A^{2}Nav : Action-Aware Zero-Shot Robot Navigation Using Vision-Language Ability of Foundation Models

Anonymous Author(s)
Affiliation
Address
email

Abstract

We tackle the challenging task of zero-shot vision-and-language navigation (ZS-VLN), where an agent learns to follow complex path instructions without annotated data. We introduce A^{2}Nav , an action-aware ZS-VLN method leveraging foundation models like GPT and CLIP. Our approach includes an instruction parser and an action-aware navigation policy. The parser breaks down complex instructions into action-aware sub-tasks, which are executed using the learned action-specific navigation policy. Extensive experiments show A^{2}Nav achieves promising ZS-VLN performance and even surpasses some supervised learning methods on R2R-Habitat and RxR-Habitat datasets.

1 Introduction

In vision-and-language navigation (VLN) tasks, an agent is required to navigate in a novel environment according to language navigation instructions. Current dominant methods (6; 14; 21; 11) attempt to learn VLN ability in a supervised learning manner. However, creating high-quality labeled data requires a significant amount of human effort, which can be time-consuming and expensive. Additionally, the labeled data may not cover all possible scenarios, making it challenging for the model to generalize to new, unseen environments. To address these challenges, exploiting the knowledge from large foundation models (3; 18; 8) for learning navigation ability without requiring downstream task annotated data is a potential solution. We call it **zero-shot navigation ability**.

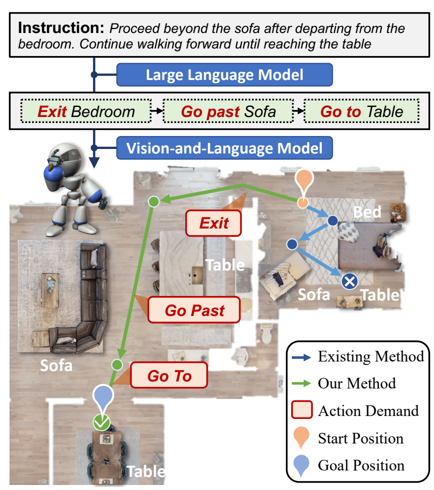

Figure 1: Existing zero-shot VLN overlooks the action demands.

Recently, researchers have made some attempts (10; 16; 1; 25) at solving object navigation tasks in a zero-shot manner. They use a foundation vision-and-language model (VLM) (18) to localize the object (10) or use it to encode the object goal features (16), enabling the agent to navigate to any object goal described by natural language. Although some progress has been made, existing methods fail to take into account the varied action demands (*e.g.*, "*proceed beyond*", "*depart from*") in navigation instructions. This may lead the agent to the

Submitted to NeurIPS 2023 6th Robot Learning Workshop: Pretraining, Fine-Tuning, and Generalization with Large Scale Models. Do not distribute.

wrong destination. For the example in Figure 1, the agent is expected to "*exist the bedroom*", but the existing methods only take the landmark "*bedroom*" into consideration. The agent mistakenly goes into the bedroom, which is in the opposite direction of the path described by the instruction.

To solve this problem, the agent must correctly figure out the expected action demand associated with each landmark and accurately execute them. In this paper, we propose an action-aware navigation method, named A^2Nav , for the zero-shot VLN task. Our method consists of two components: an instruction parser empowered by LLMs for figuring out landmarks and associated action demands; and an action-aware navigation policy empowered by CLIP for executing these action demands sequentially for navigation. Extensive experiments demonstrate that our A^2Nav achieves promising performance on zero-shot VLN task, getting 22.6% and 16.8% success rates on R2R-Habitat and RxR-Habitat, respectively.

Our main contributions are as follows: 1) Instead of treating vision-and-language navigation as a sequence of object navigation tasks, we take into account the instruction action demands and decompose the instruction into a sequence of action-specific object navigation sub-tasks, where the agent is expected to not only localize the landmarks but also navigate to different goal position according to the associated action demand. 2) To address the problem that existing zero-shot navigators cannot satisfy different action demands, we identify and summarize five fundamental action demands and learn a unique navigator for executing each one without requiring manual path-instruction annotation, leading to more accurate and explainable navigation results.

2 Related Works

Zero-Shot Object Navigation. Since the navigation instruction is often described by several landmarks, it can be decomposed into sequential object navigation tasks. The object navigation task has been explored by previous literature (10; 16; 17; 5; 19; 1; 4; 24). Among these methods, we notice that some trails that design the object navigation agent in a zero-shot manner show great potential. Gadre *et al.* (10) design a heuristic algorithm to navigate to an object using the open-world object recognition ability of the foundation vision-and-language model (*i.e.*, CLIP (18)). Some works like ZER (1) and ZSON (16) learn an image navigation agent first, and then map the image goal representation into object text goal embedding space, and thus transfer to the object navigation task.

