ZENITH: Zero-shot Exploration and Navigation in Intelligent Textual Horizons

Devanshi Garag
Department of Computer Science Engineering
d2garg@ucsd.edu

Varun Moparthi
Department of Electrical and Computer Engineering
vmoparthi@ucsd.edu

Yuyuan Wu
Halıcıoğlu Data Science Institute
yuw262@ucsd.edu

Derrick Yao
Halıcıoğlu Data Science Institute
dyao@ucsd.edu

Project and code: https://github.com/Elonian/ZENITH
Video: https://drive.google.com/drive/folders/17bsFniKkR8J3xgNIs3VTr9Mzvsd0pLyn

1 Introduction

Large Language Models (LLMs), such as GPT-4, have demonstrated strong generalization capabilities in reasoning and language tasks. Recent advances in LLMs have also enabled new possibilities in symbolic reasoning and high-level planning [1]. While LLMs have shown promise in static object navigation by reasoning over semantic maps, most existing approaches do not consider dynamic environmental states. In realistic settings, agents must often respond to time-sensitive changes such as traffic lights or movable obstacles. These situations require models to process real-time input and make state-aware decisions.

Dynamic navigation requires agents to continuously reason over both high-level goals and local observations, making long-term planning especially difficult in partially observable or unpredictable environments. One promising approach to address this challenge is waypoint-based planning, where agents generate intermediate sub-goals that serve as local navigation anchors.

In this project, we explore whether LLMs can serve as reasoning modules for waypoint-based navigation by generating intermediate waypoints based on real-time visual information. Rather than symbolic abstractions, we provide the LLM with structured visual observations extracted from raw simulation data and adopt a zero-shot setting where the LLM generates waypoints solely through prompt-based reasoning. Our long-term motivation is to assess the potential of LLM-driven waypoint generation in dynamic navigation scenarios. As a first step toward this goal, we focus on static navigation tasks that isolate the core reasoning capability of LLMs under controlled conditions.

To support these investigations, we build our testbed using SimWorld, a platform based on Unreal Engine 5 (UE5). SimWorld offers high-fidelity 3D environments and rich spatial representations, which are valuable for studying perception-informed decision making and allow us to generate realistic multimodal data streams that simulate first-person navigation scenarios. Through this work,

DSC 291 Machine Learning with Few Labels (Spring 2025).

we aim to provide initial insights into the suitability of LLMs for embodied waypoint planning using structured input and prompt-based reasoning methods, and to assess the practical viability of using UE5-based simulators as platforms for LLM-driven navigation research, laying groundwork for future extensions into dynamic, real-time navigation scenarios.

2 Background and Related Work

Recent studies have explored the use of LLMs in zero-shot navigation tasks. For example, one approach which leverages language-based reasoning for spatially-informed actions is SpatialCoT [2]. It introduces a two-stage method: 1) spatial coordinate bi-directional alignment and 2) chain-of-thought spatial grounding—which explicitly utilizes the advanced reasoning capabilities of Vision-Language Models. It achieves state-of-the-art results in both navigation and manipulation scenarios. For our project, SpatialCoT provides valuable insights into enhancing our agent's trajectory planning capabilities by effectively integrating multimodal data with language-driven reasoning frameworks.

Another approach examines the ability of LLMs to avoid both static and dynamic obstacles during zero-shot path planning[3]. The paper uses a well-structured spatial map, containing information on obstacles, to allow an LLM to generate an overall path representation. Additionally, the framework can provide trajectory adjustments when the navigated agent encounters obstacles, such as other moving entities. Our project builds on this paper by allowing navigation through visual and spatial data, rather than a predetermined map of the environment. By using improved reasoning methods, we can focus on generating instructions using only the visual and spatial data available to the agent at the time.

Open-Nav [4] is a zero-shot Vision-and-Language Navigation (VLN) framework that addresses the limitations of expensive token usage and privacy concerns associated with closed-source LLMs by utilizing locally deployed open-source models. It introduces a spatial-temporal chain-of-thought pipeline that decomposes navigation into three key stages: instruction comprehension, progress estimation, and decision-making. This process is further enhanced through the integration of fine-grained object and spatial knowledge, enabling more accurate and context-aware navigation in continuous environments. In a complementary direction, VLTNet [5] proposes a Vision-Language Tree-of-Thought Network for zero-shot object navigation guided by natural language. The model comprises four core modules: VLM-based scene understanding, semantic mapping, Tree-of-Thought (ToT) reasoning for informed frontier selection, and goal identification. By leveraging multi-path deliberation and backtracking through ToT reasoning, VLTNet enables more globally informed and robust decision-making, particularly in complex, unstructured environments.

