GC-VLN: Instruction as Graph Constraints for Training-free Vision-and-Language Navigation

Hang Yin[123]*, **Haoyu Wei**[123]*, **Xiuwei Xu**[123]†, **Wenxuan Guo**[123], **Jie Zhou**[123], **Jiwen Lu**[123]‡
[1]Department of Automation, Tsinghua University
[2]Beijing Key Laboratory of Embodied Intelligence Systems
[3]Beijing National Research Center for Information Science and Technology

Abstract: In this paper, we propose a training-free framework for vision-and-language navigation (VLN). Existing zero-shot VLN methods are mainly designed for discrete environments or involve unsupervised training in continuous simulator environments, which makes it challenging to generalize and deploy them in real-world scenarios. To achieve a training-free framework in continuous environments, our framework formulates navigation guidance as graph constraint optimization by decomposing instructions into explicit spatial constraints. The constraint-driven paradigm decodes spatial semantics through constraint solving, enabling zero-shot adaptation to unseen environments. Specifically, we construct a spatial constraint library covering all types of spatial relationship mentioned in VLN instructions. The human instruction is decomposed into a directed acyclic graph, with waypoint nodes, object nodes and edges, which are used as queries to retrieve the library to build the graph constraints. The graph constraint optimization is solved by the constraint solver to determine the positions of waypoints, obtaining the robot's navigation path and final goal. To handle cases of no solution or multiple solutions, we construct a navigation tree and the backtracking mechanism. Extensive experiments on standard benchmarks demonstrate significant improvements in success rate and navigation efficiency compared to state-of-the-art zero-shot VLN methods. We further conduct real-world experiments to show that our framework can effectively generalize to new environments and instruction sets, paving the way for a more robust and autonomous navigation framework. Project Page.

1 Introduction

Vision-and-language navigation (VLN) [1] is an essential foundational capability for various embodied tasks, which requires the robot to move in a novel environment following complex linguistic instructions. The VLN instructions typically describe the path sequence, including the direction and distance of movement along the path, as well as scene information near the path. This necessitates the robot's ability of linguistic comprehension, environment perception, and spatial reasoning. However, limited by the availability of annotated data, early data-driven VLN methods exhibit poor generalization ability in unseen scenarios. Moreover, most existing data-driven VLN methods [2, 3, 4] are trained in simulator environment, which leads to a large sim-to-real gap. Therefore, building data-independent zero-shot navigation frameworks is necessary.

Zero-shot VLN methods [5, 6] have been proposed to overcome the limitation of annotated data and narrow the sim-to-real gap. DiscussNav [7] leverages multi-expert discussions to integrate instruction understanding, environmental perception, and decision verification. MapGPT [6] builds a map-guided GPT-based agent that leverages an online linguistic-formed map for adaptive path planning. However, constrained by immature simulators, these zero-shot VLN methods can only operate in discrete environments [1], which only allows the robot to move between a set of discrete nodes

* Equal contribution. † Project lead. ‡ Corresponding author.

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

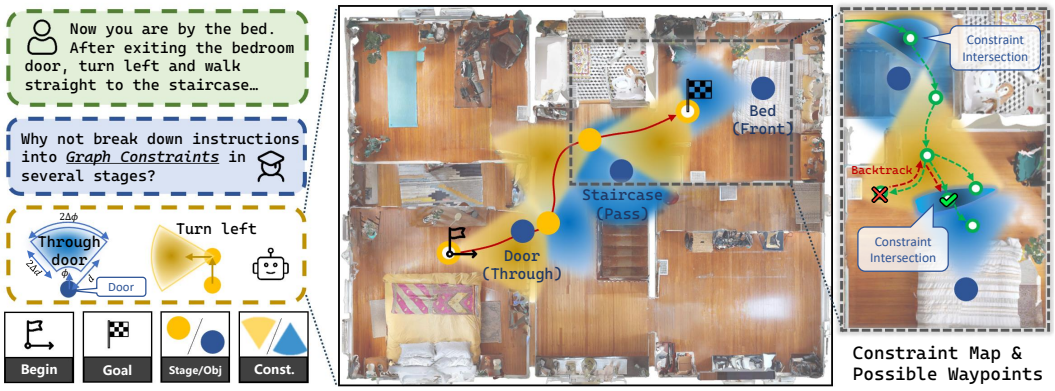

Figure 1: GC-VLN models the instructions as a graph constraint optimization problem and solve the graph constraints based on the robot's observations to obtain the robot's path, which enables training-free VLN. We illustrate how graph constraints guide the navigation path and how to re-plan the path when exploration fails.

within the environment. Methods in discrete simulator environment are impractical for deploying in real-world scenarios, introducing a significant sim-to-real gap. Vision-and-language navigation in continuous environments (VLN-CE) [8] is introduced to better simulate real-world environments, where the robot can move freely and stand at any position within the environment. In VLN-CE, the robot predicts low-level actions, including turning and moving forward, which is closer to real-world environments. The VLN-CE methods can be easily deployed in novel real-world scenarios without the adaptation. Nevertheless, most existing zero-shot VLN-CE methods still rely on unsupervised training within simulators. A^2Nav [9] constructs five subtasks for VLN-CE and trains the five subtask modules in a self-supervised manner within the simulator. The self-supervised training on simulator data means that these methods still exhibit a sim-to-real gap. Therefore, a training-free framework for VLN-CE is highly demanded.

In this paper, we propose **G**raph-**C**onstraints for **V**ision-and-**L**anguage **N**avigation (GC-VLN), a training-free framework for VLN-CE. Different from previous works which is deployed in discrete environments or self-supervised trained on simulator data, GC-VLN adopts a completely training-free approach to perceive the scene and predict the path. Specifically, we construct a constraint library to cover all types of spatial relationship constraints found in the instructions. Then the linguistic instruction is decomposed into a directed acyclic graph, which is used to query the library to construct a graph constraint. The navigation is formulated as a graph constraint optimization problem, where the coordinates of the waypoint nodes are progressively solved by the constraint solver. Additionally, we build a navigation tree to address the problem of uncertainty in the number of solutions during graph constraint solving. We conducted extensive experiments in the simulator benchmarks R2R-CE and RxR-CE, achieving state-of-the-art performance across the board. Real-world experiments further demonstrate the strong generalization ability of our method.

2 Related Work

2.1 Vision-and-Language Navigation

In real-world navigation scenarios, humans typically have prior knowledge of the path to the goal. Therefore, robots do not need to autonomously explore the goal. Using human prior knowledge, the robot follows human instructions to reach the goal, a task called Vision-and-Language Navigation (VLN) [1, 10, 11, 12]. Early VLN methods [13, 7] use discrete simulator environments to evaluate VLN approaches, where the environment is divided into multiple discrete drivable waypoints. The robot can only stand on the waypoints and selects one of the adjacent waypoints as a short-term goal at each step. Discrete environments allow the method to focus on path selection. However, due to the large sim-to-real gap, discrete VLN methods are difficult to deploy in real-world scenarios.

