AgentVLN: Enhancing Vision Language Navigation with a Reasoning Agent based on Explicit Memory Module

by

Runhao Li (Rain)

Supervisors: Prof. Steven Waslander, Dr. Mingfeng Yuan

April 7th , 2026

B.A.Sc. Thesis



Division of Engineering Science
UNIVERSITY OF TORONTO

Contents

1 Introduction **3**

2 Literature Review **5**

2.1 Vision Language Models 5

2.2 Vision Language Navigation 6

2.2.1 Tasks and Benchmarks 6

2.2.2 Related Work and Challenges 7

2.3 Scene Graph 10

2.3.1 Overview 10

2.3.2 Related work 11

2.4 Video Captioner 13

3 Problem Analysis **15**

3.1 Current Gap 18

4 Architecture and Chosen Foundation Models **20**

4.1 High-Level Architecture 20

4.2 Chosen Foundation Models 21

4.2.1 VLN Backbone: NaVILA 21

4.2.2 Scene Graph Backbone: OpenGraph 22

4.2.3 Video Captioning Backbone: NVILA 23

4.2.4 Reasoning Agent: GPT-5 nano 24

5 Methodology **26**

5.1 Overview 26

5.2 Problem Setup and Notation . . . 27
5.3 Online Scene Graph Memory . . . 29
5.4 Online Video Caption Memory . . . 31
5.5 Reasoning Agent . . . 32
5.5.1 Instruction Decomposition into Subinstructions . . . 32
5.5.2 Landmark Matching with Scene-Graph Objects . . . 33
5.5.3 Landmark Distance and Relation Tracking . . . 35
5.5.4 Caption-Based Trajectory Summarization and Current-State Inference . . . 37
5.5.5 Deviation Detection . . . 38
5.5.6 Recovery Mode for Off-Track Navigation . . . 41

6 Results **44**

6.1 Evaluation Setup Overview . . . 44
6.2 Main Quantitative Results on VLN-CE Benchmarks, R2R-CE dataset . . . 45
6.3 Ablation Study . . . 47
6.4 Deviation Detection Performance . . . 50
6.5 Recovery and Replanning Performance . . . 52

7 Discussion **54**

7.1 Strength and Limitations . . . 54
7.1.1 Effect of Explicit Memory on Navigation Robustness . . . 54
7.1.2 Runtime Overhead and Real-Time Limitations . . . 55
7.1.3 Strengths and Limitations of the Recovery Framework . . . 56
7.1.4 Limitations of the Current Scene Graph Representation . . . 58
7.1.5 Trade-off Between Robustness and Efficiency . . . 59

7.2 Future Improvements 60
7.2.1 Improving Scene-Graph Reliability through Task-Relevant Filtering 61
7.2.2 Extending Recovery from Local Correction to Task-Level Guidance 62
7.2.3 Reducing Runtime Overhead for Smoother Real-Time Execution . 63

8 Conclusion **64**

List of Figures and Tables

1 Failure cases for not realizing the target object 16

2 Failure case for two hallways and doesn't follow the correct one 17

3 AgentVLN high level architecture 20

4 OpenGraph's hierarchical layers visualization 29

5 Example of Instruction Decomposition subsection 33

6 Example of landmark matching stage 34

7 Example of Landmark Distance Description subsection 35

8 Example of Summarized Video Caption subsection 37

9 Deviation Detection architecture 38

10 Example of on-track from Deviation Detection 39

11 Example of off-track from Deviation Detection 40

12 Illustration of Recovery Mode 42

List of Tables

1 Comparison of SOTA methods on Val-Unseen of R2R-CE 15

2 Reasons for Failure cases on Val-Unseen of R2R-CE on NaVILA. 15

3 Comparison of SOTA methods on LH-VLN. 18

4 Comparison SOTA models and ours with conservative and aggressive settings 46

5 Ablation study of explicit memory and reasoning components on R2R-CE val-unseen. 49

6 Deviation detection performance under unsuccessful and successful episode settings. 52

7 Recovery and replanning performance after off-track detection. 53

Acknowledgements

I would like to sincerely thank my supervisor, Prof. Steven Waslander, for his guidance, support, and valuable feedback throughout this thesis. I am also grateful to Dr. Mingfeng Yuan for technical discussions, suggestions, and help throughout the development of this work. I would further like to thank the members of the Toronto Robotics and AI Laboratory (TRAIL) and the University of Toronto for providing a supportive and inspiring research environment. Finally, I would like to thank my family and friends for their encouragement and support throughout this project.

Abstract

Vision Language Navigation (VLN) aims to enable embodied agents to follow natural-language instructions in visually rich environments. Although recent VLN models achieve strong performance on standard benchmarks, most still rely on implicit memory represented through hidden states, temporal features, or transformer embeddings. This makes it difficult for the agent to explicitly track progress, diagnose deviation, and recover reliably in long-horizon or ambiguous navigation tasks. To address this limitation, this thesis proposes AgentVLN, a modular framework that augments a strong VLN backbone with explicit memory and a separate reasoning agent. The proposed system maintains two online memory streams: a scene-graph memory that stores objects, regions, and spatial relations, and a video-caption memory that summarizes recent observations and movement progress in natural language. A reasoning agent periodically reads these memories to decompose instructions, ground landmarks, detect off-track behavior, and generate corrective recovery instructions.

Experiments on the R2R-CE unseen-validation setting in Habitat show that AgentVLN improves task completion over the NaViLA baseline, achieving higher Oracle Success Rate and Success Rate under the best reasoning setting. Ablation studies show that caption-based semantic memory contributes strongly to robustness, while the scene graph provides additional structured spatial support but remains limited by noisy object labels. Recovery analysis further shows that once off-track behavior is detected, the replanning module successfully restores the robot toward a valid local route in a majority of cases. Overall, this thesis provides a proof of concept that explicit memory and reasoning can improve the robustness and interpretability of VLN, while also revealing important limitations in scene-graph quality, recovery design, and runtime efficiency.

1 Introduction

Vision Language Navigation (VLN) studies how an embodied agent can follow natural-language instructions to navigate through an environment using visual observations. Recent progress in vision–language models and large language models has significantly improved instruction understanding, visual grounding, and action prediction, leading to strong performance on standard VLN benchmarks. However, these systems remain brittle in practical navigation scenarios. They often fail when instructions contain ambiguous object references when multiple regions share similar structure or semantics, or when the task requires long horizon progress tracking across several subgoals. The agent must remember where it has been, relate current observations to previously seen landmarks, and determine whether it is still following the intended route in such cases. These requirements make memory and reasoning central to robust VLN.

Despite these advances, most state-of-the-art VLN systems still rely primarily on *implicit* memory. Past observations are usually carried through frame histories, hidden states, or transformer embeddings inside an end-to-end policy. Although these representations may preserve useful context, they do not provide an explicit, queryable account of objects, regions, spatial relationships, or recent semantic progress. As a result, the agent may act on past information without being able to explicitly reason about it. This makes it difficult to diagnose navigation failures and determine whether a landmark has been missed, or perform reliable replanning when the robot deviates from the intended path. Existing improvements are therefore often achieved by scaling model capacity or training data rather than by equipping the agent with structured memory and interpretable reasoning during inference.

This thesis addresses the following research problem: *can explicit memory and reasoning improve the robustness of VLN beyond standard implicit-memory pipelines?* To answer this question, this thesis proposes **AgentVLN**, a modular framework that augments a strong VLN backbone with two explicit memory representations and a separate reasoning agent. The first memory module is an online scene graph that incrementally stores objects, regions, and spatial relations in a structured form. The second is an online video-caption memory that summarizes recent observations and movement progress in natural language. A reasoning agent operates over these memories to decompose instructions into subgoals, ground landmarks, detect off-track behavior, and generate recovery instructions when deviation occurs. The resulting commands are then executed by the downstream VLN and control pipeline.

By moving beyond purely end-to-end implicit-memory navigation, this thesis aims to provide a proof of concept that explicit memory and reasoning are promising directions for building more robust embodied navigation systems. The goal of this thesis is not to replace existing VLN backbones, but to demonstrate that explicit memory can make navigation more interpretable, more diagnosable, and more recoverable. Specifically, this thesis has three objectives:

1. Design explicit online memory modules for VLN, including a scene-graph memory for structured spatial representation and a video-caption memory for semantic trajectory summarization.

2. Develop a reasoning agent that uses these memories for instruction decomposition, landmark grounding, deviation detection, and replanning.

3. Evaluate whether the proposed framework improves navigation robustness on VLN-CE benchmarks, especially under unseen and failure-prone scenarios.

2 Literature Review

2.1 Vision Language Models

Modern artificial intelligence models have made significant progress in enabling machines to understand and reason over human language. Nowadays, transformer [1] based Large language models (LLM) trained on massive textual data and learned rich semantic representations that capture syntactic structure, contextual meaning and high-level relationships between words and concepts. These models demonstrate strong capabilities in language understanding and reasoning. However, these models only take in text as input, implying they have no ability to reason the world with respect to perception information and restricted with spatial understanding. This limits their applicability to embodied tasks that require interaction with real or simulated environments. To address this limitation, Vision Language Models (VLM) extend language understanding by jointly learning from visual and textual modalities to enable a unified model for visual language tasks. There are two main approaches for VLMs for modern pipelines. The first is cross attention based VLM. This pipeline freezes the pretrained LLM model while the perception information is fused to intermediate embeddings using a cross attention mechanism [2]. This method benefits the powerful linguistic reasoning ability on pretrained LLMs and computational efficient while training. The other pipeline for VLM is auto-regressive based method which will tokenize the perception and fed alongside original LLM tokens [3],[4],[5]. It's like treating vision input as another language. This novel pipeline enables the high extensibility and generality by this multimodal training. However, building general-purpose VLM is still an issue because of the rich input distribution and diversity on tasks. Instruction tuning addresses this challenge by transforming heterogeneous multimodal data into a unified instruction–response format. This provides the VLMs better performs conversions and follow human's task and instructions [6]. One noticeable work called VILA [7], this model contains three main components: vision encoder, LLM and a projector. The vision encoder extracts visual features from input images and represents them as visual tokens, which are downsampled and projected into the language embedding space using an MLP. These projected visual tokens are concatenated with text tokens and processed by the LLM for auto-regressive generation. For video inputs, VILA uniformly samples frames at fixed intervals and prepends all frame tokens before the textual prompt, enabling the model to reason over temporal visual context. In terms of training process. VILA improves pre-training strategies for VLMs by pointing out freezing LLM perform weak in-context learning capability while updating LLMs provide

deep embedding alignment. Combining with other training strategies, VILA reaches the state-of-the-art (SOTA) performance. Building on VILA, NVILA [8] modifies the model pipeline by scaling up the spatial and temporal resolutions then compress the visual tokens. This enables efficiency on inference on high resolution videos and provides basic scene understanding and perceptual foundation for robot deployment for further task execution.

2.2 Vision Language Navigation

The long-term research on robot deployment is on building intelligent agents that could interact with humans in natural languages, capturing the perception and task executions [9]. Vision Language Navigations (VLN) [10] provides the foundation pipeline for this goal by extending the vision language modelling to robot/autonomy systems with spatial and temporal decision making and reasoning. The VLN pipeline usually contains three components: oracle (mostly human), agent (autonomy systems), and environment [11]. The agent and the oracle communicate through natural languages. The agent also explores and interacts with the environment such that it completes the corresponding tasks based on the instruction that are received from oracle and the perception information. Early VLN formulations typically model navigation as a sequence prediction or policy learning problem where the agent maps visual observations and instructions to navigation actions.

2.2.1 Tasks and Benchmarks

A classic VLN agent would receive a sequence of language instructions as language tasks at a given starting position. The agent will navigate the environment from the egocentric view. The task is to reasoning and generating a sequence of discrete trajectories of lower level control actions (like going forward 0.5 meter or turn left/right 50 degrees) to control and path following to reach the destination or target position. There are three main metrics for evaluating performance [10]. (1) Navigation Error (NE): the mean shortest path between goal location and agent's final position. (2) Success Rate (SR): the percentage of success for a sequence of tasks that the final position is close enough to goal location. (3) Success Rate Weighted Path Length (SPL): The SR is normalized by efficiency of agent's path to the correct goal location. VLNs are always tested and trained in simulators like Habitat, Matterport3D. Even though these simulation environments provide the scal-

able and generalizable platform for VLN development, these simulators still encounter significant shortcomings. For example, widely used simulators like Matterport3D [10] were built on discrete graphs that lead the agents becoming 'ghost like' entities without physical features. Subsequent simulators like Habitat [12] simplified autonomy systems' physics properties and kinematics. This led to potential bottleneck for VLN. To evaluate VLN models within a standard manner in a scalable way, the development and release of datasets that contain different kinds of tasks have been introduced. For example, Room-to-Room (R2R) [10] and Room-across-Room (RxR) [13] dataset contains fine-grained guidance instructions to agents to achieve their goal position for indoor scenarios. VLN runs in discrete environments and travel between the adjacent nodes by teleportation for these datasets on simulator like Matterport3D, referred as VLN-DE. To make this more realistic by transferring the discrete R2R and RxR environments to continuous, VLN-CE [14] was proposed accordingly and further benchmarks like Robo-VLN [15] improve with more realistic settings for robots to narrow the gap between simulation and real-world deployment.

2.2.2 Related Work and Challenges

Among these VLN research, a noticeable work named NaViLA [16] adopts a hierarchical two-level framework instead of directly predicting low-level control actions. Specifically, NaViLA first produces mid-level command that are spatially grounded action descriptions in natural language (e.g., "move forward 75 cm). These mid-level commands subsequently executed by a reinforcement learning based visual locomotion policy to get precise joint movement. These two level frameworks not only enables efficiency testing on simplified kinematics simulators for the reasoning and navigation section but also enables simulating VLN module with RL locomotion module that based on real-world robotic physics-based simulation on Issac Sim simulator and solve the issue mentioned in section 2.2.1 about bottleneck in simulation. In terms of high level, NaViLA uses VILA model mentioned in section 2.1 for understanding sequential frames' relationships. To get more structural memory for navigation, NaViLA choose to use the last frame for immediate decision making while earlier frames act as visual memory for progress. Therefore, NaViLA separate current and historical observations in the prompt using natural-language cues, avoiding additional special tokens to preserve a purely language-based LLM interface. NaViLA fine-tunes the VILA model using a blended dataset that combines real-world human navigation videos, continuous simulated VLN data, auxiliary navigation tasks, and general VQA datasets and reaches SOTA performance on VLN-CE benchmarks.

NaViLA still exhibits several limitations that motivate further investigation despite its strong performance. Firstly, the major limitation is that it doesn't provide an explicit mechanism for self-correction when the agent deviates from the intended trajectory. This limits its robustness to accumulated errors and unexpected environmental changes. Secondly, another major limitation is that the memory mechanism relies on a relatively simple long-short frame-based representation which lacks explicit structure and may be insufficient for long-horizon navigation or reasoning over complex spatial relationships. These limitations highlight the need for more structured memory representations and adaptive correction strategies in future VLN systems.

Challenge A: Error Correction

Error is often inevitable in autonomy task execution. The ability to correct from deviation from goals is essential for enhancing robustness. Error recovery has been actively studied in manipulation tasks [17], [18], [19] while navigation-oriented error correction remains relatively underexplored. Most of these error detection and recovery in manipulation field are based on LLM reasoning. The work AHA [18] detects errors by predicting task success and generating failure explanations from temporally ordered visual observations and task context. It learns this capability from procedurally generated failure data with diverse error modes. Although AHA does not perform corrections directly, its explanations guide downstream learning and planning modules to improve robustness. Another work detects errors by evaluating symbolic success conditions on simulator state that generated by LLM and marking rollouts that violate these conditions as failures [17]. Errors are addressed through repeated retries during data generation and only successful trajectories are retained for policy distillation, enabling robustness without explicit failure reasoning or online correction at inference time. Voyager is another pipeline handling error by using LLM agent [19]. It detects errors through execution feedback when a generated action program fails to produce the intended environment changes. Upon failure, the LLM performs self-reflection to revise the program and retries execution, while successful behaviors are stored in a skill library to prevent repeated mistakes and improve robustness over time. While manipulation tasks are relatively easily predicted by using LLM-generated state or environment as manipulation scenes often referred as well-defined states while navigation involves larger-scale, partially observable environments and more complex spatial reasoning. Although recent navigation methods explore trajectory reflection or explicit reasoning for error correction, they often introduce additional models or expensive inference procedures, limiting efficiency and practical deployment

in real-world settings. For example, the SmartWay work [20] proposes an online navigation error correction framework that detects failures through inconsistencies between language instructions, navigation history and current visual observations. SmartWay resolves them by explicitly allowing the agent to backtrack and replan within a continuous environment. This approach enables real-time recovery without relying on symbolic success conditions or offline retries. However, SmartWay uses large models and repeated reasoning steps during inference which introduces significant computational overhead. In addition, its error correction relies on unstructured trajectory level reflection rather than an explicit spatial memory or scene representation. This would limit the scalability and interpretability in long-horizon and complex navigation scenarios. EvolveNav [21] is another research work that explores error handling by embedding correction directly into the navigation policy through implicit reasoning over instruction and trajectory alignment. The model detects errors by cosine similarity between the intended instruction semantics, accumulated navigation history and current visual observations. The model then adapts future waypoint decisions through online replanning. This design avoids additional models and maintains inference efficiency. However, similar to SmartWay, the error handling mechanism remains unstructured, relying on latent reasoning rather than an explicit spatial memory or scene representation. Another pipeline for error detection and planning is CorrectNav [22]. This method equips the VLN model with implicit error detection and replanning capabilities by augmenting the final training stage with self-generated deviation and recovery data. This integrates correction behavior directly into the navigation policy rather than introducing a separate self-correction or replanning module, but this learning depends too heavily on black box learning and still lacks explicit structured reasoning or memory.