Zero-Shot Vision-and-Language Navigation. Based on the previous success on VLN and zero-shot object navigation, we aim to tackle the VLN task in the zero-shot manner, releasing the agent from expensive manual-labeled path-instruction training data. This problem has not been fully exploited yet. Pioneering works (22; 9; 7) have already verified the effectiveness of foundation models (LLM (3) and VLM (18)) in this scenario. These methods leverage GPT-3 (3) to extract navigation landmarks from the instruction and then initialize a heuristic object navigator using CLIP (18) to find out the landmark from visual observation and to navigate to the front of the landmark. Concurrent work (26) leverages a GPT model for inferring navigation actions on a discrete navigation graph. The performance in continuous environments has not been well explored. Our proposed A^2Nav solved these issues using a learnable action-aware object navigator.

3 Action-Aware Zero-Shot VLN

We consider a practical but challenging problem zero-shot VLN, where the agent is expected to complete the VLN task without requiring path-instruction annotation. we leverage a large language model as an instruction parser for parsing all landmarks and their associated action demands. The instruction is then decomposed into a sequence of action-specific object navigation sub-tasks, in which the agent is required to localize the landmark and navigate based on the specific action demands associated with that landmark. For executing each sub-tasks sequentially, an action-aware navigation policy comprising five action-specific navigators is learned in a zero-shot manner. The general scheme is shown in Figure 3.

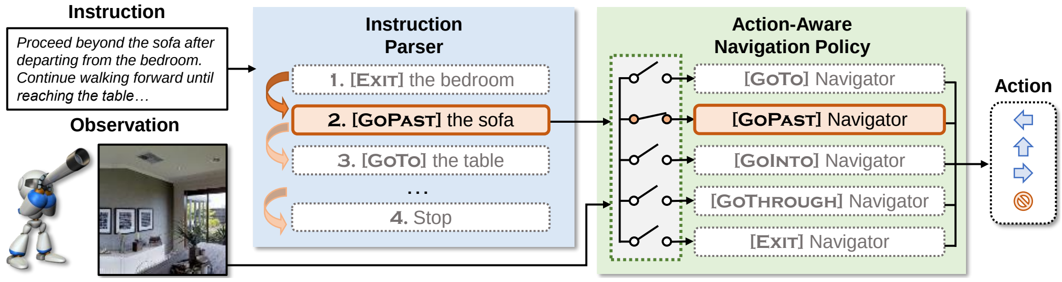

Figure 3: General scheme of A^{2}Nav for zero-shot VLN task.

3.1 Instruction Parser

Action-Specific Object Navigation Sub-Task. The instruction parser aims to transfer a complicated instruction into several sequential executable action-specific object navigation sub-tasks. Each sub-task contains a landmark and an associated action demand, such as "*departing from the bedroom*". The sub-task can be represented by a template "(ACTION, LANDMARK)". We explicitly summarize 5 basic sub-tasks shown in Figure 2, including "(GOTO, OBJECT)", "(GOPAST, OBJECT)", and "(GOINTO, REGION)", "(GOTHROUGH, REGION)", "(EXIT, REGION)".

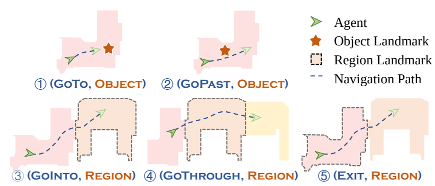

Figure 2: Illustration of sub-task types.

Decomposing Instruction into Sub-Tasks. We use the few-shot learning ability of GPT-3 LLM (3) for decomposing an instruction into a sequence of sub-tasks described above. The prompt contains several correct instruction parsing examples. As the predicted sub-tasks from GPT-3 are in the free-form language, we need to map each prediction to the predefined sub-tasks. In most cases, the "ACTION" predictions made by GPT-3 accurately match one of the "ACTION" in predefined sub-tasks, and thus we can directly map it to this sub-task type. In cases where a prediction does not match any, we follow (13) to perform mapping through semantic translation. Specifically, we use BERT (8) to encode the predicted "ACTION" and the "ACTION" in all predefined sub-tasks. Then we compute the cosine similarity between them and consider the predefined sub-task with the highest score as the predicted sub-task.

