2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

SFCo-Nav: Efficient Zero-Shot Visual Language Navigation via Collaboration of Slow LLM and Fast Attributed Graph Alignment

Chaoran Xiong[1,†], *Graduate Student Member, IEEE*, Litao Wei[1,2,†], Xinhao Hu[1,†], Kehui Ma[1], Ziyi Xia[1], Zixin Jiang[1], Zhen Sun[1], and Ling Pei[1,3,*], *Senior Member, IEEE*

***Abstract*— Recent advances in large vision-language models (VLMs) and large language models (LLMs) have enabled zero-shot approaches to visual language navigation (VLN), where an agent follows natural language instructions using only ego perception and reasoning. However, existing zero-shot methods typically construct a naive observation graph and perform per-step VLM–LLM inference on it, resulting in high latency and computation costs that limit real-time deployment. To address this, we present SFCo-Nav, an efficient zero-shot VLN framework inspired by the principle of slow–fast cognitive collaboration. SFCo-Nav integrates three key modules: 1) a slow LLM-based planner that produces a strategic chain of subgoals, each linked to an imagined object graph; 2) a fast reactive navigator for real-time object graph construction and subgoal execution; and 3) a lightweight asynchronous slow–fast bridge aligns advanced structured, attributed imagined and perceived graphs to estimate navigation confidence, triggering the slow LLM planner only when necessary. To the best of our knowledge, SFCo-Nav is the first slow-fast collaboration zero-shot VLN system supporting asynchronous LLM triggering according to the internal confidence. Evaluated on the public R2R and REVERIE benchmarks, SFCo-Nav matches or exceeds prior state-of-the-art zero-shot VLN success rates while cutting total token consumption per trajectory by over 50% and running more than 3.5× faster. Finally, we demonstrate SFCo-Nav on a legged robot in a hotel suite, showcasing its efficiency and practicality in indoor environments.**

I. INTRODUCTION

Visual language navigation (VLN) requires an embodied agent to follow natural-language navigation instructions by perceiving the environment, reasoning about the instructions, and executing a sequence of actions to reach a goal [1]. It is a fundamental task of a general embodied navigation (EN) system [2].

Recently, the success of large-scale Vision-Language Models (VLMs) [3] and Large Language Models (LLMs) [4], [5] has inspired the development of zero-shot VLN systems [6]–[10]. These zero-shot methods exploit the pre-trained reasoning and grounding capabilities of these models without task-specific fine-tuning. In contrast to conventional end-to-end trained VLN agents [11]–[13], zero-shot VLN offers significant advantages, such as no training cost, rapid deployment in new environments, and strong generalization to unseen instructions and layouts [14].

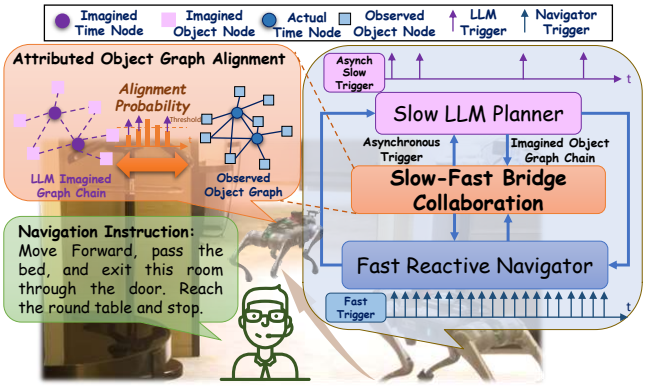

Fig. 1. SFCo-Nav is an efficient zero-shot VLN framework inspired by slow–fast cognitive collaboration. It comprises three modules: 1) a slow brain LLM-based planner; 2) a fast brain reactive navigator; and 3) a lightweight asynchronous slow–fast bridge that aligns the imagined and perceived graphs, estimates navigation confidence, and triggers LLM only when necessary. This design minimizes costly LLM calls while preserving high navigation success.