Challenge B: Long-Horizon Memory

Beyond error correction, another fundamental challenge in VLN lies in the absence of structured and persistent memory representations which also underpins many of the navigation failures discussed above. This limitation becomes particularly pronounced in long-horizon navigation, where agents must execute complex instructions composed of multiple sequential sub-tasks. These settings require sustained reasoning, explicit tracking of task progress and continuous dynamic replanning over long temporal horizons. The work LH-VLN [23] provides benchmark, long horizon generation tool, simulation platform and long horizon metrics on the multi-stage, context-rich navigation instructions. LH-VLN reveals that SOTA VLN models struggle to scale to long-horizon navigation

tasks even though they achieve strong performance on standard short-horizon benchmarks such as R2R and other continuous environments. It also reveals that most failures stem from an inability to maintain coherent progress across multiple sub-tasks rather than from isolated perception or action mistakes. Moreover, implicit and unstructured memory representations are found to be insufficient for long-horizon navigation and leads models to forget completed goals, revisit previously explored regions or lose alignment with the instruction sequence. All these results combined with failure from previous discussions about error correction show that structural representations for memories are crucial. Mem4Nav [24] tackles the structural memory bottleneck by replacing embedded flat memory information like appending past text/frames to a prompt with an explicit hierarchical spatial memory system. It builds a sparse octree for fine-grained voxel-level indexing and a semantic topological graph for high-level landmark and connectivity. On top of this map, Mem4Nav introduces a long–short memory design. Short-term memory caches recent observations in relative coordinates for fast local decisions while long-term memory stores spatially anchored at octree leaves and graph nodes using a reversible Transformer that can encode new embeddings and decode past embeddings when deeper history is required. However, memories are stored primarily as geometric or latent embeddings, the agent cannot directly reason about relationships such as inside, next to, across from. These relationships are central to language grounding and progress tracking. As a result, the model must infer these relations implicitly through attention over embeddings. This makes reasoning brittle and prone to forgetting or misalignment over long instruction sequences.

2.3 Scene Graph

2.3.1 Overview

These limitations suggest that long-horizon VLN requires memory representations that explicitly encode semantic entities and their relational structure. This motivates the adoption of structured graph-based representations to support robust reasoning and progress tracking. Occupancy mapping becomes the most prevalent model in spatial navigation for obstacle avoidance [25]. More recently, 3D scene graphs emerged as a powerful high-level representation of 3D environments. A 3D scene graph describes the environment as a layered graph where nodes represent spatial concepts at multiple levels of abstraction and edges represent relations between concepts [26]. As the development of deep learning, semantic information is integrated to such structural maps [27]. How-

ever, these semantics are only confined to predefined names and labels that are used in training stage, and this brings the limitation in effective utilization and generalization. This is called close words that semantics and descriptions are closed under certain words set. Recent advances in VLMs [28] [29] enable images and language to be embedded into a shared semantic space. This allows spatial representations to move beyond fixed labels toward open-vocabulary, concept-level understanding that better supports relational reasoning and generalization in navigation tasks. Existing open vocabulary mapping approaches often rely on directly projecting or distilling VLM features into 3D space. This provides coarse semantic recognition but lacks object centric understanding and common-sense reasoning that are required for complex and long horizon tasks. In addition, current map architectures struggle to efficiently represent and retrieve distinct objects in cluttered and repetitive environments, leading to high computational cost and ambiguous object identification that limits their scalability and practical use.

2.3.2 Related work

Hydra [26] is a baseline real-time 3D scene graph pipeline. It incrementally constructs a hierarchical 3D scene graph from odometric estimates by organizing perception into multiple spatial abstraction layers. The metric-semantic layer maintains a local volumetric Truncated Signed Distance Field (TSDF) and Euclidean Signed Distance Field (ESDF) using Voxblox [30] and integrates semantically labeled point clouds and extracting a metric-semantic mesh via marching cubes with per-voxel distance and semantic inference. The object layer forms persistent object instances through class-wise Euclidean clustering of mesh vertices such that it could estimate centroids and bounding boxes and merging overlapping instances of the same semantic class during incremental updates. The place layer models navigation free space as a sparse topological graph derived from a Generalized Voronoi Diagram (GVD) computed during ESDF integration. The nodes represent maximum clearance locations and edges encodes as straight line traversability. The room layer segments places by applying obstacle dilation to prune low-clearance nodes and identifies connected components as room candidates. At last, the building layer abstracts the entire environment as a single node connected to all rooms. Hydra provides a baseline for real-time, hierarchical 3D scene graph that explicitly organizes objects rooms, and their spatial relations for interpretable robotic reasoning. However, it is limited to predefined semantic labels from foundation segmentation models and does not support open-vocabulary or language-based caption descriptions, restricting its descriptive richness and flexibility for language-grounded tasks. OpenGraph [31] is a

framework that represents large-scale environments using a hierarchical 3D graph with open-vocabulary semantics. Instead of directly treating raw VLM features as semantic understanding, OpenGraph uses VLMs as a perceptual and cognitive front-end that visual observations are first processed to extract object instances and generate natural-language descriptions, which are then encoded by large language models to provide richer, more flexible semantic reasoning beyond simple category labels. These semantic features are subsequently projected from 2D image space into 3D LiDAR point clouds to construct object-centric representations. OpenGraph organizes these representations into a hierarchical graph structure in order to support efficient map maintenance and object retrieval. This includes (1) Point Cloud Layer: intuitive representations of the environment with metric point clouds. (2): Lane Graph Layer: contains topology lane graph. (3) Instance Layer: represents a subgraph of each instances including bounding boxes, captions, bounding boxes. (4) Segment Layer: partitioned each cluster of objects based on connectivity of lane graph like bush, road (5) Environment Layer: the entire environment node connects all the segments. However, OpenGraph relies mainly on segment-level clustering for high-level scene understanding. It lacks an explicit regional hierarchy such as room level or area level representations which limits its ability to model structured spaces in complex indoor environments with multiple subregions. In addition, although each instance is associated with a caption, these descriptions are typically shallow and object centric. It only provides limited support for deeper spatial or relational reasoning across objects and regions. Clio [32] is another 3D Scene Graph pipeline that proposes to use information bottleneck to cluster objects and regions based on tasks information. Rather than relying on fixed semantic categories, Clio formulates scene abstraction as an information bottleneck process that incrementally merges task-agnostic geometric primitives into task-relevant objects and regions. This merging favors spatially adjacent primitives with similar task relevance. It produces a compact representation that preserves essential information while discarding irrelevant details. Task relevance is estimated using open-set VLM embeddings so that it enables flexible semantic understanding without predefined labels. Task-driven clustering allows the model to more effectively extract and utilize relevant information from memory during reasoning by organizing the environment around task-relevant concepts rather than raw scene graph representations.

2.4 Video Captioner

Another explicit memory representation is video captions. Unlike latent visual features that remain difficult to interpret directly, video captions convert a sequence of observations into natural language summaries that describe salient objects, scene changes, actions, and temporal progress. Early work in video captioning and dense video captioning established the value of representing video content as text, especially for long videos containing multiple events over time. More recent video large multimodal models further improve this ability by producing richer temporally aware descriptions and detailed semantic summaries from video input. In the context of Vision Language Navigation, such captions can serve as an explicit semantic memory of what the agent has recently observed and how it has been moving through the environment. For example, captions can record high-level progress such as entering a hallway, passing a painting, or approaching a doorway, which complements the structured spatial relations stored in a scene graph. Therefore, video captioning is a natural auxiliary memory module for this thesis: it provides interpretable, language-based summaries of recent observations that can support task progress tracking, landmark grounding, deviation detection, and replanning.

Video captioning aims to translate dynamic visual content into natural language. Compared with image captioning, the video setting is more challenging because useful descriptions capture not just object appearance but temporal evolution, event order and interactions across frames. Early work in this area mainly focused on CLIP level description, supported by benchmarks such as MSR-VTT and VATEX. They provide natural language annotations for diverse short videos. As the field developed, researchers recognized that many real world videos contain multiple meaningful events rather than a single action. This motivated the task of dense video captioning that a system must both localize events in time and generate textual descriptions for each event. A representative milestone is the work of Krishna *et al.*, which introduced the dense-captioning task together with the ActivityNet Captions benchmark, enabling models to jointly reason about temporal boundaries and language generation [33, 34, 35].

Recent progress has moved from task-specific encoder-decoder pipelines toward large vision-language models that can process videos and generate richer natural-language outputs. Vid2Seq is a notable example in dense video captioning: it augments a language model with special time tokens so that temporal boundaries and captions are generated within one unified sequence, and further improves performance through large-scale

pretraining on narrated videos [36]. In parallel, video-capable large multimodal models have emerged for more general video understanding. Video-ChatGPT adapts a visual encoder to video and trains on a large set of video-instruction pairs, enabling detailed conversational understanding of video content [37]. Video-LLaVA further shows that learning a unified representation for images and videos can substantially improve video reasoning performance, while LLaVA-Video demonstrates that high-quality synthetic instruction data with detailed captioning can effectively train strong video understanding models [38, 39]. LLaVA-OneVision also suggests that strong video understanding can emerge through transfer from image-centric training, which is attractive for systems that must share components across image and video tasks [40].

Although these models are powerful, generic video captioning alone is still insufficient for embodied navigation. In navigation scenarios, the most important visual clues are often fine-grained landmarks such as a framed picture, a small sign or the relative appearance of a doorway on one side of a corridor but not the global scene semantics. These details are easy to miss when captions are generated only at the full frame level. This has motivated more fine grained forms of captioning. The recent Describe Anything Model (DAM) addresses detailed localized image and video captioning by producing descriptions for user-specified regions in images or videos while preserving both local detail and global context [41]. This localized captioning setting is especially relevant to VLN, where navigation instructions frequently refer to specific objects or spatially localized cues rather than the entire scene.

For this thesis, the role of the video captioner is not to replace the scene graph, but to complement it. A scene graph provides explicit structure, such as object instances, regions, and spatial relations, while captions provide fine-grained semantic attributes that are difficult to represent using category labels alone. For example, a graph node may identify an object as a *picture* or *cabinet*, but a captioner can enrich that node with details such as "a framed eye chart on the left wall" or "a narrow white cabinet beside the hallway entrance." Such descriptions are useful when the original VLN policy fails to ground small or visually subtle landmarks. In addition, video captions can summarize short temporal context, which may help the reasoning agent assess progress, detect whether an expected landmark has already been passed, and generate more explicit recovery instructions.

3 Problem Analysis

Building upon the limitations identified in the literature review, this section empirically examines the behavior of current SOTA VLN models under both short-horizon and long-horizon instruction settings. Although recent VLN systems demonstrate notable improvements over early baselines, navigation performance remains far from reliable even for short, single-stage instructions. On standard benchmarks such as R2R, SOTA models typically achieve low success rates shown in Table 1, indicating that there's still a huge amount of navigation episodes still fail despite limited instruction length and relatively constrained environments. To better understand the sources of these failures, NaViLA is evaluated as a representative strong baseline on unseen scenes, and its errors are analyzed qualitatively across multiple episodes.

Table 1: Comparison of SOTA methods on Val-Unseen of R2R-CE

Models	**Observations**	**NE↓**	**OS↑**	**SR↑**	**SPL↑**
Reborn [42]	Pano+Depth+Pose	5.40	57.0	50.0	46.0
DreamWalker [43]	Pano+Depth+Pose	5.53	59.0	49.0	44.0
CMA [14]	RGB+Depth	7.37	40.0	32.0	30.0
NaVid [44]	RGB	5.47	49.0	37.0	35.0
NaVILA	RGB	5.22	62.5	54.0	49.0

The results of NaViLA on unseen scenes on R2R-CE dataset is shown in Table 2. There're 143 scenes that have been studied by visualizing the failure case and determining the reason for failure by hand. This chart includes top 6 reasons for failure in these unseen scenarios and reveals the major issues of NaVILA.

Table 2: Reasons for Failure cases on Val-Unseen of R2R-CE on NaVILA.

Reasons of failure	**Numbers of failure**	**Percentage wrt total failure**
Fail to understand object description	45	31.4%
Not realize Region / multiple regions with same name	63	44.0%
Didn't see the described object (e.g. behind the initial position)	17	11.9%
Unclear description	32	29.4%
False Negative	6	4.2%

Failure to understand object description This issue accounts for approximately one third of all failure cases. In these scenarios, the instructions explicitly require the agent to execute an action such as turning or stopping upon observing a specified object. However, the model operates as an end-to-end learning system and there is no explicit mechanism to verify whether the referenced object has been successfully recognized during inference stage. To further diagnose this issue, additional analysis was conducted. It was observed that the agent generally follows instructions correctly when the referenced objects are large or common, such as desks or chairs. In contrast, instructions involving smaller or more descriptive objects like 'paintings on walls' or 'eye charts' (showing in Figure 1, the robot didn't realize the existence of eye chart and go pass instead of following command to turn) frequently cause the policy to fail. This failure pattern highlights a clear limitation in object grounding when semantic recognition relies solely on implicit visual features. Further experiments are being done with foundation models such as the Describe Anything model to support this observation and the result shows that these objects can be correctly identified when the corresponding frames are provided as input. This limitation presents a natural opportunity for improvement using a scene graph memory with fine-grained captions generated by foundation video captioning models.

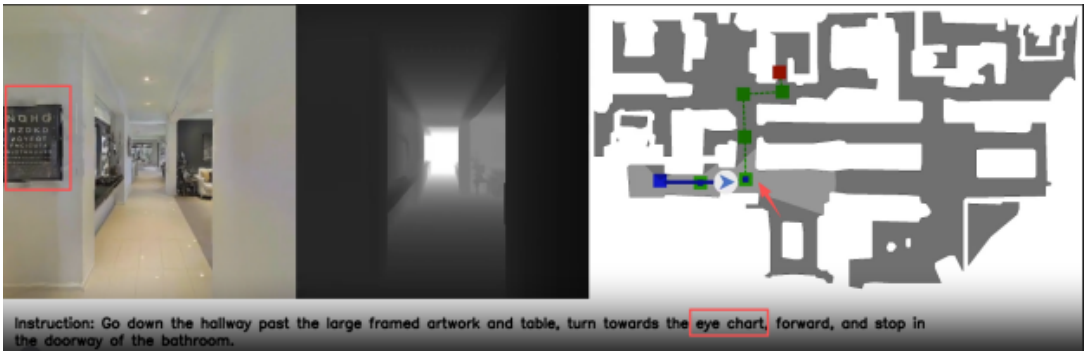


Figure 1: Failure cases for not realizing the target object

Not realize Region/multiple regions with same name This is the most frequent issue and this issue commonly occurs in open areas such as living rooms that contain multiple exits or adjacent regions with similar semantic labels. For example, the agent is been told to go through the hallway without mentioning which specific hallway to follow like showing in Figure 2. Disambiguating such regions is difficult even for humans without precise descriptions and current VLN models lack the structured memory needed to reason about these ambiguities. From a human perspective, navigation errors are typically handled by reasoning about whether a mistake has been made and deciding

whether to continue forward or return to the last confident location. This form of higher-level reflection motivates the introduction of an additional reasoning agent that operates over a structured scene-graph representation enabling systematic analysis and informed replanning. This is also well-discussed in Section 2.2.2, error detection and replanning are critical in these scenarios and constitute a key focus of this thesis.

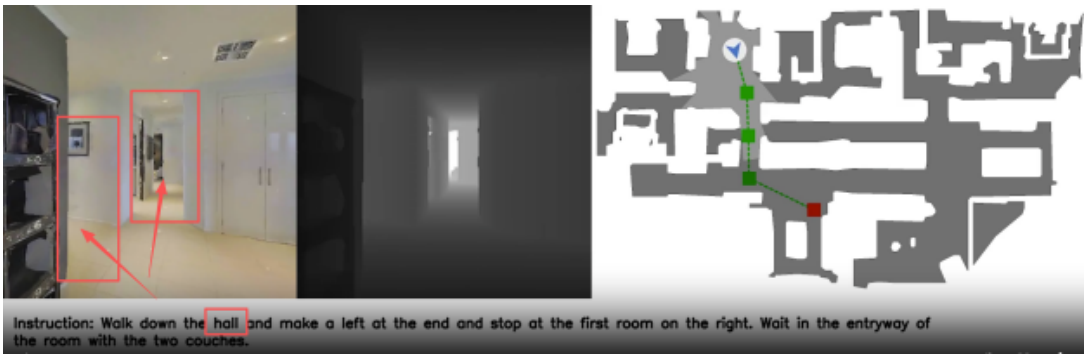


Figure 2: Failure case for two hallways and doesn't follow the correct one

Didn't see the described object the whole path This failure occurs when the agent is told to see a specific object and make actions while the object is not observed. For example, the target object lies outside the initial field of view or behind the starting position. This situation is similar to previous section on region disambiguation failures. Effective error detection and replanning are required for the agent to recognize the missing landmark and actively search for it instead of proceeding with an incorrect assumption.

Unclear instructions and false negative These failure cases are largely attributable to ambiguities in the dataset rather than model behavior. Some instructions describe objects or paths in ways that are difficult to interpret or underspecified. In false negative cases, the agent may reach a semantically correct location but be penalized for stopping at an unintended side of the object when the instruction does not precisely specify the target position.