Another approach introduced VoroNav Luo et al. (2023)[6], which builds a Reduced Voronoi Graph (RVG) from a 2D semantic map and associates each path with multimodal prompts, including object lists and image captions. GPT-3.5 is used to score each path's relevance to the target object, enabling zero-shot path planning without training. ChatNav [7] proposes a semantic frontier-based planner that prompts an LLM to judge which region in a clustered semantic map is most likely to contain the target object. The reasoning results are used to guide exploration through a gravity-repulsion utility model. Both methods demonstrate the potential of LLMs in symbolic navigation without training, and highlight the importance of prompt design and spatial abstraction.

Some work has been done in the field of navigation in dynamic environments. DynamicRouteGPT[8] is a LLM-driven framework which addresses path planning in a traffic environment. The model initially selects an optimal route with traditional algorithms like Dijkstra. However, as conditions dynamically change, it uses Llama3 to explore causal graphs and use counterfactual reasoning to continually update its routing choice. This shows that LLMs are capable of providing up-to-date navigation in rapidly changing environments. In our project, we plan to implement a similar navigation task, for agents in a simulated environment rather than vehicles on a road network.

3 Methods

3.1 Scene Construction in Unreal Engine 5

3.1.1 Simulated Environment

We use the SimWorld simulation platform to provide a detailed, open-ended environment and map for the agent to traverse. SimWorld is a platform built on Unreal Engine 5 (UE5), which combines high-fidelity physical dynamics with language-driven procedural generation, allowing users to programmatically construct city-scale scenes including buildings, road networks, and roadside assets.

Agents in the SimWorld environment can interact with the world, receiving multimodal observations of the environment, and returning agent actions including navigation and environment interaction. Using this, agents can practice embodied reasoning, learning to navigate to a specified target while avoiding both static and dynamic obstacles.

SimWorld's combination of procedural generation, photorealistic rendering, and dynamic control makes it an ideal testbed for developing and evaluating LLM-driven navigation agents under conditions that approximate complex real-world environments.

3.1.2 Data Capture

The UE5 interface exposes several endpoints that allow real-time access to agent data. These can be categorized into positional data and image-based data.

To allow our navigation agent to recognize its current and target location, we retrieve the agent's current pose and the target's position. Both agent and target position consist of their respective x, y, z coordinates. Additionally, the agent's orientation contains the camera's pitch, yaw, and roll values. Finally, we obtain the camera intrinsic matrix for processing between gathered images and spatial locations.

To provide our agent with its current environmental context, we retrieve an RGB image, depth map, and segmentation mask. The RGB image consists of a standard camera view, showing the true view of what the agent currently can observe. The depth map records per-pixel distances from the camera plane, while the segmentation mask isolates and labels objects within the scene. Our navigation agent uses the RGB image to understand and reason on the high-level environmental context around it, while using the depth map and segmentation mask to further understand and process specific locations.

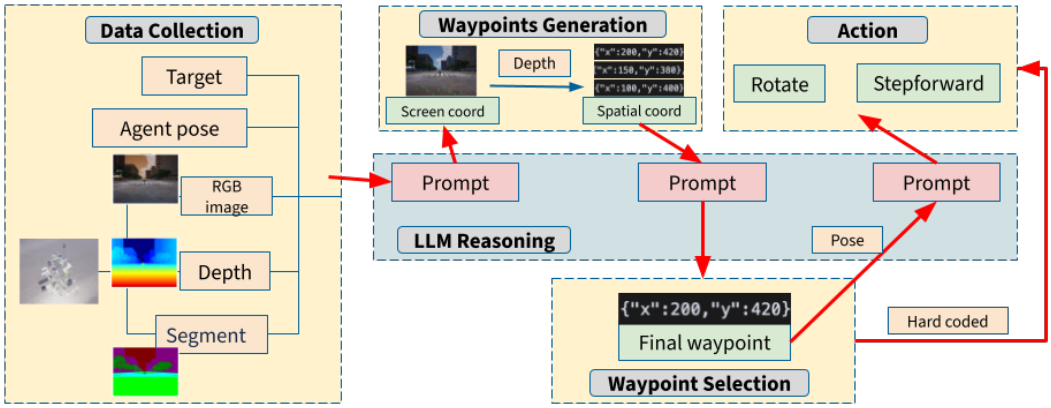


Figure 1: Framework of the ZENITH navigation agent

3.2 LLM-Driven Navigation

3.2.1 Waypoint-Based Navigation

One major obstacle with VLM-based reasoning is converting visual cues the VLM recognizes in the image into numerical values suitable for navigation instructions. Vision models are poor at converting their images into coordinates; while a VLM is able to take coordinates as input and context for an image, it is unable to directly return coordinates (either in-environment or on-screen).