3.2 Action-Aware Navigation Policy

With the sub-task sequence parsed by LLMs, we learn an action-aware navigation policy to execute them sequentially. The policy consists of five action-specific navigators, each of which is responsible for a specific sub-task type. We follow ZSON (16) to transform the question of learning such a navigator into learning an image-goal navigator on a freely collected action-specific image-path dataset. Each sample in this dataset contains an image sampled from the environment and a navigation path that is compatible with the action demands. Specifically, for training GOPAST navigator that expects the agent to go to the object and keep going forward past the object, we capture the goal image in the middle of the path. For the **GOINTO** action demand that expects the agent to go cross a doorway into the target region, we sample the path that crosses over two regions and sample the goal image at the end of this path. For the **GOTHROUGH** action demand that expects the agent to go from one side to the other side of a region, we randomly sample the path that starts near one entrance and ends near the other one of a region. The goal image is captured in the middle of the path. For the **EXIT** action demand, the path is sampled in the same way as the GOINTO action demand, while the goal image is captured at the beginning of the path. We fine-tune the trained ZSON model on these datasets using the same learning pipeline as ZSON, which is shown in Figure 4. For the **GOTO** action demands, we directly utilize a trained ZSON model as a navigator.

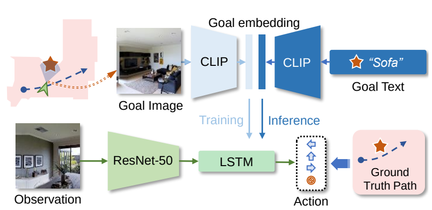

Figure 4: Paradigm of navigators.

	Method	Extra Info.	R2R-Habitat		RxR-Habitat		CSR
			SR	SPL	SR	SPL	
Supervised	Seq2Seq (14)	Depth	25.0%	22.0%	-	-	-
	LAW (21)	Depth	35.0%	31.0%	10.0%	9.0%	28.6%
	WS-MGMap (6)	Depth	38.9%	34.3%	15.0%	12.1%	38.6%
Zero-Shot	Random	-	0.0%	0.0%	6.0%	6.0%	0.0%
	CLIP-Nav (9)	Panoramic	5.6%	2.9%	9.8%	3.2%	57.4%
	Seq CLIP-Nav (9)	Panoramic	7.1%	3.7%	9.1%	3.3%	77.8%
	Cow (10)	Depth	7.8%	5.8%	7.9%	6.1%	**98.3%**
	ZSON (16)	-	19.3%	9.3%	14.2%	4.8%	73.6%
	A^{2}Nav (Ours)	-	**22.6%**	**11.1%**	**16.8%**	**6.3%**	74.3%

Table 1: Comparisons with zero-shot and supervised methods on VLN datasets.

4 Experiments

We compare our A^{2}Nav with existing zero-shot and supervised-learning navigation methods on R2R-Habitat and RxR-Habitat datasets. We introduce the experimental setup, agent configurations, and baselines in Appendix.

Comparisons with Zero-Shot Methods. In Table 1, our A^{2}Nav outperforms other zero-shot methods. On R2R-Habitat, it surpasses CLIP-Nav, Seq CLIP-Nav, CoW, and ZSON by 17.0%, 15.5%, 14.8%, and 3.3% in success rate, respectively. On RxR-Habitat, it outperforms them by 7.0%, 7.7%, 8.9%, and 2.6%, respectively.

Comparisons with Supervised Learning Methods. We compare our zero-shot A^{2}Nav with three supervised VLN methods: vanilla Seq2Seq (15), LAW (21), and WS-MGMap (6). In Table 1, our zero-shot A^{2}Nav achieves comparable performance compared with the vanilla Seq2Seq on R2R-Habitat and outperforms all supervised learning methods on RxR-Habitat, indicating that A^{2}Nav is more effective at generalizing to different datasets and can adapt more easily to varying environments. We also compare A^{2}Nav with supervised methods trained on limited data. In Figure 5, A^{2}Nav outperforms the SOTA method (*i.e.*, WS-MGMap) if less than 50% training episodes are available for it.

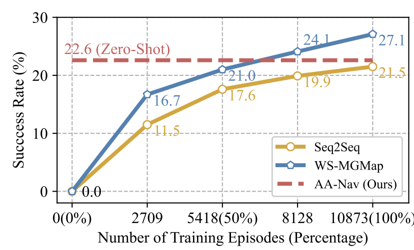

Figure 5: Comparison with the supervised learning methods that are trained on partial training data.

Effectiveness of Action-Aware Navigation Policy. To verify the effectiveness of each navigator, we create multiple navigation policy variants that progressively include 5 navigators in an order of GoTo, GoPast, GoInto, GoThrough, and Exit. By default, the sub-task is executed by the GoTo navigator if its corresponding navigator is not included. In R2R-Habitat, the SR of policy with 1$\sim$5 navigators are 19.3%, 20.9%, 21.5%, 22.3%, and 22.6%, respectively, demonstrating the importance of each navigator.