Furthermore, to evaluate VLN performance under long-horizon settings, NaViLA was adapted from the Habitat simulator to the LH-VLN simulation environment. The results exhibit a consistent trend across state-of-the-art VLN models, with all methods achieving very low success rates, as shown in Table 3. These results suggest that performance under the LH-VLN setting is highly unstable and influenced by stochastic factors rather than systematic algorithmic improvements. This limits its effectiveness as

a reliable benchmark. In contrast, most existing VLN methods are evaluated on well-established benchmarks such as R2R-CE and RxR-CE within the Habitat framework. Considering both benchmarking consistency and implementation feasibility, this thesis therefore focuses on evaluation using R2R-CE and other VLN-CE datasets.

Table 3: Comparison of SOTA methods on LH-VLN.

Models	NE↓	ISR↑ (Independent SR)	CSR↑ (Conditional SR)	SR↑	CGT↑ (GT weighted CSR)
Random	10.91	0	0	0	0
GLM-4v prompt	10.97	0	0	0	0
NaviLLM	9.79	3.54	2.53	0	5.24
MGDM	1.23	4.69	3.30	0	5.83
NaVILA	7.91	4.03	2.78	0	5.14

3.1 Current Gap

Despite recent progress in VLN, the central limitation is no longer simply whether a model can map observations and instructions to the next action, but whether it can preserve and use past information in a form that supports reliable reasoning during execution. In most current VLN systems, historical observations are propagated through hidden states, attention features, temporal tokens, or transformer embeddings rather than being stored as an explicit and queryable world representation. This means that the model may still retain information from earlier observations, but that information remains compressed inside latent feature vectors rather than externalized into a readable structure such as a scene graph, landmark memory, region map, or semantic trajectory summary. NaViLA, for example, improves continuous VLN by separating current and historical observations in a language-based prompting interface, but its history is still fundamentally represented as prior frames and latent visual-language context rather than as explicit spatial memory [16]. Likewise, recent recovery-oriented methods such as SmartWay, EvolveNav, and CorrectNav improve navigation robustness by introducing stronger reasoning, backtracking, or correction behavior, yet they still rely primarily on implicit internal representations of the past rather than a persistent semantic memory that can be directly inspected or queried [20, 21, 22]. As a result, these methods improve policy behavior without fully solving the representational problem of how navigation history should be stored.

This representational issue is especially important because many failure cases in VLN are not purely local action failures, but failures of memory organization and reference grounding over time. When an instruction depends on seeing a specific object, distinguishing between two visually similar regions, or recognizing that an expected landmark

has already been passed, the agent must do more than preserve raw visual information. It must relates what was seen, where it was seen, and how that observation to the current subgoals. Latent embeddings may contain some of this information implicitly but they do not provide a stable mechanism for explicit queries such as whether a particular object has appeared before or whether the robot is still inside the correct region or whether the current trajectory remains aligned with the intended instruction sequence. This is why long-horizon navigation remains difficult even for strong modern models. LH-VLN shows that state-of-the-art VLN systems degrade substantially when tasks require sustained progress tracking across multiple sub-instructions rather than a single short-horizon decision process [23]. Mem4Nav addresses part of this issue by introducing a hierarchical spatial memory system with long-short memory design, but even there the stored experience is mainly encoded as geometric structures and latent memory features rather than as directly interpretable semantic entities and relations that can support explicit reasoning in natural language [24]. Therefore, the problem is not only that current VLN models forget, it is that what they remember is often not represented in a form that is transparent, structured, or usable for downstream reasoning.

The gap addressed by this thesis is therefore the absence of explicit, interpretable memory in VLN systems. Existing methods have improved instruction following, trajectory correction, and long-horizon robustness, but most still entangle perception, memory, and decision making inside an end-to-end learned policy. This makes it difficult for the agent to diagnose why a failure occurred, to verify whether an expected landmark has been observed, or to generate recovery instructions grounded in a readable representation of the scene. In contrast, this thesis argues that memory should be externalized into dedicated modules that preserve past information in structured and human-interpretable form. Specifically, the proposed framework combines a hierarchical scene-graph memory with a video-caption-based memory. The scene graph captures persistent object, region, and relation structure, while video captions summarize recent observations and high-level movement progress in natural language. Together, these modules provide a complementary explicit memory layer that current VLN models generally lack. Rather than forcing the navigation policy to compress all past experience into opaque embeddings, the proposed system separates remembering from acting, so that the reasoning agent can retrieve landmarks, track progress, detect deviations, and replan using memory that is both machine-usable and human-readable. This is the main methodological gap between prior VLN systems and the framework proposed in this thesis.

4 Architecture and Chosen Foundation Models

4.1 High-Level Architecture

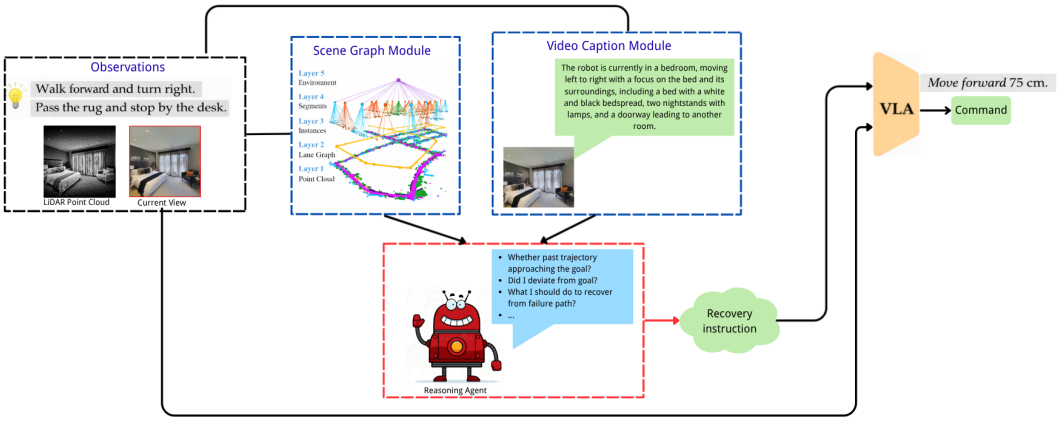


Figure 3: AgentVLN high level architecture

The proposed AgentVLN system is built as a modular architecture rather than a single end-to-end model. At a high level, the system contains four roles: a VLN backbone that produces navigation commands from instructions and observations, an online scene-graph module that incrementally builds explicit spatial memory, an online video understanding module that summarizes recent observations in language, and a higher-level reasoning component that reads these memory sources to assess progress and support self-correction as shown in Figure. The motivation behind this design is to separate action generation from memory construction and reasoning. Instead of forcing one model to internally preserve all past observations in latent features, the system explicitly stores useful information in structured and interpretable representations that can be queried during execution.

To instantiate this design, this thesis selects one core model for each major role in the system. NaVILA is used as the VLN backbone because it already provides a strong navigation policy with a language-based intermediate action interface. OpenGraph is selected as the base scene-graph model because it supports open-vocabulary, incremental

3D graph construction from visual and geometric observations. NVILA is selected as the video-captioning backbone because it is an efficient video-capable visual language model that can process higher-resolution visual input and longer temporal context than earlier VLM designs. Finaly, GPT-5-nano is choosed to be the model for reasoning agent. Together, these four components define the architectural foundation of the proposed system, while the next section explains how they are later integrated into the final methodology.

4.2 Chosen Foundation Models

4.2.1 VLN Backbone: NaVILA

NaVILA is selected as the VLN backbone because it provides a strong and practical solution to the central interface problem in robot navigation: how to convert language instructions and egocentric observations into actions that can actually be executed on a physical robot. Rather than predicting raw low-level control signals directly from a large multimodal model, NaVILA uses a two-level architecture. The first level is a vision-language-action model that interprets the navigation instruction together with the visual input and predicts a *mid-level* action expressed in language with explicit spatial content, such as moving forward by a given distance or turning by a certain amount. The second level is a learned locomotion policy that converts this mid-level command into executable robot motion. This decomposition is the key design choice of NaVILA. It avoids asking a large language-conditioned model to solve both semantic reasoning and low-level motor control at the same time, which would be difficult to train and hard to transfer to real robots [16].

At the perception side, NaVILA builds on a video-capable visual language model and uses both current and historical observations to support navigation decisions. Instead of treating navigation as a purely single-frame problem, it keeps a short visual history so that the model can reason about progress over time. However, this temporal context is still handled inside the VLN model itself, mainly as prior frames and multimodal context tokens. This design works well for strong short-horizon navigation performance, but it does not produce an external memory structure that can be directly read by another module. For this thesis, NaVILA is therefore a strong starting backbone rather than a complete solution. It provides the action-generation module of the architecture, while the explicit memory and reasoning layers proposed in this thesis are added on top of it. This choice is important architecturally: the goal is not to replace a competitive VLN model,

but to augment a strong baseline with explicit memory and self-correction capabilities [16].

Another reason for choosing NaVILA is that its intermediate command interface is especially suitable for integration with external reasoning. Because the model outputs interpretable mid-level actions rather than only hidden control decisions, the reasoning module in this thesis can intervene at the level of instruction reformulation and command correction. In other words, if the reasoning agent detects that the robot has deviated from the intended route, it can republish a revised instruction or a corrected subgoal in a form that remains compatible with the NaVILA backbone. This makes NaVILA not only a strong VLN model, but also an architecturally convenient one for a modular explicit-memory system.

4.2.2 Scene Graph Backbone: OpenGraph

OpenGraph is chosen as the base scene-graph model because it offers an explicit and open-vocabulary representation of the environment, which is precisely the type of memory missing from standard VLN systems. The main idea of OpenGraph is to move beyond dense geometric maps and instead build a hierarchical graph whose nodes and edges capture semantically meaningful entities in the world. Its pipeline begins with visual perception: object instances are extracted from images, and textual captions are associated with them to enrich their semantics. These instance-level semantics are then projected into 3D using LiDAR, allowing the system to perform incremental object-centric mapping in metric space. Finally, the mapped environment is organized into a hierarchical graph through lane-graph-based segmentation, which gives the representation structure at multiple levels rather than leaving it as a flat object list [31].

This architecture is especially attractive for the present thesis because it combines three properties that are difficult to obtain at once: incrementality, 3D grounding, and open-vocabulary semantics. First, OpenGraph is incremental, which means the map can be built online as the robot moves instead of only after the entire trajectory is available. Second, its representation is grounded in 3D geometry rather than remaining purely image level. This makes it appropriate for navigation and spatial reasoning. Third, it can support more flexible language grounded queries than traditional fixed label semantic maps that associates instances with captions and open vocabulary embeddings. These properties make OpenGraph a natural candidate for the explicit memory branch of AgentVLN. The reasoning agent in this thesis needs access to objects, their approximate locations,

and their relations to higher-level environment structure, and OpenGraph provides a foundation for all three [31].

At the same time, OpenGraph is not used in this thesis without qualification. Its original design is aimed at large-scale outdoor environments, and its hierarchy is built around point clouds, lane graphs, instances, segments, and the overall environment. That structure is powerful, but it is not yet perfectly aligned with the needs of indoor VLN, where room-level organization, doorway transitions, and repeated indoor regions are especially important. In addition, although OpenGraph associates instances with captions, the original captions are not sufficiently detailed for many of the subtle landmark-grounding problems observed in VLN failure cases. For this reason, OpenGraph is chosen here as the *scene-graph backbone*, not as a finished complete memory solution. The proposed system keeps its strengths—incremental 3D object-centric open-vocabulary mapping—while later extending it with stronger semantic enrichment and task-conditioned retrieval.

4.2.3 Video Captioning Backbone: NVILA

NVILA is selected as the video-captioning backbone because the second explicit memory channel in this thesis is not purely spatial. The scene graph captures structured object and region information, but navigation also benefits from a higher-level semantic summary of what the robot has been doing recently. For this purpose, the system needs a visual language model that can process longer temporal context efficiently and still retain enough detail to describe subtle scene changes and landmarks. NVILA is well suited to this role. It is built on top of VILA and introduces a central architectural idea called *scale-then-compress*. Instead of aggressively compressing visual input at the start, NVILA first increases the spatial and temporal resolution so that more detail is preserved, and then compresses the visual tokens afterward to maintain efficiency. This allows the model to handle high-resolution images and longer videos much more effectively than earlier designs that sacrificed too much detail too early in the pipeline [8].

This design matters directly for the captioning role envisioned in AgentVLN. Many navigation-relevant cues are small, local, and easy to lose under coarse visual tokenization: a picture on a wall, a narrow doorway, a sign, or a specific object placed near a corridor entrance. At the same time, the model must also summarize temporal context over multiple frames rather than treating each image independently. NVILA is attractive because it improves both sides of this tradeoff. By scaling first, it preserves more of the fine spatial details needed for grounded descriptions. By compressing later, it avoids the

computational cost that would otherwise make online video understanding impractical. The result is a model family aimed not only at strong accuracy, but also at end-to-end efficiency across training, fine-tuning, and deployment [8].

In this thesis, NVILA is not used as a generic chatbot-style visual assistant, but as the backbone for a caption-based explicit memory stream. Its role is to periodically convert recent observations into compact natural-language summaries that can complement the structured scene graph. The scene graph tells the system what objects and spatial relations have been mapped, while the NVILA-based captioner provides a more semantic narrative of recent motion and scene evolution. This is useful because some navigation evidence is easier to express in natural language than in graph edges alone, for example whether the robot appears to be leaving a room, moving down a hallway, or passing a landmark on one side. Therefore, NVILA is chosen not because it is the only model capable of video description, but because its efficient high-resolution and long-context design makes it especially suitable for online caption generation inside a larger embodied navigation pipeline.

4.2.4 Reasoning Agent: GPT-5 nano

The final component in the architecture is the reasoning agent, which is responsible for reading the explicit memory modules and making higher-level judgments about task progress. In this thesis, the reasoning agent is implemented using GPT-5 nano. This model is selected not because it replaces the VLN backbone, but because it provides a lightweight language-based reasoning module that can operate periodically on top of the navigation system. According to OpenAI's official model documentation, GPT-5 nano is designed for settings where speed and cost are especially important, including classification, extraction, ranking, and sub-agent style workloads. It also supports text and image inputs and provides a large context window, making it suitable for reasoning over structured memory, recent captions, and summarized navigation state without introducing the cost of a larger frontier model [45].

Within AgentVLN, GPT-5 nano is not used to directly predict robot motion. Instead, it serves as a decision-making layer above the VLN backbone. At each reasoning step, it receives the current task instruction, the active subgoal, selected scene-graph information, recent caption based summaries, and the current navigation status. Based on this information, it determines whether the robot is still on track and whether a landmark has been missed, and whether a corrective instruction should be issued. This role is well

matched to a smaller but fast reasoning model: the agent must repeatedly make focused, structured decisions during navigation rather than solve the entire perception-control problem end to end. OpenAI's documentation specifically positions GPT-5 nano as a good fit for fast, high-volume sub-agent workloads, which aligns closely with the intended use here [45].

Another advantage of using GPT-5 nano is modularity. Because it functions as a separate reasoning component, it can be prompted with explicit structured memory rather than relying on implicit hidden states inside the VLN model. This supports the central design philosophy of the thesis: navigation, memory construction, and reasoning should be separated into interpretable layers. The scene graph provides structured spatial evidence, the captioning branch provides semantic summaries of recent movement, and GPT-5 nano consumes these representations to produce discrete reasoning outputs such as on-track/off-track judgments and corrected navigation instructions. As a result, the reasoning process becomes easier to analyze and revise than if it were entirely absorbed into the VLN backbone itself.

5 Methodology

5.1 Overview

The proposed AgentVLN framework is designed as a modular closed-loop navigation system that augments a strong Vision–Language Navigation backbone with explicit memory and periodic high-level reasoning. The key idea is to separate *acting*, *remembering*, and *reasoning* into different but cooperative modules. Instead of forcing the VLN model to preserve all past information only through internal hidden features, the system continuously writes useful observations into external memory while the robot is moving. These memories are then read by a separate reasoning agent, which determines whether navigation is still consistent with the intended instruction and, if necessary, issues a corrective instruction. This design directly targets the representational gap identified in the previous section, namely that conventional VLN systems often retain past information only implicitly and therefore have limited ability to diagnose errors or justify recovery decisions.

At the action level, the VLN backbone serves as the main policy that converts the natural-language instruction and current egocentric observation into the next navigation command. In this thesis, a NaViLA-style formulation is adopted, where the backbone outputs mid-level navigation commands that are subsequently executed by the downstream control stack [16]. While this command-generation loop is running, two explicit memory branches are updated online in parallel. The first branch is an online scene-graph memory that incrementally builds a structured representation of the explored environment using the incoming visual and geometric observations. The second branch is an online video-caption memory that produces semantic summaries from the RGB stream at a lower temporal frequency. In the current design, the captioning module is invoked every 20 frames, so that the system maintains a compact description of recent landmarks, scene context, and high-level movement progress without requiring caption generation at every frame.

Above these two memory branches sits the reasoning agent. Rather than reasoning continuously at the full sensor rate, the agent operates periodically after enough new memory has accumulated. In the current implementation, it is invoked every 10 mapping steps. At each reasoning step, the agent receives the original instruction, the current subinstruction, the relevant scene-graph memory, and the recent caption summaries. It then determines whether the robot is still progressing toward the intended subgoal or has

deviated from the expected route. If the available evidence indicates that the trajectory is still aligned with the instruction, the VLN system continues normally. If the reasoning agent determines that the robot is off track, it enters a recovery stage and generates a corrected short-horizon instruction grounded in the current scene state. This corrected instruction is then republished to the VLN backbone, and the closed loop continues while both memory branches keep updating online.

Overall, the framework can be viewed as a layered navigation architecture with explicit memory. The VLN module generates actions, the online scene graph stores structured spatial memory, the captioning branch stores semantic summaries of recent observations, and the reasoning agent periodically reads both memory sources to evaluate progress and trigger self-correction. This organization allows the system to move beyond a purely end-to-end VLN formulation and instead support interpretable progress tracking, deviation detection, and recovery during execution.