To mitigate this issue, we use a waypoint based system to help the LLM recognize targets with known coordinates in an image. Waypoints are coordinates generated in the field of view of the agent, with known numerical locations. When prompting the VLM during the navigation process, the waypoints are rendered over the RGB image the agent provides. Each waypoint represents a short-horizon local planning target; the LLM determines whether the waypoint is a valid target for the agent to navigate to, and whether the waypoint moves the agent towards its target. By prompting the LLM to select waypoints rather than locations, we can harness the image processing capabilities of VLMs, while also obtaining precise 3D spatial coordinates for the agent to precisely move to.

The waypoints are generated as screen coordinates during each movement step. By using the camera intrinsic matrix, agent position, and depth map, these screen coordinates can be converted into spatial coordinates.

3.2.2 Multi-Step Reasoning

To navigate to the specified goal, the LLM-based navigation system uses an iterative process. During each iteration, the system determines the best step to eventually reach the target. In addition to the information gathered from the current iteration, the LLM is given a short history of its previous moves, to avoid loops and maintain a natural flow of movement.

Each iteration is broken down into four steps: generation, conversion, selection, and execution.

1. In the generation step, the VLM is provided with the RGB image, depth map, and segmentation mask as input. Here, the VLM generates a large batch of 10 to 12 waypoints, as the foundation for the rest of the iteration. In this step, the VLM is not yet given its target; it is prompted to select a widely distributed batch, to provide a variety of options for future steps.
2. In the conversion step, the agent prepares the screen-coordinate waypoints for further processing. First, it converts the screen coordinates created in the generation step into spatial coordinates. Here, the agent uses the depth map to get the (x,y,z) coordinates of each waypoint projected onto the terrain, relative to the camera; then, it uses the camera intrinsic matrix and the agent position to convert the camera coordinates into spatial world coordinates. Additionally, it processes the screen coordinates onto the RGB image, adding markers and labels for VLM processing. The agent stores the spatial coordinates of each waypoint, and passes the waypoint labels, depth, and modified RGB image to the next step.
3. In the selection step, the VLM is provided with the modified RGB image, the waypoint positions and labels, and the distance to each waypoint. Additionally, the VLM gains access to the agent's trajectory, including its past position, current orientation, and its target coordinates. The VLM is instructed to select the best waypoint, given this information, under the following guidelines:
    - Distance to destination
    - Avoiding previously visited areas
    - Clear path without obstacles
    - Natural movement flow
4. In the execution step, the agent is given the waypoint chosen in the selection step, which it retrieves the spatial coordinates for. Given its current pose and waypoint coordinates, the agent directly calculates the straight-line movement to the waypoint, by calculating the rotation to face the waypoint and the stepforward distance to reach it. In our final pipeline we use the hardcoded method, while the LLM-based approaches are only evaluated separately in controlled experiments to analyze their reasoning capabilities.

3.2.3 Pixel to World Coordinate Transformation

The required camera parameters are obtained using the `get_camera_information` method from the `Communicator` class, which interacts with the simulation environment in Unreal Engine to retrieve:

- The camera intrinsics matrix (including focal lengths f_x, f_y, and principal point c_x, c_y)
- The camera position $\mathbf{t}_{\text{cam}}$ in world coordinates
- The camera orientation in Euler angles, later converted to a rotation matrix $\mathbf{R}_{wc}$

The `pixel_to_world` function converts 2D image-plane waypoints into corresponding 3D world coordinates using the depth image and camera parameters. Given the camera intrinsic matrix, camera position, and orientation, the function performs the following steps:

- Extracts camera intrinsic parameters: focal lengths (f_x, f_y) and principal point (c_x, c_y).
- Parses the camera position and rotation (in degrees) and computes the rotation matrix.
- For each waypoint pixel coordinate (x, y), retrieves the depth value from the depth image.
- Projects the 2D pixel location into 3D camera coordinates using the depth and intrinsic parameters.
- Transforms the 3D camera coordinates into world coordinates by applying the camera rotation and translation.
- Returns a list of corresponding 3D points in the world frame.

$$
\begin{aligned}
x_{\text{cam}} &= \frac{(u - c_x) \cdot z}{f_x} \\
y_{\text{cam}} &= \frac{(v - c_y) \cdot z}{f_y} \\
z_{\text{cam}} &= z
\end{aligned}
$$

- Convert the point to the world coordinate frame using:

$$
\mathbf{p}_{\text{world}} = \mathbf{R}_{wc} \cdot \mathbf{p}_{\text{cam}} + \mathbf{t}_{\text{cam}}
$$

where $\mathbf{p}_{\text{cam}} = [x_{\text{cam}}, y_{\text{cam}}, z_{\text{cam}}]^\top$.