5 Conclusion

In this paper, we take into account the instruction action demands and decompose the VLN task into a sequence of action-specific object navigation sub-tasks. To execute these sub-tasks, we further propose an action-aware navigation policy that learns different navigation abilities without requiring any manual path-instruction annotation. The proposed A^{2}Nav achieves the best zero-shot VLN performance on two benchmark datasets (*i.e.*, R2R-Habitat and RxR-Habitat) and outperforms the state-of-the-art supervised learning methods on RxR-Habitat. Furthermore, our A^{2}Nav is able to more accurately follow navigation instructions that contain specific action demands, demonstrating its potential for the scenario that needs human-robot communication and interaction.

References

[1] Ziad Al-Halah, Santhosh K. Ramakrishnan, and Kristen Grauman. Zero experience required: Plug & play modular transfer learning for semantic visual navigation. In *CVPR*, pages 17010–17020, 2022.

[2] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian D. Reid, Stephen Gould, and Anton van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *CVPR*, pages 3674–3683, 2018.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *NeurIPS*, 2020.

[4] Devendra Singh Chaplot, Dhiraj Gandhi, Abhinav Gupta, and Ruslan Salakhutdinov. Object goal navigation using goal-oriented semantic exploration. In *NeurIPS*, 2020.

[5] Peihao Chen, Dongyu Ji, Kunyang Lin, Weiwen Hu, Wenbing Huang, Thomas H. Li, Mingkui Tan, and Chuang Gan. Learning active camera for multi-object navigation. *NeurIPS*, 2022.

[6] Peihao Chen, Dongyu Ji, Kunyang Lin, Runhao Zeng, Thomas H. Li, Mingkui Tan, and Chuang Gan. Weakly-supervised multi-granularity map learning for vision-and-language navigation. In *NeurIPS*, 2022.

[7] Zhenfang Chen, Qinhong Zhou, Yikang Shen, Yining Hong, Hao Zhang, and Chuang Gan. See, think, confirm: Interactive prompting between vision and language models for knowledge-based visual reasoning. *arXiv*, 2023.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pages 4171–4186, 2019.

[9] Vishnu Sashank Dorbala, Gunnar A. Sigurdsson, Robinson Piramuthu, Jesse Thomason, and Gaurav S. Sukhatme. Clip-nav: Using CLIP for zero-shot vision-and-language navigation. *CoRR*, abs/2211.16649, 2022.

[10] Samir Yitzhak Gadre, Mitchell Wortsman, Gabriel Ilharco, Ludwig Schmidt, and Shuran Song. CLIP on wheels: Zero-shot object navigation as object localization and exploration. *CoRR*, abs/2203.10421, 2022.

[11] Georgios Georgakis, Karl Schmeckpeper, Karan Wanchoo, Soham Dan, Eleni Miltsakaki, Dan Roth, and Kostas Daniilidis. Cross-modal map learning for vision and language navigation. In *CVPR*, pages 15439–15449, 2022.

[12] Yicong Hong, Cristian Rodriguez Opazo, Qi Wu, and Stephen Gould. Sub-instruction aware vision-and-language navigation. In *EMNLP*, pages 3360–3376, 2020.

[13] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *ICML*, pages 9118–9147, 2022.

[14] Jacob Krantz, Erik Wijmans, Arjun Majumdar, Dhruv Batra, and Stefan Lee. Beyond the nav-graph: Vision-and-language navigation in continuous environments. In *ECCV*, pages 104–120, 2020.

[15] Jacob Krantz, Erik Wijmans, Arjun Majumdar, Dhruv Batra, and Stefan Lee. Beyond the nav-graph: Vision-and-language navigation in continuous environments. In *ECCV*, pages 104–120, 2020.

[16] Arjun Majumdar, Gunjan Aggarwal, Bhavika Devnani, Judy Hoffman, and Dhruv Batra. ZSON: zero-shot object-goal navigation using multimodal goal embeddings. In *NeurIPS*, 2022.

[17] So Yeon Min, Yao-Hung Hubert Tsai, Wei Ding, Ali Farhadi, Ruslan Salakhutdinov, Yonatan Bisk, and Jian Zhang. Object goal navigation with end-to-end self-supervision. *CoRR*, abs/2212.05923, 2022.

[18] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya

Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, pages 8748–8763, 2021.

[19] Santhosh Kumar Ramakrishnan, Devendra Singh Chaplot, Ziad Al-Halah, Jitendra Malik, and Kristen Grauman. PONI: potential functions for objectgoal navigation with interaction-free learning. In *CVPR*, pages 18868–18878, 2022.

[20] Santhosh Kumar Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alexander Clegg, John Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X. Chang, Manolis Savva, Yili Zhao, and Dhruv Batra. Habitat-matterport 3d dataset (HM3D): 1000 large-scale 3d environments for embodied AI. In *NeurIPS Datasets and Benchmarks*, 2021.