5.2 Problem Setup and Notation

Let the navigation instruction be denoted by I, where I is a natural-language command describing the overall task to be completed by the robot. During execution, the robot receives a sequential observation stream

$$o_{1:T} = \{o_1, o_2, \ldots, o_T\},$$

where each observation o_t contains the sensor information available at step t. In the proposed system, o_t includes at least an RGB observation r_t for VLN and captioning, together with the geometric input used for mapping, such as depth or LiDAR measurements, and the pose estimate required for online map integration. For simplicity, the full multimodal observation at step t is written as

$$o_t = \{r_t, g_t, p_t\},$$

where r_t denotes the RGB image, g_t denotes the geometric observation, and p_t denotes the robot pose or pose estimate used by the scene-graph memory.

The VLN backbone takes the instruction I and the current observation context as input

and outputs a navigation command a_t at each decision step:

$$a_t = f_{\text{VLN}}(I, o_{\leq t}).$$

Here, a_t denotes the mid-level navigation command produced by the VLN module, such as a forward motion or turning action, which is then executed by the downstream control system. Since the full instruction may contain multiple stages, the system maintains a current active subinstruction s_k, where k indexes the current step in the instruction decomposition:

$$I \rightarrow \{s_1, s_2, \ldots, s_K\}.$$

The active subinstruction s_k is the portion of the task currently being monitored by the reasoning agent.

To store explicit memory during navigation, the system maintains two time-varying memory states. The first is the online scene graph,

$$G_t,$$

which stores the structured spatial memory accumulated up to step t. The second is the caption memory,

$$C_t,$$

which stores the caption-based summaries generated from recent visual observations up to step t. Both G_t and C_t are updated online as the robot moves. The scene graph is updated at each mapping step, while the caption memory is updated every $M = 20$ frames in the current design.

Reasoning is performed periodically rather than at every frame. Let the reasoning interval be denoted by R. In the current system,

$$R = 10$$

mapping steps. At each reasoning point, the agent consumes the current instruction context, the active subinstruction, and the explicit memories:

$$y_t = f_{\text{reason}}(I, s_k, G_t, C_t),$$

where y_t denotes the reasoning output. This output includes at least an on-track/off-track

judgment and, if required, a corrected instruction for recovery. Therefore, the overall system can be summarized as an iterative loop in which the VLN backbone produces actions, the memory modules update G_t and C_t, and the reasoning agent periodically evaluates whether the executed trajectory remains consistent with the intended instruction.

5.3 Online Scene Graph Memory

The online scene-graph memory is the main structured memory component in the proposed system. Its purpose is to convert the continuous stream of observations into a persistent and queryable representation of the explored environment. Rather than preserving history only as past frames or hidden features, the scene graph stores objects, regions, and their relationships in a form that can be directly accessed by the reasoning agent. The design is based on the chosen OpenGraph-style object-centric mapping paradigm [31] with hierarchical layers showing in Figure 4, but it is reorganized for indoor VLN, where room-level structure, repeated regions, and small navigation-relevant landmarks are especially important.

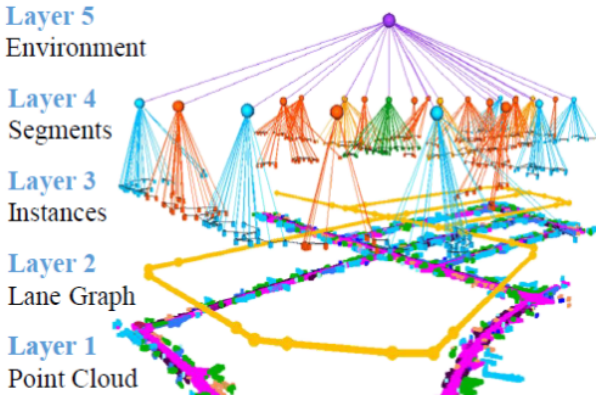


Figure 4: OpenGraph's hierarchical layers visualization

At each mapping step, the scene-graph module receives synchronized multimodal inputs from the current robot state. These include the RGB image r_t, the geometric input g_t used for 3D grounding, and the current pose estimate p_t. The RGB input is used for visual perception and semantic description, while the geometric input and pose are used to localize perceived entities in the world frame and update the graph consistently over time. Instead of storing raw frames directly, the system first extracts candidate object instances and their associated semantic features from the observation. These detections are then projected into the current 3D map and matched against previously stored nodes.

If an observed instance corresponds to an existing node, that node is updated. Otherwise, a new node is created. In this way, the scene graph is updated incrementally throughout the navigation episode.

The maintained graph in the proposed framework is organized into three practical levels that are directly suited to indoor VLN: *object nodes*, *region nodes*, and *relation edges*. Object nodes represent concrete landmarks or entities such as couches, paintings, cabinets, tables, or doorways. Each node stores its semantic label or open-vocabulary description, estimated 3D centroid, bounding volume, visually embeddings and an observation confidence or count reflecting how often the object has been detected across time. These nodes may also be augmented with richer natural-language descriptions from the captioning branch. To provide higher level indoor structure, our AgentVLN performs room segmentation every 10 mapping steps and uses the resulting segmentation to form region nodes. Each region node corresponds to a navigable indoor area, such as a bedroom, hallway, kitchen, entrance, or bathroom, and maintains the set of object nodes currently assigned to that room segment. This region level abstraction is important for indoor VLN because many instructions refer not only to individual landmarks, but also to room transitions, hallway choices, and spatial context at the area level. Relation edges then encode the connectivity and semantics required for downstream reasoning, including object-to-region membership, and simple spatial relations such as being inside a room. This multi-level graph therefore supports both fine-grained object grounding and coarse-grained spatial reasoning throughout navigation.

This graph design is more suitable for indoor VLN than a flat object list because navigation instructions are rarely grounded only at the object level. Many failures arise because the agent must distinguish between semantically similar regions, determine which room it is currently in or decide whether an observed object belongs to the correct part of the trajectory. Therefore, the graph is explicitly organized to support both fine grained and coarse grained queries. If the instruction refers to a small landmark, the reasoning agent can retrieve relevant object nodes directly. If the instruction refers to a room transition or an ambiguous corridor choice, the agent can first retrieve the relevant region and then search within it for supporting landmarks. This hierarchical use of regions and objects is one of the main reasons the memory is designed specifically for indoor VLN rather than copied directly from an outdoor mapping structure.

To improve semantic richness, the graph is also augmented with language-oriented information. Whenever available, object nodes are associated with open-vocabulary embeddings and short natural-language descriptions. These descriptions are not treated as a

separate standalone memory, but as semantic attributes attached to the corresponding graph entities. This allows the reasoning agent to retrieve not only geometric object identities but also descriptive cues that may be important for instruction grounding. For example whether a picture is a framed painting, whether a doorway is narrow, or whether a cabinet is located beside a hallway entrance. Over time, repeated observations can refine these node attributes, making the graph more stable and more descriptive as the episode progresses.

5.4 Online Video Caption Memory

The online video caption memory is the second explicit memory representation in the proposed framework. While the scene graph provides structured spatial memory through objects, regions, and their relations, it does not fully capture the high-level semantic context of how the robot has been moving through the environment over time. In particular, many navigation decisions depend not only on what objects are present, but also on whether the robot has recently passed a landmark, entered a new area, or is still progressing toward the intended subgoal. These aspects are often easier to represent as short natural-language summaries than as graph structure alone. Therefore, AgentVLN maintains a caption-based memory stream in parallel with the scene graph, so that the system preserves both structured spatial information and semantic trajectory history in explicit form.

The captioning branch operates online on the RGB observation stream at a lower temporal frequency than the mapping branch. In the current design, a new caption is generated every 20 frames. This interval is chosen to reduce redundancy between adjacent frames while still preserving meaningful temporal progress. Each caption summarizes the current scene and nearby landmarks with emphasis on navigation relevant cues such as doorways, hallways, furniture, or other distinctive objects. In addition to describing the visible scene, the captioning branch also captures recent movement context, for example whether the robot appears to be moving down a hallway, entering a room, or passing an object on one side. As a result, the caption memory provides a compact semantic record of both local surroundings and short-horizon trajectory progress.

5.5 Reasoning Agent

The reasoning agent is the decision-making module built on top of the two explicit memory branches. Its role is not to replace the VLN backbone, but to periodically interpret the current task state using structured scene-graph memory and caption-based semantic memory. At each reasoning step, the agent receives the original instruction, the current active subinstruction, the matched landmarks and their recent distance relationships, together with the summarized caption history describing recent movement and the current scene. Based on this information, it determines whether the robot is still making valid progress toward the intended route or has deviated from it. If the robot is still consistent with the current task stage, the system continues normal navigation. If the reasoning agent detects that the robot is no longer aligned with the expected subgoal or overall instruction sequence, it enters recovery mode and generates a corrected local instruction to guide the robot back on track.

This reasoning process is organized into several stages. First, the original instruction is decomposed into a sequence of ordered subinstructions. Second, the landmarks mentioned in each subinstruction are matched to the scene graph. Third, the system tracks how the robot's distance to these landmarks evolves over time and converts the raw signals into descriptive relations. Fourth, the caption memory is summarized to produce both a movement summary and an estimate of the robot's current semantic location. Finally, the reasoning agent combines all of this evidence to determine whether the robot is on track or off track and whether recovery is required.

5.5.1 Instruction Decomposition into Subinstructions

The reasoning process begins by decomposing the original instruction I into an ordered sequence of subinstructions

$$I \rightarrow \{s_1, s_2, \ldots, s_K\}.$$

Each subinstruction corresponds to a short-horizon semantic goal that can be monitored more reliably than the full instruction at once. This decomposition is important because the original instruction often contains multiple stages, room transitions, and landmark-dependent actions. If the reasoning agent attempts to judge the whole task at once, it becomes difficult to determine which part of the instruction is currently relevant and what evidence should be expected from the robot's recent behavior. By contrast, subinstruction-level reasoning makes the process more structured and more interpretable.

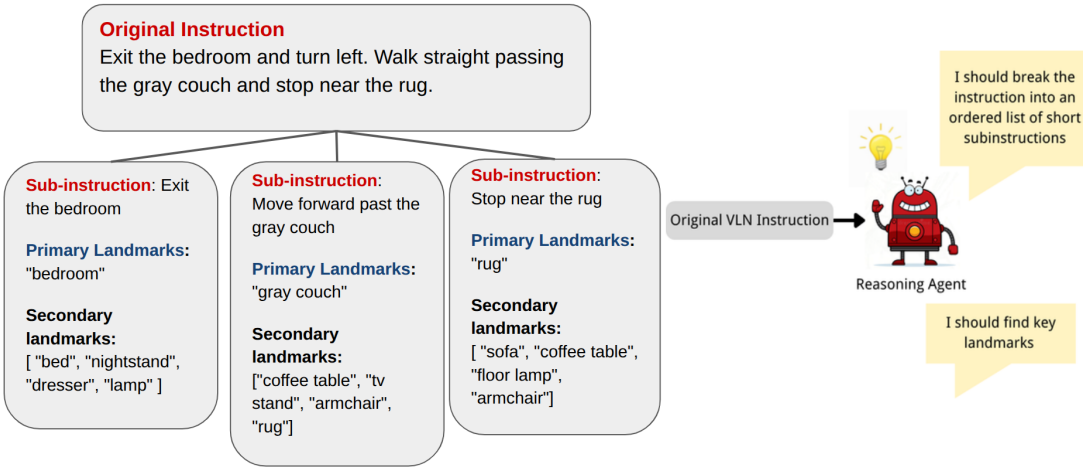


Figure 5: Example of Instruction Decomposition subsection

For each subinstruction, the system extracts two types of landmarks: *primary landmarks* and *secondary landmarks*. Primary landmarks correspond to the main entity or region that defines the goal of that step such as a bedroom, a gray couch, or a rug. Secondary landmarks provide supporting context that helps disambiguate the target and stabilize matching, such as beds, nightstands, lamps, coffee tables, or other nearby objects likely to occur with the primary landmark. This distinction is useful because instructions often refer to a target landmark directly. However, the surrounding scene context is also important for deciding whether the robot is in the correct place. The decomposition stage therefore produces a structured representation for each subinstruction consisting of the subinstruction text, its primary landmark set, and its secondary landmark set.

The resulting sequence of subinstructions is processed in order, and the reasoning agent maintains an index for the currently active subinstruction. This allows the system to judge progress locally while still respecting the overall instruction sequence. A subinstruction is considered active until the available evidence suggests that it has been completed or replaced by a recovery instruction. In this way, the decomposition stage provides the scaffold for all subsequent reasoning. The visualization of this subsection is shown in Figure 5.

5.5.2 Landmark Matching with Scene-Graph Objects

After decomposition, the landmarks associated with the active subinstruction are matched against entities in the online scene graph. This step grounds the language instruction into explicit spatial memory so that later reasoning can operate on concrete mapped en-

tities rather than only symbolic landmark names. Since navigation decisions are usually most influenced by the robot's local surroundings, the matching process is restricted to a local neighborhood around the robot instead of searching the entire graph. Specifically, at each reasoning step, the system defines a circular search region with a radius of 2 meters centered at the robot's current position and gathers all scene graph entities that lie within this local area. These entities form the candidate pool for landmark grounding.

The matching process is applied to both object nodes and region nodes depending on the type of landmark. If the landmark refers to a room or navigable area, such as a bedroom or hallway, the system searches among the region level candidates in the local neighborhood. If the landmark refers to a concrete object, such as a couch, rug, bed, or cabinet, the system searches among the object level candidates. For each primary or secondary landmark phrase extracted from the current subinstruction, the text embedding of the landmark is compared with the stored semantic embedding of each candidate node inside the 2 meter neighborhood. Cosine similarity is then used as the matching score, and the candidates are sorted from highest to lowest similarity.

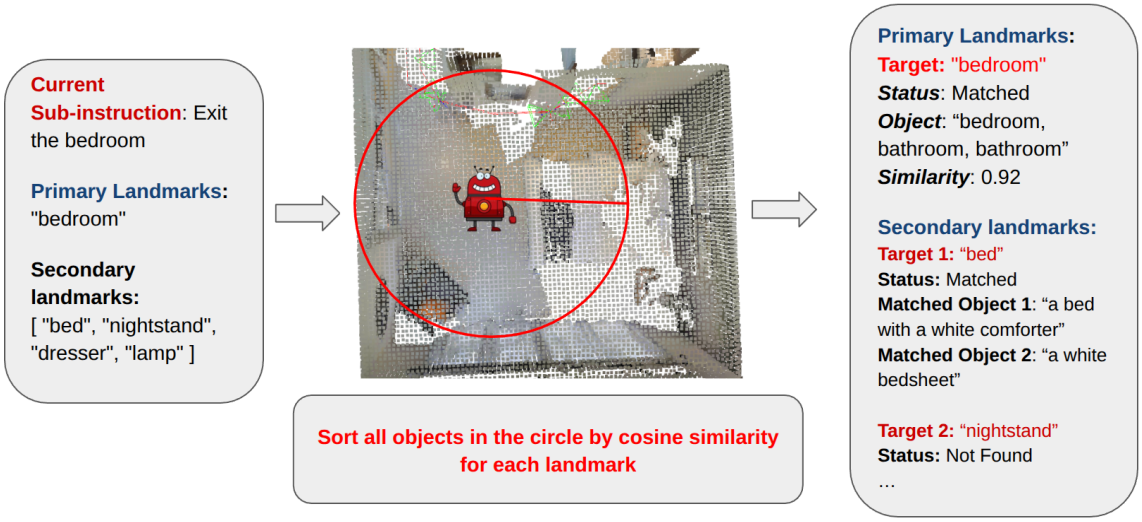


Figure 6: Example of landmark matching stage

Rather than selecting only the single best match, the system retains all candidates whose cosine similarity exceeds a predefined threshold. In this way, each landmark is associated with a candidate set instead of a single hard assignment. This is important because local observations may be noisy, multiple nearby objects may have similar semantic descriptions, and room level grounding can be ambiguous when the robot is close to boundaries or transitions. Primary landmarks are matched first because they define the main semantic target of the current subinstruction. Secondary landmarks are matched in the

same way and provide contextual support for judging whether the primary landmark grounding is reasonable. For example, if the primary landmark is matched to a bedroom region, the presence of nearby secondary candidates such as a bed, nightstand, or lamp strengthens the confidence that the current semantic grounding is correct. The visualization of current subsection with example is showing in Figure 6

The output of this stage is therefore a structured grounding result for each landmark, including the landmark phrase, the local candidate set, the cosine similarity scores, and whether each candidate passes the threshold. This design allows the later reasoning stages to operate not only on the selected semantic grounding, but also on its uncertainty and supporting context. The purpose of this step is to translate the symbolic landmark descriptions in the instruction into locally grounded scene-graph entities that can then be tracked through distance relations and used for deviation detection.

5.5.3 Landmark Distance and Relation Tracking

Once landmarks have been matched to scene-graph entities, the system monitors how the robot moves relative to them over time. For each matched landmark candidate, the system stores a short temporal sequence of robot-to-landmark distances across recent mapping steps. This temporal signal is important because navigation progress cannot be judged from a single observation alone. Instead, the reasoning agent must examine how the relationship evolves over time, for example whether the robot is approaching the target, remaining near it, moving away from it, or passing beyond it.