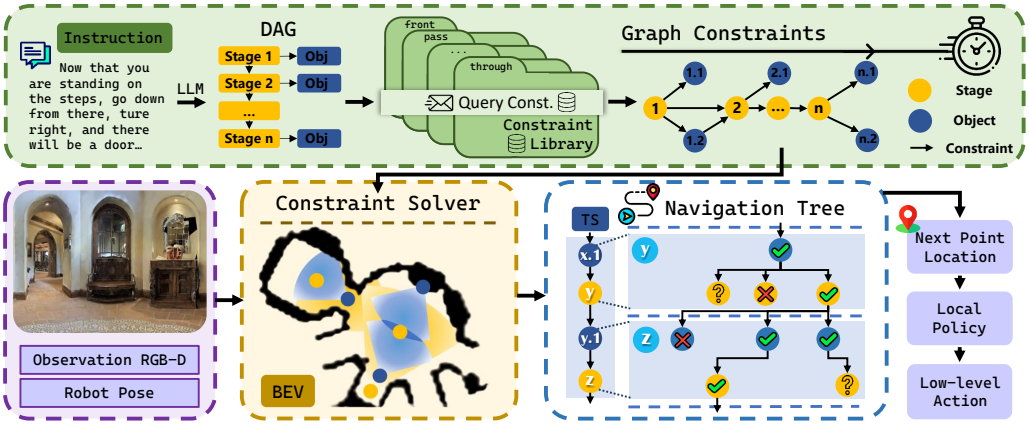

Figure 2: Framework of GC-VLN. We construct a constraint library, containing all the spatial relationship mentioned by navigation instruction. The instruction is decomposed into a directed acyclic graph (DAG) and used to query the library to get the graph constraints. The constraint solver determines the path by solving the graph constraint optimization. According to the topological sort (TS) of graph constraint, we build the navigation tree, where the number of leaf nodes equals the number of graph constraint solutions. In graph constraint and TS, $t.i$ means the i-th object node in stage t.

Towards the aim of closely approximating real-world scenarios, Vision-and-Language Navigation in Continuous Environments (VLN-CE) [8] is proposed, which allows the robot to move freely within the scene by predicting low-level actions. Methods designed [14, 15, 16, 17, 2] for VLN-CE can be directly deployed in real-world scenarios without modification. To balance the simplification of methods in discrete VLN with the approximation of real environments in VLN-CE, the waypoint predictor [18] is proposed to predict discrete waypoints online within a continuous environment, and becomes the mainstream technical approach for VLN-CE task.

2.2 Training-free Navigation

Conventional navigation methods are task-training, involving modules such as LSTM [19] and transformer [20]. The limit of annotated data results in these methods having restricted generalization abilities for more diverse goals or human instructions. Moreover, task-training methods also exhibit sim-to-real gap, which limits their performance in real-world scenarios.

Zero-shot navigation [21, 22] methods have been proposed to address the generalization problem. In the field of goal-oriented navigation, zero-shot methods for object-goal navigation [23, 24, 25, 21, 26], image-goal navigation [27, 28, 29, 30], and text-goal navigation [31] have already reached a relatively mature stage. For VLN in discrete environments, zero-shot methods [32] have also seen significant development. However, for the VLN-CE task, zero-shot methods remain unsatisfactory, which is the goal of our method.

3 Approach

We first present the task definition and the overall pipeline of our approach, followed by an explanation of how the graph constraint $\mathcal{K}$ is constructed. Finally, we elaborate on how we sequentially solve the coordinates of nodes in $\mathcal{K}$ and utilize $\mathcal{K}$ to guide the navigation direction.

3.1 Overview

In VLN, a robot is initialized in an unknown environment. The robot is required to follow the linguistic instruction $\mathcal{I}$ provided by a human, so that it can move within the environment and reach the final destination. As shown in Figure 1, the instruction is typically a piece of text that describes

how to navigate from the starting point to the destination point, including navigation directions, objects encountered along the navigational path, and spatial relationships between the navigation path and objects. If the agent reaches within r meters of the navigation endpoint in no more than t steps, the navigation is successful.

Pipeline. As shown in Figure 2, the pipeline of GC-VLN contains two main modules, the graph constraint construction module and constrained optimization module. First, the original instruction is decomposed into a multi-stage directed acyclic graph $\mathcal{G}$, which contains all the information required for navigation. We construct a constraint library that encompasses all types of spatial relationships in VLN instruction. The graph $\mathcal{G}$ is used to query this library to obtain the constraint types between nodes, thereby the graph constraint $\mathcal{K}$ is constructed. The determination of the node coordinates in $\mathcal{K}$ is formulated as a constrained optimization problem based on their constraints, and the navigation tree handles the uncertain number of coordinate solutions from the constraint solver, to backtrack when no solution meets the constraint conditions.

3.2 Graph Constraint Construction

To handle the long sequence characteristics and complex spatial relationships in the instruction $\mathcal{I}$, we convert $\mathcal{I}$ into a structured representation, namely the graph constraint, which is required to meet three criteria: 1. It must not lose any information from $\mathcal{I}$. 2. It must explicitly contain all the objects mentioned in $\mathcal{I}$. 3. It must provide explicit navigation directions, as well as spatial relationships between objects and the navigation path.

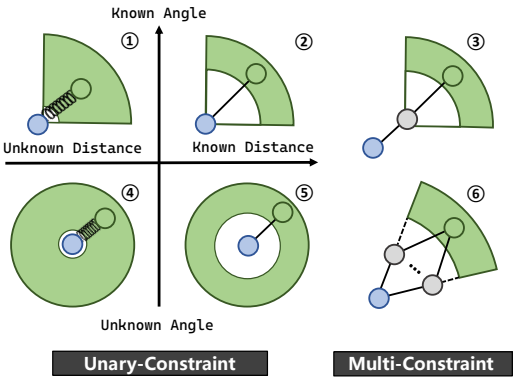

Figure 3: Diagram of the Constraint Library containing six types of constraint. For constraint $c(v \mid u)$, u and v are colored blue and green, respectively. The green region is the possible region for v.

Instruction Decomposition. The LLM is prompted to decompose the instruction $\mathcal{I}$ into multiple navigation stages, where each stage involves exactly one displacement (no requirement for the number of rotations). Each stage has two attributes: the navigation direction and the objects that appear in that stage. The navigation direction is categorized as one of the following: `["front", "right", "left", "back", "unknown"]`. Every object mentioned in $\mathcal{I}$ belongs to and belongs only to one navigation stage. Each object has one attribute: its spatial relationship with the navigation path. These nodes and edges constitute a directed acyclic graph $\mathcal{G}$.