[21] Sonia Raychaudhuri, Saim Wani, Shivansh Patel, Unnat Jain, and Angel X. Chang. Language-aligned waypoint (LAW) supervision for vision-and-language navigation in continuous environments. In *EMNLP*, pages 4018–4028, 2021.

[22] Dhruv Shah, Blazej Osinski, Brian Ichter, and Sergey Levine. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action. volume abs/2207.04429, 2022.

[23] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. DD-PPO: learning near-perfect pointgoal navigators from 2.5 billion frames. In *ICLR*, 2020.

[24] Karmesh Yadav, Ram Ramrakhya, Arjun Majumdar, Vincent-Pierre Berges, Sachit Kuhar, Dhruv Batra, Alexei Baevski, and Oleksandr Maksymets. Offline visual representation learning for embodied navigation. *CoRR*, abs/2204.13226, 2022.

[25] Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B Tenenbaum, Tianmin Shu, and Chuang Gan. Building cooperative embodied agents modularly with large language models. *arXiv preprint arXiv:2307.02485*, 2023.

[26] Gengze Zhou, Yicong Hong, and Qi Wu. Navgpt: Explicit reasoning in vision-and-language navigation with large language models. *arXiv preprint arXiv:2305.16986*, 2023.

Appendix

In the supplementary, we provide more implementation details and visualization results of our method. We organize the supplementary as follows.

- In Section A, we present the implementation details, datasets, metrics, and baselines.
- In Section B, we present more details on action-specific image-path dataset collection.
- In Section C, we present more details on action-specific navigator training and inference.
- In Section D, we present more details on prompt design for the instruction parser.
- In Section E, we present more details on zero-shot navigation baselines.
- In Section F, we present more visualization results.

A Experimental Setup

Evaluation Datasets and Metrics. We conduct experiments on the validation unseen split of three datasets, namely R2R-Habitat (15), RxR-Habitat (15), and Fine-Grained R2R (FG-R2R) (12). These three datasets contain 1,839, 1,079, and 1,839 validation episodes on 11 scenes in Matterport3D, respectively. RxR-Habitat contains instructions in three languages, and we only use the English split in our experiments. FG-R2R is an extension of R2R (2), where instructions are chunked into several sub-instructions and each sub-instruction is labeled with a corresponding sub-path, resulting in 6,687 sub-instruction-sub-path pairs. Following the existing works (14; 21; 6), we evaluate VLN performance using Success Rate (**SR**) and Success weighted by inverse Path Length (**SPL**). Besides, to evaluate the generalization ability among datasets, we follow Dorbala *et al.* (9) to propose Consistency on SR (**CSR**) for computing the relative change in SR among datasets. Specifically, $\mathrm{CSR} = 1 - \frac{|\mathrm{SR_a} - \mathrm{SR_b}|}{\max\{\mathrm{SR_a},\, \mathrm{SR_b}\}} \times 100\%$, where $\mathrm{SR_a}$ and $\mathrm{SR_b}$ are success rates for different datasets.

Agent Configurations. Following ZSON (16), the agent has a height of 1.25m, with a radius of 0.1m. It is equipped with one 128×128 RGB sensor with $90°$ horizontal field of view. The agent can execute four low-level actions, namely STOP indicating the end of an episode, FORWARD that moves itself forward by 0.25 meters and TURNLEFT and TURNRIGHT that turn itself by $30°$. The m_s is empirically set to 100 and 50 for R2R-Habitat and RxR-Habitat, respectively. The m_e is set to 500 for all three datasets following exiting works (6; 21).

Baselines. We decompose the instruction into an object navigation sub-task sequence using GPT-3 and execute these sub-tasks sequentially using four zero-shot object navigation methods.

- **CLIP-Nav** (9) is designed for navigating among discrete navigable nodes. The agent uses CLIP to determine which adjacent node has the highest possibility of containing landmarks and then moves to this node. We adapt it to continuous environments using a waypoint navigation algorithm. More details can be found in Appendix.
- **Seq CLIP-Nav** (9) is an extended version of CLIP-Nav with an additional backtracking mechanism, which allows the agent to go back to the previous location if it cannot find the landmarks for several steps.
- **CoW** (10) uses CLIP gradient for object localization and a path-planning algorithm for action determination.
- **ZSON** (16) uses the CLIP for encoding both image and landmark text to the same semantic feature space and then trains an image navigator for object navigation.