This conversion is essential for reasoning about the spatial layout in the environment, enabling navigation decisions in the next step based on real-world positions rather than image pixels.

3.2.4 Waypoint Refinement

In cases where the LLM-predicted pixel-coordinate waypoints fall outside the image boundaries, we apply a simple post-processing step to ensure all waypoints are valid. Let (x, y) be a predicted waypoint and (W, H) be the width and height of the image. The coordinates are clamped as follows:

$$
x' = \min(\max(x, 0), W - 1), \quad y' = \min(\max(y, 0), H - 1)
$$

This refinement ensures that the final set of waypoints lie strictly within the valid image bounds $[0, W - 1] \times [0, H - 1]$, avoiding indexing errors or invalid navigation targets during downstream processing.

However, this limitation does not resolve all problems. The current method sometimes selects waypoints that do not strictly meet the criteria: for example, waypoints that appear floating, overlap with obstacles, or are clustered too closely. As a result, the pruning and selection process needs further refinement to robustly enforce the constraints and improve the quality and reliability of the chosen waypoints.

4 Experiments

4.1 Experimental Setup

We evaluate our LLM-based navigation agent, **ZENITH**, in the **SimWorld** environment, a city-scale, procedurally generated simulation built in Unreal Engine 5. Each test scene contains a unique configuration of buildings, road networks, and dynamic obstacles such as moving vehicles or pedestrians. The agent is assigned a random start and target location in each episode, and after this is required to navigate to the target solely based on its multimodal observations (RGB, depth, and segmentation).

We configure each navigation episode with the following constraints:

- **Success threshold:** Agent is considered successful if it reaches within 50 units of the target.
- **Obstacle collision:** Collisions with dynamic or static objects are checked.

Waypoints are rendered over the RGB image at each iteration. The VLM used in the LLM-driven system is GPT-4o-mini.

4.2 Waypoint Selection Success Rate Evaluation

To evaluate the effectiveness of our waypoint based navigation strategy, we conducted a waypoint optimality analysis across 8 navigation episodes. In each iteration, we recorded the set of candidate waypoints generated by the VLM and the final waypoint selected for execution. We then compared the chosen waypoint's proximity to the final target against all other candidates. A waypoint was deemed optimal if it was closest to the goal and led to a valid, obstacle free path.

We define the waypoint selection success rate as the average proportion of correctly selected waypoints within each experiment. Formally:

$$SR_{\text{waypoint}} = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{1}{M_i} \sum_{j=1}^{M_i} \mathbb{I}(\text{success}_{i,j}) \right)$$

Where N is the total number of experiments, M_i is the number of steps in trial i, and $\mathbb{I}(\text{success}_{i,j})$ is an indicator function that equals 1 if the waypoint selection at step j of trial i was successful, and 0 otherwise.

4.3 Goal Success Rate Evaluation

We evaluate the overall effectiveness of the ZENITH navigation agent by measuring its success rate in reaching the assigned final target across a diverse set of simulated environments. In each episode, the agent is initialized at a fixed starting location within a procedurally generated scene and is tasked with navigating to a specific goal location using only its visual inputs (RGB, depth, and segmentation) and internal reasoning.

An episode is marked as successful if the agent reaches within 50 units of the goal without becoming stuck or entering a collision loop.

The goal success rate is defined as the proportion of trials where the agent successfully reaches the final destination:

$$SR_{\text{goal}} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\left(\text{agent reaches goal}_i\right)$$

Where $\mathbb{I}(\cdot)$ equals 1 if the agent successfully reaches the goal within the allowed conditions, and 0 otherwise. N is the total number of experiments.

We achieved a good success rate with dynamic waypoint selection. These results demonstrate the robustness of our waypoint based reasoning framework and its ability to guide the agent effectively through complex environments. The agent's success rate reflects not only its ability to understand spatial relationships but also its capacity to iteratively refine its trajectory using past context and obstacle awareness.

4.4 Local Execution Strategies Comparison

To further evaluate the impact of different reasoning strategies on movement execution, we design an additional standalone experiment outside the full navigation pipeline. In this experiment, a set of target waypoints are pre-selected and provided to the agent. Given the selected waypoint, the agent is asked to navigate from its current position to the waypoint under three different control strategies: (i) hardcoded movement controller, (ii) LLM reasoning with standard prompting, and (iii) LLM reasoning with Chain-of-Thought prompting. This experiment isolates the local navigation capability under each strategy without involving the upstream waypoint generation and selection modules.

5 Results

5.1 Unreal Engine Views

RGB Image (Agent View): This view represents the visual input captured from the perspective of the autonomous agent navigating the environment. The agent perceives a first-person street-level scene with urban elements such as buildings, crosswalks, trees, and lighting, replicating real-world conditions for navigation and perception tasks.

