Thinking in 360°: A Zero-Shot Navigation Agent for Photorealistic Indoor Spaces

Supervisor: Dr. Muhammad Tahir

Lahore University of Management Sciences (LUMS)

Ahmad Faraz [1] **Muhammad Musa Zulfiqar** [1] **Hasan Hameed** [1]

Abstract

We present a modular, zero-shot Vision-and-Language Navigation (VLN) system that uses large language models (LLMs) to interpret natural language instructions and navigate photorealistic indoor environments without finetuning or visual pretraining. Built on the Room-to-Room (R2R) benchmark and Matterport3D simulator, our approach introduces a graph-based spatial memory, panoramic scene summarization, and reasoning-centric prompting to guide step-wise navigation. Through careful engineering of visual description generation, hierarchical plan formation, and goal detection, we achieve competitive results in unseen environments and outperform several state-of-the-art baselines. A live web-based demo complements our reproducibility-focused methodology, providing an accessible interface for interactive exploration. Our results highlight the feasibility and interpretability of LLMs in real-world navigation tasks, setting a foundation for scalable, instruction-driven embodied AI.

1. Introduction

Vision-and-Language Navigation (VLN) is a multimodal learning task at the intersection of computer vision, natural language processing, and robotics, in which an embodied agent is required to navigate complex visual environments by following natural language instructions. For instance, an agent may be instructed to "Exit the bedroom, turn right down the hallway, and stop in front of the kitchen," demanding it to interpret linguistic commands, perceive its surroundings through visual input, reason about spatial relationships, and execute an appropriate sequence of actions. VLN represents a crucial capability for real-world embodied AI systems, enabling applications such as home robotics, assistive navigation, and human-robot collaboration, where seamless integration of perception and language is essential.

Formally, the task can be described as follows: given a natural language instruction $I = \{w_1, w_2, ..., w_n\}$, and an embodied agent initialized at a start position s_0 within a partially observable environment E, the objective is to generate a sequence of actions $A = \{a_1, a_2, ..., a_T\}$ that moves the agent from s_0 to a goal location s_T, in a manner that fulfills the intent of I. At each time step t, the agent receives egocentric visual observations V_t and must make decisions under constraints such as limited field of view, incomplete environmental knowledge, and instruction ambiguity. This problem setting introduces a number of challenges including multimodal alignment, long-horizon reasoning, grounding under partial observability, and robust generalization to unseen environments.

Early methodologies in Vision-Language Navigation (VLN) generally employed recurrent neural networks (RNNs) to translate natural language instructions into sequences of navigation actions. These systems often combined language encoders with visual features extracted from pretrained convolutional networks and relied on attention mechanisms to align instruction tokens with visual observations. Despite their effectiveness in constrained or familiar environments, these models exhibited significant limitations. They often lacked true semantic understanding, struggled with grounding language in complex visual scenes, and showed poor generalization to unseen environments. Additionally, their dependence on static visual representations and deterministic policies limited their ability to reason spatially or adapt to new situations.

In response to the limitations of early VLN methods, a second wave of research introduced more powerful architectures aimed at improving semantic understanding, temporal reasoning, and generalization. These approaches moved beyond recurrent models, adopting transformer-based architectures and leveraging pretrained vision-language encoders to better align instructions with visual inputs. This shift enabled agents to reason over longer horizons, incorporate richer contextual information, and exhibit greater robustness to variation in language and environment. However, despite these advances, many models remained dependent on task-specific supervision and struggled with abstract language,

compositional instructions, or sparse rewards in complex scenes. Their semantic representations, while more aligned, were still limited in flexibility and breadth. These persistent challenges motivated the next evolution of VLN systems, which turned toward large-scale language models (LLMs) to further enhance language grounding, reasoning capabilities, and general-purpose understanding.

The most recent phase of VLN research builds upon prior architectural advances by integrating Large Language Models (LLMs) to enhance reasoning, generalization, and instruction understanding. This paradigm shift reframes navigation as a language-centric task, leveraging the strong compositional and commonsense reasoning capabilities of LLMs to interpret complex, abstract instructions and generate step-by-step action plans. Unlike earlier models that relied heavily on supervised learning and fixed visual-language mappings, LLM-based approaches allow for more flexible and interpretable planning through prompting, intermediate reasoning, and dynamic goal decomposition. Additionally, prompt-based domain adaptation techniques and external knowledge integration have shown promise in bridging the gap between pretrained model capabilities and the embodied navigation domain. These models not only improve robustness to linguistic variation and unseen environments but also introduce mechanisms for error correction, subgoal discovery, and anticipation of future observations.

While LLM-based VLN systems have advanced reasoning and instruction understanding, they remain constrained by reliance on synthetic environments, heavy pretraining pipelines, and limited evaluation on realistic benchmarks. These models often struggle with generalization, semantic grounding in complex scenes, and adapting to unseen, real-world environments.

To address these challenges and bridge the gap between high-level language reasoning and grounded spatial navigation, we propose a novel system that combines structured spatial memory with explicit, step-by-step reasoning in realistic indoor environments. Our method centers on a dynamically constructed graph-based spatial representation, which captures the agent's local environment through panoramic scene descriptions and directional relationships. A lookahead exploration mechanism enriches this graph with short-term future context, while selective summarization ensures scalability within model context limits. Decision-making is framed as a structured reasoning task, where powerful language models are prompted to select the next viewpoint based on current observations, navigation history, and graph information, with enforced reasoning justifications to guide deliberate spatial choices. Additional modules for cycle detection, goal validation, and dual-level scene descriptions further enhance reliability and prevent common navigation failures. This integrated approach ensures that each navigation step is informed, interpretable, and grounded in both visual context and high-level task objectives.