Graph $\mathcal{G} = \text{LLM}(\mathcal{I})$, where $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and $\mathcal{V}, \mathcal{E}$ represent the nodes and the directed edges, respectively. $\mathcal{V}$ can be categorized into waypoints $\mathcal{V}^w$ and object nodes $\mathcal{V}^o$, where $\mathcal{V}^w = \{v_1^w, v_2^w, \ldots, v_n^w\}, \mathcal{V}^o = \bigcup_{i=1}^{n} \{v_{i1}^o, v_{i2}^o, \ldots, v_{ik_i}^o\}$. Edges can be categorized into $\mathcal{E}^w$ and $\mathcal{E}^o$ based on whether they are connected to an object node, where $\mathcal{E}^w = \left\{\left(v_i^w, v_{i+1}^w\right) \mid 1 \le i \le n-1\right\}, \mathcal{E}^o = \bigcup_{i=1}^{n} \left(\left\{\left(v_i^w, v_{ij}^o\right) \mid 1 \le j \le k_i\right\} \cup \left\{\left(v_{ij}^o, v_i^w\right) \mid 1 \le j \le m_i\right\}\right)$. v_i^w and v_{ij}^o represent the waypoint nodes and the j-th object nodes in stage i, and $e = (u, v)$ represents the edge pointing from node u to node v. For details of instruction decomposition, please refer to the supplementary materials.

Constraint Library. The spatial relationships between nodes mentioned in $\mathcal{I}$ are considered spatial constraints $c(v \mid u)$ of their coordinates u, v. The type of constraint is distinguished based on the description of the spatial relationship topology in $\mathcal{I}$. As shown in Figure 3, we construct a constraint library $\mathcal{L}$ containing six types of constraint, which covers all types of spatial relationship topologies involved in VLN instructions.

Assuming u is known, the second type of constraint as an example, the constraint $c(v \mid u)$ can be expressed as:

$$c(v \mid u) = (c^a(v \mid u), c^d(v \mid u)), \quad \text{sum}(c) = c^a + c^d, \quad \min(c) = \min(c^a, c^d) \tag{1}$$

$$c^a(v \mid u) = \cos(\Delta\phi)\|v - u\| - [\|v - u\| - (v - u) \cdot (\cos\phi, \sin\phi)] \tag{2}$$

$$c^d(v \mid u) = \Delta d^2 - (\|v - u\| - d)^2 \tag{3}$$

where $\phi, \Delta\phi$ and $d, \Delta d$ are the baseline and tolerance of angle and distance of e, and c^a, c^d are the sub-constraints of angle and distance. The complete expressions of the sub-constraints of the other five types of constraint are provided in the supplementary materials.

Graph Constraint. We query the constraint library $\mathcal{L}$ using $e \in \mathcal{E}$ to obtain the type of constraint $c = \mathcal{L}(e)$ corresponding to e. All nodes and constraints together form the graph constraint $\mathcal{K}$:

$$\mathcal{K} = (\mathcal{V}, \mathcal{C}), \quad \mathcal{C} = \{\mathcal{L}(e) \mid e \in \mathcal{E}\} \tag{4}$$

The direction of c represents the direction of causality and inferring, meaning that the coordinates of v can only be inferred when the coordinate of its parent node u is known. At the beginning of the navigation, the coordinate of v_1^w is $(0, 0)$ and that of other nodes in $\mathcal{V}$ are all uncertain. At the end of the navigation, all coordinates of $v_i^w \in \mathcal{V}^w$ are determined and v_n^w is the endpoint of this episode.

3.3 Constrained Optimization for Node Coordinates

The graph constraint $\mathcal{K}$ is used to guide the robot's direction of motion. Therefore, the coordinates of the nodes $v \in \mathcal{V}$ are determined according to the navigation order. We propose a graph-constrained optimization framework to determine the coordinates and a navigation tree to handle the uncertainty in the number of solutions.

Constraint Solver. Graph constraint $\mathcal{K}$ is utilized to solve the coordinates of waypoints $v_i^w \in \mathcal{V}^w$. We determine the coordinates of object nodes $v_{ij}^o \in \mathcal{V}^o$ by perceiving RGB-D observations with a pre-trained vision model. Solving for the coordinate of a node v requires knowledge of the coordinates of all its parent nodes in $\mathcal{K}$. In the initial state of navigation, only the starting point of stage 1, which serves as the root node v_1^w of $\mathcal{K}$, has known coordinates. v_1^w acts as the original reference for solving the coordinates of all subsequent nodes. We perform a topological sort (TS) on $\mathcal{K}$ to determine the order in which the node coordinates are solved during navigation. The order of the topological sort ensures that for any constraint $c(v \mid u)$, v always follows after u. Therefore, when determining the node coordinates in this order, the coordinates of the current node's parent nodes are guaranteed to be already solved. The details of the topological sort can be found in the supplementary materials.

For the waypoint v_i^w, multiple constraints $\mathcal{C}_{v_i^w} = \{c(v_i^w \mid v_{i-1}^w)\} \cup \{c(v_i^w \mid v_{ij}^o) \mid 1 \le j \le m_i\}$ jointly restrict it, corresponding to all edges that points to v_i^w. We formulate the coordinate determination as a *nonlinear constrained optimization problem* (P1):

$$\text{(P1): } \underset{v_i^w}{\text{Maximize}} \sum_{c \in \mathcal{C}_{v_i^w}} \text{sum}(c) \tag{5}$$

$$\text{subject to} \quad \min(c) \ge 0, \ \forall c \in \mathcal{C}_{v_i^w} \tag{6}$$

$$\|v_i^w - x_j\| \ge L, \ \forall j \in \{1, ..., k-1\} \tag{7}$$

where x_j represents the j-th solution of v_i^w. The functions $\text{sum}(c)$ and $\min(c)$ represent the sum and minimum values of the sub-constraints in c, respectively. This constraint is solved for multiple times, and in the k-th iteration, we incorporate the first $k-1$ solutions into the constraint conditions. For object nodes v_{ij}^o, we retain only the constraint conditions (without the optimization objective). A pre-trained vision model perceives objects in the observation and the objects are projected onto the BEV map. The coordinates of objects satisfying the constraint conditions are selected as solutions for v_{ij}^o.

Navigation Tree. The constraint solver generates an uncertain number of solutions for nodes $v \in \mathcal{V}$. We construct the navigation tree $\mathcal{T}$ to handle this uncertainty, where each node in $\mathcal{T}$ has a specific

Table 1: Results of R2R-CE and RxR-CE in Habitat simulator. We mainly compare the SR and SPL of SOTA methods of VLN-CE.