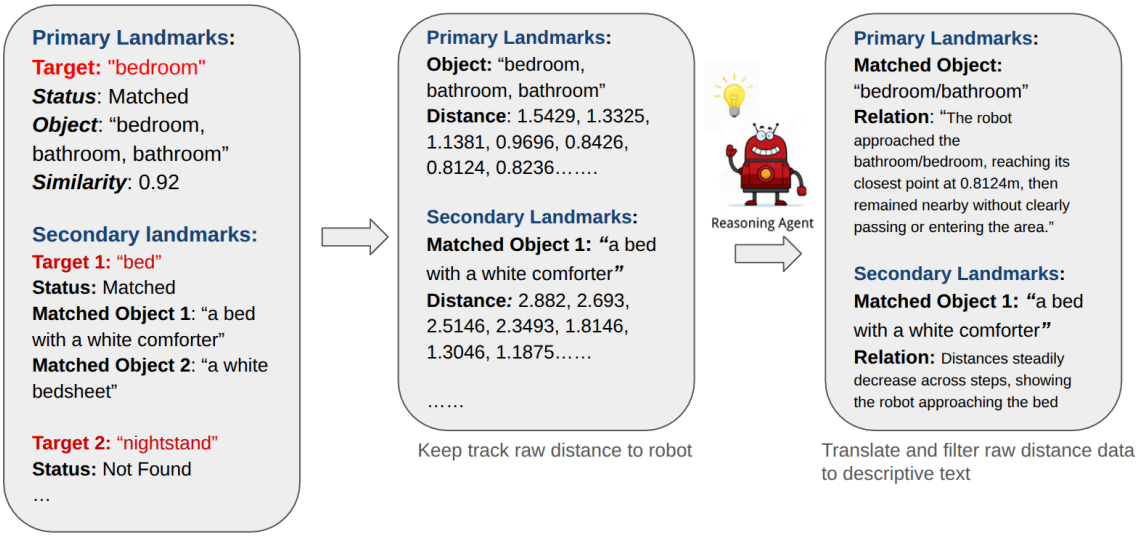


Figure 7: Example of Landmark Distance Description subsection

For object landmarks, the tracked value is the distance between the robot and the matched object location or bounding volume. For region landmarks, the tracked value reflects the robot's relationship to the relevant room or area, such as whether it is moving closer to the region, remaining inside it, or transitioning away from it. However, the raw distance values are not used directly as reasoning inputs. The main reason is that the matched candidates may correspond to entities with very different spatial scales. A small numerical change can have very different meanings depending on the candidate type. For a large region such as a room, a small distance change may say very little about meaningful progress, since the robot may still remain within the same area. By contrast, for a small object such as a lamp, painting, or artwork, a similar distance change may strongly indicate that the robot is approaching, passing, or moving away from that object. Therefore, the same raw geometric signal cannot be interpreted uniformly across all candidate types. There's an example showing the process of this subsection in Figure 7.

To handle this, the system converts the raw distance sequences into descriptive relation summaries. Rather than passing only numerical values to the reasoning model, the recent trend is filtered and translated into text descriptions such as "the robot is approaching the bed," "the robot remained inside the bedroom region," or "the robot moved away from the couch after reaching its closest point." This conversion makes the evidence more semantically meaningful and more robust to differences in spatial scale across matched candidates. It also reduces noise and compresses repeated geometric measurements into a form that is easier for the reasoning agent to interpret.

These relation summaries are generated for both primary and secondary landmarks whenever reliable matches exist. The primary landmark relations provide the main evidence for whether the robot is progressing toward the current subgoal, while the secondary landmark relations provide supporting context when the primary landmark alone is ambiguous. This descriptive filtering step is especially useful for the later deviation detection stage, because the reasoning agent benefits more from interpretable relation-level evidence than from raw numeric distances alone. As a result, this stage transforms low-level geometric tracking into a unified semantic representation that can be combined naturally with the caption memory during higher-level reasoning.

5.5.4 Caption-Based Trajectory Summarization and Current-State Inference

In parallel with scene-graph matching and distance tracking, the reasoning agent also reads from the caption memory. Because captions are generated every 20 frames, they provide a lower-frequency but semantically rich record of the robot's recent experience. Before deviation reasoning is performed, the recent caption history is summarized into two higher-level text representations. The first is a *movement summary*, which compresses several recent captions into a short description of how the robot has been moving across the environment. The second is a *current-state description*, which answers the question of where the robot appears to be now.

The movement summary focuses on trajectory-level semantics. It describes the sequence of recently observed areas and salient transitions, such as moving through a hallway, entering a bedroom, passing a bed, or approaching an open living area. This summary provides temporal continuity that is difficult to infer from the current frame alone. The current-state description, by contrast, emphasizes the robot's most likely present location and the dominant nearby landmarks visible now. Together, these two outputs provide a compact semantic account of both recent movement history and present context. The example of this subsection is visualized in Figure 8.

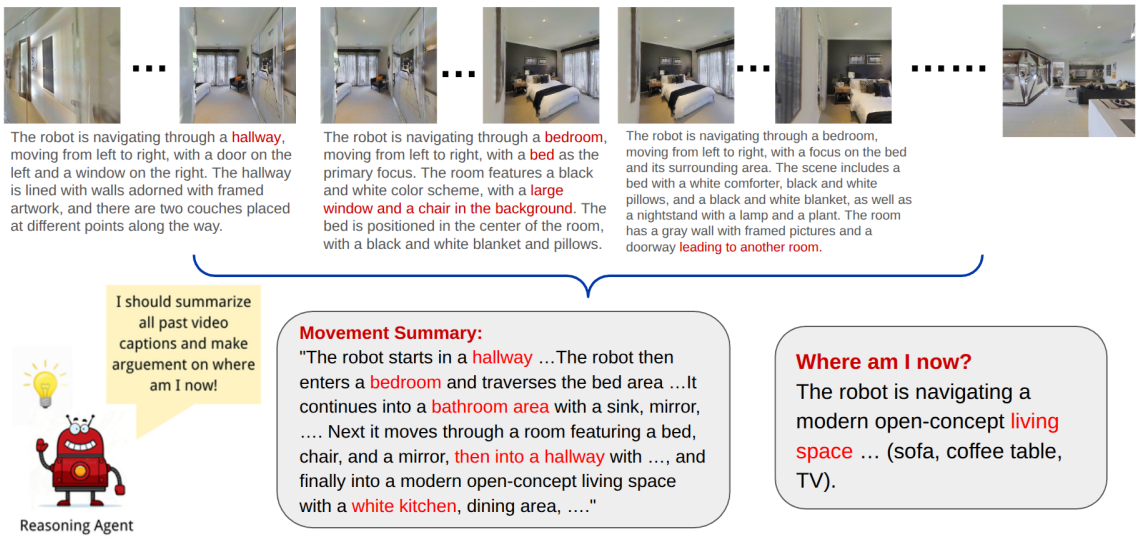


Figure 8: Example of Summarized Video Caption subsection

This caption-based reasoning input is especially important because not all useful evidence is naturally expressed through graph structure. The scene graph records what objects

and regions exist and how they are spatially organized, but the caption memory is better suited to describing the recent semantic flow of the episode. For example, the graph may indicate that both a bedroom and a bathroom are nearby, while the caption summary may reveal that the robot has recently moved from the bedroom toward the bathroom and is now focused on a sink area. This gives the reasoning agent a more coherent semantic picture of what has happened and where the robot likely is now.

5.5.5 Deviation Detection

The deviation detection stage is the core decision point of the reasoning agent. Its purpose is to determine whether the robot is still progressing consistently with the current subinstruction, whether that subinstruction has already been completed, and what the active subinstruction should be for the next reasoning step. The input to this stage includes the current subinstruction, the finished subinstruction summary, the next expected subinstructions, the matched landmark evidence from the scene graph, the distance-based relation descriptions, the current scene description, and the summarized movement history from the caption memory. Rather than making a direct binary judgment from one signal alone, the reasoning agent evaluates these inputs jointly and returns a structured verdict with three possible outcomes: *on-track*, *off-track* or *finished subinstruction*. This design is more suitable for embodied navigation than a simple binary classifier, since the available evidence is often incomplete and the robot may be between subgoals or in an ambiguous transition state.

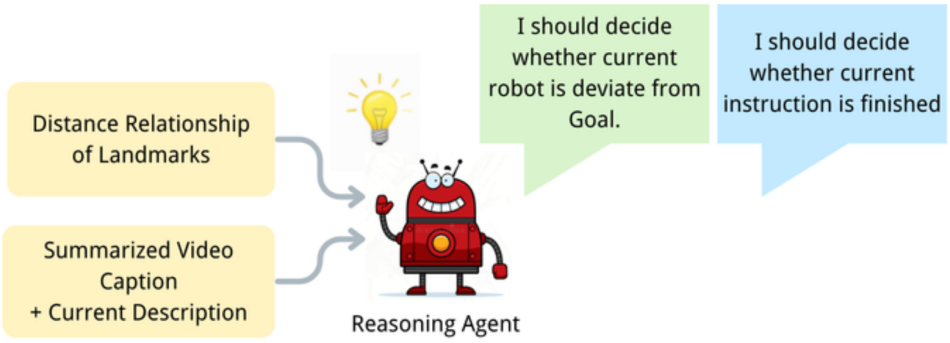


Figure 9: Deviation Detection architecture

The reasoning process follows a fixed order. First, the agent interprets the movement summary to understand how the robot has been progressing over the recent interval. Second, it examines the current scene description to infer where the robot most likely is

now. Third, it checks whether the available landmark evidence supports or contradicts that interpretation. If explicit landmark matches are weak or unavailable, the agent does not immediately conclude failure. Instead, it uses the current description and movement summary to judge whether the robot appears to be in a location where the expected landmarks are likely to be found soon. If the recent trajectory and current scene are still plausibly aligned with the active subinstruction, the robot is treated as still on track. If the available evidence suggests that the robot has moved into an unrelated area, is no longer progressing toward the intended landmark sequence, or is unlikely to encounter the expected landmarks from its present state, the robot is classified as off track. If the evidence is mixed or insufficient, the agent returns an uncertain verdict rather than forcing a confident but unsupported decision.

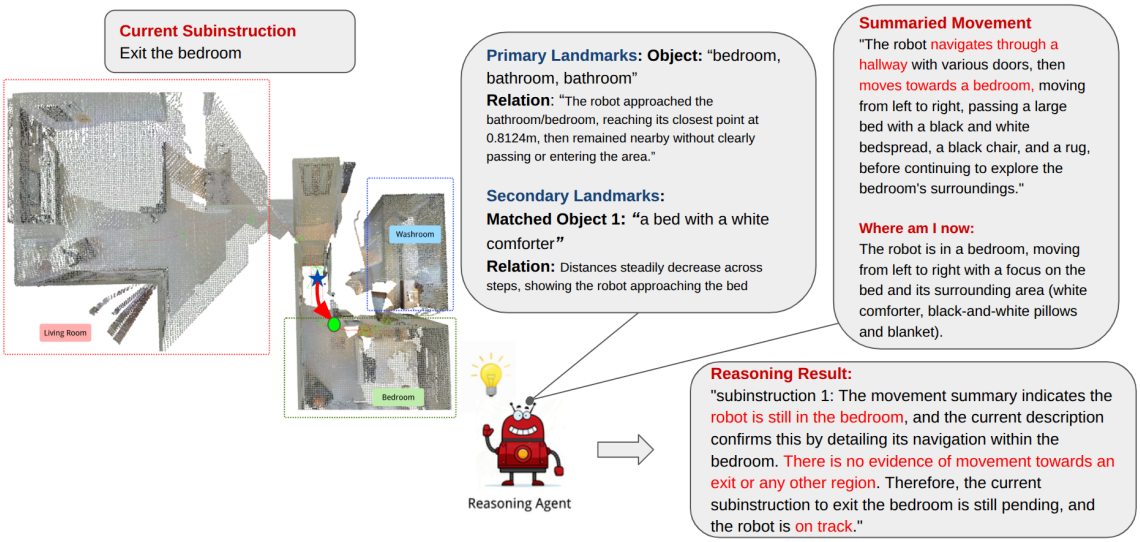


Figure 10: Example of on-track from Deviation Detection

An important feature of this reasoning stage is that deviation detection and subinstruction completion are treated as related but distinct decisions. A robot can remain on track even when the current subinstruction has not yet been completed, as long as the available evidence still supports valid progress toward it. For example in Figure 10, if the active subinstruction is to exit a bedroom and the recent summaries indicate continued movement within the bedroom without contradictory scene evidence, the robot may still be judged on track even though the subinstruction remains unfinished. Conversely, the robot should not be considered to have completed a subinstruction simply because the earlier region is no longer visible. In particular, the reasoning agent is explicitly instructed not to assume that a region has been exited merely because it is not currently observed. Completion is only declared when there is positive evidence that the previous

subgoal has been achieved and that the next expected subinstruction is now semantically supported by the current scene. For example in Figure 11, consider the case where the current subinstruction is *"Exit the bedroom"* and the next expected subinstruction is *"Move forward past the gray couch."* The movement summary indicates that the robot has moved from the bedroom into a bathroom, and the current description may further confirm that the robot is now observing bathroom-related landmarks such as sinks and mirrors. In this case, the reasoning agent can infer that the robot has likely completed the current subinstruction, since it is no longer in the bedroom and has entered a distinct new region. However, this does not imply that the robot is still on track with respect to the overall instruction sequence. Since the next expected landmark, namely the gray couch, is not supported by either the current scene description or the landmark evidence and the couch is not likely will showing up in a bathroom area. The robot is judged to be off track at the task level. This example illustrates why subinstruction completion and deviation detection must be treated separately: the robot may successfully finish one local stage while still deviating from the intended global route.

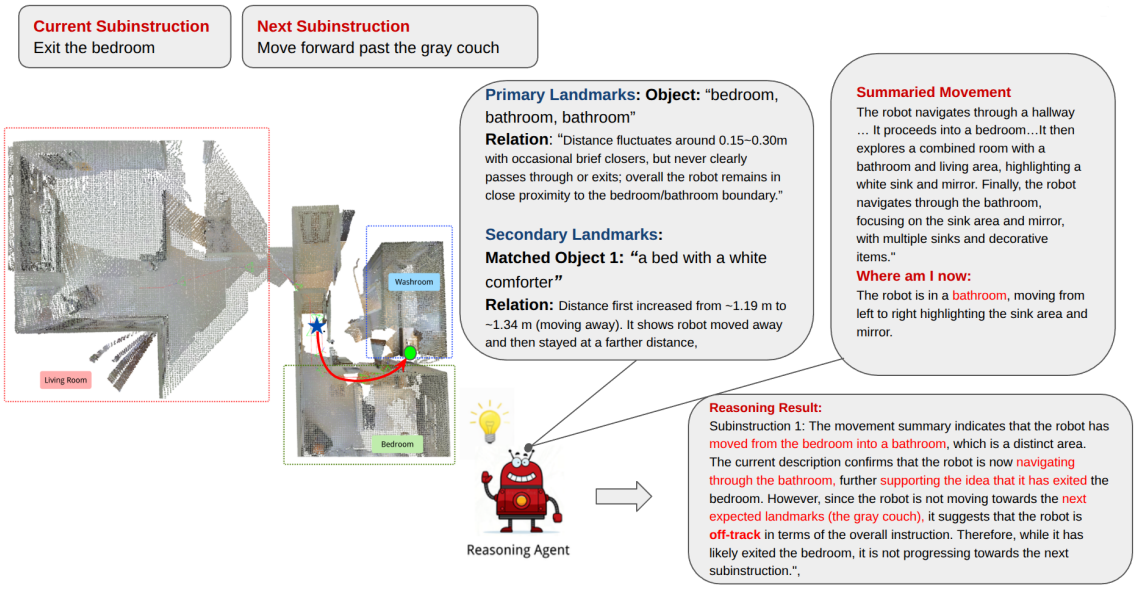


Figure 11: Example of off-track from Deviation Detection

This conservative completion rule is especially important for region transitions. Hallways and similar areas are treated as transitional spaces rather than strong evidence of subinstruction completion. For example, if the robot is supposed to leave a bedroom and proceed toward a living-room landmark, merely observing hallway structure is not sufficient to conclude that the bedroom-related subinstruction is finished. Instead, the agent checks whether the current scene already contains evidence that the next subinstruc-

tion's landmarks are likely present or immediately relevant. Only when the reasoning agent is confident that the previous stage has been passed and the next stage is now active does it mark the current subinstruction as finished and advance the subinstruction index. Otherwise, it keeps the current subinstruction active. This prevents premature stage transitions and reduces the risk of incorrectly declaring success while the robot is still between regions.

The final deviation decision therefore depends on both *local consistency* and *sequential consistency*. Local consistency refers to whether the current scene, movement summary, and landmark relations remain compatible with the active subinstruction. Sequential consistency refers to whether the trajectory also fits the expected order of the full instruction sequence, including whether the next subinstruction is becoming relevant at the correct time. These two forms of consistency are evaluated together in the structured prompt. The output includes the verdict, a confidence level, an updated subinstruction index, a new active subinstruction if needed, and a short reasoning trace summarizing the evidence used. In this way, deviation detection is not merely a yes-or-no check, but a conservative reasoning process that jointly evaluates progress, ambiguity, completion, and failure using both scene-graph evidence and caption-based semantic history.

5.5.6 Recovery Mode for Off-Track Navigation

Once the robot is classified as off track, the system enters recovery mode. The purpose of this stage is not to restart the full episode, but to use the gathered memory to generate a short corrective instruction that can guide the robot back toward the intended route. In the current design, recovery mode is triggered in two cases: either the reasoning agent produces a single high-confidence off-track decision, or it produces two consecutive off-track signals over successive reasoning steps. This rule makes the recovery process both responsive and stable. Strong evidence can trigger immediate correction, while weaker but repeated evidence is used to avoid ignoring persistent deviation.