2. Related Works

Recent advancements in embodied instruction following have introduced innovative approaches to planning and localization. LLM-Planner, for instance, leverages large language models to perform few-shot grounded planning, enabling agents to interpret natural language instructions and generate high-level action plans with minimal training data. This method demonstrates competitive performance on the ALFRED dataset, despite utilizing less than 0.5% of paired training data, highlighting its sample efficiency and adaptability . In contrast, the "Are We There Yet?" study focuses on enhancing localization capabilities by augmenting the agent's field of view with multiple perspectives and integrating a pre-trained object detection module to better ground language instructions to visual inputs. These enhancements lead to improved navigation and task execution within the ALFRED benchmark . However, it's important to note that both approaches are evaluated within synthetic environments like AI2-THOR, which may not fully capture the complexities and unpredictability of real-world settings, potentially limiting the direct applicability of these methods to real-world scenarios.

Extending beyond modular planners like LLM-Planner, recent research has focused on enhancing VLN by incorporating predictive scene understanding and reasoning capabilities. The work "Improving Vision-and-Language Navigation by Generating Future-View Image Semantics" proposes VLN-SIG, which improves navigation by predicting future visual scenes through proxy tasks such as Masked Panorama Modeling and Action Prediction with Image Generation. Similarly, "NavCoT: Boosting LLM-Based Vision-and-Language Navigation via Learning Disentangled Reasoning" introduces a chain-of-thought guided approach, where LLMs iteratively imagine next observations and actions, effectively narrowing the domain gap between VLN tasks and LLM capabilities. Despite their success on R2R and CVDN benchmarks which represent more real scenarios compared to AI2-THOR, both methods demand extensive pre-training and fine-tuning, raising concerns of high computational cost and potential overfitting, particularly due to the synthetic nature of their training environments.

Unlike methods requiring extensive task-specific training, recent works have explored zero-shot and few-shot planners for vision-and-language navigation (VLN), such as NavGPT, KERM, and DAP. These approaches capitalize on the inherent reasoning abilities of large language models (LLMs) to interpret navigation instructions and make sequential decisions with minimal or no fine-tuning. NavGPT leverages GPT models to predict navigation actions by pro-

cessing textual descriptions of visual observations, history, and explorable directions, showcasing effective zero-shot performance. KERM enhances reasoning by incorporating external knowledge, allowing agents to make contextually informed decisions, while DAP introduces dynamic adversarial patches to improve robustness against visual perturbations. The key advantage of these methods lies in their ability to generalize across tasks and environments without the heavy reliance on large annotated datasets or expensive training pipelines. However, their lack of environment-specific adaptation can limit performance in complex, real-world scenarios, where ambiguous or novel situations may lead to suboptimal actions.

Despite significant progress, VLN methods still face key challenges: reliance on synthetic datasets limits real-world applicability, heavy training introduces computational burdens and risks overfitting, and existing zero/few-shot approaches struggle in complex scenarios. These limitations highlight the need for a novel approach that is efficient, generalizable, and robust to real-world complexities.

3. Experimental Setup

3.1. Dataset

Selecting an appropriate dataset is critical in Vision-and-Language Navigation (VLN) research, as it significantly influences the model's capacity for generalization, the realism of the learned tasks, and ultimately the validity of our experimental results. At the onset of our research, we undertook a thorough exploration of existing VLN datasets, each offering unique strengths and targeting distinct aspects of the navigation problem.

We began by analyzing the state-of-the-art datasets that have emerged in recent years:

1. **TEACh (2022)**: Emphasizes multi-turn dialogue for household tasks. Focuses heavily on dialogue management rather than pure navigation.
2. **AI2-THOR (2019)**: Provides realistic physics and object manipulation across diverse rooms, but lacks explicit language-guided navigation.
3. **ALFRED (2020)**: Integrates language-instructed, multi-step object manipulation—strong for manipulation but not focused on spatial navigation reasoning.
4. **RxR (2021)**: Multilingual expansion of R2R with richer grounding annotations. Too complex for our zero-shot navigation focus.
5. **MINOS (2018)**: Flexible multimodal simulator, yet diverges from clear instruction-following scenarios due to its general-purpose design.
6. **Touchdown & Talk the Walk**: Outdoor Google Street View tasks—valuable but outside our indoor navigation scope.
7. **SOON & ObjectNav**: Excellent for object-goal navigation but lack complex natural-language instructions.
8. **VLN-CE & CHALET (2020)**: High realism and continuous control, yet emphasize motion and manipulation over language-guided paths.

After this survey, we selected the *Room-to-Room (R2R)* dataset. for our experiments based on the following considerations:

1. **Benchmark status:** R2R is the seminal VLN dataset, enabling direct comparison with prior and concurrent methods.
2. **Photorealism:** Uses real Matterport3D scans, capturing genuine indoor layouts, lighting variations, and occlusions.
3. **Natural language quality:** Instructions collected via Mechanical Turk are varied, natural, and realistically ambiguous.
4. **Balanced complexity:** Paths are challenging yet manageable, with clear splits for train, validation, and unseen test environments.
5. **Extensibility:** Integrates seamlessly with simulators (Matterport3D, Habitat) and modern multimodal models.
6. **Community support:** Widely adopted in VLN challenges and publications, aligning our work with established standards.
7. **Resource efficiency:** Moderate scale and structure facilitate rapid zero-shot prototyping without excessive compute.

This methodical selection process ensures that our zero-shot VLN experiments are conducted on a dataset offering realism, linguistic richness, and community comparability, yielding results with strong real-world relevance.

3.2. Simulator

The Room-to-Room (R2R) dataset is a pioneering benchmark for Vision-and-Language Navigation (VLN), introduced by Anderson et al. (Anderson et al., 2018a). It is constructed atop the Matterport3D dataset, which provides richly annotated, photorealistic 3D reconstructions of real indoor environments captured using RGB–D panoramic

cameras. The R2R task requires an embodied agent to interpret free-form natural language instructions and navigate within a simulated environment to reach a specified goal location.

Each R2R navigation episode consists of:

- A starting viewpoint and a natural language instruction describing the navigation path.
- A goal location that is spatially distant (average path length $\sim$9 m), often requiring multiple discrete movements.
- No explicit path annotation, allowing multiple correct trajectories as long as the goal is reached within a 3 m threshold.
- Instructions collected via Amazon Mechanical Turk exhibiting natural ambiguity, co-reference, and implicit spatial relations.