World View (Top-Down): This bird's-eye view shows the global layout of the environment created for the navigation purpose, including the full city block with multiple buildings and road networks. This perspective is useful for debugging, monitoring agent behavior, and evaluating spatial coverage during navigation or exploration experiments.

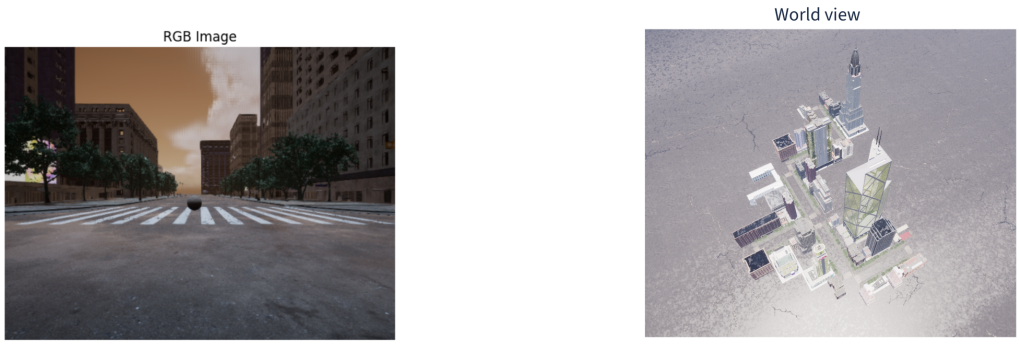


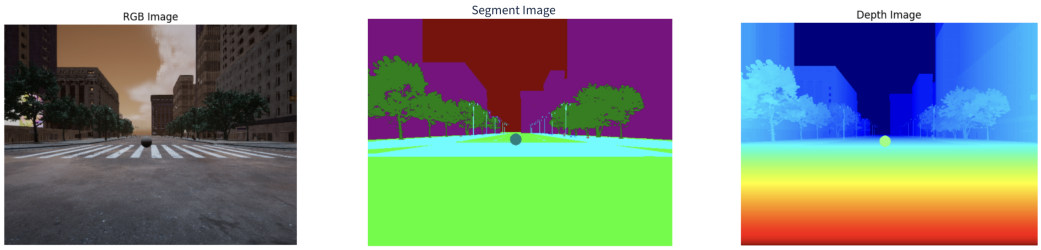
Figure 2: Unreal Views.

5.2 Multimodal Input and Waypoint Generation Visualization

Our pipeline successfully captures real-time first-person visual observations from the agent at each navigation step, including the RGB image (agent view), depth map, and semantic segmentation mask. These multimodal observations provide structured scene information to support the subsequent reasoning modules, including waypoint generation, waypoint selection, and coordinate mapping.

Based on the generated waypoint outputs, we observe that the LLM is able to propose valid candidate waypoints across diverse navigation scenarios. When manually inspecting the generated waypoints against the environmental context, we notice that in some cases, some waypoints are not generated in traversable areas, and may overlap regions marked as obstacles or float off the ground. However, in all cases, a majority of waypoints are placed on valid, traversable areas.

The generated waypoints exhibit good spatial coverage, often spanning a broad range of positions surrounding the agent. The waypoints appear near the bottom of the image and close to the agent, fulfilling their purpose as short-distance goals rather than long-distance commitments. This diversity in candidate waypoints provides sufficient options for the downstream waypoint selection module to determine the most appropriate intermediate goal toward the final target.



(a) RGB Image (b) Segment Map (c) Depth Map

Figure 3: Agent view: RGB image, segmentation map, and depth map.

(a) Depth map for Step 1 (b) Depth map for Step 3

Figure 4: Way points projected on the image.

5.3 Evaluation of Waypoint and Goal Success Rate

We conduct 8 independent navigation expeiments to evaluate the performance of LLM-based waypoint selection and final goal-reaching ability. As shown in Table 1, the average waypoint selection success rate reaches 71.04%, indicating that the LLM is generally effective in reasoning over real-time scene inputs to select appropriate intermediate waypoints. This demonstrates the LLM's ability to interpret structured visual information and make plausible local navigation decisions.

Table 1: Success rates of waypoint selection and final goal-reaching

Model	**Waypoint SR**	**Goal SR**
GPT-4o mini	71.04%	12.50%

However, the overall goal-reaching success rate is relatively low at 12.50%. Failure cases are attributed to two main causes: (1) timeout and (2) divergence from the target, where the agent's predicted next waypoint leads to a position farther from the goal than its current location. This is especially apparent when the agent is facing away from the destination, and all generated waypoints increase the agent's distance. Our analysis reveals that all failures in our experiments fall into the latter category, indicating that the agent eventually drifts away from the target rather than remaining stuck or timing out directly.