Method	Zero-shot	Training-free	R2R Unseen				RxR Unseen			
			NE	OSR	**SR**	**SPL**	NE	OSR	**SR**	**SPL**
Seq2Seq [8]	×	×	7.8	37.0	24.0	22.0	-	-	-	-
WS-MGMap [3]	×	×	6.3	47.6	38.9	34.3	9.8	29.8	15.0	12.1
NaVid [4]	×	×	5.5	49.1	37.4	35.9	8.4	34.5	23.8	21.2
Uni-NaVid [33]	×	×	5.6	53.3	47.0	42.7	6.2	55.5	48.7	40.9
ETPNav [2]	×	×	4.7	65.0	57.0	49.0	5.6	-	54.8	44.9
Cow [34]	✓	×	-	-	7.8	5.8	-	-	7.9	6.1
ZSON [35]	✓	×	-	-	19.3	9.3	-	-	14.2	4.8
A²Nav [9]	✓	×	-	-	22.6	11.1	-	-	16.8	6.3
NavGPT-CE [5]	✓	✓	8.4	26.9	16.3	10.2	-	-	-	-
CA-Nav [36]	✓	✓	7.6	**48.0**	25.3	10.8	10.4	-	19.0	6.0
InstructNav [37]	✓	✓	**6.9**	-	31.0	**24.0**	-	-	-	-
GC-VLN (Ours)	✓	✓	7.3	41.8	**33.6**	16.3	**8.8**	**44.4**	**33.8**	**13.8**

coordinate in the environment. Assuming that the topological sort (TS) of $\mathcal{K}$ is $[v_1, v_2, ..., v_{|\mathcal{V}|}]$, the constraint solver progressively solves for the node coordinates. According to the number of solutions from the constraint solver, TS is extended to a navigation tree, where the nodes at level i in $\mathcal{T}$ are the coordinate solutions of the i-th node v_i in $\mathcal{K}$. Standing at the i-th level of $\mathcal{T}$, the number of branches in the $(i+1)$-th level of the navigation tree corresponds to the number of solutions for v_{i+1}. The path from the root node in $\mathcal{T}$ to a leaf node in $\mathcal{T}$ corresponds to a specific path in the environment.

If the constraint solver fails to find a feasible solution, there will be no branches at that level for v_i, indicating that this particular navigation branch is unsuccessful. In this case, the robot will backtrack in $\mathcal{T}$ until it finds a branch point with unexplored siblings. At this branch point, the robot selects the next unexplored sibling as the future path. The backtracking mechanism enhances the fault tolerance of GC-VLN, allowing the robot to explore potentially missed correct paths. Navigation terminates if the robot reaches the last level of $\mathcal{T}$, corresponding to the endpoint $v_{|\mathcal{V}|}$.

4 Experiments

We conducted extensive experiments using simulators and real-world scenarios to validate the effectiveness of our method. In this section, we will introduce our experimental setup, comparison with state-of-the-art methods, ablation studies, and qualitative analysis, respectively.

4.1 Benchmarks and Implementation Details

Datasets: We conduct simulator experiments on the mainstream R2R-CE [38] and RxR-CE [39] datasets of VLN-CE tasks. R2R-CE and RxR-CE are derived by converting the discrete trajectories from the R2R and RxR VLN datasets into continuous trajectories within the Habitat simulator [40]. The scenes in the R2R-CE and RxR-CE datasets are sourced from the MP3D [41] dataset. We use the validation-unseen split of R2R-CE and RxR-CE, including 1,839 and 11,006 episodes. The instructions in R2R-CE are entirely in English, while those in RxR-CE are in three languages. The average path length and the instruction length in RxR-CE are greater than those in R2R-CE.

Evaluation Metrics: Following [42, 43], we use *success rate (SR)* and *success rate weighted by path length (SPL)* as the main evaluation metrics. SR represents the proportion of episodes where the agent successfully reaches within m meters of the endpoint, where $m = 3$. SPL builds upon the success rate by incorporating path length, reflecting the similarity between the actual path and the ground truth path. Besides, *navigation error (NE)* and *Oracle Success Rate (OSR)* are also reported.

Compared Methods: We compare with state-of-the-art methods for training-free VLN-CE. NavGPT-CE [5, 44] is the NavGPT adapted version for continuous environments. CA-Nav [36]

Table 2: Effect of pipeline design in GC-VLN on R2R-CE benchmark.

Graph Constraint					Consrtraint Solver and Navigation Tree				
Method	NE	OSR	**SR**	**SPL**	**Method**	NE	OSR	**SR**	**SPL**
Relax constraints in $\mathcal{K}$	8.7	25.7	21.5	10.8	Rearrange topological sort	9.0	36.4	26.2	12.1
Remove waypoint constraints	8.7	30.7	23.8	10.9	Remove solution order	8.8	41.8	33.6	14.9
Remove object constraints	8.6	35.7	30.3	15.9	Random Constraint Solver	9.1	9.1	7.0	2.6
Remove unary-constraint	8.7	37.9	32.6	16.3	Simplify $\mathcal{T}$	8.9	38.2	29.6	11.4
Remove multi-constraint	8.7	36.9	30.9	15.2	Remove Backtracking	9.0	23.2	21.3	13.6
Full Approach	**7.3**	**41.8**	**33.6**	**16.3**	**Full Approach**	**7.3**	**41.8**	**33.6**	**16.3**

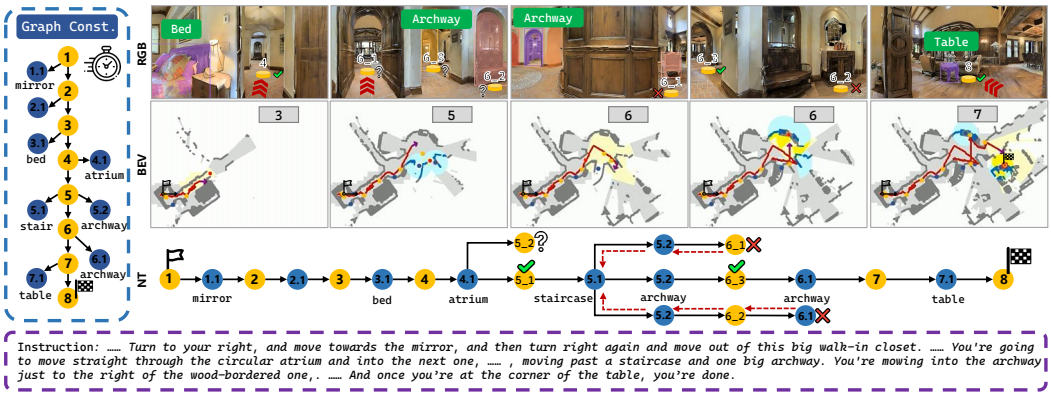

Figure 4: Demonstration of the graph constraints solving of GC-VLN. Here t_i is the i-th branch node of the t-th level of navigation tree.

is a constraint-aware zero-shot VLN-CE method. InstructNav [37] is a generic instruction navigation method that is applicable to VLN-CE, ojbect-goal navigation, and demand-driven navigation.