The Matterport3D Simulator is an interactive, high-fidelity simulation environment built to navigate the panoramic scans of the Matterport3D dataset. It provides:

- **Discrete action space:** Agents can execute one of six actions (move forward, turn left/right, look up/down, stop).
- **Graph-based topology:** Nodes are panoramic viewpoints; edges represent physically traversable paths (precomputed via ray tracing).
- **RGB panoramic observations:** Each node contains 36 discretized views spanning the full 360° field of view.
- **Environment realism:** Includes natural occlusions, complex room layouts, and real lighting conditions.

(a) Kitchen Scene

(b) Corridor Scene

(c) Dining Scene

Figure 1. Key locations of an episode in the Matterport3D simulator.

JSON Episode Example


```
{
    "distance": 8.24,
    "scan": "17DRP5sb8fy",
    "path_id": 6833,
    "path": [
        "77a1a11978b04e9cbf74914c98578ab8",
        "b185432bf33645aca813ac2a961b4140",
        "5e9f4f8654574e699480e90ecdd150c8",
        "08c774f20c984008882da2b8547850eb",
        "1a41339ece1846eda6a924cdb4c417dd",
        "558ba0761bf24428b9cf91e60333ea25",
        "00ebbf3782c64d74aaf7dd39cd561175"
    ],
    "heading": 2.96,
    "instructions": [
        "Walk forward in the direction of
    the dining room. Veer right, and go
    down the hall into the bathroom. Stop
    in front of the sink. ",
```

```
        "Head to the left through the
    dining area, then veer right into the
    bathroom. Stop near the sink. ",
        "Go straight and follow the bar, go
     to the right and continue through. Go
    to the bathroom and then stop by the
    sink. "
    ]
}
```

4. Forming Our Methodology

To rigorously investigate the potential of Large Language Models (LLMs) in Vision-and-Language Navigation (VLN) without reliance on pretraining or finetuning, we developed a lightweight prototype pipeline focused on a single Room-to-Room (R2R) episode. This early-stage exploration was essential for validating core design assumptions, exposing infrastructure constraints, and uncovering the practical challenges of grounding spatial instructions using multimodal reasoning.

The primary objective of this prototype was to assess the ability of off-the-shelf LLMs to reason over visual input and produce coherent step-by-step plans that navigate an agent toward a defined goal. The system was composed of multiple components, each tailored to handle specific sub-tasks in the VLN pipeline.

4.1. Image Acquisition and Filtering

We began by selecting an R2R training episode with scan ID `5LpN3gDmAk7`, ensuring that path lengths exceeded 15 meters to enforce sufficient reasoning complexity. Panoramic RGB images were extracted from the `matterport_color_images` dataset, prioritizing front-facing and downward-facing views while discarding upward-facing images, which were found to be less relevant for navigational planning.

4.2. Visual Description Generation with Llava

To convert the collected images into a form suitable for textual prompting, we employed the LLaVA-Next v1.6 (Mistral-7B) vision-language model. Each image was passed through a detailed captioning prompt tailored for spatial description. The prompts emphasized key navigational features such as doorways, stairways, halls, and relative object positions. This conversion facilitated the use of text-based LLMs by providing structured and navigationally useful scene summaries derived from raw visual input.

4.3. Hierarchical Plan Generation via LLM

The generated scene descriptions were fed into the Mixtral-8x7B model through the ChatGroq interface to incrementally produce Hierarchical Language Plans (HLPs). Each planning step was conditioned on three inputs: the agent's current navigation goal, the accumulated history of prior steps, and the textual scene description corresponding to the current view. The prompting strategy was carefully structured to ensure atomic action generation, avoidance of direct image referencing, and encouragement of spatially grounded instructions such as "turn left," or "walk through the hallway."

4.4. Goal and Step Embedding Evaluation

To ensure progression toward the goal while minimizing redundancy, each generated instruction was embedded using the all-MiniLM-L6-v2 model from SentenceTransformers. Cosine similarity was computed between the current step and both the goal embedding and the previous step. Instructions that demonstrated semantic novelty while remaining aligned with the navigation goal were retained. This filtering mechanism aimed to enforce coherence and reduce repetitive step patterns within the generated plans.

4.5. Memory Consolidation through Summarization

To simulate long-term memory and context retention, we implemented a lightweight summarization module using OpenAI GPT-3.5. At each stage, navigation history was condensed into coherent past-tense summaries, which were then injected into future prompts. This surrogate memory served to enhance prompt conditioning by maintaining continuity and logical flow across planning steps.

4.6. Execution Infrastructure and Constraints

Due to limited local GPU access, our experiments were executed on Kaggle cloud notebooks. These platforms offered temporary GPU access with adequate memory to support quantized deployments of Llava and Mistral-7B. We used 4-bit quantization via BitsAndBytesConfig to reduce VRAM consumption and performed all tensor operations in `torch.float16` to maintain compatibility with the deployed models. This configuration proved resource-efficient and enabled fast prototyping and iteration during the early phases of experimentation.

4.7. Initial Observations and Failure Analysis

Preliminary results revealed that the system could occasionally produce plausible navigation steps aligned with the intended goal. However, overall plan quality suffered significantly in complex environments. A key failure mode was step repetition, where identical actions were issued across multiple time steps, suggesting insufficient memory of previous progress. Furthermore, reasoning breakdowns were common, particularly when spatial consistency was re-

