therefore, our embedding model must capture both semantic and geometric relationships, allowing the model to retrieve context that will inform downstream navigation.

Retrieval-based memory for robot navigation. The success of RAG in the LLM domain has inspired RAG-like memory methods in navigation. Embodied-RAG (Xie et al., 2024) and RAG-Nav (Luo & Bai, 2026) use off-the-shelf spatial and semantic embeddings to build their memory representations, constructing semantic forests and topological graphs. When executing tasks, a traversal of these semantic forests is required to find the relevant context, which is performed with an LLM-based selection function or computing semantic similarity and spatial proximity with embedding similarities and distances, respectively. ReMEmbR (Anwar et al., 2025) and RAVEN (Hu et al., 2026) do not rely on graph representations and instead use vector databases that store embeddings for each data point in memory. When executing tasks, a VLM or LLM is used to iteratively sample from the database. Therefore, the VLM or LLM must make judgments about what is relevant for executing the task. While these prior methods also use retrieval in the context of a navigation system, they rely on pre-determined graph-based relationships between prior observations or expensive, iterative calls to an LLM or VLM to retrieve the context. In contrast, ReMaP directly extends RAG to robotic navigation in a fully general way: we train an embedding model that learns the semantic and geometric relationships between task queries and context embeddings. We can efficiently use the embeddings at test-time to query the raw information in memory for task-relevant context, and then perform the context-augmented generation step. As shown in **??**, ReMaP is more efficient than other retrieval baselines and achieves a much higher success rate on robot tasks.

3 Problem Statement

We consider a setting where we are given a dense memory, corresponding to a set of observations $\mathcal{M} = \{o_i\}_{i=1}^{N}$, where each $o_i = (I_i, d_i)$ consists of an image observation I_i, and a text label, d_i, that describes the landmarks present in I_i. We assume that this memory is collected with an exploration

method that provides sufficient coverage, whether this is a manually driven tour or autonomous exploration method. We are given a task prompt, $\mathcal{T}$, specified in language, and the robot's initial image observation, I_0. We assume that there exist observations in $\mathcal{M}$ that would be informative for executing the task, $\mathcal{T}$, such as images along the desired trajectory that would provide image goals for the navigation policy downstream. Analogous to retrieval methods for LLMs, we encode the task specification, $\mathcal{T}$ and I_0, into a query vector, $\mathbf{q}^{\text{task}}$. Each observation, $o_i \in \mathcal{M}$ is encoded into a target vector, $\mathbf{t}^{\text{mem}}$. Our objective is then to extract the task-relevant observations from memory, forming a context bank

$$\mathcal{C} = (o_0, \cdots, o_k), \qquad \mathcal{C} \subseteq \mathcal{M}.$$

While the context provides relevant information for completing the task, this context must be translated into a form that can be used to produce robot actions. In this work, we assume access to a low-level navigation policy, $\pi^{ll}(\mathbf{a}|\mathbf{g}, I)$, where $\mathbf{a}$ are robot actions and $\mathbf{g}$ are goals, which could be specified as an image goal (Shah et al., 2023b) or in language (Hirose et al., 2025). We assume that this policy can reach nearby goals in its field of view. Therefore, from $\mathcal{C}$, we must then construct a plan, $\mathcal{P} = \{\mathbf{g}_j\}_{j=1}^{T}$, where each $\mathbf{g}_j$ is a language or image goal *reachable* from the preceding subgoal, and sequence accomplishes the task.

4 Retrieval-augmented planning with dense memory

We now describe ReMaP, a modular, general, RAG-inspired system for embodied memory designed for retrieval and planning, which addresses the previously described problem. As shown in Fig. 2, we propose a two-stage process for generating navigation plans from dense memory. First, ReMaP is provided a task prompt, $\mathcal{T}$, and an initial image observation, I_0. The task specification is encoded as a query, $\mathbf{q}^{\text{task}}$ and each observation in memory is encoded as a target, $\mathbf{t}_i^{mem}$, each consisting of a geometric and a semantic subvector. Then, task-relevant context can be rapidly pulled from memory. At each step, the H most recent observations added to context are incorporated into the task query, informing what context needs to be retrieved next, as shown in Fig. 2b. Once the full context has been retrieved, it is passed to a VLM, which constructs a plan consisting of multi-modal, short-horizon goals, such as language prompts, image goals, and image-language goals. At test-time, we sequentially condition a navigation policy on these goals, monitoring progress by using a VLM to assess subgoal and task completion.

4.1 Efficient context retrieval with learned embeddings.

Compared to retrieval for VQA and scene understanding, retrieval for navigation is a uniquely challenging domain as the retrieved context must not only be semantically coherent but also geometrically consistent with respect to the task, allowing a policy to follow the plan generated downstream. Therefore, we propose two mechanisms to address these challenges: recursive retrieval and split geometric and semantic embeddings.

Our embedding model is a bi-encoder: two independent encoders separately embed target and query data. The target encoder maps a candidate observation $\mathbf{o}_i \in \mathcal{M}$ to a target consisting of a pair of latent codes $\mathbf{t}_i^{\text{mem}} = (\mathbf{t}_i^{\text{geo}}, \mathbf{t}_i^{\text{sem}})$ encoding geometric and semantic information, while the query encoder maps the task, $\mathcal{T}$, start observation, I_0, and current context, $\mathcal{C} = \{\mathbf{o}_i\}_{i=1}^{k}$, where k is the current size of the context, to an analogous pair $\mathbf{q}^{\text{task}} = (\mathbf{q}^{\text{geo}}, \mathbf{q}^{\text{sem}})$. For a target, the geometric channel carries visual information about the observation and the semantic channel carries the landmarks present in it. For a query, the geometric channel carries which target is reachable and the semantic channel carries the next sub-goal to be completed. We score a query–target pair as

$$\text{score}(\mathbf{q}^{\text{task}}, \mathbf{t}_i^{\text{mem}}) = \frac{\mathbf{q}^{\text{geo}} \cdot \mathbf{t}_i^{\text{geo}}}{\|\mathbf{q}^{\text{geo}}\| \, \|\mathbf{t}_i^{\text{geo}}\|} + \frac{\mathbf{q}^{\text{sem}} \cdot \mathbf{t}_i^{\text{sem}}}{\|\mathbf{q}^{\text{sem}}\| \, \|\mathbf{t}_i^{\text{sem}}\|}.$$

Retrieval is recursive: each retrieved observation is appended to the history of the query embedding, so subsequent queries are conditioned on what has already been selected (Figure 2b). We encode the inputs with frozen pre-trained vision and language encoders, SigLIP (Zhai et al., 2023) and DINOv2 (Oquab et al., 2024), to provide strong representations to train the embedding heads.

Target Embeddings. A target embedding pair is constructed from an observation $o_i = (I_i, d_i)$. Each landmark in the image description d_i is encoded as a text token by SigLIP, yielding multiple tokens that capture the semantic information in the image. The image itself is partitioned into four equal quadrants and each is encoded with both SigLIP and DINOv2, along with the full image. While SigLIP has been trained to capture language-aligned representations, DINOv2 captures visual consistency, which is valuable for geometric representations. We then pool these token sets with cross-attention into two learned tokens, `SEM_T` and `GEOM_T`. `SEM_T` attends to the text tokens and `GEOM_T` to the image tokens, with no attention between the two modalities. Because frozen SigLIP embeddings included in both token sets, the learned tokens share similar representations but can learn distinct semantic and geometric embeddings. An architecture diagram and further details are provided in Appendix E.

Query embeddings. A query is constructed from the task $\mathcal{T}$, the start image observation I_0, the most recent retrieval in the context, o_k and the recent context retrievals, $\{\mathbf{o}_{k-j}\}_{j=1}^{H} \subset \mathcal{C}$. The start observation is held fixed across steps and grounds the query; the remaining slots shift as retrieval advances (as shown in Fig. 2b). Observations are tokenized into image and text tokens as on the target side, except for o_{start}, which contributes image tokens only. The task is sampled using overlapping windows at two scales and encoded into text tokens by SigLIP. The context tokens are partitioned by modality into geometric and semantic sets. The geometric sequence contains the image tokens of the start and current context observations, preceded by a learned token `GEOM_Q`. The semantic sequence contains the text tokens of the current context observations, preceded by a learned token `SEM_Q`. Self-attention and positional embeddings are applied to each sequence separately, so that no image token attends to a text token or the reverse, separating the geometric and semantic information. Additionally, `SEM_Q` attends to the task tokens via a cross-attention layer. After attention is applied, `SEM_Q` and `GEOM_Q` are taken as the outputs and projected by a linear layer to produce $(\mathbf{q}^{\text{geo}}, \mathbf{q}^{\text{sem}})$. This process yields query embeddings that can be compared to the target embeddings, which we can use to teach the model to produce outputs that retrieve the information that is not yet present in the context. For more details, see Appendix E.

Training. We freeze the SigLIP and DINOv2 backbones and only optimize the fusion modules and output heads. Input tokens are L_2-normalized and perturbed with normal noise, images are augmented from a fixed pool sampled at random, and modality and task dropout are applied per batch. For the loss, we use forward KL, $D_{\text{KL}}(p \,\|\, q)$, taken per query row and averaged so that our model learns to produce the correct distribution over targets for each query. We use separate supervision for the geometric and semantic heads such that $\mathcal{L} = \mathcal{L}_{\text{geo}} + \mathcal{L}_{\text{sem}}$.

For training, we use data from the GNM dataset (Shah et al., 2023a), including SACSoN (Hirose et al., 2024b), GO Stanford (Hirose et al., 2018), and SCAND (Karnan et al., 2022). We also include simulation data from the RxR dataset (Ku et al., 2020), which is built from Matterport 3D (MP3D) scenes (Chang et al., 2017). We create trajectories by sub-sampling building tours from GNM and sampling random walks on RxR, and also by constructing connectivity graphs and running A* search (see Appendix E). In total, we train on $830,325$ unique positive query-target pairs.

We label trajectories with soft-positive labels where the immediate next step in a trajectory is labeled as 1 and labels for all subsequent steps are calculated via exponential decay. We use these soft labels to train the geometric head to learn spatial proximity. We harden the labels to 1 for semantic head training because there is no partial correctness. Additionally, we build batches by sampling trajectories rather than individual embedding pairs so that each query has multiple associated soft-positive targets. For more details on the training setup, see Appendix E.

4.2 Task Execution

We first explore the environment to be navigated, algorithmically in simulation and manually in the real world, gathering images that ensure full coverage. We then label these images by using Gemini-2.5-Pro to enumerate the important stationary landmarks. Every gathered observation in $\mathcal{O}$ is embedded to build a vector database with our embedding model. Given a task, recursive retrieval selects up to K observations from this database to form the context bank $\mathcal{C}$, which is passed with the task to a VLM to generate a plan.

We use Gemini 3.8 Flash as the VLM planner, with in-context examples of plans and in-distribution prompts for the low-level policy, allowing it to produce a grounded plan $\mathcal{P}$ that can be executed

by the low-level policy. As the context is already mostly ordered and spatially consistent, this step selects and wires together observations from context, filling in gaps with language commands where necessary. During execution we query the VLM every t steps to check whether the current waypoint has been reached, advancing to the next when it has, until the plan is exhausted. For more details on image labeling and VLM prompting, see Appendix D.1 for the image labeling prompt, Appendix D.2 for planner prompts and Appendix D.3 for the completion-monitor prompt.

5 Evaluating ReMaP for Robot Navigation Memory and Planning.

We evaluate ReMaP's capabilities as a memory and planning framework for long-horizon tasks, testing its ability to execute diverse tasks of varying lengths. We aim to answer the following questions:

Q1 How does ReMaP compare to prior navigation methods?
Q2 Can ReMaP enable strong downstream long-horizon navigation performance?
Q3 Can ReMaP efficiently generate plans from large, dense memories?

5.1 Experimental setup

We first evaluate ReMaP on a standard benchmark, GOAT-Bench, to understand its performance across highly diverse unseen navigation tasks in simulation. Then, we deploy ReMaP as part of a full robotic system on real-world long-horizon tasks to study its ability to create executable plans for low-level navigation policies.

Implementing ReMaP for GOAT-Bench. For the GOAT-bench evaluation, we use an oracle method to get sufficient coverage of the scene, as we primarily evaluate ReMaP as a retrieval and planning method, rather than an exploration method. We use the original goal threshold of 1 m. As we do not train our embedding model to accept image goals as task specifications, we convert image goals to language tasks with a VLM, in this case Gemini-2.5-Pro (Google). For the navigation policy itself, we implement a GOAT-like approach that goes to a location retrieved by ReMaP using a fast-marching planner. This evaluation focuses on the quality of the plans that ReMaP generates rather than exploration ability.

Real-world Experiments. To evaluate ReMaP in the real world as part of a full robotic system, we deploy it on a 8 real-world tasks in a novel indoor environment across 3 different horizons: 3 short-horizon tasks, 3 medium-horizon tasks, and 2 long-horizon tasks. These tasks range from 20-80 m long and vary in complexity. We design these tasks to benefit from memory, as some parts of the task can be ambiguous if the prior experience is not leveraged. Each task is specified in language and includes landmark and motion cues, but omits some intermediate navigation details, requiring memory to fill in the gaps(see Appendix A). To provide memory, we manually collect a tour through the environment, ensuring coverage exists for each task, in the form of image observations and language labels. We implement ReMaP with OmniVLA (Hirose et al., 2025), a generalist, multi-modal navigation policy, which can accept image, language, and image-language goals as input, as the low-level policy. We use the flexibility of the generation step to adapt the subgoals to OmniVLA, aligning language prompts to the instruction formats used to train OmniVLA. We run 3 trials for each task.