Implementation Details: We evaluate GC-VLN in the habitat simulator and real-world robot Hexmove. We deploy DeepSeek-R1 [45] as LLM for instruction decomposition and Grounded-SAM-2 [46, 47, 48] for object perception. The local policy is the Fast Marching Method [49].

4.2 Comparison with State-of-the-art

We compare GC-VLN with the state-of-the-art VLN-CE methods in three settings for VLN-CE: supervised, zero-shot and training-free in Table 1. GC-VLN surpasses previous zero-shot methods, and surpassed SOTA training-free method InstructNav by 2% success rate on the R2R-CE benchmark. On the RxR-CE, we outperform all zero-shot methods for which performance has been reported. In the supervised setting, we also outperform some methods, such as NaVid on RxR-CE.

In particular, RxR is more challenging than R2R. However, our method maintains high performance on RxR, which demonstrates its stronger ability to handle complex instructions.

4.3 Ablation Study

We conduct ablation studies on R2R-CE to validate the effectiveness of each module in GC-VLN.

Effect of graph constraint: In Table 2, we first relax the graph constraints, which means removing the angle constraint and retaining only the maximum distance for the distance constraint. The results show a significant drop in SR and SPL. Then we remove the waypoint constraints $c\left(v_{t+1}^{w} \mid v_{t}^{w}\right)$, object constraints $c\left(v_{t+1}^{w} \mid v_{tj}^{o}\right)$, unary-constraints (type $1, 2, 4, 5$) and multi-constraints (type $3, 6$), respectively, replacing them with the weakest constraint (type 4). The performance of GC-VLN shows varying degrees of decline, demonstrating the effectiveness of graph constraints.

Effect of constraint solver and navigation tree: We rearrange the topological sort by making object nodes no longer belong to a specific stage. For solution order, we no longer use the order of

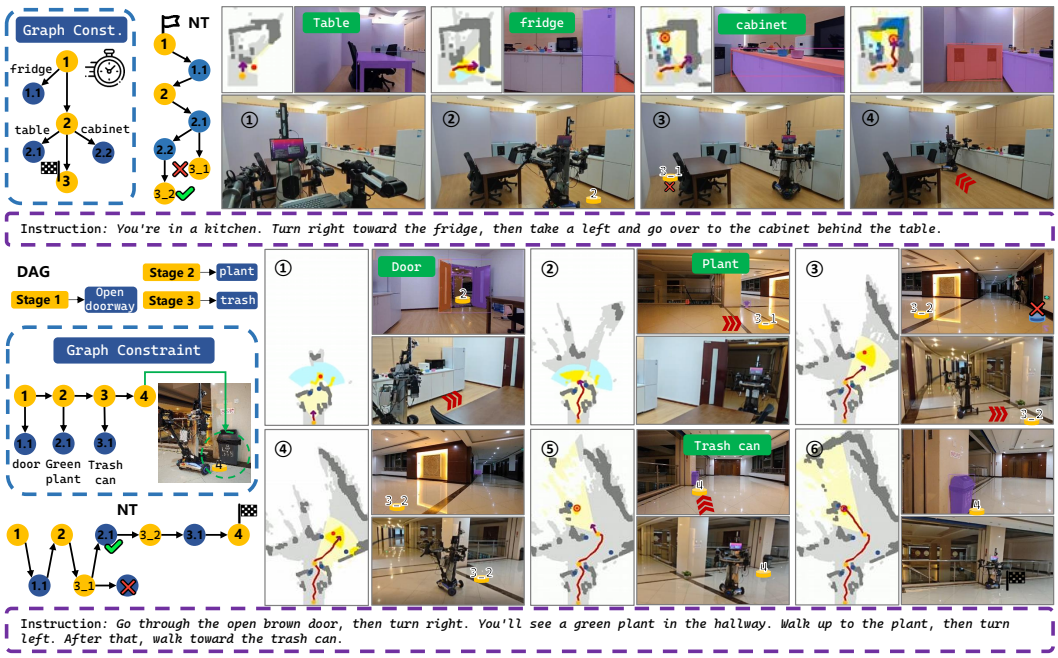

Figure 5: Demonstration of deployment in real-world environment.

coordinate solutions for a node to construct the navigation tree branches but adopt a random order. For the constraint solver, instead of using the maximization objective function to solve coordinates, we randomly sample points within the region defined by the constraint conditions. We simplify the navigation tree $\mathcal{T}$ by removing the earlier unexplored branches, and remove backtracking by not saving unexplored branches at all. The performance declines across the board, demonstrating the effectiveness of constraint solver and navigation tree.

4.4 Qualitative Results

To provide a more intuitive view of our method, we present visualizations of the navigation process in both the simulator and real-world environments. As shown in Figure 4, the robot progressively explore the scene in the simulator and solve the position of each waypoint based on spatial constraints. As illustrated in Figure 5, we deploy GC-VLN in the real world, demonstrating its strong generalization ability in real-world deployment.

5 Conclusion

In this paper, we propose a graph constraint-guided training-free vision-and-language navigation framework. Since zero-shot VLN methods for discrete environment and self-supervised VLN methods for continuous environment have a large sim-to-real gap, it is challenging to deploy them in real-world scenarios. We construct a spatial constraint library that encompasses all possible spatial constraints and utilize LLM to decompose the instruction into a directed acyclic graph. The graph is used to query the constraint library and obtain the graph constraints. The positions of nodes in the directed acyclic graph are determined by the constraint solver, which solves the graph constraint optimization problem. Navigation tree is used to handle the issue of uncertainty in the number of solutions. Extensive experiments conducted in both simulated and real-world scenarios demonstrate the performance and generalization ability of GC-VLN, for which it can be easily deployed in real-world environment without performance degradation.

Acknowledgments

This work was supported in part by the Beijing Natural Science Foundation under Grant No. L247009, the National Natural Science Foundation of China under Grant 62125603, and the Beijing National Research Center for Information Science and Technology.

References

[1] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[2] D. An, H. Wang, W. Wang, Z. Wang, Y. Huang, K. He, and L. Wang. Etpnav: Evolving topological planning for vision-language navigation in continuous environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[3] P. Chen, D. Ji, K. Lin, R. Zeng, T. H. Li, M. Tan, and C. Gan. Weakly-supervised multi-granularity map learning for vision-and-language navigation. *arXiv preprint arXiv:2210.07506*, 2022.

[4] J. Zhang, K. Wang, R. Xu, G. Zhou, Y. Hong, X. Fang, Q. Wu, Z. Zhang, and H. Wang. Navid: Video-based vlm plans the next step for vision-and-language navigation. *Robotics: Science and Systems*, 2024.

[5] G. Zhou, Y. Hong, and Q. Wu. Navgpt: Explicit reasoning in vision-and-language navigation with large language models. *arXiv preprint arXiv:2305.16986*, 2023.