The core challenge in zero-shot VLN lies in efficiently transforming visual observations and textual instructions into accurate navigation actions. Early work, such as NavGPT [8], preprocesses all navigable viewpoints with a VLM BLIP-2 [15] to produce textual descriptions of the scene. Then the agent queries an LLM to output the next target viewpoint ID. While NavGPT verified LLMs' navigation capability, it is an offline system that demands heavy computation. More recently, MapGPT [9] employs a naive structured topological map representation and integrated VLMs like GPT-4V [3] or BLIP-2 [15] for online, step-by-step image-to-text conversion and direct action prediction. Although online-capable, this approach still requires per-step VLM processing over multiple candidate viewpoints. This leads to substantial token usage and slow inference. To achieve higher efficiency, NavCoT [14] partially mitigates the cost by fine-tuning the LLM to reduce token consumption, yet retains the same expensive VLM multi-point perception at each step.

Current zero-shot VLN methods [6]–[10], [14] usually adopt VLM–LLM paradigm. This paradigm can be referred to as a full slow-brain strategy: at every navigation step, the

[†]Chaoran Xiong, Litao Wei, and Xinhao Hu contribute equally to this work.
[*]Corresponding Author: Ling Pei.
This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant No. 62273229, and in part by the Science and Technology Commission of Shanghai Municipality under Grant Nos. 24DZ3101300, 24TS1402600, and 24TS1402800.
[1]The authors are with the Shanghai Key Laboratory of Navigation and Location Based Services, Shanghai Jiao Tong University (SJTU), Shanghai 200240, China. [2]Litao Wei is also with Zhiyuan College, SJTU. [3]Ling Pei is also with the State Key Laboratory of Submarine Geoscience, SJTU. (e-mail: {sjtu4742986; oscar0731; xinhaohu; khma0929; matcha.latte; zhensun; yan.xiang; ling.pei}@sjtu.edu.cn; jiangzixin0214@163.com).
Demo and code will be released at `https://anonymous.4open.science/r/GQ-Nav-5EB5/`.

system exhaustively queries a VLM for all visible viewpoints and uses an LLM to directly decide the next action. This results in high token cost and slow decision process. Conversely, human navigators typically perform an initial phase of slow thinking: parsing instructions, setting subgoals, and forming an internal plan. Then the fast thinking executes the subgoals using perceptual intuition. Slow reasoning is invoked only when confidence drops in unfamiliar or altered situations. Details may refer to [16]. Such a slow–fast collaboration mechanism has the potential to improve navigation efficiency, but has been overlooked in zero-shot VLN.

Inspired by human slow–fast cognitive synergy [16], we present SFCo-Nav, a zero-shot VLN framework combining an LLM-based slow brain with a lightweight reactive fast brain for efficient embodied navigation. The slow brain parses instructions into a strategic decision chain including target objects, navigation skills, and an imagined object graph. On the other hand, the fast brain perceives a real-time object graph and executes the plan using object-skill primitives. An asynchronous triggering mechanism monitors a structured attributed graph [17] matching confidence and engages the slow brain only when confidence drops below a threshold, minimizing unnecessary reasoning. SFCo-Nav achieves comparable or higher success rates than strong zero-shot baselines on R2R [1] and REVERIE [18], with significantly lower inference costs. Additionally, SFCo-Nav demonstrates practical efficiency in a real-world hotel-suite deployment on a legged robot. To the best of our knowledge, it is the first slow-fast zero-shot VLN system with asynchronous LLM triggering based on internal confidence. Our main contributions are as follows:

1) A zero-shot slow-fast collaborative navigation framework SFCo-Nav for embodied navigation. Our slow component, an LLM, generates a strategic decision chain of target objects, required skills and imagined object graph. The fast component, a reactive navigator, then executes these steps by efficiently aligning a real-time, perceived object graph with the target objects and imagined object graphs generated by LLM.
2) A lightweight, asynchronous triggering mechanism that governs the slow-fast collaboration for superior efficiency. This is achieved by computing an advanced structured, attributed graph matching probability as the navigator confidence based on the alignment of the perceived graph with the LLM imagined graph chain.
3) State-of-the-art efficiency in zero-shot embodied navigation, demonstrated on the R2R and REVERIE benchmarks. SFCo-Nav achieves comparable or higher task success rates than existing methods while reducing average total consumed tokens per trajectory by over 50% and running more than 3.5× faster.