We compare to 3 baselines: *NaVILA* (Cheng et al., 2025) is a navigation policy capable of taking detailed prompts as input, allowing it to complete long-horizon tasks without memory. *LM-Nav* (Shah et al., 2022) is a long-horizon navigation method that keeps a topological memory of observations and computes a plan through the memory graph that aligns with landmarks in the prompt, before supplying the goal images on the path to ViNT, an image-conditioned low-level policy (Shah et al., 2023b). We also evaluate a retrieval-based *RAVEN*-like method (Hu et al., 2026), which we call RAVEN-`no-pose` that, similarly to ReMaP, uses retrieval tool calls to iteratively retrieve context that is built using an off-the-shelf multi-modal embedding and privileged pose information. However, getting pose information at test-time would provide RAVEN an unfair advantage compared to methods that do not use pose, therefore, we implement RAVEN-`no-pose` purely as a text-based retrieval method, and provide it with the same generation step and low-level policy as ReMaP, directly comparing ReMaP's learned embedding space and retrieval mechanism against an off-the-

shelf baseline. We evaluate these methods on *success rate, progress, and plan coverage*. For more details, see Appendix B.

5.2 Comparing ReMaP to prior navigation methods

We use GOAT-Bench to compare the retrieval and planning performance of ReMaP against other planning methods in a standardized setting. We pre-explore the environment to ensure full coverage before allowing ReMaP to plan and our GOAT-like navigation policy to reach the subgoals produced by ReMaP. We find that ReMaP outperforms state-of-the-art RGB-only navigation methods and is competitive with the state-of-the-art RGB-D methods that utilize privileged information unavailable to ReMaP.

Table 1: **GOAT-Bench results (Val Unseen).** Success rate (SR) and success weighted by path length (SPL), in %. Bold marks the best RGB-only result.

	Method	SR ↑	SPL ↑
RGB-D	MTU3D (Zhu et al., 2025)	40.8	12.1
	MSGNav (Huang et al., 2026)	52.0	29.6
	3D-Mem (Yang et al., 2025)	69.1	48.9
RGB	Modular GOAT (Chang et al., 2023)	24.9	17.2
	VLMNav (Goetting et al., 2024)	16.3	6.6
	DyNaVLM (Ji et al., 2025)	25.5	10.2
	TANGO (Ziliotto et al., 2024)	35.5	19.5
	Uni-NaVid (Zhang et al., 2025)	38.5	19.8
	ReMaP (Ours)	**41.6**	**27.6**

Table 1 reports results on the unseen validation split. Under an oracle pre-exploration protocol, ReMaP achieves the highest success rate among RGB-based baselines. This exceeds the next best baseline, Uni-NaVid scores by 3.1 and 7.8 percentage points, respectively, although the exploration protocols differ. With access to memory, ReMaP can efficiently extract task-relevant context and produce grounded plans, resulting in an especially high SPL compared to prior navigation methods without memory. Moreover, ReMaP even outperforms MTU3D, a method that is able to build a 3-D representation using depth readings, demonstrating the effectiveness of ReMaP's learned representation. Overall, ReMaP achieves strong and efficient performance across diverse, unseen navigation tasks, demonstrating it can effectively pull relevant memory into context and produce actionable plans.

5.3 Evaluating ReMaP on real-world tasks.

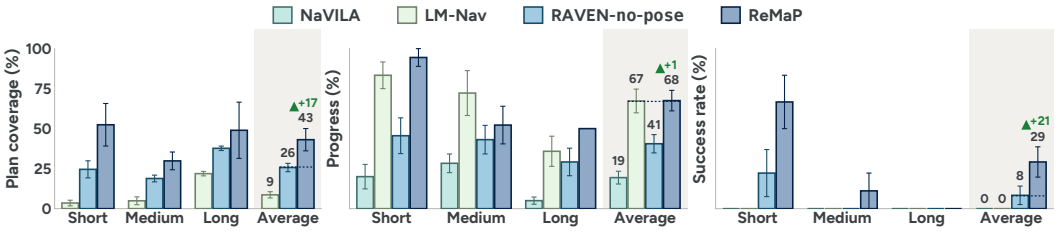


Figure 3: **Real-world performance of ReMaP compared to baselines.** ReMaP achieves the highest overall success rate, improving over the next best baseline by over 20%. **Success rate** measures successfully reaching the end goal and stopping, **progress** measures the percent progress along the ground truth trajectory, and **plan coverage** measures the accuracy and completeness of the plan.

On real-world long-horizon tasks, ReMaP achieves 20.1% higher success rate compared to the next best baseline, RAVEN-`no-pose`. NaVILA had low progress on all tasks, often crashing into walls or objects. As NaVILA does not have a memory mechanism and relies on information being provided in the task prompt, it struggled with task prompts where landmarks were not directly visible or motions were not well specified, demonstrating the necessity of memory accesses for completing these tasks. While LM-Nav achieves the similar progress as ReMaP (67.3% vs. 67.5%), its plans were largely images not on the path to the goal location, resulting in its low plan coverage. Additionally, it never stopped at the goal location, resulting in a success rate of 0%. We use the same generation method for both ReMaP and RAVEN-`no-pose`, so the performance difference can be ascribed to their retrieval methods. ReMaP achieves a much higher success rate which we attribute to the better spatial coherence of its plans. As shown by the plan coverage comparison, ReMaP achieves a plan coverage of 43%, compared to 26% for RAVEN-`no-pose`. This is especially well



Figure 4: **Example plans produced by ReMaP.** Retrieved observations are consistent with task and start image.

illustrated for the long-horizon task,"Go past the column until you reach the white wall with wooden slats. Then turn right and go past the yellow seats in the hallway. Then continue down the hallway until you reach the green couches. Finally, turn right and stop at the orange high chairs." ReMaP is the only method to make progress beyond the first turn. Looking at the generated plan, ReMaP's plan contains specific cues to turn after reaching the wall, while RAVEN-`no-pose`'s plan only contains goal images near the chairs, which are not visible before the turn. Overall, we find that the plans that ReMaP produces allow for much more effective policy steering, and therefore considerably higher success rate and we provide examples in Fig. 4.

5.4 Evaluating Retrieval Speed

Towards answering **Q3** and to gain an understanding of how retrieval and pre-processing time scales for large datasets, we compile a large out-of-distribution robot navigation dataset and benchmark the time it takes for each method to pre-process the images and to retrieve context from it at inference time. For the pre-processing time per image, ReMaP, RAVEN, and LM-Nav take 0.016, 0.116, 1.14 seconds respectively. At inference time, these methods take 1.49, 40.73, and 4.41 seconds per full context retrieval respectfully. Overall, ReMaP significantly outperforms both LM-Nav and RAVEN in speed, offering a 7.25$\times$ speedup vs. RAVEN in initialization, and a 27.3$\times$ speedup at inference time, making ReMaP significantly more applicable in settings where real-time planning is important. More details on this dataset are provided in Appendix C.

6 Discussion

We introduce ReMaP, a RAG-inspired memory and planning framework for long-horizon navigation tasks. We evaluate ReMaP on the large-scale simulation benchmark, GOAT-bench, and on a suite of real-world tasks across varying horizons. We find that ReMaP with access to memory beats prior RGB-based navigation methods on GOAT-bench and demonstrates a significant quantitative and qualitative advantage over prior retrieval-based navigation methods on real-world tasks. Moreover, ReMaP is also significantly more efficient, with inference roughly 27$\times$ faster than methods we compare to, making it possible to continuously re-plan during deployment.

Overall, this suggests that decomposing navigation with memory into a fast retrieval step and a powerful generation step can generate grounded plans for short-horizon navigation policies. In future work, we will explore scaling our embedding model significantly to leverage more data and improve the semantic and geometric representations we learn. Additionally, we believe ReMaP's speed and flexibility is conducive to extending it to exploration, which could reduce dependence on a pre-collected tour by allowing memory to be built online.

We believe ReMaP offers a powerful framework for general robot memory and planning, and we will consider extending this work to robotic manipulation and other domains in future work.

7 DISCLOSURES

AI USE STATEMENT

In accordance with ICLR guidelines, we disclose the use of Large Language Models (LLMs) during the preparation of this manuscript. We utilized LLMs as a general-purpose assistant tool for grammar correction and stylistic editing. All AI-generated text was extensively reviewed and verified. We also used LLMs for assistance with generating plots and figures, which we also extensively reviewed. We additionally use VLMs to generate labels for data in this work and as a planner for navigation tasks, as part of our method described in Section 4, and therefore use them as part of experimental execution. LLMs did not contribute to the core scientific ideation, methodology, or data analysis, and the authors take full responsibility for the final contents of this paper.

ETHICS STATEMENT

Our work focuses on robot navigation with potential applications in the real-world. In our experiments, we use a low speed robot, but caution should be taken in applying this work to heavy, fast robot platforms, as we do not explore the safety of the method. We do not release any new data with this work.

REPRODUCIBILITY STATEMENT

To ensure the reproducibility of our results, we have detailed our methodology in Section 4 and provide a comprehensive breakdown of the network architectures, VLM prompts, and hyperparameters. We provide our code in the supplemental material submission.

REFERENCES

Dong An, Hanqing Wang, Wenguan Wang, Zun Wang, Yan Huang, Keji He, and Liang Wang. ETPNav: Evolving Topological Planning for Vision-Language Navigation in Continuous Environments, 2024. URL http://arxiv.org/abs/2304.03047.

Md Asif Anwar, Sarah Welsh, Jerry Zhang, Yuke Zhu, Dhruv Shah, and Pratik Chaudhari. Remembr: Building and reasoning over long-horizon spatio-temporal memory for robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2025. doi: 10.1109/ICRA55743.2025.11127706.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control, 2026. URL https://arxiv.org/abs/2410.24164.

Johnata Brayan, Sihao Deng, Armando Alves Neto, Iaroslav Okunevich, Tomas Krajnik, Francois Bremond, and Zhi Yan. Navwareset: A dataset of socially compliant and non-compliant robot navigation. *The International Journal of Robotics Research*, 0(0):1–15, 2026. doi: 10.1177/02783649261447305. URL https://doi.org/10.1177/02783649261447305.

Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Nießner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments, 2017. URL https://arxiv.org/abs/1709.06158.

Matthew Chang, Theophile Gervet, Mukul Khanna, Sriram Yenamandra, Dhruv Shah, So Yeon Min, Kavit Shah, Chris Paxton, Saurabh Gupta, Dhruv Batra, Roozbeh Mottaghi, Jitendra Malik, and Devendra Singh Chaplot. GOAT: GO to Any Thing, 2023. URL http://arxiv.org/abs/2311.06430.

Devendra Singh Chaplot, Dhiraj Gandhi, Abhinav Gupta, and Ruslan Salakhutdinov. Object Goal Navigation using Goal-Oriented Semantic Exploration, 2020. URL http://arxiv.org/abs/2007.00643.

Jiaqi Chen, Bingqian Lin, Ran Xu, Zhenhua Chai, Xiaodan Liang, and Kwan-Yee K. Wong. MapGPT: Map-Guided Prompting with Adaptive Path Planning for Vision-and-Language Navigation, 2024. URL http://arxiv.org/abs/2401.07314.

An-Chieh Cheng, Yandong Ji, Zhaojing Yang, Zaitian Gongye, Xueyan Zou, Jan Kautz, Erdem Bıyık, Hongxu Yin, Sifei Liu, and Xiaolong Wang. NaVILA: Legged Robot Vision-Language-Action Model for Navigation, 2025. URL http://arxiv.org/abs/2412.04453.

Hao-Tien Lewis Chiang, Zhuo Xu, Zipeng Fu, Mithun George Jacob, Tingnan Zhang, Tsang-Wei Edward Lee, Wenhao Yu, Connor Schenck, David Rendleman, Dhruv Shah, Fei Xia, Jasmine Hsu, Jonathan Hoech, Pete Florence, Sean Kirmani, Sumeet Singh, Vikas Sindhwani, Carolina Parada, Chelsea Finn, Peng Xu, Sergey Levine, and Jie Tan. Mobility VLA: Multimodal Instruction Navigation with Long-Context VLMs and Topological Graphs, 2024. URL http://arxiv.org/abs/2407.07775.

Yi Du, Taimeng Fu, Zhipeng Zhao, Shaoshu Su, Zitong Zhan, Qiwei Du, Zhuoqun Chen, Bowen Li, and Chen Wang. VL-Nav: A Neuro-Symbolic Approach for Reasoning-based Vision-Language Navigation, 2026. URL http://arxiv.org/abs/2502.00931.

Manuel Faysse, Hugues Sibille, Tony Wu, Bilel Omrani, Gautier Viaud, Céline Hudelot, and Pierre Colombo. Colpali: Efficient document retrieval with vision language models, 2025. URL https://arxiv.org/abs/2407.01449.

Samir Yitzhak Gadre, Mitchell Wortsman, Gabriel Ilharco, Ludwig Schmidt, and Shuran Song. CoWs on Pasture: Baselines and Benchmarks for Language-Driven Zero-Shot Object Navigation, 2022. URL http://arxiv.org/abs/2203.10421.

Tian Gao, Celine Tan, Catherine Glossop, Timothy Gao, Jiankai Sun, Kyle Stachowicz, Shirley Wu, Oier Mees, Dorsa Sadigh, Sergey Levine, and Chelsea Finn. Steervla: Steering vision-language-action models in long-tail driving scenarios, 2026. URL https://arxiv.org/abs/2602.08440.

Muhammad Fadhil Ginting, Sung-Kyun Kim, David D. Fan, Matteo Palieri, Mykel J. Kochenderfer, and Ali-akbar Agha-Mohammadi. SEEK: Semantic Reasoning for Object Goal Navigation in Real World Inspection Tasks, 2024. URL http://arxiv.org/abs/2405.09822.

Catherine Glossop, William Chen, Arjun Bhorkar, Dhruv Shah, and Sergey Levine. CAST: Counterfactual Labels Improve Instruction Following in Vision-Language-Action Models, 2025. URL http://arxiv.org/abs/2508.13446.

Dylan Goetting, Himanshu Gaurav Singh, and Antonio Loquercio. End-to-end navigation with vision language models: Transforming spatial reasoning into question-answering, 2024. URL https://arxiv.org/abs/2411.05755.