During recovery mode showing in Figure 12, the reasoning agent first explains *why* the robot is off track based on the available scene graph and caption evidence. The goal is to identify not only that deviation has occurred, but also what semantic cues the robot is currently following instead of the correct route. For example, the robot may be classified as off track because it is currently following bathroom cues rather than returning to the intended bedroom to living room route. This explanation is important because it anchors the recovery action in the robot's present state instead of generating a generic instruction

disconnected from the current environment.

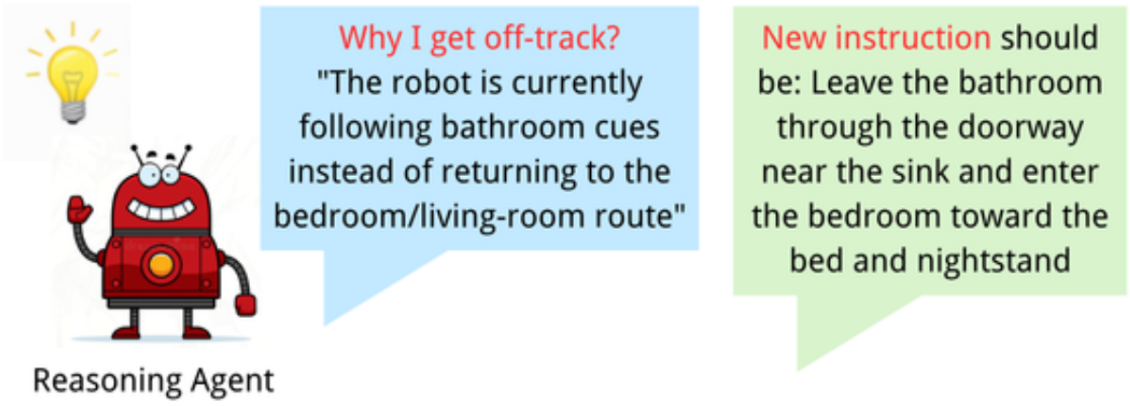


Figure 12: Illustration of Recovery Mode

After identifying the reason for failure, the reasoning agent generates a new locally grounded recovery instruction. This instruction is intentionally short-horizon and explicit. Rather than restating the entire original task, it directs the robot using landmarks that are currently nearby and semantically meaningful from its present location. For instance, if the robot has moved into the bathroom when it should instead return toward the bedroom and continue along the living-room route, the recovery instruction for this episode is: *"Leave the bathroom through the doorway near the sink and enter the bedroom toward the bed and nightstand."* This form of recovery instruction is easier for the VLN backbone to follow because it is based on currently observable landmarks and immediate spatial transitions.

In the current design, recovery instructions are handled through an optional `queue_LIFO` mechanism. Under this setting, the newly generated recovery instruction is inserted back into the instruction queue and processed with higher priority than the remaining original instructions. In other words, although the recovery instruction is inserted last, it is extracted and reasoned on first, following a last-in-first-out strategy. The recovery instruction therefore goes through the same reasoning pipeline as a normal instruction, including decomposition into subinstructions, landmark matching, relation tracking, and deviation detection, until the recovery task is judged complete. After that, the system returns to the original instruction sequence and continues execution from the interrupted task stage.

If the robot deviates again while already in recovery mode, another recovery instruction can be generated and pushed onto the same queue, making recovery itself recursively correctable. This design increases the overall computational cost because additional

instructions must pass through the full reasoning loop, but it provides a more stable mechanism for long-horizon execution. Rather than treating recovery as a one-time intervention, the `queue_LIFO` option allows recovery to behave as a temporary high-priority subtask that must be completed before the robot resumes the main route. The corrected instruction is then republished to the VLN backbone, while the explicit memory branches continue updating throughout the process. In this way, recovery mode remains part of the same closed-loop architecture and offers a more robust execution option for the full navigation task but with higher computational cost and longer execution time.

6 Results

6.1 Evaluation Setup Overview

The evaluation of AgentVLN is conducted directly in the Habitat simulator under the same continuous-environment setting used by NaViLA. This design choice keeps the experimental setup consistent with the baseline and allows the proposed system to be compared within the same VLN-CE evaluation framework rather than introducing changes from a different simulator or task formulation. Following the evaluation plan established earlier in this thesis, the main benchmark setting focuses on Habitat-based VLN evaluation using R2R-CE and other VLN-CE style tasks, where standard navigation metrics can be reported directly.

More specifically, AgentVLN is implemented as an extension of the NaViLA pipeline rather than as a separate navigation stack. The underlying VLN backbone remains the same, while the proposed scene-graph memory, caption memory, and reasoning agent are added alongside it during execution. Because the environment, observation format, and command-generation interface are kept aligned with NaViLA, the same evaluation pipeline can be reused for testing. This ensures that any change in performance is primarily attributable to the added explicit memory and reasoning components rather than to differences in the simulator or evaluation protocol. NaViLA itself is designed and evaluated in the VLN-CE setting with a continuous navigation policy and a language-based intermediate command interface, making it a natural baseline for direct comparison in Habitat.

Accordingly, the reported results in this chapter use the same evaluation procedure and standard VLN metrics adopted by the NaViLA setup. The main quantitative comparison is therefore made under a matched simulation setting, with AgentVLN and the baseline sharing the same Habitat environment and evaluation pipeline. For the R2R-CE unseen-validation split, the full set contains approximately 3400 episodes. However, since the experiments are conducted using only a single NVIDIA RTX 5090 GPU, evaluating the entire split is computationally costly. Therefore, for the experiments reported in this thesis, evaluation is performed on the first 1000 episodes of the unseen validation set. Although this is a reduced subset rather than the full benchmark split, it still provides a controlled and consistent test set to measure whether the added explicit memory and reasoning modules improve navigation robustness, deviation handling, and overall task performance.

6.2 Main Quantitative Results on VLN-CE Benchmarks, R2R-CE dataset

The quantitative evaluation in this thesis follows the standard VLN metrics used in prior continuous-environment benchmarks. Specifically, we report Navigation Error (NE), Oracle Success Rate (OS), Success Rate (SR), and Success weighted by Path Length (SPL). Navigation Error measures the shortest-path distance between the robot's final position and the goal location. Lower values indicate more accurate navigation. Oracle Success Rate evaluates whether the robot ever reaches a successful position at any point along its trajectory even if it does not stop there finally. This reflects whether the correct region was visited during execution. Success Rate measures the fraction of episodes in which the robot stops within the success threshold of the target, making it the main metric for final task completion. Success weighted by Path Length further incorporates path efficiency by penalizing unnecessarily long trajectories, so it captures not only whether the task is completed, but also how efficiently the robot reaches the goal. Together, these metrics provide a balanced evaluation of final accuracy, intermediate reachability, task completion, and trajectory efficiency, and are consistent with the standard evaluation protocol used for VLN-CE benchmarks.

The quantitative performance of AgentVLN is closely tied to the behavior of the reasoning agent, especially the prompt design and how confidently the agent makes on-track and off-track judgments. In practice, the recovery frequency, subinstruction switching behavior, and overall navigation stability are all affected by whether the reasoning policy is too aggressive or too conservative. To study this effect, two prompt settings were evaluated. The first uses a more aggressive reasoning style, which is more likely to classify ambiguous situations as on-track or off-track early and therefore triggers subinstruction transitions or recovery actions more readily. The second uses a more conservative reasoning style, which requires stronger supporting evidence before changing subinstruction state or declaring deviation. As a result, the aggressive setting tends to react faster but is more sensitive to noisy or incomplete evidence, while the conservative setting reduces premature decisions at the cost of slower intervention. The following two tables report the corresponding results under these two reasoning settings showing in 4.

Table 4: Comparison SOTA models and ours with conservative and aggressive settings

Models	**Observations**	**NE↓**	**OS↑**	**SR↑**	**SPL↑**
DreamWalker [36]	Pano+Depth+Pose	5.53	59.0	49.0	44.0
CMA [14]	RGB+Depth	7.37	40.0	32.0	30.0
NaVILA	RGB	5.22	62.5	54.0	49.0
Ours cons.	RGB+Depth+Pose	5.31	68.2	62.6	45.7
Ours aggr.	RGB+Depth+Pose	5.77	60.5	58.1	54.2

From Table 4, the conservative reasoning setting achieves the strongest task completion performance, improving Oracle Success Rate from 62.5 to 68.2 and Success Rate from 54.0 to 62.6 compared with NaVILA. This suggests that a more cautious reasoning policy is more effective at maintaining correct task progress and avoiding premature deviation or subinstruction switching. However, this comes at the cost of lower path efficiency, as reflected by the reduced SPL of 45.7. By contrast, the aggressive setting achieves a higher SPL of 54.2, indicating more efficient trajectories on successful runs, but its OS and SR are lower than the conservative setting. This suggests that aggressive reasoning can react more quickly, but is also more vulnerable to noisy evidence and premature off-track or completion decisions. Overall, these results indicate that the reasoning policy introduces a clear trade-off between navigation robustness and trajectory efficiency, with the conservative setting being more favorable for reliable task completion in the current system.

It is also notable that Navigation Error does not improve as strongly as Success Rate. In the conservative setting, NE is slightly worse than NaVILA, while in the aggressive setting the degradation is larger. This suggests that the main benefit of AgentVLN is not in producing consistently shorter geometric paths, but in increasing the likelihood that the robot eventually reaches or revisits the correct target region through explicit reasoning and recovery. In other words, the proposed system improves robustness of task completion more than raw endpoint accuracy under direct policy rollout.

A comparison caveat should also be noted. The proposed AgentVLN system uses RGB, depth, and pose information for explicit memory construction, whereas the NaViLA baseline in Table 4 is reported with RGB observations only. Therefore, the comparison should be interpreted as a system level evaluation of the proposed explicit memory framework instead of a strictly input matched backbone comparison. At the same time, this difference does not undermine the main claim of the thesis. In the current setting, depth and pose are supporting signals rather than the core source of improvement: depth can

in principle be approximated from modern foundation models, and pose can be obtained through standard SLAM or odometry pipelines. More importantly, the main contribution of AgentVLN is not fine-grained geometric estimation itself, but the construction of an explicit scene representation and the use of a reasoning agent to operate over that representation. The reasoning stage also does not depend on precise metric pose values at a detailed level. It mainly relies on filtered spatial relations and coarse movement trends, such as whether the robot is approaching, leaving, or entering a landmark or region. Since the proposed method is designed around explicit memory, semantic filtering, and reasoning rather than exact geometry alone, the comparison remains fair at the architectural level for evaluating whether explicit scene representation and reasoning improve navigation robustness in practice.

6.3 Ablation Study

To better understand which parts of AgentVLN contribute most to the final performance, an ablation study is conducted by progressively disabling or modifying key components of the proposed framework. The goal of this study is not only to measure the performance of the full system, but also to isolate the effect of each explicit memory and reasoning design choice. In particular, four factors are examined: removing the video caption memory, removing the scene-graph memory, changing the reasoning model, and enabling the optional `queue_LIFO` recovery mechanism. These ablations are directly motivated by the system design proposed in this thesis, where the final behavior depends on the interaction between structured scene memory, semantic caption memory, and the reasoning agent operating over both. For each ablation studies, we only random select 200 episodes due to computation resources. All prompts set to be the conservative version.

W/O Video Caption Memory The first ablation removes the online video caption memory while keeping the scene-graph memory and reasoning pipeline unchanged. This setting evaluates whether semantic trajectory summaries provide useful information beyond the structured object-region representation stored in the scene graph. Without the caption branch, the reasoning agent loses access to natural-language summaries of recent movement and current scene description, and must rely only on graph-based landmark matching and relation tracking. This ablation is expected to reveal whether the caption memory is important for resolving ambiguous progress states, especially in cases where the graph alone cannot fully express the semantic flow of the recent trajectory.

W/O Scene Graph Memory The second ablation removes the online scene graph while retaining the caption memory and reasoning agent. This setting evaluates the complementary role of structured spatial memory. Without the scene graph, the system no longer has persistent object-region organization or explicit spatial relations, and the reasoning agent must instead depend mainly on caption-based semantic descriptions of recent observations. This ablation is particularly important because it tests whether language summaries alone are sufficient for deviation detection and recovery, or whether explicit structured memory is necessary for stable landmark grounding and region-level reasoning. Since one of the main claims of this thesis is that explicit scene representation improves robustness, the no-scene-graph setting acts as a direct test of that hypothesis.

Other reasoning models The third ablation studies the effect of the reasoning model itself. In the full system, the reasoning stage is performed by GPT-5 nano, which is selected as the default reasoning model because it provides a good tradeoff between speed and reasoning capability for repeated online calls. To evaluate how sensitive the framework is to the reasoning backbone, additional experiments are performed by replacing the default model with alternative GPT variants. This comparison helps determine whether the observed gains come mainly from the explicit memory design or are strongly dependent on the capacity of the reasoning model. If the system remains effective across multiple reasoning backbones, this would support the claim that the main contribution lies in the explicit memory and reasoning pipeline rather than in one particular model choice.

Disable queue_LIFO The final ablation evaluates the optional `queue_LIFO` recovery mechanism. In the default recovery design, an off-track recovery instruction can be inserted back into the instruction queue and treated as a temporary high-priority subtask. Under `queue_LIFO`, the most recently generated recovery instruction is processed first, meaning that recovery itself goes through the same reasoning loop of decomposition, landmark matching, progress tracking, and deviation detection before the system returns to the interrupted original instruction. This mechanism increases computational cost, since recovery instructions may recursively generate additional reasoning cycles, but it may also improve execution stability by ensuring that the robot fully resolves the immediate deviation before continuing the main route. The ablation therefore measures whether this added computation produces a meaningful gain in robustness and task completion.

Table 5: Ablation study of explicit memory and reasoning components on R2R-CE val-unseen.

Method Variant	**NE↓**	**OS↑**	**SR↑**	**SPL↑**
Full AgentVLN	5.31	68.2	62.6	45.7
w/o Video Caption Memory	6.27	57.4	54.1	46.3
w/o Scene Graph Memory	5.23	64.7	60.2	49.1
GPT 4o mini	5.46	67.9	62.8	43.9
Disabled `queue_LIFO`	5.13	65.3	59.7	44.5

The ablation results in Table 5 show that the full AgentVLN configuration achieves the strongest overall task completion performance, with the highest Oracle Success Rate and Success Rate among all variants. Among the ablations, removing the video caption memory causes the largest degradation in task success, reducing OS from 68.2 to 57.4 and SR from 62.6 to 54.1. This indicates that caption-based semantic memory is one of the most important components of the proposed system. In the current framework, the caption branch provides high-level trajectory summaries together with useful local scene details, allowing the reasoning agent to better understand what kind of region the robot is currently in, what landmarks have recently been observed, and whether the current movement is still consistent with the intended subgoal. Without this semantic memory stream, the reasoning process becomes much less informed about recent progress, leading to a substantial drop in completion-related metrics.

Secondly, removing the scene graph leads to a smaller reduction in performance. Compared with the full model, the no scene graph setting decreases OS from 68.2 to 64.7 and SR from 62.6 to 60.2, but still remains relatively close to the full system. This suggests that the scene graph is useful, but currently less stable than the caption memory branch in practice. A likely reason is that the scene-graph pipeline is still brittle: some object nodes may be assigned incorrect semantic names or weak descriptions, which becomes problematic when those objects are later used as landmark candidates in the reasoning stage. In such cases, the scene graph may still provide useful structural memory, but inaccurate object naming can weaken the reliability of landmark grounding. This explains why the full system still benefits from the scene graph, while the gain from this module is not yet as large as the gain from the caption memory.

The comparison between GPT-5.4 nano and GPT-4o mini shows that the framework is relatively robust to the exact reasoning backbone. Replacing the default reasoning model with GPT-4o mini leads to only a moderate decrease in performance, with OS dropping

from 68.2 to 67.9 and SR from 62.6 to 62.8. This indicates that the observed gains do not depend entirely on one specific reasoning model, and instead mainly arise from the explicit memory and structured reasoning design of AgentVLN itself.

Disabling the `queue_LIFO` recovery mechanism also reduces performance, lowering OS from 68.2 to 65.3 and SR from 62.6 to 59.7. This suggests that prioritizing recovery instructions improves stability of long-horizon task execution by forcing the robot to resolve the current deviation before resuming the interrupted main route. However, this gain comes with a clear computational trade-off. In the current implementation, turning off `queue_LIFO` reduces the average execution time per task by approximately a factor of two. Therefore, `queue_LIFO` should be understood as a robustness-oriented option: it increases reasoning and execution cost, but provides more stable overall task performance.

Overall, the ablation study suggests that the strongest contributor in the current system is the online video caption memory, followed by the recovery strategy enabled by `queue_LIFO`. The scene graph still provides measurable benefit, but its contribution is currently limited by brittle object naming and imperfect landmark grounding. However, The reason that scene-graph performs limited is because the scene graph's accuracy needed to be improved. This contribute to the core future steps. Because we have shown even though under limited scene graph performance, noticable improvements are also noted. At the same time, the relatively small gap between GPT-5.4 nano and GPT-4o mini supports the claim that the main improvement of AgentVLN comes from the explicit memory and reasoning framework rather than from a single reasoning backbone alone.

6.4 Deviation Detection Performance

Since deviation detection is one of the central functions of AgentVLN, its performance is evaluated separately from the standard VLN success metrics. In this thesis, deviation detection is analyzed from two complementary perspectives. The first focuses on unsuccessful NaViLA episodes, where the main question is whether the reasoning agent can correctly identify that the robot has gone off track. The second focuses on successful episodes, where the system still produces off-track detections during execution. This second case is important because a successful final outcome does not necessarily imply that the intermediate reasoning signals are always correct. Some off-track detections in successful episodes may correspond to real temporary deviation, while others may be

false alarms caused by noisy or incomplete evidence.