B More Details on Action-Specific Image-Path Dataset Collection

For learning a navigator for executing each action demand, we need to collect an action-specific image-path dataset for fine-tuning a trained ZSON model. In Section 3.3.2 of the paper, we have

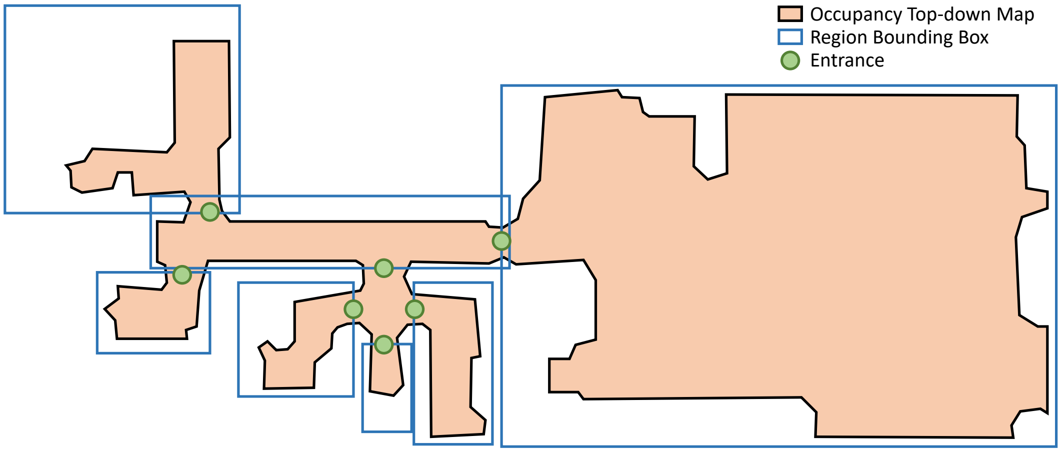

Figure 6: Obtaining entrance positions from the intersections between regions and top-down map.

introduced the basic principle for collecting the episodes (*i.e.*, the path and the corresponding goal image) in the dataset. In this section, we present more data collection details.

- **GoPast Dataset.** We randomly sample two points whose geometric distance is longer than 1.5m, and consider the shortest navigation path as the ground truth path. The goal image is sampled in the middle of the path facing the direction of the agent's advancement. We introduce some randomness to the angle by jittering it by $\pm 45°$.
- **GoInto Dataset.** We randomly choose a region from the scene. The start point is sampled near the entrance of this region (the geometric distance is less than 1.5m). The goal point is sampled randomly inside this region. The goal image is taken in a random direction at the goal point.
- **GoThrough Dataset.** We randomly select a region with two different entrances and sample a random point near each entrance respectively to form a path. The geometric distance from the start or end point to the entrance is less than 1.5m. Goal image is taken in the middle of the path and faces the direction of the agent's advancement.
- **Exit Dataset.** The ground truth path of the EXIT action is similar to the GoInto action besides switching the position of start point and goal point.

We utilize room region bounding box annotations for obtaining the entrance position of regions. Specifically, we consider the intersection between the room region bounding box and the occupancy top-down map as the entrance of a region. An example is shown in Figure 6. Since collecting GoInto, GoThrough, and Exit datasets requires the entrance position, we collect these three datasets from 131 scenes in HM3D (20) dataset that have region bounding box annotations. For the GoPast dataset, we collect from all 800 scenes in the train split of HM3D. We sample 9,000 image-path pairs from each scene.

C More Details on Action-Specific Navigator Training and Inference

We fine-tune a trained ZSON model on the action-specific dataset for learning an action-specific navigator. We use the ZSON model that is trained on agent configuration A described by Majumdar *et al.* (16), *i.e.*, the agent has a height of 1.25m, with a radius of 0.1m and is equipped with one 128 × 128 RGB sensor with 90° horizontal field of view. We fine-tune this model using a reinforcement learning algorithm (*i.e.*, DD-PPO (23)) for 100M steps with the same navigation reward as ZSON. This reward encourages the agent to go to the end of the path in an episode and look toward the goal image:

$$r_t = r_{\text{success}} + r_{\text{angle-success}} - \Delta_{\text{dtg}} - \Delta_{\text{atg}} + r_{\text{slack}} \quad (1)$$

where $r_{\text{success}} = 5$ if STOP is predicted when the agent is within 1m of the goal position, $r_{\text{angle-success}} = 5$ if the agent is within 1m of the goal position and within $25°$ of the goal orientation (and 0 otherwise). Besides, Δ_{dtg} is the change in the agent's distance-to-goal, and Δ_{atg} is the change in the agent's angle-to-goal. Δ_{atg} is set to 0 if the agent is not within a circle of 1m radius from the goal position. We also use a slack reward $r_{\text{slack}} = -0.01$ to encourage the agent to reach the goal as soon as possible.

Q: You are a robot walking in a house. You should parse the navigation instruction into several subtasks and then execute them one by one.