The remainder of this paper is organized as follows. Section II reviews related work on zero-shot VLN and slow–fast systems. Section III formalizes the proposed slow–fast collaboration framework for VLN. Section IV presents the architecture and technical details of SFCo-Nav. Section V reports experimental results on public VLN benchmarks, including comparisons with state-of-the-art zero-shot methods, ablation analyses, and a real-world case study. Finally, a conclusion is given in Section VI.

II. RELATED WORK

In this section, we review relevant literature in zero-shot VLN. Section II-A examines recent LLM-based approaches that leverage large-scale pretrained models to interpret navigation instructions. Section II-B discusses slow–fast system principles in perception and decision-making, and their potential to improve the efficiency of zero-shot VLN.

A. LLM-Based Zero-Shot VLN

Advances in foundation Vision–Language Models (VLMs) [3] and Large Language Models (LLMs) [4], [5] have enabled agents to follow natural language navigation instructions without task-specific training. Unlike conventional end-to-end VLN [12], [13], [19], zero-shot VLN leverages pretrained models directly, avoiding costly data collection and fine-tuning, which is a key advantage for real-world deployment [8]–[10].

NavGPT [8] first demonstrated LLM navigation capabilities by converting all viewpoints into BLIP-2 textual descriptions offline and prompting the LLM to select the next viewpoint. Despite its effectiveness, this per-step multi-view processing made online use impractical. A2Nav introduced Action Awareness so the LLM could reason over both scene observations and available navigation skills [20], while Console [21] used the LLM to filter and refine landmarks, boosting success but still relying on scene-trained VLN models.

To enable online execution, MapGPT [9] used a simple topological map linking navigable viewpoints, converting multiple observations per step into text for LLM-driven action prediction. However, the naive structure of scene description and multi-view VLM processing per step caused high token cost and latency. NavCoT [14] reduced LLM token usage and improved reasoning via LLM fine-tuning, but retained the same expensive multi-view VLM computation, undermining the zero-shot advantage. These limitations highlight the need for a real-time, low-cost, deployable zero-shot VLN approach.

B. Slow–Fast Systems

Current zero-shot VLN methods, such as [8]–[10], [14], usually adopt a full slow-brain strategy, invoking VLM–LLM reasoning at every step to output actions. While comprehensive, this leads to redundant perception, high computation cost, and low time efficiency. Inspired by human fast and slow thinking [16], slow–fast systems have gained attention in robotics, particularly in manipulation, where a slow high-level planner guides a fast low-level controller for efficient execution [22].

In zero-shot visual language navigation scenario, however, slow–fast collaboration is largely unexplored. The longer distances and larger spatial scope demand efficient confidence-based triggers for re-planning. Existing methods [8], [9], [23]