For the first analysis, deviation detection accuracy is measured over NaViLA episodes that are unsuccessful at the final task level. In these episodes, if the reasoning agent produces an off-track judgment during execution, the episode is counted as a correct detection. This metric evaluates whether the system can recognize major navigation failures before or by the time the episode ends. Formally, this detection rate can be interpreted as

$$\text{Detection Rate}_{\text{failed}} = \frac{\#\text{ unsuccessful episodes detected as off-track}}{\#\text{ total unsuccessful episodes}}.$$

This measure is useful because it directly evaluates whether the reasoning module can identify trajectories that are inconsistent with the intended instruction, even when the baseline policy would otherwise continue until failure.

A second evaluation is performed over all successful episodes. In this setting, the quantity of interest is the fraction of successful runs in which the reasoning module nevertheless produces an off-track judgment at some point during execution. This metric reflects how frequently the deviation detector intervenes even when the overall task is eventually completed. However, such detections cannot be interpreted directly as errors, because a successful episode may still contain a real temporary deviation in the middle of the trajectory. Therefore, an additional manual diagnosis is performed on this category.

Specifically, from the successful episodes that were detected as off-track, 100 episodes are randomly sampled and manually examined. The purpose of this analysis is to determine whether the detected off-track signal corresponds to a real intermediate deviation or to a false detection. Among these 100 sampled cases, 44 are judged to contain real deviation, meaning that the detector correctly identified a temporary off-track event even though the robot later recovered and finished successfully. The remaining 56 cases are judged to be false detections. This manual analysis is important because it shows that off-track judgments inside successful episodes should not automatically be treated as mistakes. Some of them capture meaningful temporary drift that standard final success metrics do not reflect.

Table 6: Deviation detection performance under unsuccessful and successful episode settings.

Evaluation setting	**Rate / Proportion (%)**
Unsuccessful episodes detected as off-track	73
Successful episodes detected as off-track	24
Manual diagnosis of 100 successful episodes detected as off-track	
Real intermediate deviation (correct detection)	44
False detection	56

As shown in Table 6, the detector identifies 73% of unsuccessful episodes as off track, indicating that the reasoning agent captures a substantial portion of major navigation failures. For successful episodes, the system also produces off-track detections at a rate of 24%. Manual inspection of 100 such successful-detected cases shows that 44% correspond to real intermediate deviation, while 56% are false detections. This suggests that the current detector is useful for capturing genuine temporary drift, but still tends to over-trigger in some successful runs. The main causes are the brittleness of the scene graph, especially incorrect object labels affecting landmark grounding, and the difficulty of interpreting ambiguous scene transitions from limited evidence.

6.5 Recovery and Replanning Performance

To evaluate the effectiveness of the recovery module after deviation is detected, this thesis measures how often the recovery mechanism successfully brings the robot back toward a valid local navigation state after an off-track judgment has been triggered. Unlike the standard VLN metrics, which only evaluate the final outcome of the full episode, this analysis focuses specifically on the behavior of the replanning module itself. In the current implementation, off-track detection may occur multiple times within a single episode, so the recovery analysis is performed at the level of recovery events rather than only at the episode level. This makes the metric more suitable for evaluating whether the reasoning agent can repeatedly provide useful correction once deviation has already occurred.

Table 7 shows that off-track was detected 796 times in total across the evaluated episodes, and the corresponding recovery success rate is 75.3%. This indicates that, once deviation is identified, the replanning mechanism is able to guide the robot back toward a locally reasonable route in a substantial majority of cases. Therefore, the recovery functionality itself is performing reasonably well: the reasoning agent is not only detecting

off-track events, but is also able to generate corrective instructions that are often useful for redirecting the robot toward the intended navigation flow.

At the same time, this relatively high recovery rate does not translate into an equally large improvement on the final VLN-CE benchmark metrics. The main reason is that the current recovery mechanism is primarily local: it can often bring the robot back toward an on-track state, but after recovery the system still returns to the original VLN instruction sequence. If the underlying VLN backbone is still unable to successfully complete that original subinstruction, the robot may deviate again and trigger another recovery cycle. In other words, recovery is often effective at correcting the immediate local mistake, but it does not yet fully resolve the deeper task-level failure that caused the deviation in the first place. This is the main bottleneck limiting final benchmark performance. The episode time limit of 300 seconds further amplifies this issue, since repeated deviation and recovery cycles can consume substantial time budget and prevent the robot from finishing the full task within the allowed duration. Therefore, benchmark performance depends not only on whether recovery works locally, but also on whether the system can continue and successfully complete the original instruction after recovery.

Another factor that limits the benefit of recovery is the quality of the scene graph. Since recovery instructions are generated from the current scene representation and landmark evidence, errors in scene-graph object naming or region interpretation can weaken the quality of downstream replanning. Therefore, the current results suggest that recovery and replanning are already reasonably strong components of AgentVLN, while the larger remaining bottlenecks lie in the reliability of the explicit scene representation and the efficiency of repeated correction within the episode time budget.

Table 7: Recovery and replanning performance after off-track detection.

Metric	**Value**
Times of off-track detected (may multiple in one episode)	796
Recovery Success Rate	75.3%

7 Discussion

This chapter interprets the experimental results presented in the previous section and relates them back to the central research question of the thesis: whether explicit memory and reasoning can improve the robustness of Vision Language Navigation beyond standard end-to-end VLN pipelines. Overall, the results suggest that the answer is yes, but with important qualifications. The proposed AgentVLN framework improves task completion performance and provides meaningful recovery capability by introducing explicit memory and staged reasoning. At the same time, the experiments also reveal that the gains are not uniformly distributed across all components. Some parts of the system, especially the caption-based memory and the reasoning-driven recovery mechanism, already provide clear benefit, while other parts, especially the current scene-graph representation, remain a significant source of error. Therefore, the contribution of this thesis is not simply a better benchmark number, but a clearer demonstration that explicit memory and reasoning can change how VLN systems fail, recover, and interpret their own progress.

7.1 Strength and Limitations

7.1.1 Effect of Explicit Memory on Navigation Robustness

One of the clearest findings of this thesis is that explicit memory improves navigation robustness in a way that standard implicit VLN history does not fully achieve. In a conventional VLN pipeline, past observations are usually carried through hidden states, transformer embeddings, or short visual history windows. These representations may preserve useful information, but they do so in a latent and opaque way. By contrast, AgentVLN separates memory into two readable forms: a scene-graph memory that stores objects, regions, and spatial relations, and a caption-based memory that stores semantic summaries of recent observations and movement. This design allows the reasoning agent to make decisions from explicit evidence rather than only from compressed hidden features.

The quantitative results support this design choice. In particular, the ablation study shows that removing the video caption memory causes the largest drop in success-related metrics among the tested components. This indicates that explicit semantic trajectory memory is highly useful for understanding what the robot has recently seen, which region

it is likely in now, and whether the recent movement is consistent with the current subinstruction. This is an important result because it suggests that, for long-horizon navigation, explicit natural-language summaries of recent scene context are not merely auxiliary information, but a major source of robustness. The caption memory appears to compensate for one of the main weaknesses of standard VLN pipelines: even when the robot has enough local perception to act, it may still lack a stable representation of recent progress.

The scene graph also improves robustness, although its effect is smaller than that of caption memory in the current implementation. This still supports the main thesis claim that explicit spatial structure is beneficial. A scene graph gives the system a persistent notion of what entities have been observed, where they are located, and how they relate to one another and to larger indoor regions. This is especially useful for reasoning about room transitions, nearby landmark consistency, and candidate subgoal grounding. Even though the current graph is imperfect, the results suggest that explicit scene structure helps convert raw observations into memory that can be reused by the reasoning module across time. In this sense, the scene graph is valuable not only as a map, but also as an interface between perception and reasoning.

More broadly, the results suggest that explicit memory changes the failure mode of the navigation system. Instead of failing silently inside a latent policy rollout, the system now has intermediate representations that make it possible to diagnose why the robot may be drifting. The reasoning agent can inspect current scene descriptions, matched landmarks, and movement summaries to ask whether the current behavior still makes sense. This does not eliminate failure, but it makes failure more structured and more recoverable. Therefore, the main effect of explicit memory on robustness is not only better final success rate, but also better interpretability of progress, clearer diagnosis of deviation, and improved opportunity for intervention during execution.

7.1.2 Runtime Overhead and Real-Time Limitations

A major practical limitation of the current AgentVLN framework is its runtime overhead. Compared with the original NaViLA pipeline, the proposed system is significantly slower because multiple modules are executed together during navigation. In addition to the VLN backbone itself, AgentVLN runs online scene-graph construction, periodic video captioning, and the reasoning agent within the same navigation loop. As a result, each episode takes approximately three to five times longer than the baseline system. This

increase in computation is substantial and directly affects the practicality of the method for real-time embodied deployment.

This overhead is not only a matter of total runtime, but also of motion quality. Since the robot must wait for scene-graph updates, caption summaries, and reasoning decisions, navigation becomes less smooth than in the original VLN pipeline. In the current implementation, the robot does not move as continuously or responsively as an end-to-end baseline, because the control flow is repeatedly interrupted by memory construction and high-level reasoning. Therefore, although the proposed framework improves robustness and interpretability, it also introduces a significant systems-level cost that cannot be ignored in practical deployment.

The source of this cost is closely related to the main design philosophy of the thesis. AgentVLN intentionally externalizes memory and inserts reasoning into the execution loop, which makes the system more interpretable and more capable of self-correction, but also more computationally demanding. Scene-graph construction requires geometric grounding and semantic updating, caption generation adds another vision-language inference step, and the reasoning agent must repeatedly process explicit memory and instruction context to produce deviation judgments and recovery decisions. When these components are combined, the resulting navigation loop becomes much heavier than the baseline VLN rollout.

This limitation is important because it introduces a tension between robustness and deployability. The explicit memory and reasoning design improves navigation behavior in difficult cases, but the resulting runtime overhead reduces responsiveness and makes the robot less smooth in execution. Therefore, one of the main open challenges for AgentVLN is not only to improve reasoning quality, but also to make the full pipeline significantly lighter. In its current form, the framework demonstrates the value of explicit memory and reasoning at the algorithmic level, but still falls short of the efficiency needed for smoother real-time robotic operation.

7.1.3 Strengths and Limitations of the Recovery Framework

Another major contribution of the thesis is the reasoning-based recovery framework. The experiments show that once off-track behavior is detected, the replanning mechanism is able to bring the robot back toward a valid local route in a substantial majority of cases. This is an important result because many VLN systems can only fail or succeed

through the backbone policy itself, without an explicit mechanism for semantic self-correction during execution. In AgentVLN, the reasoning agent does not simply classify the current state, it also interprets the cause of the deviation and uses explicit memory to generate a locally grounded corrective instruction. This makes the recovery process more transparent and more task-aware than a purely reactive low-level correction.

The recovery results suggest that this local intervention mechanism is already reasonably strong. Once a deviation is detected, the system can often redirect the robot using current landmarks and scene context. This confirms that the combination of explicit memory and periodic reasoning is not only descriptive, but operational: the memory representations are useful enough to support action-level recovery decisions. The framework therefore demonstrates that reasoning can serve as a practical control layer above a strong VLN backbone, rather than existing only as an analysis tool.

At the same time, the current recovery design also reveals an important limitation. In its present form, recovery is mainly local. It often succeeds in bringing the robot back toward an on-track state, but after that the system still returns to the original VLN instruction sequence. If the VLN backbone is still unable to execute that original subinstruction reliably, the robot may deviate again and trigger another recovery cycle. This means that recovery can correct the immediate local mistake without fully resolving the deeper task-level failure. The result is a mismatch between strong local recovery behavior and only moderate improvement in the final benchmark metrics. In other words, the recovery framework works well as route correction, but not yet as full task-level replanning.

The conservative reasoning design further highlights this tension. A conservative off-track policy helps reduce unstable or premature intervention and is beneficial for reliable task completion. However, it also means that some true deviations are only detected after stronger evidence has accumulated. This makes the system more stable, but less sensitive. The aggressive setting, by contrast, reacts faster and can improve path efficiency, yet it is more vulnerable to noisy evidence and premature decisions. The results therefore suggest that the reasoning based recovery framework is already valuable but still operates under a delicate balance between decisiveness and caution.

Overall, the recovery module should be understood as a strong proof of concept rather than a complete solution. It demonstrates that explicit reasoning can improve embodied navigation after deviation occurs, but it also shows that local correction alone is not sufficient for full long-horizon task robustness. The next stage of improvement should not only make recovery better at returning to the route, but also better at revising the

current task state after recovery has occurred.

7.1.4 Limitations of the Current Scene Graph Representation

The results consistently show that the current scene graph is one of the main bottlenecks limiting the full performance of AgentVLN. Although the graph provides explicit spatial structure and persistent object-region memory, its quality is not always reliable enough for high-confidence reasoning. The most important weakness is semantic brittleness at the object level. Some graph nodes are assigned incorrect or weak object names, and these errors directly affect landmark grounding. When the reasoning module attempts to match a landmark phrase to a scene-graph candidate, incorrect node labels can lead to misleading similarity scores or poor contextual support. Because landmark grounding is one of the main inputs to deviation detection and recovery generation, such errors propagate through the whole reasoning pipeline.

This issue is particularly important in indoor VLN, where many instructions depend on fine-grained landmarks rather than only on large room categories. A small error in naming a subtle object may not matter much for generic mapping, but it matters greatly when that object is used as evidence for whether the robot is near the correct turn point or has entered the expected area. This helps explain why the scene-graph ablation still performs relatively close to the full system: the graph is useful, but its current noise level prevents it from contributing as strongly as a fully reliable structured memory would.

Another limitation is that the current scene graph is more effective at storing structure than at expressing semantic richness. While it can represent object nodes, region nodes, and simple relations, many navigation-relevant distinctions are still easier to capture through the caption memory. For example, whether a robot is moving through a transition space, whether a room feels semantically more like a bedroom or bathroom from the current viewpoint, or whether a landmark is likely to appear next are not always naturally encoded in the graph. In the present system, this burden is partly shifted to the captioning branch, which is one reason the caption memory contributes so strongly in ablation. This does not mean the scene graph is unnecessary. It rather suggests that the current graph representation is structurally useful but semantically incomplete.

There is also a practical limitation related to local candidate retrieval. Because the reasoning stage only retrieves nearby graph entities within a local radius around the robot, the system may miss useful larger-context evidence that lies slightly farther away

but is still important for the route. This local design is computationally sensible, but it can reduce the ability of the graph to support broader planning decisions. In addition, room segmentation and region interpretation are themselves imperfect, which can create ambiguity near boundaries and transitional spaces. Since much of the reasoning logic depends on whether the robot is inside, leaving, or entering a meaningful region, such uncertainty weakens the reliability of the graph as a reasoning substrate.

Taken together, these results suggest that the scene graph is already valuable as an explicit memory representation, but it is not yet strong enough to fully realize the intended benefit of structured spatial reasoning. In the current system, the main limitation is not the idea of the scene graph itself, but the quality and stability of the graph construction pipeline. Therefore, improving semantic labeling, region consistency, and landmark reliability is likely one of the most promising directions for improving overall AgentVLN performance.

7.1.5 Trade-off Between Robustness and Efficiency

A central theme emerging from the results is that robustness and efficiency are not improved in the same way by the proposed system. AgentVLN improves success-related metrics and provides stronger recovery behavior, but these gains often come with additional computation, longer reasoning loops, and lower trajectory efficiency in some settings. This trade-off is visible in several parts of the system design.

First, the conservative and aggressive reasoning settings reveal a clear robustness–efficiency balance. The conservative setting leads to stronger final task completion, suggesting that cautious reasoning is more effective at avoiding premature subinstruction switching and unnecessary recovery triggers. However, this comes at the cost of lower SPL, meaning that successful trajectories are often less efficient. By contrast, the aggressive setting reacts more quickly and can produce more efficient successful paths, but it is also more vulnerable to noisy evidence and therefore less reliable for overall completion. This indicates that the reasoning policy itself acts as a control knob between stable completion and efficient progress.

Second, the optional `queue_LIFO` mechanism provides another example of this trade-off. Prioritizing the most recent recovery instruction improves stability because it forces the system to resolve the immediate deviation before resuming the interrupted task. This is beneficial for long-horizon robustness, especially when repeated errors would otherwise

accumulate. However, it also increases computational cost because the inserted recovery instruction must pass through the full reasoning loop again. The experiments show that disabling `queue_LIFO` reduces the per-task runtime by roughly a factor of two, even though it lowers performance. This makes `queue_LIFO` a robustness-oriented option rather than a universally optimal setting.

Third, the repeated-recovery issue further amplifies the cost of robustness. Because the current recovery mechanism can restore a local on-track state without fully solving the original subinstruction failure, the system may cycle between correction and renewed deviation. This behavior consumes both time and compute budget. The 300 seconds episode time limit prevents these cycles from continuing indefinitely, but it also means that repeated local recovery can still prevent successful task completion within the allowed duration. In this sense, the efficiency cost is not only about raw computation time, but also about how quickly the system can convert recovery into durable progress on the full task.

Therefore, the current results suggest that AgentVLN should be interpreted as a system that prioritizes robustness and interpretability over lightweight execution. This is not a weakness in itself, because the thesis is explicitly motivated by the limitations of purely end-to-end VLN under deviation and long-horizon ambiguity. However, it does mean that future improvements should pay closer attention to how explicit memory and reasoning can be made more selective and more efficient. The long-term goal is not simply to add more reasoning, but to add the minimum reasoning necessary to improve robustness without incurring excessive computational and trajectory cost.