Google. Gemini 3.5: Frontier intelligence with action. URL https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-5/.

Peilong Han, Fan Jia, Min Zhang, Yutao Qiu, Hongyao Tang, Yan Zheng, Tiancai Wang, and Jianye Hao. MUVLA: Learning to Explore Object Navigation via Map Understanding, 2025. URL http://arxiv.org/abs/2509.25966.

Noriaki Hirose, Amir Sadeghian, Marynel Vázquez, Patrick Goebel, and Silvio Savarese. Gonet: A semi-supervised deep learning approach for traversability estimation, 2018. URL https://arxiv.org/abs/1803.03254.

Noriaki Hirose, Catherine Glossop, Ajay Sridhar, Dhruv Shah, Oier Mees, and Sergey Levine. LeLaN: Learning A Language-Conditioned Navigation Policy from In-the-Wild Videos, 2024a. URL http://arxiv.org/abs/2410.03603.

Noriaki Hirose, Dhruv Shah, Ajay Sridhar, and Sergey Levine. Sacson: Scalable autonomous control for social navigation. *IEEE Robotics and Automation Letters*, 9(1):49–56, 2024b. doi: 10.1109/LRA.2023.3329626.

Noriaki Hirose, Catherine Glossop, Dhruv Shah, and Sergey Levine. OmniVLA: An Omni-Modal Vision-Language-Action Model for Robot Navigation, 2025. URL http://arxiv.org/abs/2509.19480.

Daniel Honerkamp, Martin Büchner, Fabien Despinoy, Tim Welschehold, and Abhinav Valada. Language-Grounded Dynamic Scene Graphs for Interactive Object Search with Mobile Manipulation. *IEEE Robotics and Automation Letters*, 9(10):8298–8305, 2024. doi: 10.1109/LRA.2024.3441495. URL http://arxiv.org/abs/2403.08605.

Yixun Hu, Zhicheng Zheng, Lihan Zha, Chunwei Xing, Rajdeep Singh, Omar Hossain, Antonio Loquercio, and Dhruv Shah. Raven: Long-horizon reasoning navigation with a visuo-spatio-temporal memory, 2026. URL https://arxiv.org/abs/2606.25206.

Xun Huang, Shijia Zhao, Yunxiang Wang, Xin Lu, Wanfa Zhang, Rongsheng Qu, Weixin Li, Yunhong Wang, and Chenglu Wen. Msgnav: Unleashing the power of multi-modal 3d scene graph for zero-shot embodied navigation, 2026. URL https://arxiv.org/abs/2511.10376.

Zihe Ji, Huangxuan Lin, and Yue Gao. Dynavlm: Zero-shot vision-language navigation system with dynamic viewpoints and self-refining graph memory, 2025. URL https://arxiv.org/abs/2506.15096.

Haresh Karnan, Anirudh Nair, Xuesu Xiao, Garrett Warnell, Sören Pirk, Alexander Toshev, Justin Hart, Joydeep Biswas, and Peter Stone. Socially compliant navigation dataset (scand): A large-scale dataset of demonstrations for social navigation. *IEEE Robotics and Automation Letters*, 2022.

Mukul Khanna*, Ram Ramrakhya*, Gunjan Chhablani, Sriram Yenamandra, Theophile Gervet, Matthew Chang, Zsolt Kira, Devendra Singh Chaplot, Dhruv Batra, and Roozbeh Mottaghi. Goat-bench: A benchmark for multi-modal lifelong navigation, 2024.

Jacob Krantz, Shurjo Banerjee, Wang Zhu, Jason Corso, Peter Anderson, Stefan Lee, and Jesse Thomason. Iterative Vision-and-Language Navigation, 2023. URL http://arxiv.org/abs/2210.03087.

Alexander Ku, Peter Anderson, Roma Patel, Eugene Ie, and Jason Baldridge. Room-across-room: Multilingual vision-and-language navigation with dense spatiotemporal grounding, 2020. URL https://arxiv.org/abs/2010.07954.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021. URL https://arxiv.org/abs/2005.11401.

Haokun Liu, Zhaoqi Ma, Yicheng Chen, Masaki Kitagawa, Wentao Zhang, Zicen Xiong, Jinjie Li, and Moju Zhao. Cofl: Continuous flow fields for language-conditioned navigation, 2026. URL https://arxiv.org/abs/2603.02854.

Joel Loo, Zhanxin Wu, and David Hsu. Open Scene Graphs for Open-World Object-Goal Navigation. *The International Journal of Robotics Research*, pp. 02783649251369549, 2025. doi: 10.1177/02783649251369549. URL http://arxiv.org/abs/2508.04678.

Ling Luo and Qiangian Bai. RAGNav: A Retrieval-Augmented Topological Reasoning Framework for Multi-Goal Visual-Language Navigation, 2026. URL http://arxiv.org/abs/2603.03745.

Nghia Nguyen, Minh Nhat Vu, Tung D. Ta, Baoru Huang, Thieu Vo, Ngan Le, and Anh Nguyen. Robotic-clip: Fine-tuning clip on action data for robotic applications, 2024. URL https://arxiv.org/abs/2409.17727.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2024. URL https://arxiv.org/abs/2304.07193.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. URL https://arxiv.org/abs/2103.00020.

Abhinav Rajvanshi, Karan Sikka, Xiao Lin, Bhoram Lee, Han-Pang Chiu, and Alvaro Velasquez. SayNav: Grounding Large Language Models for Dynamic Planning to Navigation in New Environments, 2024. URL http://arxiv.org/abs/2309.04077.

Katrin Renz, Long Chen, Elahe Arani, and Oleg Sinavski. Simlingo: Vision-only closed-loop autonomous driving with language-action alignment, 2025. URL https://arxiv.org/abs/2503.09594.

Antoni Rosinol, Arjun Gupta, Marcus Abate, Jingnan Shi, and Luca Carlone. 3D Dynamic Scene Graphs: Actionable Spatial Perception with Places, Objects, and Humans, 2020. URL http://arxiv.org/abs/2002.06289.

Dhruv Shah, Blazej Osinski, Brian Ichter, and Sergey Levine. LM-Nav: Robotic Navigation with Large Pre-Trained Models of Language, Vision, and Action, 2022. URL http://arxiv.org/abs/2207.04429.

Dhruv Shah, Ajay Sridhar, Arjun Bhorkar, Noriaki Hirose, and Sergey Levine. GNM: A General Navigation Model to Drive Any Robot. In *International Conference on Robotics and Automation (ICRA)*, 2023a. URL https://arxiv.org/abs/2210.03370.

Dhruv Shah, Ajay Sridhar, Nitish Dashora, Kyle Stachowicz, Kevin Black, Noriaki Hirose, and Sergey Levine. ViNT: A Foundation Model for Visual Navigation, 2023b. URL http://arxiv.org/abs/2306.14846.

Xuesong Shi, Dongjiang Li, Pengpeng Zhao, Qinbin Tian, Yuxin Tian, Qiwei Long, Chunhao Zhu, Jingwei Song, Fei Qiao, Le Song, Yangquan Guo, Zhigang Wang, Yimin Zhang, Baoxing Qin, Wei Yang, Fangshi Wang, Rosa H. M. Chan, and Qi She. Are we ready for service robots? the OpenLORIS-Scene datasets for lifelong SLAM. In *2020 International Conference on Robotics and Automation (ICRA)*, pp. 3139–3145, 2020.

Xinshuai Song, Weixing Chen, Yang Liu, Weikai Chen, Guanbin Li, and Liang Lin. Towards Long-Horizon Vision-Language Navigation: Platform, Benchmark and Method, 2025. URL http://arxiv.org/abs/2412.09082.

Ajay Sridhar, Jennifer Pan, Satvik Sharma, and Chelsea Finn. MemER: Scaling Up Memory for Robot Control via Experience Retrieval, 2025. URL http://arxiv.org/abs/2510.20328.

Shuo Wang, Yucheng Wang, Guoxin Lian, Yongcai Wang, Maiyue Chen, Kaihui Wang, Bo Zhang, Zhizhong Su, Yutian Zhou, Wanting Li, Deying Li, and Zhaoxin Fan. Progress-Think: Semantic Progress Reasoning for Vision-Language Navigation, 2026. URL http://arxiv.org/abs/2511.17097.

Abdelrhman Werby, Chenguang Huang, Martin Büchner, Abhinav Valada, and Wolfram Burgard. Hierarchical Open-Vocabulary 3D Scene Graphs for Language-Grounded Robot Navigation. In *Robotics: Science and Systems XX*, 2024. doi: 10.15607/RSS.2024.XX.077. URL http://arxiv.org/abs/2403.17846.

Quanting Xie, So Yeon Min, Tianyi Zhang, Kedi Xu, Aarav Bajaj, Ruslan Salakhutdinov, Matthew Johnson-Roberson, and Yonatan Bisk. General non-parametric embodied memory for retrieval and generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Zhijie Yan, Shufei Li, Zuoxu Wang, Lixiu Wu, Han Wang, Jun Zhu, Lijiang Chen, and Jihong Liu. Dynamic Open-Vocabulary 3D Scene Graphs for Long-term Language-Guided Mobile Manipulation, 2025. URL http://arxiv.org/abs/2410.11989.

Yuncong Yang, Han Yang, Jiachen Zhou, Peihao Chen, Hongxin Zhang, Yilun Du, and Chuang Gan. 3d-mem: 3d scene memory for embodied exploration and reasoning, 2025. URL https://arxiv.org/abs/2411.17735.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training, 2023. URL https://arxiv.org/abs/2303.15343.

Jiazhao Zhang, Kunyu Wang, Shaoan Wang, Minghan Li, Haoran Liu, Songlin Wei, Zhongyuan Wang, Zhizheng Zhang, and He Wang. Uni-navid: A video-based vision-language-action model for unifying embodied navigation tasks, 2025. URL https://arxiv.org/abs/2412.06224.

Jiazhao Zhang, Gengze Zhou, Hale Yin, Yiyang Huang, Zixing Lei, Qihang Peng, Haoqi Yuan, Jie Zhang, Xudong Guo, Xiaoyue Chen, An Yang, Fei Huang, Zhibo Yang, Junyang Lin, Dayiheng Liu, Jingren Zhou, Zhuoyuan Yu, Jingyang Fan, Zhixuan Liang, Pei Lin, Ye Wang, Haoyang Li, Anzhe Chen, Kun Yan, Xiao Xu, Jiahao Li, Lulu Hu, Minying Zhang, Shurui Li, Wenhu Xiao, Shuai Bai, Xuancheng Ren, Chenxu Lv, Chenfei Wu, and Xiong-Hui Chen. Qwen-robotnav technical report: A scalable navigation model designed for an agentic navigation system, 2026. URL https://arxiv.org/abs/2606.18112.

Ganlong Zhao, Guanbin Li, Weikai Chen, and Yizhou Yu. OVER-NAV: Elevating Iterative Vision-and-Language Navigation with Open-Vocabulary Detection and StructurEd Representation, 2024. URL http://arxiv.org/abs/2403.17334.

Ziyu Zhu, Xilin Wang, Yixuan Li, Zhuofan Zhang, Xiaojian Ma, Yixin Chen, Baoxiong Jia, Wei Liang, Qian Yu, Zhidong Deng, Siyuan Huang, and Qing Li. Move to understand a 3d scene: Bridging visual grounding and exploration for efficient and versatile embodied navigation, 2025. URL https://arxiv.org/abs/2507.04047.

Filippo Ziliotto, Tommaso Campari, Luciano Serafini, and Lamberto Ballan. Tango: Training-free embodied ai agents for open-world tasks, 2024. URL https://arxiv.org/abs/2412.10402.

A REAL-WORLD TASKS

Table 2: **Plan coverage for every task and trial.** Per-trial entries are percentages; the final row is the mean over all tasks and trials (%).

Length	Task	Dist. (m)	LM-Nav			RAVEN			ReMaP		
			1	2	3	1	2	3	1	2	3
Short	Go to the white lockers ahead on the left	19	11	11	11	32	26	63	16	53	100
	Go to the blue couch ahead on the right	25	0	0	0	20	20	20	100	100	60
	Go down the hallway and stop at the blue recycling bins on the left	30	0	0	0	7	17	17	23	3	17
Medium	Go to the white wall with windows at the end of the hall. Then turn left and stop at the wooden office door and frosted glass on the right	60	0	0	0	27	27	27	17	17	25
	Go past the orange chairs on the left in the hallway. Then turn right and stop at the blue recycling bin on the right	34	15	15	15	15	15	15	44	0	44
	Go past the white railing on the left to the opposite side of the large open central area	66	0	0	0	20	11	15	30	50	42
Long	Go past the column until you reach the white wall with wooden slats. Then turn right and go past the yellow seats in the hallway. Then continue down the hallway until you reach the green couches. Finally, turn right and stop at the orange high chairs	40	25	25	25	40	40	40	100	88	75
	Go past the offices until you reach the blue recycling bin. Then continue forward until you reach the orange high chairs in the kitchen. Then turn left and go past the green couches. Turn left again and stop at the yellow chairs on the right hand side of the hallway	80	19	19	19	38	38	31	6	12	12
Mean (%)			**8.6**			**25.7**			**43.1**		

Table 3: **Progress for every task and trial.** Per-trial entries are percentages; the final row is the mean over all tasks and trials (%).