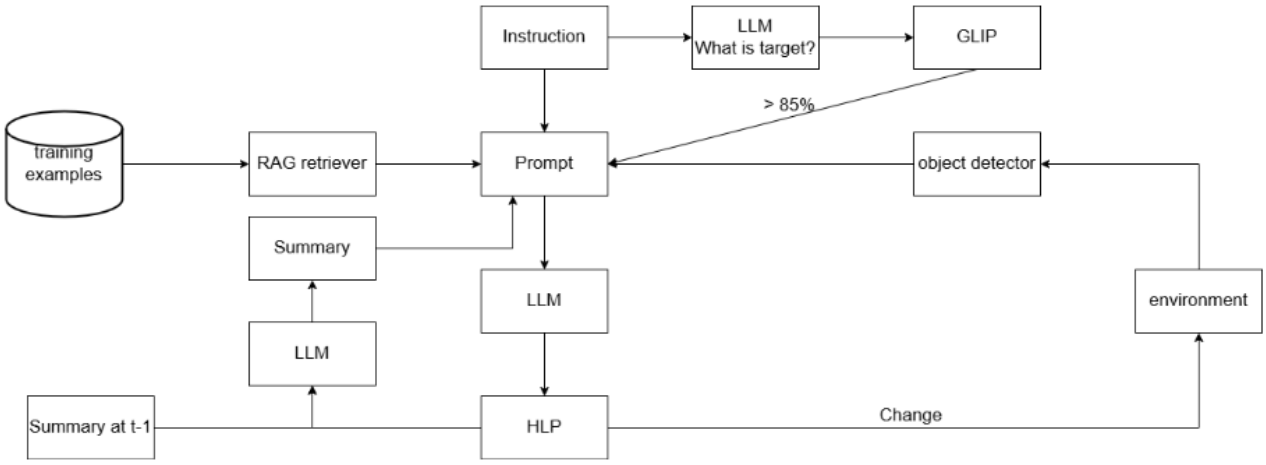

Figure 2. Initial pipeline

quired across occluded or partially visible areas of the scene. Hallucinated instructions—such as referencing nonexistent hallways or objects—were frequently observed when visual context was ambiguous.

Attempts to pass multiple panoramic images into a single prompt led to overloading and semantic confusion, demonstrating a clear limitation in multi-image reasoning capabilities. In response, we explored auxiliary strategies to mitigate these issues. Cosine similarity–based filtering was initially used to reject semantically inconsistent steps, but it frequently misjudged irrelevant steps as high-quality due to surface-level similarity. An example included a faulty step receiving a misleadingly high similarity score of 0.8556.

We also tested an image-stitching approach, where multiple frames were combined into a grid to provide richer context. However, LLaVA continued to misground objects and hallucinate spatial elements, indicating inherent limitations in its multi-image integration. A more promising strategy involved dividing scene descriptions into separate prompts distributed across multiple LLM calls. This divide-and-conquer method reduced hallucinations and improved logical flow, though occasional spatial inconsistencies persisted.

In summary, while the prototype showed initial promise in enabling zero-shot VLN planning through LLMs, it faced critical limitations due to the absence of persistent memory, poor visual context aggregation, and semantic drift across planning steps. These insights directly motivated the development of our final methodology, which integrates structured spatial representations and step-wise justifications to address the shortcomings observed in this exploratory phase.

5. Method

Our approach focused on the systematic development and refinement of a zero-shot Vision-and-Language Navigation (VLN) agent capable of interpreting natural language instructions and executing them in complex indoor environments. The methodology leveraged the Matterport3D Simulator for realistic visual grounding and integrated advanced multimodal language models (LLMs) for step-wise spatial reasoning and planning. This section outlines the architecture, implementation strategy, and engineering considerations of our proposed system.

5.1. Simulator and Execution Environment

The foundation of our navigation pipeline was the Matterport3D Simulator, a photorealistic environment based on panoramic RGB imagery from real-world indoor scenes. Navigation episodes were defined over discrete viewpoint graphs provided by the simulator API (MatterSim), where the agent executed transitions between viewpoint nodes rather than continuous physical motion. This abstraction allowed us to isolate reasoning and planning capabilities without introducing control dynamics.

To resolve environment-level dependency conflicts—particularly between the simulator's legacy Python Docker setup and modern ML libraries such as `transformers`, `torch`, and `sentence-transformers`—we split the system into two modular components:

- **Simulation Module (`driver.py`)**: Responsible for managing the simulator, executing navigation commands, capturing panoramic views, and transmitting data to the external inference server via REST APIs.

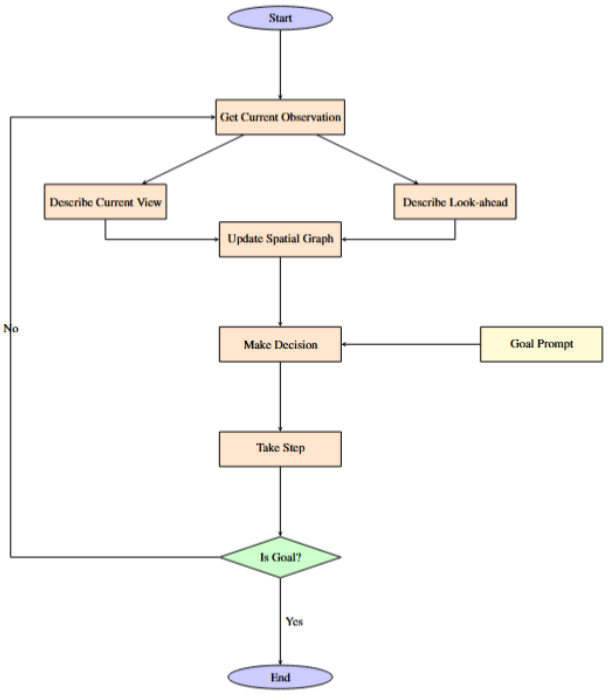

Figure 3. Current Pipeline

- **Inference Server (`server.py`)**: A Flask-based microservice hosting captioning, LLM inference, and reasoning logic, executed on an NVIDIA RTX 4090 GPU.

This separation of concerns allowed asynchronous interaction between the simulator and compute-heavy language models, improving reliability and compatibility across components.

5.2. Visual Perception and Scene Description

5.2.1. Panoramic Scene Capture and Filtering

At each navigation step, the agent captures a full 360-degree view of the environment using the simulator's `get_images()` function, which returns seven RGB images at 60-degree intervals. To eliminate redundant perspectives and reduce computational overhead, we applied a consistent filtering strategy via `filter_images()`, selecting the front-facing image first, followed by alternating views from opposing ends (i.e., front-left and front-right). This typically yields three directional images representing left, forward, and right perspectives—an arrangement that maintains sufficient spatial context while minimizing prompt length.