This issue may stem from limitations in the current LLM-based reasoning framework. Because the model operates solely based on first-person visual observations, its ability to plan is constrained when the final target lies beyond the agent's limited field of view. In such cases, the LLM may generate waypoints that are locally reasonable but globally suboptimal, ultimately leading to compounding errors as navigation progresses. Additionally, in some episodes, the LLM selects suboptimal waypoints from the provided candidate list, which can further increase deviation from the goal.

Despite these limitations, our results provide valuable insights into the feasibility of using LLMs for real-time visual navigation. The model demonstrates strong waypoint reasoning ability under partial observability, highlighting its potential as a zero-shot planner without explicit model fine-tuning. Further improvements may involve richer scene memory, history tracking, or integrating global guidance signals to address long-horizon planning challenges.

5.4 Local Execution Comparison (Ablation)

In addition to the end-to-end pipeline evaluation, we qualitatively compare different control strategies for local movement execution given pre-selected waypoints. The three strategies include: (i) hardcoded controller, (ii) LLM reasoning with standard prompting, and (iii) LLM reasoning with Chain-of-Thought (CoT) prompting.

Overall, the hardcoded controller performs most efficiently and accurately. It directly decomposes movement into two deterministic steps: rotating toward the target direction, followed by move-forward movement for a pre-computed distance, resulting in consistently short and precise navigation trajectories.

The standard prompting approach shows success on simple navigation tasks, such as moving from (0,0) to (0,100), where little or no rotation is required. However, when navigating toward targets that involve non-trivial angles, the LLM often struggles to accurately infer the correct rotation angle. In many cases, this leads to the agent repeatedly rotating in place without making forward progress.

Chain-of-Thought prompting demonstrates improved reasoning over standard prompting. In several cases, it successfully navigates toward targets that require both rotation and forward movement. Nevertheless, CoT-generated trajectories tend to be less direct compared to the hardcoded policy. For example, when facing a target located approximately 45 degrees to the left, CoT may first issue a forward movement command, then rotate 90 degrees, and finally move forward again, rather than directly rotating by 45 degrees and proceeding forward. This behavior may be influenced by limitations in how available movement functions are explained to the LLM during prompting.

While these qualitative results show that LLM reasoning can partially handle local movement decisions, especially with CoT prompting, the hardcoded controller remains more reliable for precise low-level control. These findings highlight the gap between LLM high-level reasoning and fine-grained control, suggesting that future improvements may require better prompt engineering or hierarchical planning structures.

6 Conclusion

In this project, we investigate the feasibility of using LLMs for waypoint-based navigation within physically grounded simulation environments. Rather than relying on precomputed full maps, we explore whether LLMs can infer navigation paths based on structured scene input and general reasoning capabilities. Besides, waypoint-based planning may also serve as a flexible and scalable paradigm for dynamic navigation tasks, where agents continuously update their local plans in response to changing environmental states.

We successfully implement goal-oriented VLM prediction, prompting the model to select the best waypoint to reach the desired destination. Our VLM agent uses varied, multimodal data, including various image views of its surroundings and numerical data on its movement, to reason based on context, past actions, and future goal.

Additionally, we implement a closed-loop control system where the agent automatically interfaces with the LLM, providing the LLM with the desired context and carrying out the instructions it provides. In this way, we allow the agent to move autonomously towards its specified goal.

6.1 Limitations

One limitation we observed is that the agent struggles when it is no longer facing the destination. This can happen if the agent moves at an angle or overshoots the goal and ends up oriented away from it. In such cases, the agent is limited to generating waypoints within its current field of view and cannot propose waypoints that redirect it back toward the destination. This issue is especially

noticeable when the agent overshoots the goal, as it continues selecting forward-facing waypoints that further increase the distance to the target.

In the current approach, we terminate navigation if the next selected waypoint results in an increased distance to the destination.

To improve this behavior, future work could involve providing the LLM with additional context about the agent's orientation relative to the destination. This may include dynamically modifying the prompt to include heading information and instructing the LLM to suggest waypoints that help realign the agent toward the goal. Additionally, slowing down the agent near the destination and generating denser, finer-grained waypoints could help prevent overshooting and improve final accuracy.

Another area for improvement is the evaluation of optimal waypoint selection. We evaluate whether a waypoint is optimal depending on whether it minimizes the distance to the destination. However, in complex navigation tasks, this may not always apply; a waypoint that does not minimize distance may be optimal so that the agent can avoid obstacles. In the future, we can use more advanced pathfinding algorithms to establish a more accurate ground truth of optimality, incentivizing waypoints that follow the shortest path rather than the straight-line distance.