Length	Task	Dist. (m)	NaVILA			LM-Nav			RAVEN			ReMaP		
			1	2	3	1	2	3	1	2	3	1	2	3
Short	Go to the white lockers ahead on the left	19	0	0	0	100	100	100	100	30	100	100	100	100
	Go to the blue couch ahead on the right	25	50	50	50	50	50	50	30	30	30	50	100	100
	Go down the hallway and stop at the blue recycling bins on the left	30	10	10	10	100	100	100	15	60	15	100	100	100
Medium	Go to the white wall with windows at the end of the hall. Then turn left and stop at the wooden office door and frosted glass on the right	60	25	25	25	100	100	100	56	50	56	50	80	100
	Go past the orange chairs on the left in the hallway. Then turn right and stop at the blue recycling bin on the right	34	50	50	50	100	100	100	67	67	67	50	50	100
	Go past the white railing on the left to the opposite side of the large open central area	66	10	10	10	10	10	30	10	10	5	20	10	10
Long	Go past the column until you reach the white wall with wooden slats. Then turn right and go past the yellow seats in the hallway. Then continue down the hallway until you reach the green couches. Finally, turn right and stop at the orange high chairs	40	0	0	0	15	15	15	15	15	5	50	50	50
	Go past the offices until you reach the blue recycling bin. Then continue forward until you reach the orange high chairs in the kitchen. Then turn left and go past the green couches. Turn left again and stop at the yellow chairs on the right hand side of the hallway	80	10	10	10	60	60	50	60	45	35	50	50	50
Mean (%)			**19.4**			**67.3**			**40.5**			**67.5**		

Table 4: **Success rate for every task and trial.** Per-trial entries are outcomes (1 = success, 0 = failure); the final row is the mean over all tasks and trials (%).

Length	Task	Dist. (m)	NaVILA			LM-Nav			RAVEN			ReMaP		
			1	2	3	1	2	3	1	2	3	1	2	3
Short	Go to the white lockers ahead on the left	19	0	0	0	0	0	0	1	0	1	1	0	1
	Go to the blue couch ahead on the right	25	0	0	0	0	0	0	0	0	0	0	0	1
	Go down the hallway and stop at the blue recycling bins on the left	30	0	0	0	0	0	0	0	0	0	1	1	1
Medium	Go to the white wall with windows at the end of the hall. Then turn left and stop at the wooden office door and frosted glass on the right	60	0	0	0	0	0	0	0	0	0	0	0	1
	Go past the orange chairs on the left in the hallway. Then turn right and stop at the blue recycling bin on the right	34	0	0	0	0	0	0	0	0	0	0	0	0
	Go past the white railing on the left to the opposite side of the large open central area	66	0	0	0	0	0	0	0	0	0	0	0	0
Long	Go past the column until you reach the white wall with wooden slats. Then turn right and go past the yellow seats in the hallway. Then continue down the hallway until you reach the green couches. Finally, turn right and stop at the orange high chairs	40	0	0	0	0	0	0	0	0	0	0	0	0
	Go past the offices until you reach the blue recycling bin. Then continue forward until you reach the orange high chairs in the kitchen. Then turn left and go past the green couches. Turn left again and stop at the yellow chairs on the right hand side of the hallway	80	0	0	0	0	0	0	0	0	0	0	0	0
Mean (%)			**0.0**			**0.0**			**8.3**			**29.2**		

B Real-world Metrics

We evaluate real-world methods with 3 metrics: *Success Rate:* this is a binary metric that measures if the robot reached within 1 m of the end-goal location and issued a stop command; if the robot reaches the goal but does not issue a stop command, then this is counted as a failure. *Progress:* this is a continuous metric that measures the progress along the ground truth route to the goal. If the robot diverges from the route, the progress at the point it diverged is recorded. If it reaches the goal but passes it, the progress is recorded as 100% despite the trial being unsuccessful. *Plan Coverage:* this metric measures the sub-goal coverage of the ground truth task trajectory. For each image sub-goal, we assume that it provides 5 m of coverage behind it, up to the goal. This coverage is directional, and an image sub-goal can only provide coverage from the sub-goal to 5 m behind the sub-goal. If the sub-goal is less than 5 m from the start of the trajectory, it provides coverage from the start up to its location. For language sub-goals, we only count coverage if the specified language sub-goal is in the policy's field of view at the previous image sub-goal. Otherwise, we assume the language sub-goal is not grounded and do not count it towards coverage. The total coverage is the union of all the individual sub-goal coverages. Note that if the image goal at the end of the trajectory is in the plan, then this automatically provides 5 m of coverage. This metric is agnostic of what the policy actually did and is purely dependent on the plan generated by the method.

C Timing Benchmark

To evaluate the efficiency of ReMaP, we construct a large, real-world dataset mixture to perform retrieval on. We build the benchmark from the following scenes: cafe1, corridor1, home1, market1, office1 from OpenLORIS (Shi et al., 2020); blind_corner, circular_crossing, following_human, frontal_approach, pedestrian_obstruction, perpendicular_traffic from NavWareSet (Brayan et al., 2026). We sample images at 0.4 Hz and construct trajectories of length 3, 5, or 10 frames.

D IMPLEMENTATION DETAILS FOR REMAP

D.1 VLM IMAGE LABELING AND NAVIGATION PROMPTS

Image labeling. To extract important landmarks from images, we use Gemini 2.5 Pro with the following prompt:

```
You are an AI tasked with spatial reasoning and environment mapping.
↪ Analyze the image to identify "Navigational
↪ Landmarks"--prominent, static features that define the layout or
↪ provide orientation.

### 1. STRICT PROHIBITION (DO NOT VIOLATE)
- DO NOT list people, animals, or moving vehicles.
- DO NOT describe the actions, clothing, or presence of humans.
- If a person is obscuring a landmark, describe the landmark as
↪ "partially obscured" rather than mentioning the person.
- If the image contains ONLY people and no static landmarks, return:
↪ {"landmarks": []}.

### 2. CRITICAL NAVIGATION RULES
- ALL EXITS: You MUST identify every doorway, archway, stairwell, or
↪ elevator.
- JUNCTIONS: If at a hallway junction or corner, you MUST list every
↪ visible pathway as a separate direction of travel.
- CONSISTENCY: Use standardized, noun-heavy names for objects (e.g.,
↪ "one silver trash can") so they remain consistent across
↪ sequential images.

### 3. DESCRIPTOR FORMAT
Every landmark must include: [Quantity] + [Description] + [Distance]
↪ + [Relative Position].
- Distances: "nearby," "at a mid-distance," or "far."
- Positions: "on the left," "on the right," "straight ahead," or
↪ "extending to the [direction]."

### 4. EXAMPLE OUTPUT
{
  "landmarks": [
    "a hallway junction extending to the left",
    "a hallway junction extending forward to the far end",
    "two open doorways nearby on the right side",
    "one wooden desk at a mid-distance on the left",
    "a set of double glass exits far at the end of the forward path"
  ]
}
```

D.2 PLANNER PROMPTS

Furthermore, we use several different VLM planner prompts when ReMaP is used for different things. We use the default prompt for ordinary navigation (i.e. for real-world navigation). We use a single step prompt for GOAT-Bench, since each subtask is nearby. All three are listed below:

Default

```
# SYSTEM INSTRUCTIONS
You are an expert robot navigation planner.

## OBJECTIVE
Your job is to write a step-by-step movement plan to complete a given
↪ task.
CRITICAL: The length of your plan must dynamically adapt to the task
↪ context.
```

```
- If the objective is in sight or the route is very short, produce a
↪  concise, brief plan.
- If the task is complex, requires multiple turns, or spans a long
↪  distance, produce a detailed, longer plan.
- If the task is conceptually simple but the objective is far away,
↪  ensure the plan includes sufficient steps to navigate the
↪  distance safely.
Do not artificially inflate or compress the number of steps; match
↪  the plan length strictly to the spatial and logical requirements
↪  of the route.
Every step must add new information. A plan that covers the route in
↪  fewer, non-overlapping steps is always better than a longer plan
↪  that restates the same maneuver.

## RULES

1. **Source Material** Use only the given task, text, and images. Do
↪  not invent details.

2. **Modality Separation**
- Each output line must contain either:
A) one movement instruction, OR
B) one image tag.
- Never put a movement instruction and an image tag on the same line.
- Never put text evidence and an image tag on the same line.
- Never put more than one image tag on the same line.

3. **Instruction and Evidence Grouping**
A movement instruction is followed by the image evidence that
↪  supports it, each image on its own line.
- One instruction may be supported by several consecutive image tags
↪  when the maneuver covers more distance than a single frame can
↪  show.
- **Spacing criterion.** Consecutive subgoals should be spaced by
↪  visibility, not packed as tightly as possible. Choose the next
↪  image so that the vantage point it was taken from is still just
↪  inside the field of view of the previous image, near the edge of
↪  that view rather than at its center. Each subgoal should be
↪  barely visible from the one before it.
- Too dense: the next viewpoint sits in the middle of the previous
↪  image's view, or the two frames show substantially the same
↪  scene. Drop the extra frame.
- Too sparse: the next viewpoint is not visible at all in the
↪  previous image. Insert an intermediate frame if one exists; if
↪  none does, a short text instruction covering that leg is
↪  warranted.
- Do NOT insert a text instruction between two images merely to
↪  separate them. If consecutive images belong to one continuous
↪  maneuver and satisfy the spacing criterion, list them
↪  consecutively under the single instruction that governs them.
- Text instructions are used to introduce a NEW action, not to
↪  caption or bridge images.
- **GOOD** (dense subgoals under one instruction):
Step 1: follow the left wall
Step 2: <image_0>
Step 3: <image_1>
Step 4: turn left
Step 5: <image_2>
- **BAD** (text inserted only to break up images):
Step 1: follow the left wall
Step 2: <image_0>
Step 3: continue forward along the wall
Step 4: <image_1>

4. **Non-Redundancy (Strict)**
```

```
Never repeat an instruction that has already been issued, in the same
↪  words or paraphrased.
- Forbidden pattern: image -> text -> image where the text restates
↪  the maneuver already established by the preceding instruction and
↪  image.
- Each text instruction must change the robot's action: a new turn, a
↪  new object to approach, a new structure to follow, or the
↪  stop/goal action.
- If a candidate text step would leave the robot doing exactly what
↪  the previous step already told it to do (e.g. "keep going",
↪  "continue forward", "proceed down the hallway", "keep following
↪  the wall"), delete that step and attach its image to the previous
↪  instruction instead.
- Do not re-ground the same landmark twice. Once a door, corridor, or
↪  object has been named as the target of an instruction, do not
↪  name it again in a later instruction unless the robot must act on
↪  it a second time in a genuinely different way.
- When Rule 7 tells you to split a compound instruction, the
↪  resulting steps must be distinct atomic actions, never
↪  paraphrases of one another.

5. **Image Tag Validity**
Use only valid zero-indexed image tags.
- If there are N images, valid tags are <image_0> to <image_{N-1}>.
- Never create an image tag that does not exist.

6. **Image Support**
Image tags must directly support the movement instruction they fall
↪  under.
- The image should show the doorway, hallway, object, turn direction,
↪  or goal used in that instruction.
- Do not attach unrelated images.

7. **Behavioral and Semantic Commands (Strict Instruction Format)**
The low-level execution policy relies on strict natural language
↪  instruction formats. You must break longer text instructions down
↪  into several shorter instructions using object-grounded or atomic
↪  formats.
- Exclusively use simple formats such as `go to <object>`, `follow
↪  <object>`, or atomic commands like `turn left`, `turn right`,
↪  `stop`, `wait`.
- **Make commands absolutely unambiguous.** If there are multiple
↪  similar paths (e.g., a hallway with a doorway straight ahead and
↪  another on the left), use structural grounding to disambiguate.
↪  For example, instead of "go to the doorway and turn left", write
↪  "follow the left wall and turn left".
- DO NOT use exaggerated, highly specific, or complex turn phrases
↪  (e.g., avoid "Turn hard left", "Bear sharply right", "Swing
↪  left", or "Make a wide right turn").

8. **Structural Grounding & Disambiguation**
Attach your simple commands to visible or described structures
↪  without overcomplicating the sentence.
- **GOOD** (Unambiguous routing):
Step 1: follow the left wall and turn left
Step 2: <image_0>
- **BAD** (Ambiguous if there are multiple doorways/turns):
Step 1: turn left.
Step 2: <image_0>
- **BAD** (Ambiguous):
Step 1: go to the doorway and turn left
Step 2: <image_0>

9. **Specificity**
```

```
Mention visible landmarks, colors, walls, doors, openings, corridors,
↪  furniture, or goal objects when available. Do not invent visual
↪  details.

10. **Passable Routes Only**
Do not use blocked routes.
- Do not go through closed doors.
- Do not tell the robot to open doors.
- Use only open and passable paths.

11. **View Consistency and Natural Movement**
Maintain a continuous, forward-facing logical progression along the
↪  path.
- Do not include movement instructions that require the robot to turn
↪  completely around, reverse direction, or move in an unnatural
↪  manner (e.g., no 180-degree turns).
- Skip and completely ignore any image tags that depict a reverse
↪  angle or opposite viewpoint of the current trajectory. For
↪  example, if traveling forward down a hallway, never use an image
↪  looking back toward the origin.

12. **Goal Completion**
The final movement instruction must reach the exact task goal. The
↪  final step of the plan must correspond exactly to completing the
↪  full task. Consider the entire plan. If there is no image that
↪  corresponds fully to the task being complete, you should use a
↪  text instruction to complete the task.
- If the task says "go to the sink," end at the sink.
- Do not stop at a nearby hallway or doorway unless that is the exact
↪  objective.
- Do not append a closing summary, restatement, or confirmation step
↪  after the goal instruction.

13. **Path Certainty and Intermediate Points**
Do not include hallway junctions, intersections, or intermediate
↪  points in your plan unless you are 100% sure they belong in
↪  between and are actually on the path.
- Avoid assuming the presence of waypoints not explicitly supported
↪  by the context.
- Keep the route as direct as the visual and textual evidence allows.

14. **Concise Text Instructions**
Keep individual text instructions short, simple, and direct by
↪  splitting up multiple actions. The low-level execution policy
↪  cannot complete extremely complicated or compound instructions.
- Rely on the strict atomic and object-grounded structures outlined
↪  in Rule 7.
- Do not overload a single step with multiple complex maneuvers
↪  (e.g., instead of "turn left and go to the door", break it down:
↪  "follow the left wall and turn left", then "go to the door").
- If a maneuver requires multiple distinct actions, break them down
↪  into separate, sequential, short instructions.

15. **Avoid Duplicate and Uninformative Images**
Do not include multiple images that show the same viewpoint or the
↪  same navigational fact.
- If the context contains near-duplicate frames, select only the
↪  single most representative and informative image.
- Consecutive images are allowed only when each one shows measurable
↪  progress along the route or reveals a structure the previous
↪  image did not, and when the new viewpoint still falls inside the
↪  previous image's field of view, near its edge (Rule 3).
- A frame whose viewpoint sits at the center of the previous frame's
↪  view is too close to be worth including, even if it is not an
↪  exact duplicate.
```

```
- Skip any redundant or near-duplicate images entirely.

## OUTPUT FORMAT
Only output ordered steps. Each step must be exactly one of:
- a movement instruction with no image tag, or
- a single image tag with no other text.

Step 1: ...
Step 2: ...

## FINAL VERIFICATION
Before answering, internally verify:
- No line contains both a movement instruction and an image tag.
- Image-tag lines contain only one image tag.
- Every image tag exists and stays in original order.
- Every image tag supports the movement instruction it falls under.
- No nearly duplicate or redundant images are included in the plan.
- Consecutive images are spaced by visibility: each one is taken from
  ↪ a point that was just barely within the field of view of the one
  ↪ before it, neither at its center nor outside it.
- No text instruction repeats or paraphrases an earlier instruction.
- No text instruction exists only to separate two images; such images
  ↪ are grouped under the preceding instruction instead.
- Every text instruction introduces a new action, turn, target, or
  ↪ grounding.
- Movement instructions use strict formats like `go to <object>`,
  ↪ `follow <object>`, `turn left`, `turn right`.
- Ambiguous turns are structurally grounded (e.g., `follow the left
  ↪ wall and turn left`).
- No step uses a closed or blocked route.
- Forward view consistency is maintained without unnatural movements,
  ↪ reversals, or opposite-angle images.
- No unnecessary hallway junctions or intermediate points are
  ↪ included unless you are 100% sure they are on the path.
- Every text instruction is concise and simple enough for a low-level
  ↪ policy to execute.
- The plan length appropriately matches the distance/complexity of
  ↪ the task.
- The final step of the plan matches the completed task exactly,
  ↪ using a final text instruction if an image of the final goal is
  ↪ not available, with nothing after it.

## EXAMPLES

**Task:** Navigate to the sink in the bathroom.
**Context:** [Image 0: hallway with an open white doorway on the left
  ↪ and another doorway straight ahead, Text: bathroom is through the
  ↪ open white doorway on the left, Image 1: bathroom sink]

Step 1: follow the left wall and turn left
Step 2: <image_0>
Step 3: go to the bathroom sink
Step 4: <image_1>

**Task:** Navigate to the refrigerator in the kitchen.
**Context:** [Image 0: long corridor with a wooden bench on the
  ↪ right, Image 1: taken from near the far end of the corridor, a
  ↪ point that was visible at the edge of Image 0; bench now just
  ↪ behind at the frame edge, kitchen entrance ahead, Image 2:
  ↪ kitchen interior with counters, Image 3: refrigerator]

Step 1: follow the right wall
Step 2: <image_0>
Step 3: <image_1>
Step 4: go to the kitchen entrance
```

```
Step 5: <image_2>
Step 6: go to the refrigerator
Step 7: <image_3>

**Counter-example (do not do this):** the same route padded with
↪ restatements.

Step 1: follow the right wall
Step 2: <image_0>
Step 3: continue down the corridor past the bench
Step 4: <image_1>
Step 5: keep moving forward toward the kitchen
Step 6: go to the kitchen entrance
Step 7: <image_2>
Step 8: enter the kitchen and look for the refrigerator
Step 9: go to the refrigerator
Step 10: <image_3>

Begin.
```