Each subtask consists of an [action] and a {landmark}. Action should be chosen from [turn left], [turn right], [go to], [go past], [go into], [go through], and [exit]. Landmark should be {a specific object} or {a region}. Here is the definition of each action:
[go to] means go to the front of {a specific object};
[go past] means go to {a specific object} and then go pass it;
[go into] means go into {a region};
[go through] means walk along {a region} or through {a region};
[exit] means find a door and go out of {a region}.

Now help me parse the following instruction: "Exit the bedroom and turn left. Walk straight passing the gray couch and stop near the rug. "

A: Let's think step by step:
Subtask 1: [exit] the {bedroom};
Subtask 2: [turn left];
Subtask 3: [go past] the {gray couch};
Subtask 4: [go to] the {rug}.

Figure 7: An example of prompt design 1: a brief description of sub-task.

Q: Parse the instruction using the following subtasks: 1. {go to} [landmark], 2. {go past} [landmark], 3. {turn left}, 4. {turn right}, 5. {go through} [region], 6. {go into} [region], 7. {exit} [region], 8. {stop}.

Here are several examples.
(1) Instruction: "Walk into the hallway and turn left. Walk to the left of the railing and across the hall past the plant. Stop to the left of the stairs.", and the subtasks should be:
1. [go into] {the hallway}.
2. [turn left].
3. [go to] {the left of therailing}.
4. [go past] {the plant}.
5. [go to] {the left of the stairs}.
6. [stop].
……(with another 4 examples)

Now help me parse the following instruction: "Exit the bedroom and turn left. Walk straight passing the gray couch and stop near the rug. "

A: Let's think step by step:
1. [exit] {the bedroom}.
2. [turn left].
3. [go past] {the gray couch}.
4. [go to] {the rug}.
5. [stop].

Figure 8: An example of prompt design 2: a collection of parsing examples. This prompt design performs the best.

After fine-tuning, we use the trained navigator for executing a sub-task. Specifically, we feed the landmark (*i.e.*, text description of an object or a region) to the CLIP for extracting goal embedding. The navigator take the current RGB observation and the goal embedding as input for predicting a low-level action for this sub-task.

D More Details on Prompt Design for the Instruction Parser

We have tried three prompt designs for parsing instructions using the large language model GPT-3.

- **Prompt Design 1:** a brief description of each sub-task definition.
- **Prompt Design 2:** a collection of instruction parsing examples

Q: You are a robot walking in a house. You should parse the navigation instruction into several subtasks and then execute them one by one.

Each subtask consists of an [action] and a {landmark}. Action should be chosen from [turn left], [turn right], [go to], [go past], [go into], [go through], and [exit]. Landmark should be {a specific object} or {a region}. Here is the definition of each action:
[go to] means go to the front of {a specific object};
[go past] means go to {a specific object} and then go pass it;
[go into] means go into {a region};
[go through] means walk along {a region} or through {a region};
[exit] means find a door and go out of {a region}.

Here are several examples.
(1) Instruction: "Walk into the hallway and turn left. Walk to the left of the railing and across the hall past the plant. Stop to the left of the stairs.", and the subtasks should be:
1. [go into] {the hallway}.
2. [turn left].
3. [go to] {the left of therailing}.
4. [go past] {the plant}.
5. [go to] {the left of the stairs}.
6. [stop].
……(with another 4 examples)

Now help me parse the following instruction: " Exit the bedroom and turn left. Walk straight passing the gray couch and stop near the rug. "

A: Let's think step by step :
1. [exit] {the bedroom}.
2. [turn left].
3. [go past] {the gray couch}.
4. [go to] {the rug}.
5. [stop].

Figure 9: An example of prompt design 3: a combination of both sub-task definition description and examples.

- **Prompt Design 3:** a combination of both brief description and examples

Experimental results in Table 4 in the paper show that the second prompt design performs the best. We show the examples of these prompt designs in Figures 7, 8 and 9, respectively. We mark the GPT-3 output in brown color.

E More Details on Zero-Shot Navigation Baselines

We decompose the instruction into an object navigation sub-task sequence using GPT-3 and execute these sub-tasks sequentially using four zero-shot object navigation methods.