[6] J. Chen, B. Lin, R. Xu, Z. Chai, X. Liang, and K.-Y. K. Wong. Mapgpt: Map-guided prompting with adaptive path planning for vision-and-language navigation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

[7] Y. Long, X. Li, W. Cai, and H. Dong. Discuss before moving: Visual language navigation via multi-expert discussions. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 17380–17387. IEEE, 2024.

[8] J. Krantz, E. Wijmans, A. Majundar, D. Batra, and S. Lee. Beyond the nav-graph: Vision and language navigation in continuous environments. In *European Conference on Computer Vision (ECCV)*, 2020.

[9] P. Chen, X. Sun, H. Zhi, R. Zeng, T. H. Li, G. Liu, M. Tan, and C. Gan. a^2nav: Action-aware zero-shot robot navigation by exploiting vision-and-language ability of foundation models, 2023. URL https://arxiv.org/abs/2308.07997.

[10] W. Hao, C. Li, X. Li, L. Carin, and J. Gao. Towards learning a generic agent for vision-and-language navigation via pre-training. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[11] P. Anderson, A. Shrivastava, J. Truong, A. Majumdar, D. Parikh, D. Batra, and S. Lee. Sim-to-real transfer for vision-and-language navigation. In *Conference on Robot Learning*, pages 671–681. PMLR, 2021.

[12] K. Chen, J. K. Chen, J. Chuang, M. Vázquez, and S. Savarese. Topological planning with transformers for vision-and-language navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11276–11286, 2021.

[13] R. Liu, X. Wang, W. Wang, and Y. Yang. Bird's-eye-view scene graph for vision-language navigation. In *ICCV*, pages 10968–10980, 2023.

[14] J. Krantz and S. Lee. Sim-2-sim transfer for vision-and-language navigation in continuous environments. In *European conference on computer vision*, pages 588–603. Springer, 2022.

[15] H. Wang, W. Liang, L. Van Gool, and W. Wang. Dreamwalker: Mental planning for continuous vision-language navigation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10873–10883, 2023.

[16] Z. Wang, X. Li, J. Yang, Y. Liu, and S. Jiang. Gridmm: Grid memory map for vision-and-language navigation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15625–15636, 2023.

[17] Z. Wang, X. Li, J. Yang, Y. Liu, J. Hu, M. Jiang, and S. Jiang. Lookahead exploration with neural radiance representation for continuous vision-language navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13753–13762, June 2024.

[18] Y. Hong, Z. Wang, Q. Wu, and S. Gould. Bridging the gap between learning in discrete and continuous environments for vision-and-language navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022.

[19] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[21] B. Yu, H. Kasaei, and M. Cao. L3mvn: Leveraging large language models for visual target navigation. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3554–3560. IEEE, 2023.

[22] H. Yin, X. Xu, Z. Wu, J. Zhou, and J. Lu. Sg-nav: Online 3d scene graph prompting for llm-based zero-shot object navigation. *arXiv preprint arXiv:2410.08189*, 2024.

[23] K. Zhou, K. Zheng, C. Pryor, Y. Shen, H. Jin, L. Getoor, and X. E. Wang. Esc: Exploration with soft commonsense constraints for zero-shot object navigation. In *ICML*, pages 42829–42842. PMLR, 2023.

[24] W. Cai, S. Huang, G. Cheng, Y. Long, P. Gao, C. Sun, and H. Dong. Bridging zero-shot object navigation and foundation models through pixel-guided navigation skill, 2023. URL https://arxiv.org/abs/2309.10309.

[25] P. Wu, Y. Mu, B. Wu, Y. Hou, J. Ma, S. Zhang, and C. Liu. Voronav: Voronoi-based zero-shot object navigation with large language model. *arXiv preprint arXiv:2401.02695*, 2024.

[26] Y. Kuang, H. Lin, and M. Jiang. Openfmnav: Towards open-set zero-shot object navigation via vision-language foundation models. *arXiv preprint arXiv:2402.10670*, 2024.

[27] W. Guo, X. Xu, H. Yin, Z. Wang, J. Feng, J. Zhou, and J. Lu. Igl-nav: Incremental 3d gaussian localization for image-goal navigation. *arXiv preprint arXiv:2508.00823*, 2025.

[28] J. Krantz, T. Gervet, K. Yadav, A. Wang, C. Paxton, R. Mottaghi, D. Batra, J. Malik, S. Lee, and D. S. Chaplot. Navigating to objects specified by images. *arXiv preprint arXiv:2304.01192*, 2023.

[29] H. Yin, X. Xu, L. Zhao, Z. Wang, J. Zhou, and J. Lu. Unigoal: Towards universal zero-shot goal-oriented navigation. *arXiv preprint arXiv:2503.10630*, 2025.

[30] M. Wei, T. Wang, Y. Chen, H. Wang, J. Pang, and X. Liu. Ovexp: Open vocabulary exploration for object-oriented navigation. *arXiv preprint arXiv:2407.09016*, 2024.

[31] X. Sun, L. Liu, H. Zhi, R. Qiu, and J. Liang. Prioritized semantic learning for zero-shot instance navigation, 2024. URL https://arxiv.org/abs/2403.11650.

[32] D. Li, W. Chen, and X. Lin. Tina: Think, interaction, and action framework for zero-shot vision language navigation, 2024. URL https://arxiv.org/abs/2403.08833.

[33] J. Zhang, K. Wang, S. Wang, M. Li, H. Liu, S. Wei, Z. Wang, Z. Zhang, and H. Wang. Uninavid: A video-based vision-language-action model for unifying embodied navigation tasks, 2024.

[34] S. Y. Gadre, M. Wortsman, G. Ilharco, L. Schmidt, and S. Song. Cows on pasture: Baselines and benchmarks for language-driven zero-shot object navigation. *CVPR*, 2023.

[35] A. Majumdar, G. Aggarwal, B. Devnani, J. Hoffman, and D. Batra. Zson: Zero-shot object-goal navigation using multimodal goal embeddings. In *Neural Information Processing Systems (NeurIPS)*, 2022.

[36] K. Chen, D. An, Y. Huang, R. Xu, Y. Su, Y. Ling, I. Reid, and L. Wang. Constraint-aware zero-shot vision-language navigation in continuous environments. *arXiv preprint arXiv:2412.10137*, 2024.

[37] Y. Long, W. Cai, H. Wang, G. Zhan, and H. Dong. Instructnav: Zero-shot system for generic instruction navigation in unexplored environment, 2024.

[38] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. Van Den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3674–3683, 2018.

[39] A. Ku, P. Anderson, R. Patel, E. Ie, and J. Baldridge. Room-across-room: Multilingual vision-and-language navigation with dense spatiotemporal grounding. *arXiv preprint arXiv:2010.07954*, 2020.