5.2.2. Image Captioning with LLaVA

Filtered images were passed individually to LLaVA-Next (Mistral-7B) deployed on the inference server. We used the following structured prompt to encourage navigationally relevant descriptions:

```
img_desc_prompt = """
Describe the image with a focus on
    navigational elements
such as hallways, doorways, stairs, and
    open spaces.
Pay attention to the positions of these
    elements and their
relationships to each other.
Use directional cues like 'left,' 'right,'
    'ahead,' or 'behind'
to provide spatial context.
"""
```

The `getImageDescriptions()` function returned concise spatially grounded captions. These descriptions were then unified using `unify_descriptions()` through an additional LLM call to yield a coherent and compact representation of the entire scene.

5.3. Graph-Based Spatial Memory and Reasoning

5.3.1. Motivation and Structural Overview

Initial prototypes revealed that LLMs alone could not handle long-horizon spatial reasoning, especially in cluttered or repetitive environments. To overcome this, we introduced a dynamic spatial memory encoded as a directed graph using `networkx`. This graph stored structured knowledge of explored viewpoints, their textual descriptions, and interconnections representing agent movement.

5.3.2. Lookahead-Based Graph Construction

To expand the agent's understanding of its immediate surroundings, we implemented a lookahead mechanism using `get_lookahead_desc()`. For each navigable neighbor, the agent initialized temporary episodes via `sim.newEpisode()` without committing to movement. Descriptions of these adjacent nodes were generated and added to the navigation graph as new nodes, with edges representing bidirectional navigability. This process incrementally constructed a local subgraph of depth $k+1$ at each step, allowing the agent to reason about short-term futures during decision-making.

5.3.3. Graph Summarization and Context Pruning

To ensure scalability within the input limits of LLMs, we developed a summarization utility `graph_to_text_shortened()` that invoked an external LLM endpoint (`/summarize_desc`) to compress textual information from distant or irrelevant nodes. Immediate neighboring nodes retained full descriptions, while peripheral ones were represented with summaries. This approach preserved fine-grained local context and enabled memory-efficient long-horizon planning.

5.4. Decision-Making and Viewpoint Selection

5.4.1. Reasoning-Centric Prompting

Viewpoint selection was framed as a reasoning task. At each step, the LLM was prompted with a JSON input structured as follows:

```
{
  "current_img_desc": "...",
  "viewpoint_descs": {"viewpoint_id1": "
    description1", ...},
  "goal": "...",
  "history": "...",
  "graph_info": "..."
}
```

The model was instructed to output:

```
{
  "viewpointID": "<chosen viewpoint>",
  "reasoning": "<justification for this
    choice>"
}
```

This strict response format, enforced through prompt design, yielded better interpretability and helped minimize impulsive or inconsistent choices.

5.4.2. Cycle Detection and Recovery

To avoid redundant backtracking, we implemented a loop detection mechanism based on the `visited[]` list. When a previously visited viewpoint was selected again, a dedicated endpoint (`/detect_cycle`) prompted the LLM for a justification and request for alternative candidates. This led to context-aware rerouting that avoided stagnation and improved overall navigation efficiency.

5.5. Goal Detection and Validation

Recognizing goal attainment was often hampered by ambiguous phrasing and visual similarities. To address this, we deployed a goal-checking function `reached_goal_unified()`, which computed semantic similarity between the current unified scene description and the goal statement. When multiple viewpoints matched the goal semantically, we used a confirmation threshold to validate goal achievement and prevent false positives.

5.6. Prompt Engineering and Model Optimization

5.6.1. Dual Description Strategy

Each node in the graph maintained a tuple of descriptions: one simplified and one detailed. The simplified version enabled quick filtering and reduced token usage during LLM calls, while the detailed one supported nuanced reasoning when ambiguity arose.

5.6.2. Model Scaling and Tuning

We began with GPT-4o-mini and Mixtral-8x7B but encountered reasoning failures in complex scenes. Migrating to larger models such as DeepSeek-R1 and GPT-4o provided significant improvements in output quality due to increased context windows and more robust inferencing. Prompt formats were incrementally refined to ensure consistent output schema, enforce logical justifications, and maintain step coherence across long navigation trajectories.

5.7. Summary

Our complete methodology combines structured spatial memory, step-wise planning, robust visual-language grounding, and prompt-based reasoning into an efficient zero-shot VLN framework. By decomposing the problem across sim-

ulation, perception, reasoning, and execution, and engineering interfaces between modules via modular prompts and RESTful APIs, we ensure adaptability, reproducibility, and strong generalization to realistic indoor environments. The subsequent section will provide empirical results and qualitative analysis validating the effectiveness of our approach.

6. Experiment

This section presents our comprehensive evaluation of the proposed zero-shot Vision-and-Language Navigation (VLN) pipeline. We describe the dataset and evaluation protocol, define the metrics used, explain implementation details, and report quantitative results alongside a comparative analysis with existing state-of-the-art (SOTA) methods.

6.1. Dataset and Evaluation Setup

We evaluate our system on the Room-to-Room (R2R) benchmark, focusing specifically on the validation unseen split. This subset features environments that do not appear during training, providing a rigorous test of the model's generalization ability in a zero-shot setting. Notably, our method does not rely on any finetuning on R2R environments, preserving the zero-shot constraint across all evaluation stages.

6.2. Evaluation Metrics

The evaluation adheres to the standardized VLN metrics widely adopted by the R2R leaderboard. Trajectory Length (TL) measures the average path length in meters, capturing agent exploration tendencies. Navigation Error (NE) quantifies the distance in meters between the final agent position and the goal location, reflecting accuracy. Oracle Success Rate (OSR) computes the success rate if the agent were to stop at the closest point to the goal along its path, isolating trajectory planning from stopping behavior. Success Rate (SR) evaluates the percentage of episodes where the agent ends within 3 meters of the goal, indicating task completion. Success weighted by Path Length (SPL) combines SR with path efficiency by penalizing unnecessarily long trajectories. These metrics collectively provide a comprehensive assessment of both the accuracy and efficiency of VLN agents. While many state-of-the-art systems optimize explicitly for SPL or NE, often leveraging curriculum learning and task-specific training, our method remains finetuning-free.