7.2 Future Improvements

Although AgentVLN demonstrates that explicit memory and reasoning can improve navigation robustness, the experimental results and qualitative analysis also reveal several clear directions for future improvement. In particular, the current limitations concentrate in three areas: the reliability of the scene-graph representation, the task-level ability of the recovery mechanism, and the runtime cost of the full pipeline. These limitations do not invalidate the main contribution of the thesis, but they show that the current system should be viewed as a strong proof of concept rather than a fully optimized final solution. The next stage of this research is therefore not to change the overall idea of AgentVLN, but to make each major component more accurate, more stable, and more practical for

long-horizon embodied navigation.

7.2.1 Improving Scene-Graph Reliability through Task-Relevant Filtering

The first and most immediate direction for improvement is to make the scene graph more reliable. As shown throughout the results and in the qualitative examples, one of the main weaknesses of the current system is that scene-graph object names can be noisy or semantically unstable. This becomes especially harmful when those object nodes are later used as landmark candidates by the reasoning agent. In several failure cases, the graph contains objects whose labels are inaccurate or too weakly grounded, which can mislead the reasoning process and cause incorrect off-track decisions or poor recovery instructions. The bedroom–bathroom confusion discussed earlier is a representative example of this problem: if the scene graph provides unreliable semantic evidence, then even a strong reasoning agent can make the wrong judgment because it is reasoning from corrupted memory.

A promising next step is to introduce an information bottleneck style filtering mechanism over the scene graph before it is passed to the reasoning stage. The main idea is that the reasoning agent should not be exposed to all nearby scene-graph objects equally. Instead, the memory should be filtered to preserve only the most task-relevant and reliable semantic cues. In practice, this could mean removing nodes with unstable labels, suppressing objects with low observation confidence, and prioritizing entities that are most strongly aligned with the current subinstruction. For example, when the robot is trying to leave a bedroom and move toward a living-room landmark, it is not useful to send every nearby object to the reasoning agent. What is more useful is a smaller set of reliable objects and regions that are clearly relevant to the current navigation objective.

This filtering process could also be fused with the caption memory. Since the captioning branch already provides strong high-level semantic summaries, it can help determine which graph objects are likely to be most important for the current task. In other words, instead of treating the scene graph and caption memory as two completely separate sources, future work could use caption-based context to select or reweight scene-graph candidates before reasoning. This would make the explicit memory more selective and more semantically coherent. Overall, the goal of this improvement is not only to make the graph denser or more detailed, but to make it more useful for downstream decision making by retaining only the most stable and task-relevant semantic evidence.

7.2.2 Extending Recovery from Local Correction to Task-Level Guidance

The second major direction for improvement concerns the recovery mechanism. In the current framework, recovery is mainly local in nature. Once the robot is classified as off track, the reasoning agent generates a short-horizon corrective instruction based on the current scene and nearby landmarks, and this often succeeds in guiding the robot back toward a locally valid route. However, the system still has an important limitation: after the recovery instruction is completed, the robot returns to the original VLN instruction sequence. If the original subinstruction remains difficult for the VLN backbone, the robot may fail again and trigger another recovery cycle. Therefore, the current recovery design is effective at correcting immediate route mistakes, but it does not yet fully resolve the deeper task-level failure that caused the deviation.

A more complete future design should allow recovery to not only redirect the robot, but also revise how the original task is interpreted after recovery occurs. In other words, recovery should provide hints back to the original instruction progression rather than acting only as a temporary detour. Once the robot has been brought back to a reasonable local state, the system should use the updated scene knowledge to reformulate or refine the active original subinstruction. This would allow recovery to function as task-aware replanning instead of only route-aware correction. For example, if the robot has drifted into a bathroom when it should have continued along a bedroom-to-living-room route, the future recovery module should not only tell it how to leave the bathroom, but also explicitly guide how the next original instruction should now be interpreted from the robot's updated position.

This change is important for long-horizon navigation because many failures are not purely local. A robot may return to the correct neighborhood but still be uncertain about what to do next under the original task sequence. If recovery can update the original instruction state, then the system would no longer depend entirely on the old subinstruction remaining valid after deviation. Such a mechanism would likely reduce repeated recovery loops and improve final benchmark performance. Therefore, a key future goal is to transform recovery from a local correction module into a full task-level replanning mechanism that interacts more directly with the original instruction sequence.

7.2.3 Reducing Runtime Overhead for Smoother Real-Time Execution

The third major direction for improvement is runtime efficiency. In the current implementation, AgentVLN runs scene-graph construction, video captioning, and reasoning together within the same navigation loop. This makes the full system significantly more computationally expensive than the original baseline. In practice, each episode takes roughly three to five times longer than before, which creates a serious limitation for real-time deployment. This overhead does not only increase total runtime. It also makes the robot's motion less smooth, since navigation is repeatedly interrupted by memory construction and high-level reasoning. As a result, even though the system becomes more robust and interpretable, it becomes less responsive from a robotics systems perspective.

Future work should therefore focus on making the pipeline more selective and more asynchronous. One possible direction is to reduce the frequency of expensive operations and trigger them only when needed. For instance, captioning and reasoning do not necessarily need to run at fixed intervals if the current state is stable and unambiguous. Instead, they could be activated only when uncertainty becomes high, when a landmark match becomes inconsistent, or when a possible deviation is detected. Another direction is to cache or reuse intermediate scene-graph states rather than fully updating all graph candidates at every cycle. Similarly, low-confidence or clearly irrelevant graph entities could be pruned earlier so that later reasoning stages receive a smaller and cleaner memory input.

Model-level optimization is also important. The current system already uses a lightweight reasoning backbone, but further gains may come from using more efficient captioning schedules, batching memory operations, or running different modules asynchronously rather than forcing them into a single blocking control loop. The long-term goal is to preserve the benefit of explicit memory and reasoning while making the overall system fast enough for smoother embodied navigation.

8 Conclusion

This thesis studies a central limitation of current Vision Language Navigation systems: although modern VLN models can use past observations during execution, this history is usually preserved only implicitly in hidden states, temporal features, or transformer embeddings. As a result, current VLN systems often struggle to explicitly track progress, diagnose deviation, and recover reliably in long-horizon or ambiguous navigation tasks. To address this issue, this thesis proposes AgentVLN, a modular framework that augments a strong VLN backbone with explicit memory and a separate reasoning agent. Specifically, the proposed system combines an online scene-graph memory, an online video-caption memory, and a reasoning-based recovery module that operates over these explicit representations during navigation.

The experimental results provide a proof of concept that this design is effective. Compared with the baseline VLN setting, the proposed framework improves success-related metrics and demonstrates meaningful deviation detection and recovery capability. The ablation study further shows that explicit memory, especially caption-based semantic memory, contributes substantially to navigation robustness. In addition, the recovery results show that once off-track behavior is detected, the system can often guide the robot back toward a locally valid route. These findings support the main claim of the thesis: explicit memory and reasoning can improve the robustness and interpretability of VLN beyond standard end-to-end implicit-memory pipelines.

At the same time, the current system still has important limitations. The scene graph remains brittle because object labels and semantic grounding can be noisy, which weakens landmark matching and downstream reasoning. The recovery mechanism is also mainly local. This means that it can often return the robot to an on-track state but does not yet fully revise the original task progression. In addition, the full pipeline is computationally expensive, since scene-graph construction, video captioning, and reasoning are all performed during navigation, which limits real-time smoothness. Therefore, while AgentVLN demonstrates that explicit memory and reasoning are promising directions for embodied navigation, further work is needed to improve memory quality, make recovery more task-aware, and reduce runtime overhead for practical deployment.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

[2] Anas Awadalla, Irena Gao, Josh Gardner, Jack Hessel, Yusuf Hanafy, Wanrong Zhu, Kalyani Marathe, Yonatan Bitton, Samir Gadre, Shiori Sagawa, Jenia Jitsev, Simon Kornblith, Pang Wei Koh, Gabriel Ilharco, Mitchell Wortsman, and Ludwig Schmidt. Openflamingo: An open-source framework for training large autoregressive vision-language models, 2023. URL https://arxiv.org/abs/2308.01390.

[3] Rohan Bavishi. Fuyu-8b: A multimodal architecture for ai agents, 2023. URL https://www.adept.ai/blog/fuyu-8b/.

[4] OpenAI et al. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

[5] Enneng Yang, Li Shen, Guibing Guo, Xingwei Wang, Xiaochun Cao, Jie Zhang, and Dacheng Tao. Model merging in llms, mllms, and beyond: Methods, theories, applications and opportunities, 2025. URL https://arxiv.org/abs/2408.07666.

[6] Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale Fung, and Steven Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning, 2023. URL https://arxiv.org/abs/2305.06500.

[7] Ji Lin, Hongxu Yin, Wei Ping, Yao Lu, Pavlo Molchanov, Andrew Tao, Huizi Mao, Jan Kautz, Mohammad Shoeybi, and Song Han. Vila: On pre-training for visual language models, 2024. URL https://arxiv.org/abs/2312.07533.

[8] Zhijian Liu, Ligeng Zhu, Baifeng Shi, Zhuoyang Zhang, Yuming Lou, Shang Yang, Haocheng Xi, Shiyi Cao, Yuxian Gu, Dacheng Li, Xiuyu Li, Yunhao Fang, Yukang Chen, Cheng-Yu Hsieh, De-An Huang, An-Chieh Cheng, Vishwesh Nath, Jinyi Hu, Sifei Liu, Ranjay Krishna, Daguang Xu, Xiaolong Wang, Pavlo Molchanov, Jan Kautz, Hongxu Yin, Song Han, and Yao Lu. Nvila: Efficient frontier visual language models, 2025. URL https://arxiv.org/abs/2412.04468.

[9] Jiafei Duan, Samson Yu, Hui Li Tan, Hongyuan Zhu, and Cheston Tan. A survey of embodied ai: From simulators to research tasks, 2022. URL https://arxiv.org/abs/2103.04918.

[10] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments, 2018. URL https://arxiv.org/abs/1711.07280.

[11] Jing Gu, Eliana Stefani, Qi Wu, Jesse Thomason, and Xin Wang. Vision-and-language navigation: A survey of tasks, methods, and future directions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, page 7606–7623. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.acl-long.524. URL http://dx.doi.org/10.18653/v1/2022.acl-long.524.

[12] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied ai research, 2019. URL https://arxiv.org/abs/1904.01201.

[13] 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.

[14] Jacob Krantz, Erik Wijmans, Arjun Majumdar, Dhruv Batra, and Stefan Lee. Beyond the nav-graph: Vision-and-language navigation in continuous environments, 2020. URL https://arxiv.org/abs/2004.02857.

[15] Muhammad Zubair Irshad, Chih-Yao Ma, and Zsolt Kira. Hierarchical cross-modal agent for robotics vision-and-language navigation, 2021. URL https://arxiv.org/abs/2104.10674.

[16] An-Chieh Cheng, Yandong Ji, Zhaojing Yang, Xueyan Zou, Jan Kautz, Erdem Biyik, Hongxu Yin, Sifei Liu, and Xiaolong Wang. Navila: Legged robot vision-language-action model for navigation. In *RSS*, 2025.

[17] Huy Ha, Pete Florence, and Shuran Song. Scaling up and distilling down: Language-guided robot skill acquisition, 2023. URL https://arxiv.org/abs/2307.14535.

[18] Jiafei Duan, Wilbert Pumacay, Nishanth Kumar, Yi Ru Wang, Shulin Tian, Wentao Yuan, Ranjay Krishna, Dieter Fox, Ajay Mandlekar, and Yijie Guo. Aha: A vision-language-model for detecting and reasoning over failures in robotic manipulation, 2024. URL https://arxiv.org/abs/2410.00371.

[19] Zeyi Liu, Arpit Bahety, and Shuran Song. Reflect: Summarizing robot experiences for failure explanation and correction, 2023. URL https://arxiv.org/abs/2306.15724.

[20] Xiangyu Shi, Zerui Li, Wenqi Lyu, Jiatong Xia, Feras Dayoub, Yanyuan Qiao, and Qi Wu. Smartway: Enhanced waypoint prediction and backtracking for zero-shot vision-and-language navigation, 2025. URL https://arxiv.org/abs/2503.10069.

[21] Bingqian Lin, Yunshuang Nie, Khun Loun Zai, Ziming Wei, Mingfei Han, Rongtao Xu, Minzhe Niu, Jianhua Han, Hanwang Zhang, Liang Lin, Bokui Chen, Cewu Lu, and Xiaodan Liang. Evolvenav: Empowering llm-based vision-language navigation via self-improving embodied reasoning, 2025. URL https://arxiv.org/abs/2506.01551.

[22] Zhuoyuan Yu, Yuxing Long, Zihan Yang, Chengyan Zeng, Hongwei Fan, Jiyao Zhang, and Hao Dong. Correctnav: Self-correction flywheel empowers vision-language-action navigation model, 2025. URL https://arxiv.org/abs/2508.10416.

[23] 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 https://arxiv.org/abs/2412.09082.

[24] Lixuan He, Haoyu Dong, Zhenxing Chen, Yangcheng Yu, Jie Feng, and Yong Li. Mem4nav: Boosting vision-and-language navigation in urban environments with a hierarchical spatial-cognition long-short memory system, 2025. URL https://arxiv.org/abs/2506.19433.

[25] Armin Hornung, Kai Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34, 04 2013. doi: 10.1007/s10514-012-9321-0.

[26] Nathan Hughes, Yun Chang, and Luca Carlone. Hydra: A real-time spatial perception system for 3d scene graph construction and optimization, 2022. URL https://arxiv.org/abs/2201.13360.

[27] Yinan Deng, Meiling Wang, Yi Yang, Danwei Wang, and Yufeng Yue. See-csom: Sharp-edged and efficient continuous semantic occupancy mapping for mobile robots. *IEEE Transactions on Industrial Electronics*, 71(2):1718–1728, 2024. doi: 10.1109/TIE.2023.3262857.

[28] 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.

[29] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation, 2022. URL https://arxiv.org/abs/2201.12086.

[30] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, page 1366–1373. IEEE, September 2017. doi: 10.1109/iros.2017.8202315. URL http://dx.doi.org/10.1109/IROS.2017.8202315.

[31] Yinan Deng, Jiahui Wang, Jingyu Zhao, Xinyu Tian, Guangyan Chen, Yi Yang, and Yufeng Yue. Opengraph: Open-vocabulary hierarchical 3d graph representation in large-scale outdoor environments, 2024. URL https://arxiv.org/abs/2403.09412.

[32] Dominic Maggio, Yun Chang, Nathan Hughes, Matthew Trang, Dan Griffith, Carlyn Dougherty, Eric Cristofalo, Lukas Schmid, and Luca Carlone. Clio: Real-time task-driven open-set 3d scene graphs, 2024. URL https://arxiv.org/abs/2404.13696.

[33] Jun Xu, Tao Mei, Ting Yao, and Yong Rui. Msr-vtt: A large video description dataset for bridging video and language. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[34] Xin Wang, Jiawei Wu, Junkun Chen, Lei Li, Yuan-Fang Wang, and William Yang Wang. Vatex: A large-scale, high-quality multilingual dataset for video-and-language research, 2020. URL https://arxiv.org/abs/1904.03493.

[35] Ranjay Krishna, Kenji Hata, Frederic Ren, Li Fei-Fei, and Juan Carlos Niebles. Dense-captioning events in videos. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[36] Antoine Yang, Arsha Nagrani, Paul Hongsuck Seo, Antoine Miech, Jordi Pont-Tuset, Ivan Laptev, Josef Sivic, and Cordelia Schmid. Vid2seq: Large-scale pretraining of a visual language model for dense video captioning. In *CVPR*, 2023.

[37] Muhammad Maaz, Hanoona Rasheed, Salman Khan, and Fahad Shahbaz Khan. Video-chatgpt: Towards detailed video understanding via large vision and language models, 2024. URL https://arxiv.org/abs/2306.05424.

[38] Bin Lin, Yang Ye, Bin Zhu, Jiaxi Cui, Munan Ning, Peng Jin, and Li Yuan. Video-LLaVA: Learning united visual representation by alignment before projection. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 5971–5984, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.342. URL https://aclanthology.org/2024.emnlp-main.342/.

[39] Yuanhan Zhang, Jinming Wu, Wei Li, Bo Li, Zejun Ma, Ziwei Liu, and Chunyuan Li. Llava-video: Video instruction tuning with synthetic data, 2025. URL https://arxiv.org/abs/2410.02713.

[40] Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Peiyuan Zhang, Yanwei Li, Ziwei Liu, and Chunyuan Li. Llava-onevision: Easy visual task transfer, 2024. URL https://arxiv.org/abs/2408.03326.

[41] Long Lian, Yifan Ding, Yunhao Ge, Sifei Liu, Hanzi Mao, Boyi Li, Marco Pavone, Ming-Yu Liu, Trevor Darrell, Adam Yala, and Yin Cui. Describe anything: Detailed localized image and video captioning, 2025. URL https://arxiv.org/abs/2504.16072.

[42] Dong An, Zun Wang, Yangguang Li, Yi Wang, Yicong Hong, Yan Huang, Liang Wang, and Jing Shao. 1st place solutions for rxr-habitat vision-and-language navigation competition (cvpr 2022), 2022. URL https://arxiv.org/abs/2206.11610.

[43] Hanqing Wang, Wei Liang, Luc Van Gool, and Wenguan Wang. Dreamwalker: Mental planning for continuous vision-language navigation, 2023. URL https://arxiv.org/abs/2308.07498.

[44] Jiazhao Zhang, Kunyu Wang, Rongtao Xu, Gengze Zhou, Yicong Hong, Xiaomeng Fang, Qi Wu, Zhizheng Zhang, and He Wang. Navid: Video-based vlm plans the next step for vision-and-language navigation, 2024. URL https://arxiv.org/abs/2402.15852.

[45] Aaditya Singh et al. Openai gpt-5 system card, 2025. URL https://arxiv.org/abs/2601.03267.