[40] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9339–9347, 2019.

[41] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *3DV*, 2017.

[42] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.

[43] G. Ilharco, V. Jain, A. Ku, E. Ie, and J. Baldridge. General evaluation for instruction conditioned navigation using dynamic time warping. *arXiv preprint arXiv:1907.05446*, 2019.

[44] G. Zhou, Y. Hong, Z. Wang, X. E. Wang, and Q. Wu. Navgpt-2: Unleashing navigational reasoning capability for large vision-language models. *arXiv preprint arXiv:2407.12366*, 2024.

[45] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

[46] T. Ren, S. Liu, A. Zeng, J. Lin, K. Li, H. Cao, J. Chen, X. Huang, Y. Chen, F. Yan, Z. Zeng, H. Zhang, F. Li, J. Yang, H. Li, Q. Jiang, and L. Zhang. Grounded sam: Assembling open-world models for diverse visual tasks, 2024.

[47] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson, E. Mintun, J. Pan, K. V. Alwala, N. Carion, C.-Y. Wu, R. Girshick, P. Dollár, and C. Feichtenhofer. Sam 2: Segment anything in images and videos, 2024. URL https://arxiv.org/abs/2408.00714.

[48] T. Ren, Q. Jiang, S. Liu, Z. Zeng, W. Liu, H. Gao, H. Huang, Z. Ma, X. Jiang, Y. Chen, Y. Xiong, H. Zhang, F. Li, P. Tang, K. Yu, and L. Zhang. Grounding dino 1.5: Advance the "edge" of open-set object detection, 2024.

[49] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. *proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996.

A Overview

This supplementary material is organized as follows:

- Section B provides the algorithm for the overall pipeline of GC-VLN.
- Section D provides the details of the approach.
- Section C provides the details of the hardware used in real-world experiments.
- Section E reports the results of additional ablation experiments.
- Section F shows visualization results of failure cases.
- Section G details the prompts for LLM.

B Pipeline of GC-VLN

In Algorithm 1, we provide an algorithm diagram of GC-VLN.

Algorithm 1 Overall Pipeline of GC-VLN

Require: Instruction $\mathcal{I}$, Observation $\mathcal{O}$
Ensure: Goal Position (x, y)
  $\mathcal{G} \leftarrow$ DecomposeInstruction($\mathcal{I}$)
  $\mathcal{L} \leftarrow$ ConstraintLibrary()
  $\mathcal{C} \leftarrow$ ConstructGraphConstraint($\mathcal{G}, \mathcal{L}$)
  $\mathcal{T} \leftarrow$ NewNavigationTree()
  while True **do**
    $\{v_i \mid i = 1, 2, ..., k\} \leftarrow$ ConstraintSolver($\mathcal{C}, \mathcal{O}$)
    $\mathcal{T} \leftarrow$ UpdateNavigationTree($\mathcal{T}, \{v_i \mid i = 1, 2, ..., k\}$)
    if $\{v_i \mid i = 1, 2, ..., k\} == \Phi$ **then**
      $v \leftarrow$ Backtrack($\mathcal{T}$)
    else
      $v \leftarrow$ GetWaypoint($\{v_i \mid i = 1, 2, ..., k\}$)
    end if
    Go to $v\ (x, y)$
    if isFinalWaypoint (v) **then**
      Stop at $v\ (x, y)$
    end if
  end while

C Hardware Details

We present the hardware details of the robot employed in our real-world experiments in Figure 6. The robot base is the TRIGGER platform developed by Hexmove. A monocular RGB-D camera serves as our input for observations, which is Orbbec Gemini 336L. Our pose input is provided by the RealSense T265 tracking camera. We utilize the PIPER robot arm from AGILE·X for object manipulation.

D Details of Approach

D.1 Instruction Decomposition

LLM is prompted to decompose the linguistic instruction $\mathcal{I}$. The decomposition of $\mathcal{I}$ must meet several rules: 1. Each stage must contain exactly ONE position change. Rotating alone without movement is not a stage. 2. The direction of a stage is equal to the direction of the line from the start to the end of the stage. 3. Each stage comprises two attributes: position and a list of nodes. Each node consists of two attributes: name and position. 4. "waypoint_position" must belong to one of

"front", "right", "left", "back", "unknown". 5. "object_position" must belong to one of the types *"right", "left", "through", "weave", "pass", "near", "back"*.

D.2 Constraint Library

As illustrated in Figure 3 of the main text, based on the description in the instruction $\mathcal{I}$, the angle and distance information within a constraint can be identified, allowing the constraints to be classified into six types. A unary constraint involves only two nodes, while a multi-constraint involves three or four nodes. For example, the instruction "move forward 3 meters to the left" belongs to type 2, where both u (blue node) and v (green node) are waypoints. The instruction "move forward through a door" corresponds to type 3, where u_1 (blue node) and v (green node) are waypoints, while u_2 (gray node) is a door node. The instruction "walk forward through the space between two chairs" falls under type 6, where u_1 (blue node) and v (green node) are waypoints, and u_2 and u_3 (gray nodes) are chair nodes.

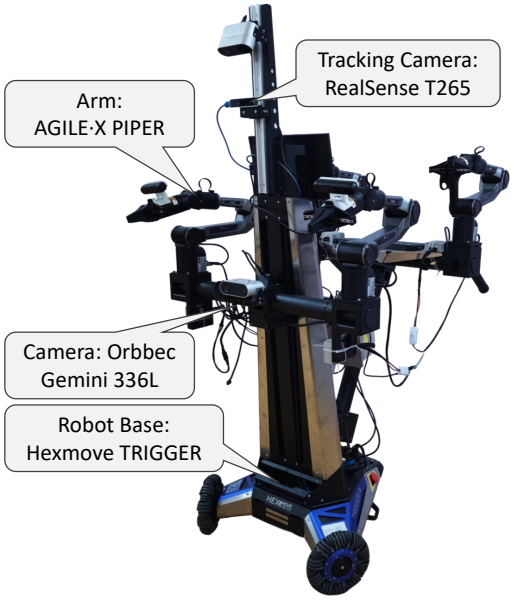

Figure 6: The robot employed for conducting the real-world experimental deployments.

We formulate the sub-constraints for all six types of constraints. Each type of constraint includes two possible sub-constraints: the angle constraint c^a and distance constraint c^d. The constraint types that include angular constraints are: 1, 2, 3, and 6. The constraint types that include distance constraints are: 2, 3, 5, and 6. We formulate the constraint as:

$$c = (\mathbf{1}^a c^a, \mathbf{1}^d c^d), \quad \text{sum}(c) = \mathbf{1}^a c^a + \mathbf{1}^d c^d, \quad \min(c) = \min(\mathbf{1}^a c^a, \mathbf{1}^d c^d) \tag{8}$$

where $\mathbf{1}^a$ and $\mathbf{1}^d$ denote indicator functions representing the presence of angle and distance sub-constraints, respectively.