6.3. Evaluation Procedure and Implementation

We adapted the official R2R evaluation codebase and implemented a customized evaluation pipeline through `evaluator.py`. Navigation graphs were constructed using the `_load_nav_graphs` function, enabling agent movement based on pre-defined panoramic viewpoints. The shortest path between viewpoints was calculated using 3D Euclidean geometry, ensuring precise measurement of navigation error and trajectory efficiency. Generated agent trajectories were matched against ground truth data from the R2R dataset using unique identifiers from `R2R_test.json`. Our evaluation was conducted across 29 episodes comprising 86 instructions.

6.4. Quantitative Results

Our system successfully followed instructions in 21 out of 29 episodes, yielding an instruction-level accuracy of 53.49%. The average trajectory length was 11.77 meters, with a navigation error of 6.28 meters. The agent achieved an OSR of 47.13%, SR of 47.13%, and SPL of 41.31%. These results demonstrate that our zero-shot VLN model, without any visual feature training or R2R-specific adaptation, achieves strong generalization and competitive navigation performance.

6.5. Comparison with State-of-the-Art

We compare our model with both language-only and cross-modal methods. Among language-only approaches, our system outperforms NavGPT, achieving higher success rate (47.13% vs. 34%) and SPL (41.31% vs. 29). Furthermore, we improve over NavCoT with LLaMA-Adapter (SR: 21.97%) and NavCoT with LLaMA-2 without augmentation (SR: 36.40%, SPL: 33.17%). Although our method does not match the highest-performing cross-modal systems such as ScaleVLN or BEVBert, which benefit from strong visual backbones and extensive pretraining, it significantly outperforms earlier supervised systems like Seq2Seq and Speaker-Follower, despite being finetuning-free and vision-free. These findings reinforce the viability of large-scale LLMs for spatial reasoning and navigation planning under zero-shot constraints.

To summarize, our method demonstrates that it is possible to achieve strong VLN performance without visual pretraining or finetuning, simply by leveraging high-capacity LLMs, robust scene descriptions, and a structured reasoning pipeline. The next section provides a complete reproduction guide, covering dependency installation, dataset preparation, simulator integration, and evaluation workflows to facilitate future research in this domain.

7. Reproducibility and Setup Guidance

Ensuring reproducibility is a cornerstone of robust scientific research. This section offers a detailed and transparent guide for replicating all experimental procedures and results of our Vision-and-Language Navigation (VLN) system. It encompasses step-by-step setup instructions, environment configurations, data acquisition, simulator deployment, inference server management, and evaluation protocols.

7.1. Environment and Software Setup

The implementation environment integrates Docker containers, Windows Subsystem for Linux (WSL), GPU-enabled inference pipelines, and a standalone Flask-based server architecture. Our experiments were conducted on Windows 11 using WSL 2 with an Ubuntu 22.04.3 LTS distribution and an NVIDIA RTX 4090 GPU for high-throughput inference.

System prerequisites include:

- Operating System: Windows 11 (WSL 2 compatible)
- GPU: NVIDIA RTX 4090 or equivalent (CUDA-enabled)
- Docker Desktop (latest version with WSL2 backend)
- Python: 3.6 for Docker compatibility; 3.11.6 for host-based server

WSL can be installed by executing the following command in PowerShell:

```
wsl --install
```

After installation, ensure Ubuntu 22.04.3 LTS is selected as the default distribution. Docker Desktop should be downloaded from the official Docker website, with WSL integration enabled in the settings. GPU accessibility must be verified by executing:

```
nvidia-smi
```

7.2. Matterport3D Simulator Setup

The simulator enables photorealistic navigation in real-world environments. To begin, clone the project repository:

```
git clone https://github.com/hasanhd555/VLN
cd VLN
```

Navigate to the simulator directory within WSL:

```
cd Matterport3dSim/Matterport3DSimulator
```

Build the Docker container using:

```
docker build -t mattersim:9.2-devel-
    ubuntu18.04 .
```

7.3. Data Acquisition and Preparation

We used the Matterport3D dataset, which is approximately 1.3 TB in size. For selective data download, execute:

```
python download_mp.py -o matterportdata --
    id 17DRP5sb8fy --type
    matterport_skybox_images
```

For the full dataset:

```
python download_mp.py -o matterportdata
```

Extract all downloaded archives using:

```
python extract_zips.py
```

To improve inference efficiency, skybox images are downscaled using:

```
python downsize_skybox.py
```

7.4. Running the Simulator

Ensure that the environment variable for the Matterport data directory is set correctly inside WSL:

```
cd Matterport3dSim/Matterport3DSimulator
export MATTERPORT_DATA_DIR=$(pwd)/
    matterportdata
```

To launch the simulator inside Docker with GPU support:

```
xhost +
docker run -it --gpus all \
  -e DISPLAY \
  -e "QT_X11_NO_MITSHM=1" \
  -v /tmp/.X11-unix:/tmp/.X11-unix \
  --mount type=bind,source=
    $MATTERPORT_DATA_DIR,target=/mnt/c/
    Users/IST/Desktop/matterport/
    Matterport3DSimulator/matterportdata,
    readonly \
  --volume $(pwd):/matterportdata \
  mattersim:9.2-devel-ubuntu18.04
```

Once inside the container, configure the Python environment:

```
cd matterportdata
cp /matterportdata/build/MatterSim.cpython
    -36m-x86_64-linux-gnu.so /usr/local/lib
    /python3.6/dist-packages/
pip3 install requests unidecode
```

Launch the driver script:

```
python3 src/driver/driver.py
```

7.5. Inference Server (Flask) Execution

The inference server runs externally on the host machine and is responsible for processing LLM calls. Begin by activating the Python virtual environment:

```
cd Matterport3dSim/Matterport3DSimulator/
    src/driver
.\venv\Scripts\activate
```

Install dependencies:

```
pip install -r requirements.txt
```

Start the server:

```
python server.py
```

Ensure the server is actively running before triggering simulation episodes so that inference requests are properly handled.