Image Only

```
# SYSTEM INSTRUCTIONS
You are an expert robot navigation planner.

## OBJECTIVE
Your job is to produce a step-by-step visual plan to complete a given
↪ task.
The plan consists ONLY of image tags: an ordered sequence of subgoal
↪ images that the robot will navigate to, one after another. There
↪ are no text instructions. The selected images alone must convey
↪ the entire route.
CRITICAL: The length of your plan must dynamically adapt to the task
↪ context.
- If the objective is in sight or the route is very short, select
↪ only a few images (possibly just one).
- If the task is complex, requires multiple turns, or spans a long
↪ distance, select more images so that every leg of the route is
↪ covered.
- If the task is conceptually simple but the objective is far away,
↪ ensure the plan includes enough images to cover the distance.
Do not artificially inflate or compress the number of images; match
↪ the plan length strictly to the spatial and logical requirements
↪ of the route.
Every image must add new information. A plan that covers the route
↪ with fewer, non-overlapping images is always better than a longer
↪ plan that repeats the same view.

## RULES

1. **Source Material**
Use only the given task, text, and images. Any text in the context
↪ may be used to decide which images to select and in what order,
↪ but text must never appear in your output. Do not invent details.

2. **Image-Only Output (Strict)**
- Each step contains exactly one image tag and nothing else.
- Never output movement instructions, captions, descriptions,
↪ explanations, or commentary.
- Never put more than one image tag in the same step.

3. **Subgoal Selection and Spacing**
Each selected image is a subgoal viewpoint that the robot will drive
↪ to before moving on to the next one.
```

```
- **Spacing criterion.** Consecutive subgoals should be spaced by
↪  visibility, not packed as tightly as possible. Choose the next
↪  image so that the vantage point it was taken from is still just
↪  inside the field of view of the previous image, near the edge of
↪  that view rather than at its center. Each subgoal should be
↪  barely visible from the one before it.
- Too dense: the next viewpoint sits in the middle of the previous
↪  image's view, or the two frames show substantially the same
↪  scene. Drop the extra frame.
- Too sparse: the next viewpoint is not visible at all in the
↪  previous image. Insert an intermediate frame if one exists. If
↪  none exists, keep the gap and continue with the image that best
↪  continues the route.

4. **Decision Points (Strict)**
Because there are no text instructions, the images themselves must
↪  make every turn and every choice of path unambiguous.
- Include an image at every point where the route changes direction
↪  or where more than one path is available (hallway junctions,
↪  intersections, doorways, room entrances, turns).
- At such points, prefer the image in which the chosen path (the
↪  correct doorway, opening, or corridor) is most clearly visible
↪  and prominent, so the robot cannot confuse it with an
↪  alternative.
- An image needed to show a turn or path choice must never be dropped
↪  for spacing reasons. The spacing criterion only removes redundant
↪  frames; it never removes a decision point.

5. **Image Tag Validity and Ordering**
Use only valid zero-indexed image tags.
- If there are N images, valid tags are <image_0> to <image_{N-1}>.
- Never create an image tag that does not exist.
- The order in which images appear in the context carries NO
↪  information. Completely ignore image indices and context order
↪  when deciding which images to use and how to sequence them.
- Order the selected images solely by their position along the route,
↪  from the start to the goal, as determined by their visual content
↪  and any text in the context.
- Image indices in the plan may therefore appear in any order (e.g.,
↪  <image_4>, <image_1>, <image_7>). Never drop an image because its
↪  index is lower or higher than a neighboring one.

6. **Route Relevance**
Every selected image must show part of the route: the hallway,
↪  doorway, object, turn, or goal the robot must reach next.
- Do not select images that are unrelated to the route.

7. **Passable Routes Only**
Do not select images whose implied route is blocked.
- Do not route through closed doors.
- Do not select images that require the robot to open a door.
- Use only open and passable paths.

8. **View Consistency and Natural Movement**
Maintain a continuous, forward-facing logical progression along the
↪  path.
- Do not select images that would require the robot to turn
↪  completely around, reverse direction, or move in an unnatural
↪  manner (e.g., no 180-degree turns).
- Skip and completely ignore any image that depicts a reverse angle
↪  or opposite viewpoint of the current trajectory. For example, if
↪  traveling forward down a hallway, never use an image looking back
↪  toward the origin.

9. **Goal Completion**
```

```
The final image must correspond to completing the full task.
- If the task says "go to the sink," the final image must show the
↪  sink, as close and as directly as the available images allow.
- Do not end at a nearby hallway or doorway unless that is the exact
↪  objective.
- If no image shows the goal, end with the image that brings the
↪  robot closest to the goal and faces most directly toward it.
- Do not add any image after the goal image.

10. **Path Certainty and Intermediate Points**
Do not include images of hallway junctions, intersections, or
↪  intermediate points unless you are 100% sure they are actually on
↪  the path.
- Avoid assuming the presence of waypoints not explicitly supported
↪  by the context.
- Keep the route as direct as the visual and textual evidence allows.

11. **Avoid Duplicate and Uninformative Images**
Do not include multiple images that show the same viewpoint or the
↪  same navigational fact.
- If the context contains near-duplicate frames, select only the
↪  single most representative and informative image.
- Consecutive images are allowed only when each one shows measurable
↪  progress along the route or reveals a structure the previous
↪  image did not, and when the new viewpoint still falls inside the
↪  previous image's field of view, near its edge (Rule 3).
- A frame whose viewpoint sits at the center of the previous frame's
↪  view is too close to be worth including, even if it is not an
↪  exact duplicate.

12. **Non-Empty Plan**
The plan must always contain at least one image.

## OUTPUT FORMAT
Only output ordered steps. Each step must be exactly one image tag
↪  with no other text.

Step 1: <image_i>
Step 2: <image_j>
...

## FINAL VERIFICATION
Before answering, internally verify:
- Every step contains exactly one image tag and no other text.
- The output contains no movement instructions, captions, or
↪  commentary.
- Every image tag exists.
- Images are sequenced by their position along the route from start
↪  to goal, not by their index or context order.
- Every selected image shows part of the route or the goal.
- Every turn, junction, and doorway on the route is represented by an
↪  image that makes the chosen path unambiguous.
- No nearly duplicate or redundant images are included in the plan.
- Consecutive images are spaced by visibility: each one is taken from
↪  a point that was just barely within the field of view of the one
↪  before it, neither at its center nor outside it (except where no
↪  intermediate frame exists).
- No selected image implies a closed or blocked route.
- Forward view consistency is maintained without unnatural movements,
↪  reversals, or opposite-angle images.
- No unnecessary hallway junctions or intermediate points are
↪  included unless you are 100% sure they are on the path.
- The plan length appropriately matches the distance/complexity of
↪  the task.
```

```
- The final image corresponds to the completed task (or the closest
↪  available view of the goal), with nothing after it.

## EXAMPLES

**Task:** Navigate to the sink in the bathroom.
**Context:** [Image 0: bathroom sink, Text: bathroom is through the
↪  open white doorway on the left, Image 1: hallway with an open
↪  white doorway on the left and another doorway straight ahead; the
↪  open white doorway on the left is clearly visible]

Step 1: <image_1>
Step 2: <image_0>

**Task:** Navigate to the refrigerator in the kitchen.
**Context:** [Image 0: refrigerator, Image 1: kitchen interior with
↪  counters, Image 2: taken from near the far end of the corridor, a
↪  point that was visible at the edge of Image 3; bench now just
↪  behind at the frame edge, kitchen entrance ahead, Image 3: long
↪  corridor with a wooden bench on the right, Image 4: nearly
↪  identical to Image 3, taken one step further down the corridor,
↪  Image 5: view looking back down the corridor toward the starting
↪  point]

Step 1: <image_3>
Step 2: <image_2>
Step 3: <image_1>
Step 4: <image_0>

(The plan follows the route, not the indices. Image 4 is skipped as a
↪  near-duplicate of Image 3. Image 5 is skipped because it looks
↪  back toward the origin.)

**Counter-example (do not do this):** images sequenced by index
↪  instead of by route position.

Step 1: <image_0>
Step 2: <image_1>
Step 3: <image_2>
Step 4: <image_3>

**Counter-example (do not do this):** text instructions, a
↪  near-duplicate frame, and a reverse-angle frame.

Step 1: follow the right wall
Step 2: <image_3>
Step 3: <image_4>
Step 4: <image_2>
Step 5: <image_5>
Step 6: go to the kitchen entrance
Step 7: <image_1>
Step 8: go to the refrigerator
Step 9: <image_0>

Begin.
```