For types 1, 2, 4, and 5, the c^a and c^d are:

$$c^a\left(v \mid u\right) = \cos(\Delta\phi)\|v-u\| - [\|v-u\| - (v-u)\cdot(\cos\phi, \sin\phi)] \tag{9}$$

$$c^d\left(v \mid u\right) = \Delta d^2 - (\|v-u\| - d)^2 \tag{10}$$

For type 3, the c^a and c^d are:

$$c^a\left(v \mid u_1, u_2\right) = \cos(\Delta\phi)\|v-u_2\| - [\|v-u_2\| - (v-u_2)\cdot(\cos\phi, \sin\phi)] \tag{11}$$

$$c^d\left(v \mid u_1, u_2\right) = \Delta d^2 - (\|v-u_2\| - d)^2 \tag{12}$$

In type 1, 2, 3, 4, and 5, if the angle and distance are not explicitly specified, then $\Delta\phi = 45°$, $d = 1.5m$ and $\Delta d = 1.5m$.

For type 6, the c^a and c^d are:

$$c^a\left(v \mid u_1, u_2, u_3\right) = \cos(\Delta\phi)\|v-u_1\| - [\|v-u_1\| - (v-u_1)\cdot(\cos\phi, \sin\phi)] \tag{13}$$

$$c^d\left(v \mid u_1, u_2, u_3\right) = \Delta d^2 - (\|v-u_1\| - d)^2 \tag{14}$$

$$\phi = \arg\left(\frac{u_2-u_1}{|u_2-u_1|} + \frac{u_3-u_1}{|u_3-u_1|}\right) \tag{15}$$

$$\Delta\phi = \frac{1}{2}\arccos\left(\frac{(u_2-u_1)\cdot(u_3-u_1)}{|u_2-u_1|\cdot|u_3-u_1|}\right) \tag{16}$$

D.3 Topological Sort

To determine the order of nodes in graph constraint $\mathcal{K}$, we perform topological sort on the $\mathcal{K}$, which ensures that the parent node u of each constraint $c(v \mid u)$ is always positioned ahead of its corresponding child node v. The topological sort satisfies several conditions: 1. The parent node must appear before its child node. 2. Among multiple child nodes of a single node, object nodes must precede waypoint nodes. 3. The order of multiple object child nodes under a single parent node must be consistent with the order in which they are mentioned in the instruction $\mathcal{I}$.

E Ablation Study

In Table 3, we report results of additional ablation experiments on hyperparameters of GC-VLN and RxR-CE benchmark. For R2R-CE, we ablate the angle tolerance $\Delta\phi$ of the constraint type $1, 2, 3, 6$, and the distance baseline d of the all constraint types. For RxR-CE, the settings of ablation are the same as those in the main text.

Table 3: Effect of angle tolerance and distance baseline on R2R-CE. Effect of constraint and constraint solver on RxR-CE.

Constraint Condition on Angle (R2R-CE)					**Consrtraint Condition on Distance (R2R-CE)**				
Angle Tolerance $\Delta\phi$	NE	OSR	**SR**	**SPL**	**Distance Baseline d**	NE	OSR	**SR**	**SPL**
$30°$	10.2	32.6	30.1	15.9	$0.8m$	10.6	35.7	28.5	14.7
$45°$ **(Ours)**	**7.3**	**41.8**	**33.6**	**16.3**	$1.5m$ **(Ours)**	**7.3**	**41.8**	**33.6**	**16.3**
$75°$	10.2	33.8	31.5	15.6	$2.5m$	10.1	39.0	30.5	14.0

Graph Constraint (RxR-CE)					**Consrtraint Solver (RxR-CE)**				
Method	NE	OSR	**SR**	**SPL**	**Method**	NE	OSR	**SR**	**SPL**
Relax constraints in $\mathcal{K}$	10.8	31.5	24.5	9.9	Random Constraint Solver	11.1	21.9	18.0	10.6
Full Approach	**8.8**	**44.4**	**33.8**	**13.8**	**Full Approach**	**8.8**	**44.4**	**33.8**	**13.8**

F Visualization of Failure Cases

In Figure 7, we further provide visualizations of failure cases for better understanding.

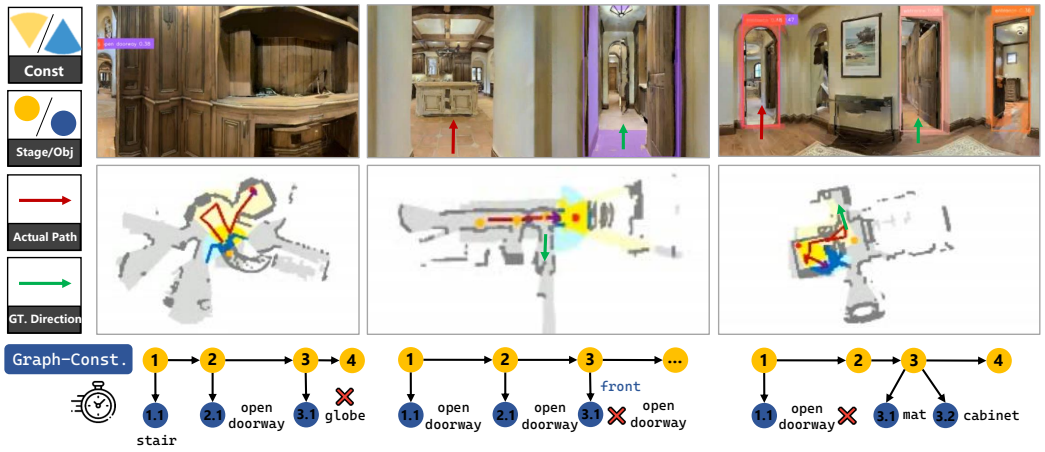

Figure 7: Visualization of three failure cases. In the first case, robot fails to locate the globe. In the second case, the robot mistakes "passing through the door to the right" for "forward" during the construction of the graph constraint. In the third case, the robot initially selects an incorrect path, and coincidentally encounters a correct object, which prevents timely backtracking.

G Prompts

We provide the prompt used for instruction decomposition in GC-VLN.

```
Parse navigation instructions into movement stages in JSON format: {
  "stage 1": {
    "waypoint position": <position>,
    "connected nodes": [
      {"node": <object>, "object position": <position>},
      ...
    ]
  },
  ...
}
Input Instruction: <Instruction>
Rules:
Each stage has one position change. The key "waypoint position" refers
to the relative position of the next waypoint with respect to the
current one, encompassing both the distance and the angle between them.
The key "connected nodes" means the objects involved in the current
stage. Each node includes two attributes: the "node" name and its
"position" which denotes the relative position with respect to the
waypoint.
```

where <Instruction> will be replaced by the input instruction.