7.6. Evaluation of Results

To evaluate agent performance using standard VLN metrics, prepare the generated trajectory JSON outputs and run:

```
python evaluator.py path_to_agent_results.
    json
```

This will produce a summary including Navigation Error, Success Rate, SPL, and Oracle Success Rate.

7.7. Troubleshooting Common Issues

WSL–Docker integration failures may arise if the NVIDIA Docker toolkit is not installed or the backend is misconfigured. Ensure that WSL 2 is selected as the backend in Docker settings. If the Docker container fails to access the host display, grant permissions using:

```
xhost +
```

7.8. Repository and Ongoing Support

All scripts, Dockerfiles, and configuration templates described here are available at:

GitHub Repository: https://github.com/hasanhd555/VLN

This repository will be maintained with updated instructions, bug fixes, and future enhancements to support ongoing reproducibility and adaptation of this work.

8. Interactive Web Demo: System Architecture and Deployment

To complement our research and enhance accessibility, we developed an interactive frontend-based web application that allows users to visualize and interact with the Vision-and-Language Navigation (VLN) agent presented in this paper. This platform transforms our research into a functional tool, enabling real-time inspection of agent behavior and intuitive exploration of the Matterport3D dataset. The interface is publicly accessible and designed to offer a deeper experiential understanding of the agent's spatial reasoning process.

8.1. System Design and Technology Stack

The frontend is implemented as a single-page application using React, chosen for its modular architecture and robust state management capabilities. To ensure an efficient development workflow and optimal production performance, we adopted Vite as our development server and bundler. Vite's fast hot module replacement (HMR) and minimal build times allowed us to iterate quickly while maintaining a high-performance user experience.

The application's state logic, including viewpoint data, agent paths, and interaction history, is managed using React Context and Hooks. These features provide a clean, maintainable, and component-scoped structure for synchronizing data across the environment viewer and control panels. Tailwind CSS was used to handle styling, supporting a utility-first approach that ensures responsiveness and visual consistency across devices. All code adheres to ES6+ standards and follows best practices in componentization and readability, ensuring that the project is both scalable and accessible for future contributors.

8.2. Purpose and Functional Objectives

This application was developed to bring our research beyond the scope of static paper figures and into an interactive browser-based format. By deploying the navigation system to the web, we enable broader engagement and facilitate inspection of the VLN pipeline by a general audience. Users can now actively explore our dataset, observe how the agent interprets instructions, and examine step-by-step reasoning in realistic 3D indoor environments. This live interface transforms the passive consumption of research findings into an interactive learning experience.

8.3. Key Features and Demonstration Capabilities

The application includes a live navigation demo hosted at https://nav-vision-sproj.vercel.app/viewpoint-demo, which presents an interactive 360° viewer of the Matterport3D scan. Users can pan, rotate, and click on waypoints to simulate movement through the environment. As the user guides the agent, the interface dynamically updates to reflect current positioning, planned trajectories, and local scene understanding, allowing observers to track the agent's reasoning process in real time.

To illustrate more challenging navigation scenarios, the main landing page https://nav-vision-sproj.vercel.app features animated replays of recorded runs. These GIF-based visualizations emphasize complex route execution and are particularly useful for highlighting situations involving spatial ambiguity or non-linear instruction sets.

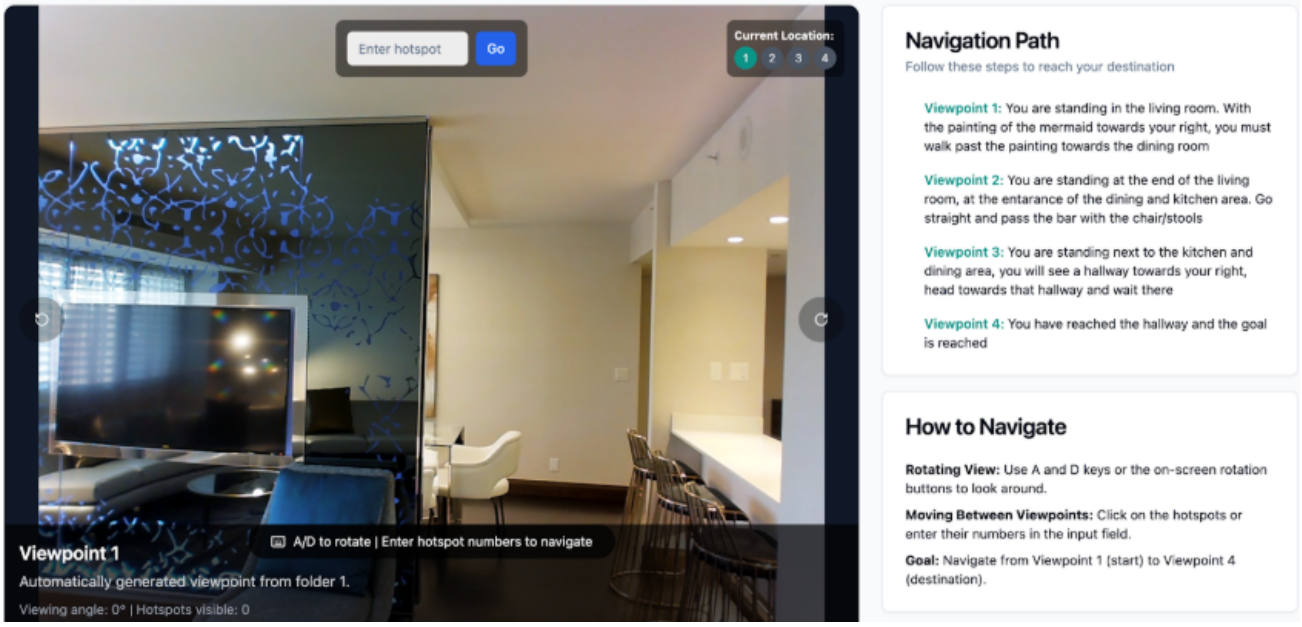

Figure 4. Website Demo

The user interface is fully responsive and has been tested across a range of screen sizes, ensuring seamless usability on desktops, tablets, and mobile devices. This accessibility makes the demo suitable for classroom presentations, academic dissemination, or broader outreach to non-specialist audiences interested in embodied AI systems.