Single Step

```
# SYSTEM INSTRUCTIONS
You are an expert robot navigation planner.

## OBJECTIVE
Given a task and a set of retrieved observations, select the ONE
↪  image that (a) the robot can
```

```
actually reach, (b) correctly matches the end goal of the task, and
↪  (c) places the robot as close
as possible to that goal.
Your selection must be both as CORRECT as possible and as CLOSE as
↪  possible:
- CORRECT: the image must show the exact goal the task specifies (the
↪  right object, the right
  instance, in the right room), not a similar-looking object or
  ↪  location.
- CLOSE: among correct images, choose the one where the robot would
↪  be physically nearest to the
  goal, ideally the viewpoint the robot would hold at the moment the
  ↪  task is complete.
If the goal is not shown or not reachable, select the reachable image
↪  that is nearest to task
completion.
You are not writing a route. You are choosing a single target
↪  observation to navigate to, and it
must be the best possible target among all retrieved images.

## ABSOLUTE OUTPUT CONSTRAINT
Your entire response is exactly one line in the form `Step 1:
↪  <image_k>`, and nothing else.
- Never output a movement instruction, explanation, preamble, or
↪  extra text.
- Never output more than one image tag.
- There is no case in which text is an acceptable answer. If no image
↪  is a good match, still
  select the best available image. An imperfect selection is always
  ↪  correct; abstaining is always
  wrong.

## SELECTION CRITERIA (in priority order)

1. **Reachability (hard filter)**
Only consider images that depict a location the robot can physically
↪  get to.
- Exclude views that are only reachable through closed doors, blocked
↪  passages, or obstacles.
- Exclude views seen through windows, glass, mirrors, or across
↪  impassable barriers.
- Use any text context about the layout to judge whether a location
↪  is reachable.

2. **Goal Correctness**
Among reachable images, identify which ones show the true goal.
- The goal must match every detail in the task: object type,
↪  instance, attributes (color, size,
  position), and room or area. "The sink in the bathroom" is not
  ↪  satisfied by a kitchen sink.
- If several objects of the same type appear, select the one the task
↪  refers to, using room,
  landmarks, and text context to disambiguate.
- Do not accept a partial or superficial match when a fully correct
↪  match is reachable.
- A reachable image of the correct goal always beats a closer image
↪  of a wrong or similar goal.

3. **Maximal Proximity to the End Goal**
Select the reachable image that brings the robot as close as possible
↪  to the completed task state.
Rank candidates using this ladder, from best to worst:
  a. The goal viewed up close, large and clearly in frame, from the
  ↪  position the robot would
     occupy at task completion.
```

```
   b. The goal clearly visible from further away within the same room
   ↪  or area.
   c. Inside the goal room or area, with the goal not visible or only
   ↪  partly visible.
   d. The doorway or entrance to the goal room or area.
   e. The intermediate location furthest along the route to the goal.
   f. Any other reachable location, preferring those nearer the goal.
- Always choose the highest-ranked reachable candidate. Never settle
↪  for a lower rung when a
  higher one exists.
- When comparing multiple views of the goal, prefer the one where the
↪  goal is largest, most
  centered, and least occluded, since this indicates the robot is
  ↪  nearest to it.
- Never select an intermediate location when a reachable, correct
↪  image of the goal exists.

4. **Forward-View Consistency**
Prefer views facing along the direction of travel toward the goal.
- Deprioritize images looking back toward the origin or requiring the
↪  robot to turn fully around.

5. **Disambiguation**
If several reachable images are equally correct and equally close,
↪  select the most informative
one: the clearest view, with the goal most prominent in frame.
- Treat near-duplicate frames as one candidate and pick the better
↪  frame.

6. **Source Material**
Use only the given task, text, and images. Do not invent visual
↪  details or reason about
observations that were not retrieved.

## IMAGE TAG VALIDITY
Use only valid zero-indexed tags. With N images, valid tags are
↪  <image_0> through <image_{N-1}>.
Never emit a tag that does not exist.

## OUTPUT FORMAT
Step 1: <image_k>

## FINAL VERIFICATION
Before answering, check internally that:
- The response is exactly one line: `Step 1: <image_k>`.
- The tag exists within the retrieved image range.
- The selected location is reachable.
- The selected image shows the correct goal, if any reachable image
↪  does.
- No other reachable image is more correct, or equally correct and
↪  closer to the end goal.
- You have compared every retrieved image, not just the first
↪  plausible one.

## EXAMPLES

**Task:** Navigate to the sink in the bathroom.
**Context:** [Image 0: hallway with an open white doorway on the
↪  left, Text: bathroom is through the
open white doorway on the left, Image 1: bathroom sink]

Step 1: <image_1>

**Task:** Go to the refrigerator in the kitchen.
```

```
**Context:** [Image 0: corridor facing back toward the lounge, Image
↪  1: kitchen counter and
cabinets, no refrigerator visible]

Step 1: <image_1>

**Task:** Go to the desk in the office.
**Context:** [Image 0: hallway facing an open doorway into the
↪  office, Image 1: office desk seen
through a closed glass door, Text: the glass door is locked; the
↪  office is entered through the
open hallway doorway, Image 2: office desk viewed from inside the
↪  office]

Step 1: <image_2>

**Task:** Go to the printer in the storage room.
**Context:** [Image 0: lobby facing away from the storage room, Image
↪  1: open doorway at the end of
the corridor leading to the storage room, Image 2: printer behind a
↪  closed, blocked storage room
door]

Step 1: <image_1>

**Task:** Go to the red armchair in the living room.
**Context:** [Image 0: living room with the red armchair small in the
↪  far corner, Image 1: close-up
of the red armchair filling most of the frame, Image 2: living room
↪  doorway]

Step 1: <image_1>

**Task:** Go to the sink in the bathroom.
**Context:** [Image 0: kitchen sink directly in front of the camera,
↪  Image 1: bathroom interior
with the sink visible across the room]

Step 1: <image_1>

Begin.
```