6.2 Future Work

In the future, we can update the environment and provide the agent with more complex tasks to carry out.

In this project, we implement navigation within a wholly static environment with no moving parts. This does not generalize to embodied navigation in the real world, where an agent must take into account dynamic obstacles such as pedestrians and vehicles. Additionally, our agent makes decisions based on the physical context, focusing on physical obstacles. In the future, we can leverage the capabilities of SimWorld to add non-physical guidelines, such as constraints like traffic lights or preferences like walking on sidewalks.

Additionally, our agent navigates through an ideal environment, with perfectly traversable surfaces and flat terrain. By going beyond the SimWorld engine, we could introduce terrain such as rough dirt or stairs, which lie between the two extremes of paved road and nontraversable obstacles.

Finally, we make some efforts for the agent to move in a natural way through LLM prompting. In the scope of our project, this is limited to providing the agent with its movement history and instructing it to continue from its previous path. In the future, we could expand this to include much more human-like behavior. This coincides with the other future goals above. For example, we could instruct the agent to slow down in rough terrain, which an embodied agent would do to keep its balance; we could also instruct the agent to prefer walking on "human" spaces like sidewalks, teaching societal rules to the agent.

References

[1] Yubo Zhao, Qi Wu, Yifan Wang, Yu-Wing Tai, and Chi-Keung Tang. Dynamic path navigation for motion agents with llm reasoning. *arXiv preprint arXiv:2503.07323*, 2025.

[2] Yuecheng Liu, Dafeng Chi, Shiguang Wu, Zhanguang Zhang, Yaochen Hu, Lingfeng Zhang, Yingxue Zhang, Shuang Wu, Tongtong Cao, Guowei Huang, Helong Huang, Guangjian Tian, Weichao Qiu, Xingyue Quan, Jianye Hao, and Yuzheng Zhuang. Spatialcot: Advancing spatial reasoning through coordinate alignment and chain-of-thought for embodied task planning. *CoRR*, abs/2501.10074, 2025.

[3] Yubo Zhao, Qi Wu, Yifan Wang, Yu-Wing Tai, and Chi-Keung Tang. Dynamic path navigation for motion agents with llm reasoning, 2025.

[4] Yanyuan Qiao, Wenqi Lyu, Hui Wang, Zixu Wang, Zerui Li, Yuan Zhang, Mingkui Tan, and Qi Wu. Open-nav: Exploring zero-shot vision-and-language navigation in continuous environment with open-source llms. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2025.

[5] Congcong Wen, Yisiyuan Huang, Hao Huang, Yanjia Huang, Shuaihang Yuan, Yu Hao, Hui Lin, Yu-Shen Liu, and Yi Fang. Zero-shot object navigation with vision-language models reasoning.

In *Proceedings of the 27th International Conference on Pattern Recognition (ICPR 2024)*, pages 389–404. Springer, 2025.

[6] Pengying Wu, Yao Mu, Bingxian Wu, Yi Hou, Ji Ma, Shanghang Zhang, and Chang Liu. Voronav: Voronoi-based zero-shot object navigation with large language model. In *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, pages 53757–53775. PMLR, 2024.

[7] Yong Zhu, Zhenyu Wen, Xiong Li, Xiufang Shi, Xiang Wu, Hui Dong, and Jiming Chen. Chatnav: Leveraging llm to zero-shot semantic reasoning in object navigation. *IEEE Transactions on Circuits and Systems for Video Technology*, PP(99):1–1, 2024.

[8] Ziai Zhou, Bin Zhou, and Hao Liu. Dynamicroutegpt: A real-time multi-vehicle dynamic navigation framework based on large language models, 2024.

7 Appendix

7.1 Generation, Refinement and Selection

7.1.1 Waypoint generation

We generate 10–12 candidate waypoints by providing the language model with the RGB image, depth map, and segmented scene image, along with the agent's current position, destination, and a list of world-coordinate waypoints. The model selects waypoints that reduce the distance to the goal, avoid obstacles, and maintain natural scene flow.