8.4. Live Deployment

The application is freely accessible at `https://nav-vision-sproj.vercel.app`, with all system logic running entirely on the client side. By hosting the system through a modern deployment platform and avoiding backend dependencies, we ensure minimal latency, high availability, and ease of use for researchers and educators aiming to explore or demonstrate VLN technologies.

9. Future Work

This research presents a proof-of-concept pipeline that establishes the feasibility of performing Vision-and-Language Navigation (VLN) in real-world, photorealistic environments using large language models (LLMs) in a zero-shot setting. While our system demonstrates strong performance and interpretability without domain-specific finetuning, several avenues remain open for future exploration and enhancement.

One immediate direction involves improving the graph-based reasoning module. Although our current approach effectively enhances spatial understanding and stability, it can be further augmented through structured memory systems or the integration of graph neural networks. Such enhancements may enable multi-hop reasoning, improved subgoal abstraction, and long-term state retention, particularly in complex or repetitive environments. Additionally, the visual perception component, which currently depends on panoramic scene descriptions, may benefit from modeling temporal consistency across frames and incorporating visual affordance cues to better interpret actions in dynamic or cluttered scenes.

Another promising area of future research lies in transitioning from simulation to physical embodiment. Integrating the pipeline with real-time robotic agents would extend the applicability of our framework to real-world settings such as assistive indoor robotics, smart home systems, or controlled disaster-response environments. This shift would introduce new challenges involving sensor noise, latency, and actuation uncertainty, necessitating further robustness in perception and planning components.

Our methodology also lends itself naturally to extensions in multi-agent collaboration or dialogue-driven navigation. In such contexts, the agent would need to reason about shared plans, interpret and respond to natural language corrections, and adapt its trajectory based on interactive feedback. These capabilities are particularly relevant in human-robot teaming scenarios where situational awareness and cooperation are crucial.

Finally, while we maintain a zero-shot constraint throughout this study, future iterations could investigate the benefits of instruction-tuning or synthetic finetuning using scene-

instruction pairs derived from R2R or similar datasets. This may improve robustness in ambiguous situations while retaining the modularity, generalizability, and explainability of our current framework. Given the flexible architecture of our system, such refinements can be introduced incrementally and evaluated independently without disrupting the interpretability and core design philosophy of our approach.

10. Conclusion

This work introduces a modular, zero-shot Vision-and-Language Navigation (VLN) framework that leverages the reasoning capabilities of large language models (LLMs) in photorealistic indoor environments. Without relying on domain-specific training, our system demonstrates that structured spatial memory, scene-aware prompting, and explicit decision rationales enable effective and interpretable navigation. By integrating panoramic scene descriptions, dynamically expanding graph representations, and reasoning-centric prompting strategies, we address key VLN challenges such as instruction grounding, trajectory planning, and goal recognition.

Our experiments on the R2R benchmark validate the generalization capability of this approach across unseen environments, achieving competitive performance against state-of-the-art baselines—despite using no pretrained visual encoders or finetuned policies. Through rigorous methodology, efficient deployment infrastructure, and transparent reproducibility protocols, we provide a practical and extensible foundation for future research in embodied AI.

This study affirms the potential of LLMs to serve as capable planners in real-world spatial reasoning tasks and motivates further exploration into scalable, training-free navigation systems. By bridging the gap between language understanding and embodied decision-making, our work takes a meaningful step toward more versatile and human-aligned autonomous agents.

References

Anderson, P., Wu, Q., Teney, D., Bruce, J., Johnson, M., Gould, S., and van den Hengel, A. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *CVPR*, 2018a.

Anderson, P. et al. Vision-and-language navigation: Interpreting visually grounded navigation instructions in real environments. In *CVPR*, 2018b.

Chen, C.-Y., Xu, X., Gordon, D., Batra, D., Parikh, D., and Lee, S. History aware multimodal transformer for vision-and-language navigation. In *NeurIPS*, 2021.

Chen, Y.-W. et al. Duet: Unified vision-language pretraining for navigation. In *CVPR*, 2023.

Fried, D., Hu, R., Cirik, V., Rohrbach, A., Andreas, J., Morency, L.-P., Berg-Kirkpatrick, T., Klein, D., and Darrell, T. Speaker-follower models for vision-and-language navigation. In *NeurIPS*, 2018.

Gupta, S. et al. Visual adversarial patches for improving navigation robustness. In *ECCV*, 2022.

Hao, H. et al. Navgpt: A generalist policy for vision-and-language navigation. *arXiv preprint arXiv:2304.14510*, 2023.

Hong, Y. et al. Improving vision-and-language navigation by generating future-view image semantics. In *CVPR*, 2023.

Liu, H. et al. Visual instruction tuning. *arXiv preprint arXiv:2304.08485*, 2023.

Pan, X. et al. Navcot: Boosting llm-based vision-and-language navigation via learning disentangled reasoning. *arXiv preprint arXiv:2312.00730*, 2023.

Qiao, S. et al. Bevbert: Bidirectional and entity-aware vln pretraining with bev. In *NeurIPS*, 2023.

Singh, A. et al. Llm-planner: Few-shot grounded planning with large language models. *arXiv preprint arXiv:2303.06355*, 2023.

Zhang, B., Chen, Y., et al. Scaling data-driven vision-and-language navigation to real-world environments. In *CVPR*, 2023.

Zhou, S. et al. Knowledge-enhanced reasoning for vision-and-language navigation. *arXiv preprint arXiv:2302.05647*, 2023.