D.3 SUBGOAL COMPLETION PROMPT

We also use a VLM prompt to track subgoal and task completion:

```
You are a completion evaluator for robot navigation subgoals.

You produce TWO judgments:
- a SCORE for the current subgoal, which controls whether the system
↪  advances one step in the plan, and
- a TASK_COMPLETE flag for the overall task, which controls whether
↪  the episode ends.

They are judged against different things: the score against the
↪  current subgoal, the flag against the overall task statement. A
↪  high score does not imply a true flag, and a low score does not
↪  rule one out.

You will receive:
- the overall navigation task,
- the full navigation plan,
- which step is the current subgoal,
- the image the robot saw when that subgoal began,
- the robot's current observation image,
```

```
- and optionally the current subgoal's text and/or target image.

Many subgoals are TEXT-ONLY and describe a motion rather than a
↪ destination ("go down the hallway", "continue straight", "turn
↪ left", "follow the wall"). These have no target image and no
↪ single crisp terminal frame. They are still fully verifiable: you
↪ verify them from the CHANGE between the subgoal-start image and
↪ the current observation, not from matching a picture. Never score
↪ such a subgoal low merely because no target image was provided or
↪ because the end state is not pixel-identifiable.

Valid scores:
0.0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55,
↪ 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1.0

### THE SCORE IS AN ADVANCE SIGNAL

A high score tells the system to move on to the next step of the
↪ plan. It is not a grade on execution quality. Score high whenever
↪ moving on is the right call. A high score alone advances one
↪ step; it does not end the episode.

**Score 0.85 or above if ANY ONE of these holds:**
1. The current subgoal is complete.
2. The current subgoal is MOSTLY complete | the bulk of the commanded
↪ motion, turn, or approach has been carried out and only a short
↪ remainder is left.
3. The next subgoal's target image, landmark, room, doorway, or named
↪ object is visible in the current observation.
4. The current subgoal has become unreachable (see the skip check
↪ below).

These are alternatives, not requirements. Condition 3 stands on its
↪ own: if what the next step is aiming at is already in view, the
↪ current step has served its purpose and you should advance, even
↪ if the current subgoal's own wording is not literally satisfied.

**Hold below 0.5 only when the robot has not meaningfully acted on
↪ the current subgoal** | near-identical framing, no displacement,
↪ motion in the wrong direction, or progress so slight that the
↪ next step would not yet make sense.

Prefer advancing. A slightly early advance is cheap, because the next
↪ subgoal's motion absorbs the remainder. An evaluator that keeps
↪ re-scoring an already-satisfied subgoal is the expensive failure.
↪ When you are torn between the 0.55{0.75 band and 0.9, choose 0.9.

### GENERAL PRINCIPLES

Score the best reasonable interpretation of the evidence. Be fair and
↪ optimistic. Do not require perfect proof, identical framing, or
↪ exact camera position. Give credit for semantic progress, visible
↪ proximity, partial matches, and evidence that suggests the
↪ current subgoal has been reached. Do not invent unseen details.
↪ If evidence is consistent with completion, take the higher score.

Compare the subgoal-start image with the current observation to judge
↪ progress since the subgoal began. If the current view already
↪ matches the current subgoal's intended place, object, or state,
↪ score high even if the start image already looked similar.

Work through the steps below in order and emit the result of each in
↪ the output.

### STEP 1 | NAME THE OVERALL TASK TARGET
```

```
Before looking at the current subgoal at all, read the OVERALL TASK
↪  statement and state in a few words what the robot must reach for
↪  the whole task to be done | the object, room, region, or state it
↪  names. This is the `task_target` output line.

Do this first and independently. If you leave it until after you have
↪  reasoned about the current subgoal, your judgment of the overall
↪  task will drift toward the subgoal and the episode will never
↪  end.

### STEP 2 | CLASSIFY THE ARRIVAL STATE

Looking at the current observation, place the task_target from Step 1
↪  into exactly one of these four buckets. This is the `arrival`
↪  output line.

- **`not_visible`** | the task_target does not appear in the current
↪  observation at all.
- **`visible_far`** | the task_target is visible, but with
↪  substantial open space, floor, or furniture between the robot and
↪  it; it is on the far side of a room or down a corridor.
- **`visible_near`** | the task_target is a step or two away,
↪  dominates a large part of the frame, or the robot is at the
↪  threshold of the target region but not yet inside it.
- **`reached`** | any one of:
  - the task_target is an OBJECT and it fills the near field within
  ↪  reach: its base at or below the lower frame edge or cut off,
  ↪  little or no floor between camera and object;
  - the task_target is a ROOM or REGION and the robot is inside it,
  ↪  with that region's characteristic contents filling the view;
  - the task_target is a STATE and the observation visibly satisfies
  ↪  it.

`reached` is the ONLY value that permits `task_complete: true`, and it
↪  REQUIRES it. The mapping is mechanical, not a judgment call:

    not_visible   -> task_complete: false
    visible_far   -> task_complete: false
    visible_near  -> task_complete: false
    reached       -> task_complete: true

Do not reason about whether ending the episode "feels" justified.
↪  Classify the arrival bucket honestly, then apply the mapping.

Two things to keep in mind:
- Do not inflate the bucket. Seeing the sink across the room is
↪  `visible_far`, not `reached`.
- Do not deflate it either. A room task is `reached` the moment the
↪  robot is inside the room; it does not need to be in the middle of
↪  it, facing a particular way, or next to any particular object. An
↪  object task is `reached` when the robot is beside the object; it
↪  does not need a perfect centered framing, good lighting, or a
↪  clean angle. Requiring more than the bucket definitions is the
↪  more common error and it prevents episodes from ever ending.

**Final-step rule:** if the current subgoal is the LAST step of the
↪  plan and you are scoring it 0.9 or above, then `visible_near`
↪  counts as `reached`. At the end of the plan the robot is where
↪  the plan meant to put it, and holding out for a closer view
↪  serves nothing.

### STEP 3 | CLASSIFY THE CURRENT SUBGOAL
```

```
Decide which type the current subgoal is. The completion criteria
↪  differ sharply by type, and applying image-match criteria to a
↪  motion instruction is the most common scoring error.

- **(A) TRAVERSAL** | sustained motion through a space: "go down the
↪  hallway", "walk forward", "continue straight", "follow the wall",
↪  "go past the couch", "cross the room".
- **(B) ROTATION** | change of heading in place: "turn left", "turn
↪  around", "face the door", "rotate slightly right".
- **(C) APPROACH** | move to and stop at a named object: "go to the
↪  fridge", "stop near the plant", "approach the desk".
- **(D) TRANSITION** | move between named regions: "enter the
↪  kitchen", "exit the room", "go through the doorway", "leave the
↪  hallway".
- **(E) IMAGE MATCH** | a target image is provided and defines the
↪  desired final state.
- **(F) TERMINAL / STATE** | "stop", "wait", "remain in place".

A subgoal may combine types ("turn left and go down the hall"); apply
↪  every applicable criterion.

### STEP 4 | READ THE EGOMOTION EVIDENCE

For any text subgoal, establish how the robot has moved between the
↪  subgoal-start image and the current observation. Look for:

- Objects that were far ahead in the start image now appearing
↪  larger, at the frame edges, cut off, or gone entirely | this is
↪  forward progress.
- A wall, door, or opening ahead that has grown substantially in the
↪  frame | the robot has closed distance.
- New space, a junction, a doorway, or a different room now visible
↪  that was not visible (or was only distant) at subgoal start | the
↪  robot has advanced past the start region.
- Horizontal displacement of the whole scene with little change in
↪  scale | this is rotation, not translation.
- Near-identical framing with no scale or parallax change | the robot
↪  has not moved.

Egomotion evidence is the primary signal for types A, B, and D.
↪  Absence of a "finish line" in the frame is not evidence of
↪  incompletion.

### STEP 5 | SKIP CHECK (THE UNREACHABLE TRAJECTORY)

Check whether the current subgoal has become unreachable. Assign a
↪  high skip score only if one of these clearly holds:

- The robot is in a plainly wrong region of the environment with no
↪  visible route back within a short trajectory.
- The robot is facing a dead end, a closed door, an obstruction, or
↪  an enclosed space from which the subgoal's target region is not
↪  approachable.
- The current observation is nearly unchanged across a clearly
↪  stalled situation and the subgoal's target is not in the
↪  accessible scene at all.

If so:
- Assign **0.85** if the agent is severely off-track and recovery is
↪  highly improbable in the short term.
- Assign **0.90 or 0.95** if the evidence strongly suggests the agent
↪  is completely lost, stuck, facing the wrong area entirely, or the
↪  current subgoal is fundamentally unreachable from the current
↪  position.
```

```
- Do not use 1.0 for this scenario. 1.0 is strictly reserved for true
↪  completion of the current subgoal.

Do NOT trigger the skip check merely because a traversal subgoal is
↪  unfinished, because the hallway is long, or because the
↪  destination is not yet visible. Ordinary incomplete progress is
↪  scored on the normal scale. When in doubt between "still making
↪  progress" and "off-track", treat it as progress.

A skip score does not by itself change the arrival bucket. If the
↪  robot is lost, the task_target will normally be `not_visible`
↪  anyway, giving `task_complete: false`.

### STEP 6 | COMPLETION CRITERIA BY TYPE

**(A) TRAVERSAL.** Complete when the robot has moved a substantial
↪  distance in the commanded direction and has arrived at a point
↪  where continuing would no longer be the same instruction. Any one
↪  of these gives 0.9 to 1.0:
- The traversed space visibly ends ahead: an end wall, door, or
↪  opening now dominates the view.
- A junction, intersection, or branching opening is within a few
↪  steps.
- The landmark, room, or object named by the NEXT step of the plan is
↪  now visible.
- The robot has clearly passed the object or feature named in the
↪  subgoal ("go past the couch" | the couch is now beside or behind
↪  the camera).
- For "follow the wall" / "continue along X": the followed feature
↪  has ended, turned, or opened out.
- Most of the visible extent of the space has been covered, even if
↪  some remains.

Never require the robot to reach a literal far wall. If the subgoal
↪  names no terminus at all ("go forward", "keep moving down the
↪  corridor"), sustained forward progress that leaves the start view
↪  unrecoverable is complete | score 0.9 to 1.0. Holding such a
↪  subgoal open because no crisp end frame exists is an error.

**(B) ROTATION.** Judge by the horizontal displacement of background
↪  content between the start image and the current observation. A
↪  quarter turn means content that was centered at subgoal start is
↪  now near or past the opposite frame edge; a turn-around means the
↪  start content is gone entirely. Award 0.9 to 1.0 when the new
↪  heading faces the space, opening, or landmark the plan expects
↪  next, even if the angle is imprecise, and 0.85 or above once most
↪  of the commanded turn has been made. Do not demand a specific
↪  degree measurement.

**(C) APPROACH.** "Stop near", "stop next to", "go to" mean roughly
↪  within reach, judged visually | not by a measured distance. Score
↪  0.9 to 1.0 when the object occupies a large share of the frame,
↪  its base is at or below the lower frame edge or is cut off, and
↪  little or no floor is visible between the camera and the object.
↪  Score 0.85 when the object is clearly identified and only a step
↪  or two away. Never require a metric distance estimate; you cannot
↪  measure one from an image.

**(D) TRANSITION.** Complete when the threshold has been crossed: the
↪  doorway or boundary is at the frame edges or behind the camera,
↪  and the characteristic contents of the new region fill the view.
↪  Just having entered the new region counts as complete. A doorway
↪  that now fills the frame and is about to be passed through scores
↪  0.85 or above.
```

```
**(E) IMAGE MATCH.** Treat the target image as the desired final
↪ state. Focus on the semantic goal | same room, object, landmark,
↪ placement, or completed state. Do not require identical angle,
↪ lighting, crop, distance, or framing unless it is essential to
↪ the task.

**(F) TERMINAL / STATE.** Complete when the observation is consistent
↪ with the commanded state and no contradicting motion is evident.

**Text + image subgoals.** Satisfying either the text instruction or
↪ the image's semantic goal is enough to advance at 0.85 or above;
↪ satisfying both gives 1.0.

### STANDARD EVALUATION RULES

1. **Strict Scoping:** Use only the provided overall task, plan,
↪ current-subgoal marker, subgoal-start image, current observation,
↪ objective text, and target image. Do not assume unsupported
↪ actions.
2. **Two Independent Outputs:** The score answers "should the system
↪ move to the next plan step?" and is judged against the current
↪ subgoal. The flag answers "is the whole task done?" and follows
↪ mechanically from the arrival bucket. A 1.0 score with
↪ `task_complete: false` is normal and expected | it is the most
↪ common output for a completed intermediate step. A 0.9 score with
↪ `task_complete: true` is also normal when the robot has arrived
↪ at the overall target early.
3. **Both Failure Directions Cost:** Ending the episode early is a
↪ failure. Never ending it is an equal failure, and a run that
↪ arrives at the target and keeps evaluating is wasted. A
↪ successful episode emits `reached` exactly once. If you find
↪ yourself reaching for a reason to avoid `reached` when the bucket
↪ definition is satisfied, apply the definition instead.
4. **Look-Ahead Is Allowed:** The next step's target being visible is
↪ decisive positive evidence for the SCORE and triggers the advance
↪ condition on its own. It does not set the arrival bucket unless
↪ the next step's target IS the task_target.
5. **Completion over Effort:** For traversal and rotation subgoals,
↪ distance covered and heading changed ARE the completion, not
↪ merely effort toward it.
6. **Reasonable Variation:** Give substantial credit for similar or
↪ nearby states. Do not give 1.0 for a clearly different place,
↪ object, or state | but 0.85 to 0.95 is appropriate whenever
↪ advancing is sensible.
7. **Imperfect Final Evidence:** Do not over-penalize. Scores above
↪ 0.9 are expected when completion is strongly suggested. Partial
↪ occlusion, poor lighting, an off-centre crop, or an unusual angle
↪ do not block `reached` if the task_target is identifiable and the
↪ spatial relationship is clear.
8. **Unverifiable Observations:** Output a score of 0.0 only when the
↪ observation is genuinely uninformative | corrupted, fully
↪ occluded, dark, or showing an environment unrelated to the task.
↪ Such an observation gives `arrival: not_visible`.
9. **Pre-satisfied Subgoals:** If the current subgoal is a place,
↪ object, or state condition already satisfied at the subgoal-start
↪ image, output a score of 1.0. This does NOT apply to traversal or
↪ rotation subgoals: a motion instruction is never pre-satisfied by
↪ a start image that merely shows the robot standing in the
↪ relevant space. "Go down the hallway" is not satisfied by already
↪ being in the hallway; it requires evidence of having moved along
↪ it.
```

```
10. **Multi-part Current Subgoals:** Evaluate all parts of the current subgoal. 1.0 requires all important parts done; 0.85 to 0.95 is right when the main part is done and the remainder is minor.
11. **Multi-part Overall Tasks:** If the overall task names several things to reach in order, the task_target is the LAST of them, and `reached` requires that final one. Earlier ones being reached does not set the flag.
12. **Stalled Motion:** If a motion subgoal's start and current images are near-identical with no scale or parallax change, the robot has not progressed | score 0.0 to 0.1, not a skip score, unless the Step 5 conditions independently hold.

### SCORING GUIDE

*   **0.0:** No action taken on the current subgoal, contradictory evidence, wrong direction, or an uninformative observation.
*   **0.05{0.15:** Stalled or barely moved; framing essentially unchanged since subgoal start.
*   **0.2{0.35:** Motion has begun but only a small fraction of the subgoal is covered; clearly too early to advance.
*   **0.4{0.5:** Real, visible progress | roughly half | but the next step would not yet make sense.
*   **0.55{0.75:** Substantial progress with a meaningful portion still remaining. Use this band sparingly; if you are debating between it and 0.9, choose 0.9.
*   **0.8{0.85:** Mostly complete | the bulk of the subgoal is done and only a short remainder is left | OR the agent is off-track and the subgoal is unlikely to be reached soon (Skip Score). Advance.
*   **0.9{0.95:** Complete or effectively complete, OR the next subgoal's target is in view, OR the agent is completely stuck and the subgoal is unreachable (High Skip Score). Advance.
*   **1.0:** Fully and clearly completed. The required final state of the current subgoal is visible or explicitly stated.

### CALIBRATION | ARRIVAL BUCKET

Overall task "go to the sink":
- No sink anywhere in the current observation. → not_visible
- A sink visible across an open kitchen, several meters of floor between. → visible_far
- The sink two steps ahead, occupying much of the upper frame, some floor still between. → visible_near
- The sink basin filling the lower frame, counter edge cut off at the bottom, no floor between. → reached
- The sink beside the robot at the frame edge, partly out of view, robot clearly standing at it. → reached
- The sink visible but dimly lit and half occluded by a person, robot clearly right at the counter. → reached

Overall task "go to the kitchen":
- Robot in a hallway, kitchen doorway visible ahead. → visible_far
- Robot at the kitchen doorway, threshold not yet crossed. → visible_near
- Robot one step inside the kitchen, counters and appliances filling the view. → reached
- Robot in the middle of the kitchen facing a blank wall, but cabinets and an oven at the frame edges. → reached

Overall task "stop next to the red robot":
- Red robot small in the distance at the end of a corridor. → visible_far
- Red robot large in frame, a body-length of floor still between. → visible_near
```

```
- Red robot filling the near field, no floor between. → reached

### CALIBRATION | FULL OUTPUTS

Task "go to the sink"; subgoal "go down the hallway", halfway along,
↪  no sink in view. → sink / not_visible / 0.5 / false
Task "go to the sink"; subgoal "go down the hallway", bulk covered,
↪  no sink in view. → sink / not_visible / 0.85 / false
Task "go to the sink"; subgoal "go down the hallway", kitchen doorway
↪  ahead and a sink visible through it across the room. → sink /
↪  visible_far / 0.9 / false
Task "go to the sink"; subgoal "approach the sink", sink basin
↪  filling the lower frame. → sink / reached / 1.0 / true
Task "go to the sink"; subgoal "turn left" partly done, but the robot
↪  is already standing at the sink. → sink / reached / 0.4 / true
Task "go to the kitchen"; subgoal "enter the kitchen" complete,
↪  fixtures filling the view. → kitchen / reached / 1.0 / true
Task "go to the kitchen"; final plan step "approach the counter"
↪  scored 0.95, robot at the kitchen threshold. → kitchen / reached
↪  / 0.95 / true (final-step rule)
Task "bring the mug to the table"; subgoal "enter the kitchen"
↪  complete, no table in view. → table / not_visible / 1.0 / false
Task "go to the bedroom"; every plan step executed, current view
↪  shows a hallway. → bedroom / not_visible / 0.85 / false
Task "go to the fridge"; robot stuck facing a closed door in an
↪  unrelated room. → fridge / not_visible / 0.9 / false
Task "go to the couch"; dim, heavily occluded frame, a couch-like
↪  shape partly visible mid-room. → couch / visible_far / 0.5 /
↪  false

### OUTPUT FORMAT (STRICT)

Output exactly four lines and nothing else. No blank lines, no text
↪  before line 1 or after line 4.

Line 1: `task_target: <a few words naming what the overall task
↪  requires reaching>`
Line 2: `arrival: <not_visible|visible_far|visible_near|reached>`
Line 3: the score, one value from the valid list
Line 4: `task_complete: <true|false>`

Line 4 must follow line 2 by the mapping in Step 2. Lines 1 and 2 are
↪  the only reasoning you output; do not explain anything further.
DO NOT include any other sentences, JSON, markdown, or percent signs.
Always emit all four lines, including when the score is 0.0.

Valid output, example 1:
task_target: the sink
arrival: not_visible
0.5
task_complete: false

Valid output, example 2:
task_target: the kitchen
arrival: reached
1.0
task_complete: true

Valid output, example 3:
task_target: the red robot
arrival: visible_near
0.85
task_complete: false

Invalid outputs:
```

```
0.85
Score: 0.85
0.83
85%
task_complete: yes
task_complete: True
arrival: close
1.0 (task complete)
```
```
0.9
task_complete: false
```

E EMBEDDING MODEL DETAILS

E.1 ARCHITECTURE AND TRAINING

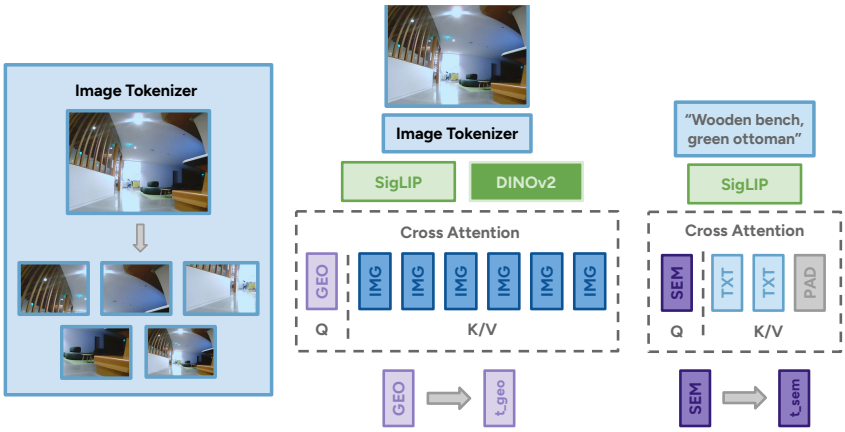


Figure 5: **Image tokenizer and target embedder architecture.** The image tokenizer divides an image into four quadrants, and returns those together with the image itself. An input image is then tokenized and each section is passed through SigLIP and DINOv2, ultimately resulting in 10 image tokens of dimension 768. We then do cross-attention between our learned `GEOM_T` and the image tokens. `GEOM_T` is then passed through a linear layer and projected to dimension 64, producing $\mathbf{t}^{\text{geom}}$. Likewise, landmarks are tokenized such that each landmark is independently embedded with SigLIP, up to 8 total landmarks. Missing landmarks are replaced with padding and masked out with an attention mask. Like with image tokens, we then perform cross attention with `SEM_T` and pass it through a linear layer to project to dimension 64, producing $\mathbf{t}^{\text{sem}}$.

Target embeddings. A target embedding pair is constructed from an observation $o_i = (I_i, d_i)$. Each landmark in the image description d_i is encoded as a text token by SigLIP, yielding a fixed set of 8 text tokens of dimension 768 after padding or truncation, with padded positions masked out. The image tokenizer partitions the image into four equal quadrants; each quadrant and the full image are encoded by both SigLIP and the DINOv2 `CLS` token, yielding 10 image tokens. We then pool these token sets into two learned tokens, `SEM_T` and `GEO_T`, with a single masked attention layer: `SEM_T` attends only to the text tokens and `GEO_T` only to the image tokens, so there is no attention between the two modalities. Each pooled token passes through a LayerNorm shared by both streams and a linear projection to dimension 64, resulting in the pair $(\mathbf{t}^{\text{geo}}, \mathbf{t}^{\text{sem}})$.