- **CLIP-Nav** (9) is designed for navigating among discrete navigable nodes. The agent uses CLIP to determine which adjacent node has the highest possibility of containing landmarks and then moves to this node. To adapt it to continuous environments, we capture 4 RGB images uniformly in different directions and use CLIP (18) to select one image that has the highest possibility of containing landmarks. Then, we randomly set a waypoint in that direction and use a path-planing algorithm to plan low-level actions for navigating to the waypoint. If the CLIP softmax score is higher than the threshold of 0.8, we switch to the next object navigation sub-task. For implementation convenience, we use the "*shortest_path_follower*" API in the Habitat simulator for path planning, which assumes the complete occupancy top-down map is available.
- **Seq CLIP-Nav** (9) is an extended version of CLIP-Nav with an additional backtracking mechanism, which allows the agent to go back to the previous location if it cannot find the landmarks for several steps. In our implementation, we directly set the agent to the position 15 step before for performing backtracking.
- **CoW** (10) uses CLIP gradient for object localization and a path-planning algorithm for action determination. For implementation convenience, we use the "*shortest_path_follower*" API in the Habitat simulator for path planning, which assumes the complete occupancy top-down map is available. Even using the oracle occupancy information, our A^{2}Nav still performs better than this baseline.

- **ZSON** (16) uses the CLIP for encoding both image and landmark text to the same semantic feature space and then trains an image navigator for object navigation. We use the model trained on the HM3D dataset using the config A setting.

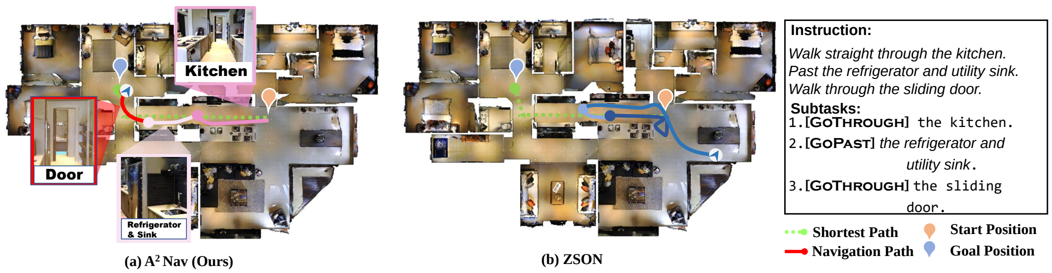

Figure 10: Visualization of the navigation path. Our method successfully goes through the kitchen, while the baseline fails to do it.

Figure 11: Visualization of the navigation path. Our method successfully exits the bedroom and goes past the stair, while the baseline is stuck in the bedroom.

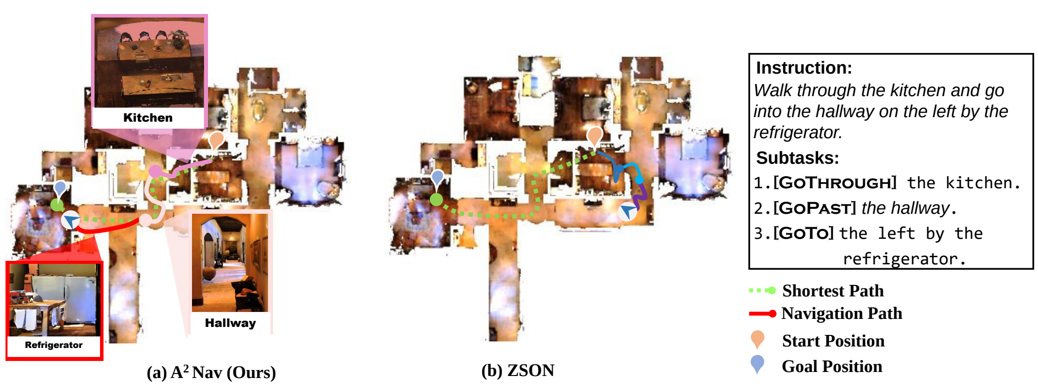

Figure 12: Visualization of the navigation path. Our method successfully goes through the kitchen and finds the refrigerator, while the baseline fails to do it.

F More Visualization Results

In this section, we provide more visualization examples for comparing the method between ZSON (16) and ours. In Figure 10, the instruction requires the agent to go across the kitchen area and exit this area through the door. Our A^2Nav successfully follows the instruction because of the learned "GoThrough" ability, which leads the agent to completely go through the area. However, ZSON (16) just goes to the kitchen area of a refrigerator, which directly causes the task to fail. In Figure 11 the instruction requires the agent to go past the stair which is outside the bedroom. Our A^2Nav successfully exits the bedroom, navigates past the stair and then stop at the correct doorway. In contrast, the ZSON model fails to exit the bedroom and finally stop at the doorway of the bedroom incorrectly. In Figure 12, the instruction requires the agent to get out of the kitchen and stop near the refrigerator. Our A^2Nav successfully walks across the kitchen and goes by the hallway, finally finding the target. ZSON (16) tries to go to the area which suggests the higher confidence of the kitchen, which is the opposite direction of the shortest path to the target. All examples demonstrate the effectiveness of our action-aware agent.