Prompt

You are given an RGB image, a depth map, and a segmentation mask, all captured from an agent navigating a simulated urban environment. Your task is to generate navigable waypoints based on visual, depth, and segmentation cues. These waypoints represent potential positions on the ground plane that the agent can safely travel to next.
Each example contains: - An RGB image showing the agent's forward-facing view. - A depth map providing per-pixel distance information in the same view. - A segmentation mask where ground pixels are strictly colored green ([0, 255, 0]), and all obstacles, objects, buildings, and pedestrians are colored red ([255, 0, 0]). - A list of waypoints: these are image-plane coordinates (x, y) corresponding to navigable ground locations.
Important constraints and guidelines for generating waypoints: - Waypoints must lie only on green pixels in the segmentation mask (navigable ground). - Never place waypoints on or near red pixels (obstacles, objects, pedestrians, or buildings). - Use the depth map to ensure waypoints are located on flat, continuous ground, and within a safe navigable distance. - Avoid placing waypoints on depth discontinuities, steep gradients, or near vertical surfaces. - **Avoid clustering**: Waypoints must be **widely distributed** across different regions of the navigable ground, covering left, center, and right fields of view. - **Do not follow a grid, diagonal, or symmetric pattern. Spread the points naturally and unevenly across the field of view.** - **Ensure all waypoints are at least 60 pixels away** from the bottom center of the image (representing the agent's current position). - Prioritize waypoints that extend forward in the direction of movement, covering diverse reachable zones. - Place waypoints in open, unobstructed regions that offer forward progress.
Output Format Requirements: - Output exactly 10 to 12 waypoints per image. - Each waypoint must be in the format: (x, y) - x is the horizontal (width) pixel coordinate. - y is the vertical (height) pixel coordinate. - The pixel origin (0, 0) is at the top-left corner of the image: - x increases left to right. - y increases top to bottom. - Do not return any text or explanation—only a JSON object in this format: "waypoints": ["x": X1, "y": Y1, ..., "x": Xn, "y": Yn]

7.2 Waypoint refinement prompt

The waypoint refinement prompt takes the initial 10–12 candidate points generated previously and refines the selection down to at most 4 prime waypoints. It ensures that the chosen waypoints are

well-distributed, obstacle-free, and aligned with the intended path. This step helps focus the agent's navigation decisions on the most promising directions while maintaining spatial diversity and progress toward the destination.

Although we implemented the waypoint refinement step early in the development of ZENITH, we do not use it in the version presented here, as it is somewhat redundant with the waypoint selection step while increasing the token cost of each iteration by an extra one half. However, because the waypoint generation step generates some nontraversable and clustered waypoints, implementing the refinement step may slightly improve performance across the entire navigation process.

Prompt

You are given:
1. An **RGB image** showing the agent's forward-facing view.
- This image is overlaid with 30-35 waypoints, as colored dots.
2. A **list of waypoints** showing potential areas the agent might move to next.
- These waypoints are ‘(x,y)‘ image-plane coordinates. Each waypoint contains three fields:
- x: position from left of image
- y: position from top of image
- color: color of dot overlaid in RGB image

Selection Criteria
- Waypoints must be **on the ground**; they must not float in the air or lie on top of walls, pedestrians, vehicles, or obstacles.
- Avoid pedestrians or dynamic objects by identifying regions with non-ground depth patterns.
- Favor waypoints that follow established paths, such as roads and sidewalks.
- Favor waypoints that are farther from the agent, or located at key points like intersections.
- Favor a selection of waypoints that spread out in different directions. If possible, avoid selecting waypoints that are clustered together.
- Do not select waypoints that are floating in the air, or lie on top of walls, obstacles, vehicles, or pedestrians.
- Only select waypoints that are on the ground.

Output Format
Output a short reasoning, and then a list of four waypoints ‘(x,y)‘ in JSON format. For example:
{
"reasoning": "<detailed reasoning here>",
"waypoints": [
"x": 360, "y": 220,
"x": 380, "y": 230,
"x": 455, "y": 90,
"x": 720, "y": 280
]
}

7.3 Waypoint selection

We evaluate the large language model's ability to select the best next waypoint for navigation by providing it with important scene information and navigation data. The input includes a segmented view of the scene, the agent's current and past positions, the destination, and a list of possible waypoints with their distances to the agent and the destination.

The model is asked to choose the waypoint that most effectively reduces the remaining distance to the destination while avoiding obstacles and previously visited areas, and following a natural path through the environment.

Prompt

Scene Analysis: {scene_analysis}
Current Position (world coordinates): {current_pos}
Destination (world coordinates): {destination}

Distance from Current Position to Each Waypoint (in world coordinates): {distances_from_current}
Distance from Each Waypoint to Destination (in world coordinates): {distances_to_destination}

Available Waypoints are in world coordinates: {waypoint_text}

Objective:
Choose the next waypoint that **effectively reduces the total distance from the new position to the destination meaning choose the waypoint which has lowest distance between waypoint coordinates and destination coordinates**, based on distances provided. Additionally, ensure the selected waypoint:
1. Avoids previously visited or nearby areas (based on history)
2. Has a clear path without obstacles
3. Aligns with natural and logical movement flow in the scene

Return only the waypoint label (e.g., A, B, C), **surrounded by double asterisks**. For example, return **B**.
Don't return any additional text or explanations, just the waypoint label.