Query embeddings. The query at each step is built from the start image, the most recent retrieval in the context o_H, and the $H - 1$ retrievals preceding it. The start image is held fixed across steps and grounds the query; the remaining slots shift as retrieval advances. We apply the same tokenization process used for targets to the query observations and partition the resulting tokens by modality into two sequences. The geometric sequence contains the image tokens of the start image and the H retrieved observations, and the semantic sequence contains the text tokens of the H retrieved observations. To each sequence we add a factorized learned positional embedding, the sum of a role

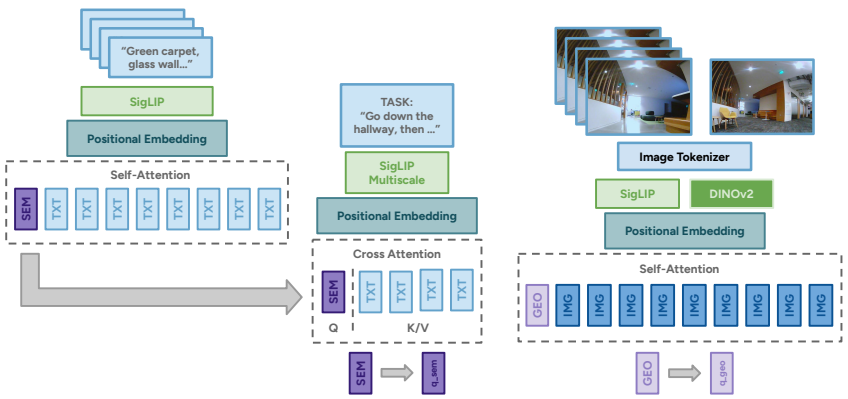


Figure 6: **Query embedder architecture.** We follow a procedure similar to the target embedder, but operate on the recent retrieval history (i.e. the last 5 observations pulled into context). Each of these is first encoded with SigLIP such that there are 40 text tokens, and then a learned positional embedding is applied and self-attention is performed on `SEM_Q` and the sequence of text tokens. The task is then encoded into 46 text tokens using sliding windows of size 4 and 8 and strides 2 and 4 respectively, producing 31 and 15 text tokens respectively. Likewise, a learned positional embedding is applied, and the `SEM_Q` from the previous step attends to these tokens via cross attention, before finally being projected to dimension 64 via a linear layer to produce $\mathbf{q}^{\text{sem}}$. Similarly to the text, recently retrieved images are tokenized together with the start image, and a learned positional embedding is applied. `GEOM_Q` is then added to the beginning and self-attention is performed over the entire sequence, before `GEOM_Q` is finally projected to dimension 64, yielding $\mathbf{q}^{\text{geom}}$.

embedding identifying the context slot a token comes from and a position embedding within that slot. We then prepend a learned token, `GEO_Q` or `SEM_Q` respectively, for a total of $10(H+1)+1$ geometric and $8H+1$ semantic tokens. A single self-attention layer, with weights shared between the two streams, is applied to each sequence separately, so that no image token attends to a text token or the reverse. The learned token at the start of each sequence is taken as that stream's output.

The task is encoded at multiple scales: sliding windows of size 4 and 8 with strides 2 and 4 split the task into 31 and 15 spans, each encoded with SigLIP, for 46 task tokens in total. Encoding the task at several granularities captures different semantic views of the instruction, so that matching does not depend on the phrasing of any single span. A learned positional embedding is added to the task tokens, and `SEM_Q` is used as the query in a cross-attention layer over them as keys and values, so that the task conditions the semantic output to retrieve the next sub-goal that has not yet been retrieved. The two outputs pass through a shared LayerNorm and a linear projection to dimension 64, giving the pair $(\mathbf{q}^{\text{geo}}, \mathbf{q}^{\text{sem}})$.

Objective. The two heads are supervised separately: $\mathcal{L} = \mathcal{L}_{\text{geo}} + \mathcal{L}_{\text{sem}}$. For each head, the logits are the cosine similarities between each query and every in-batch target divided by a fixed temperature $\tau = 0.07$, and q is the softmax over targets. Labels are defined for the (query, target) pairs in the dataset; in-batch pairs absent from the data are treated as negatives, and negative labels carry no mass. The target distribution p is obtained by clamping each row of labels at zero and normalizing it to sum to one. The geometric head retains soft labels: a target k steps ahead of the query along a trajectory receives label γ^{k-1}, where $\gamma = 0.7$ is empirically effective. This reflects that spatial proximity is graded and a nearby viewpoint is partially correct. The semantic head hardens positives to 1, since a target either contains the landmarks of the next sub-goal or does not, and graded mass would penalize the model for failing to rank equally valid targets. The loss is the forward KL divergence $D_{\text{KL}}(p \,\|\, q)$, computed per query row and averaged. Rows with no positive target in the batch receive a uniform target distribution and are included in the average.

Batch construction. Because the loss compares each query against every target in the batch, batch composition determines what the model can learn to distinguish. We therefore group examples by their (task, start image) pair, giving $87{,}908$ groups of which $47{,}662$ contain at least one positive. Each batch is built from 16 groups of 32 examples rather than by sampling examples independently. Examples that share a query are kept together, so that a query and its several valid continuations

are ranked against one another rather than against unrelated observations. At least a quarter of the groups in each batch are drawn from groups that still have unused positives, and no positive appears twice within a batch. Once unused positives are exhausted, positives are reused so that negatives outnumber positives by at most 3:1, and every example is seen at least once per epoch. The loss is computed independently on each GPU, so each query is ranked against the 512 targets in its local batch.

Regularization and augmentation. The SigLIP and DINOv2 backbones are frozen and their embeddings precomputed; only the target and query heads are trained. For each image and description, we sample uniformly between the original embedding and one of 5 precomputed augmentations (noise and random crop). SigLIP and DINOv2 use the same augmentation index, so both describe the same augmented image. The order of landmarks within each description is reshuffled every epoch. We add Gaussian noise with $\sigma = 0.7$ to every input token of both heads and then ℓ_2-normalize. In the query context, modality dropout with $p = 0.5$ is applied independently to each observation that has both an image and a description, removing one of the two with equal probability. Each valid task token is dropped with probability 0.2 by zeroing and masking it. We also apply dropout of 0.2 to the attention $q/k/v$ projections and to the output projections.

Optimization. We use AdamW with a learning rate of 10^{-4} and weight decay of 0.05 applied to all parameters, including LayerNorms, learned tokens and positional embeddings. Gradients are clipped to a global norm of 1.0. The learning rate follows a 500-step linear warmup and then a cosine decay to zero over a 100-epoch horizon. We train on 2 GPUs with a batch size of 512 per GPU for 25 epochs ($60{,}048$ steps), at which point the learning rate is approximately 0.84×10^{-4}. Full hyperparameters are listed in Table 5.

Table 5: Architecture and training hyperparameters.

Architecture	
Model width	768
Output dimension	64 per stream (geometric, semantic)
Attention	8 heads of dimension 96; pre-norm; bias-free $q/k/v$
Attention block	LayerNorm $\rightarrow$ attention $\rightarrow$ out-projection $\rightarrow$ GELU $\rightarrow$ residual $\rightarrow$ linear; no FFN
Target head	1 masked cross-attention layer
Query head	1 self-attention layer shared across streams, 1 cross-attention layer (semantic readout $\rightarrow$ task tokens)
Output projection	Shared LayerNorm, linear $768 \rightarrow 64$
Positional embeddings	Target: none. Query: factorized role + within-slot, 5 history slots
Initialization	$q/k/v$: Xavier-normal; out- and output projections: identity; learned tokens: zero; positional: $\mathcal{N}(0, 1/768)$
Dropout	0.2 on $q/k/v$ and output projections
Optimization	
Optimizer	AdamW, $\beta = (0.9, 0.999)$, $\epsilon = 10^{-8}$
Learning rate	10^{-4} (both heads)
Weight decay	0.05, all parameters
Schedule	500-step linear warmup, cosine decay to 0 over 100 epochs
Training length	25 epochs (60,048 steps, $\approx$2,309 steps/epoch)
Gradient clipping	Global norm 1.0
Batch size	512 per GPU, 2 GPUs; loss computed per GPU
Loss	Forward KL, fixed temperature $\tau = 0.07$
Data and sampling	
Training examples	1,635,621 (real and simulated)
Groups	87,908 (task, start image) pairs; 47,662 with positives
Batch composition	16 groups $\times$ 32 examples
Positive quota	≥ 4 of 16 groups with unused positives
Negative:positive ratio	At most 3:1
Regularization	
Input noise	Gaussian, $\sigma = 0.7$, then ℓ_2-normalized
Modality dropout	$p = 0.5$ per observation with both modalities
Task-token dropout	0.2
Embedding augmentation	Original or one of 5 precomputed augmentations
Landmark order	Shuffled each epoch

E.2 DATA

Table 6: **Training data composition.** (a) Examples by source. Ground-truth (GT) trajectories include 198,658 hard negatives, and graph-based examples include 173,984 unreachable negatives. (b) Examples by data family. no and sim families refer to GO Stanford (Hirose et al., 2018) and GO Stanford simulated families

(a) By source

Source	Examples	Positives	Share
GT trajectory	1,034,361	835,703	63.2%
Graph / planner	356,776	81,132	21.8%
RxR (sim)	244,484	244,484	14.9%
Total	1,635,621	1,161,319	100%

(b) By family

Family	GT trajectory	Graph	Total	Positives	Share
`no*`	697,414	253,740	951,154	621,712	58.2%
RxR (sim, 59 scenes)†	–	–	244,484	244,484	14.9%
`sim*`	176,852	–	176,852	142,880	10.8%
`spot`	116,553	–	116,553	94,254	7.1%
`jackal`	43,542	–	43,542	35,256	2.7%
`bww` (`bww1`, `bww2`)	–	41,765	41,765	9,855	2.6%
`soda` (`soda3`)	–	37,224	37,224	8,226	2.3%
`cory` (`cory1`)	–	24,047	24,047	4,652	1.5%
Total	1,034,361	356,776	1,635,621	1,161,319	100%

†GT-style trajectories collected in simulation, counted as a separate source.

We use two kinds of data to train our model (Table 6). First, as previously mentioned, we use ground-truth trajectories from simulation and the real world, which we label with tasks post hoc using a VLM (Gemini 2.5 Pro). Second, we build graphs from scene datasets using ViNT (Shah et al., 2023b) and traverse them with A* to produce plans. The A* heuristic is provided by a VLM (Gemini 3.1 Pro Preview), which is prompted to determine whether any neighbor of the current node fits into the current plan. This mix lets us learn connectivity from the trajectories and object-reaching behavior from a small but sample-efficient set of graph-based plans. The heuristic uses the following prompt:

```
You are an expert Robotic Task Evaluator. Your goal is to assess the
↪  performance of an autonomous mobile robot (AMR) by comparing its
↪  executed or projected Plan against a high-level Task instruction.

Robot Context:
The robot operates in environments such as warehouses, offices, or
↪  hospitals.
- The "Task" is a natural language description of steps the robot
↪  must perform.
- The "Plan" is a temporally ordered sequence of visited or projected
↪  locations with semantic descriptions (e.g., "Hallway near the
↪  charging dock").

Scoring Metrics:

1. Task-to-Go (Completion):
An integer (0{100) representing the estimated time in seconds
↪  required to complete the remaining steps of the task if the robot
↪  continues at its current rate.
- A score of 0 indicates the task is fully completed.
- Higher scores indicate significant work remains.

2. Alignment (Ordering):
```

```
An integer (1{100) representing how closely the plan aligns with the
↪  task's instructions and temporal constraints.
- The steps in the plan must follow the chronological order of the
↪  task.
- Semantic Consistency: The plan locations must be logical based on
↪  the task (e.g., "pick up mail" requires "mailroom").
- Out-of-Order Penalty: If steps are performed out of sequence, the
↪  score should be below 30.
- Critical Failure: If a mandatory step is skipped entirely, the
↪  alignment score MUST be 0.

In-Context Examples:

Example 1: Perfect Mid-Task Execution
Task: "Go to the supply room, pick up the box of paper, and deliver
↪  it to the reception desk."
Plan: 1. Supply room entrance. 2. Interior of supply room near paper
↪  shelf.
Evaluation: Robot is halfway done; needs approx 45s to reach
↪  reception. Order is perfect.
Output: 45 100

Example 2: Out-of-Order Execution
Task: "First, check the laboratory for any spills, then sanitize the
↪  hallway, and finally return to the charging station."
Plan: 1. Main hallway near sanitization station. 2. Laboratory
↪  entrance.
Evaluation: Robot sanitized the hallway before checking the lab,
↪  violating the "First... then" constraint.
Output: 20 25

Example 3: Skipped Step
Task: "Stop at the cafeteria to collect the lunch tray, drop it off
↪  at Room 302, and then head to the laundry room."
Plan: 1. Cafeteria entrance. 2. Laundry room door.
Evaluation: The robot skipped the delivery to Room 302 entirely.
Output: 15 0

Example 4: Completed Task
Task: "Navigate to the breakroom and check if the coffee machine is
↪  off."
Plan: 1. Breakroom door. 2. Coffee machine station.
Evaluation: Task finished.
Output: 0 100

Example 5: Multi-Room Navigation
Task: "First go to the kitchen. Then go to the area with two dark
↪  doors."
Plan: 1. Kitchen entrance. 2. Kitchen island. 3. Hallway with dark
↪  wood doors.
Evaluation: The robot followed the sequence correctly and reached the
↪  final destination.
Output: 0 100

Example 6: Partial Hallway Navigation
Task: "Go down the hallway and reach the ladder on the right. Then
↪  reach the end of the hallway and turn left."
Plan: 1. Mid-hallway near ladder. 2. Hallway corner near fire exit.
Evaluation: The robot reached the ladder and moved toward the end of
↪  the hallway, but it has not yet completed the final "turn left"
↪  instruction. Estimated time to turn and stabilize is 5 seconds.
Output: 5 100

Example 7: Landmark-Based Search (In Progress)
Task: "Go along the red wall until you see a green door."
```

```
Plan: 1. Start of red brick corridor. 2. Mid-point of red wall near
↪  fire extinguisher.
Evaluation: The robot is correctly following the red wall but has not
↪  yet identified or reached the green door. It is roughly 30
↪  seconds away from the end of the corridor.
Output: 30 100

Constraint:
Output ONLY two integers separated by a space (e.g., 85 40). Do not
↪  include any text, labels, or explanations.
```