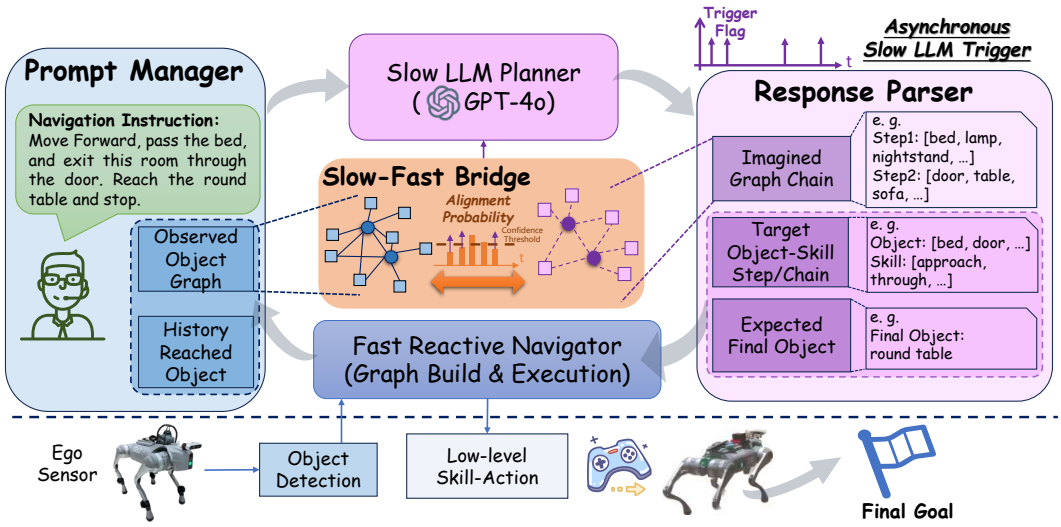

Fig. 2. System overview of SFCo-Nav, a slow–fast collaborative framework for efficient zero-shot visual language navigation. The Slow LLM Planner (Π_{slow}) decomposes the navigation instruction into subgoals, each paired with an imagined object graph G_t^i. The Fast Reactive Navigator (π_{fast}) builds a perceived object graph G_t^p in real time and executes low-level actions to align with G_t^i. The Slow–Fast Bridge computes the graph-alignment confidence C_t; high confidence ($C_t > \tau_C$) keeps control with π_{fast}, while low confidence ($C_t \leq \tau_C$) triggers replanning by Π_{slow}.

either invoke slow reasoning at every step or lack adaptive mechanisms, both of which incur high latency. This gap motivates our design of a slow–fast collaborative zero-shot VLN framework that reduces slow LLM reasoning overhead for practical deployment.

III. PROBLEM FORMULATION

To solve the slow-fast problem in VLN, our zero-shot slow–fast collaboration framework is formalized as follows. We consider a zero-shot navigation setting in which an embodied agent follows a natural language instruction I to reach a target object or location in an unknown environment. The task is to produce a sequence of low-level actions $\mathbf{A} = (a_0, a_1, \ldots, a_T)$ that guides the agent to the instruction-specified goal while minimizing use of computationally expensive reasoning components.

At each timestep t, the agent perceives the environment from egocentric sensory input and maintains an internal state S_t that summarizes observations and task progress. The navigation policy is realized as a hybrid slow–fast architecture as follows:

1) Slow High-Level Planner (Π_{slow}): A high-capacity but costly reasoning module that, when invoked, generates or updates a high-level plan $\mathscr{P}_t$ from I and S_t.
2) Fast Low-Level Controller (π_{fast}): A low-latency policy that executes local control actions using recent observations and the current high-level plan $\mathscr{P}_t$.
3) Collaboration Module: A decision mechanism that determines when to switch navigation policy between π_{fast} and Π_{slow} based on agent internal indicators, which seeks to maximize task success while minimizing reliance on Π_{slow}.

Navigation terminates when the agent reaches the goal within a stopping distance d_{stop}. The design objective is to jointly optimize Π_{slow}, π_{fast}, and the slow-fast bridge rule to maximize task success while reducing the reliance on the slow planner for efficiency.

IV. METHODOLOGY

In this section, in order to address the zero-shot navigation problem formulated in Section III, we present SFCo-Nav, our proposed slow–fast collaborative framework. Firstly, the overall architecture of SFCo-Nav is introduced. Then a detailed description of its three core components is provided.

A. System Overview

The architecture of SFCo-Nav, shown in Fig. 2, is designed around a slow–fast collaboration principle to achieve efficient embodied navigation. The system is composed of three primary modules:

- Slow LLM Planner (Π_{slow}) serves as the "slow brain," decomposing a user's instruction I into a sequence of subgoals. Each subgoal contains a target object and a required skill, together with an imagined object graph G_t^i, representing the LLM's semantic and spatial prior for the expected subgoal scene.
- Fast Reactive Navigator (π_{fast}) acts as the "fast brain," operating in real time. It builds a local perceived object graph G_t^p from ego-centric sensor inputs (e.g., RGB-D) and outputs low-level actions a_t to greedily maximize the alignment between G_t^p and the subgoal's G_t^i.
- Slow–Fast Bridge implements asynchronous collaboration by computing a graph-alignment confidence score $C_t = \text{Align}(G_{1:t}^p, G_{1:t}^i)$. If $C_t > \tau_C$, control remains with π_{fast}. Otherwise, the bridge triggers Π_{slow} to replan.

The following subsections detail the design and implementation of each of these three modules.

B. Slow LLM Planner: Object-Skill-Graph Chain Generation

Let the user's navigation instruction be denoted by a string I. The agent's perception of the environment is represented by a perceived object graph G_t^p. The full navigation plan generated by the Slow Planner Π_{slow} is denoted by $\mathscr{P}$.

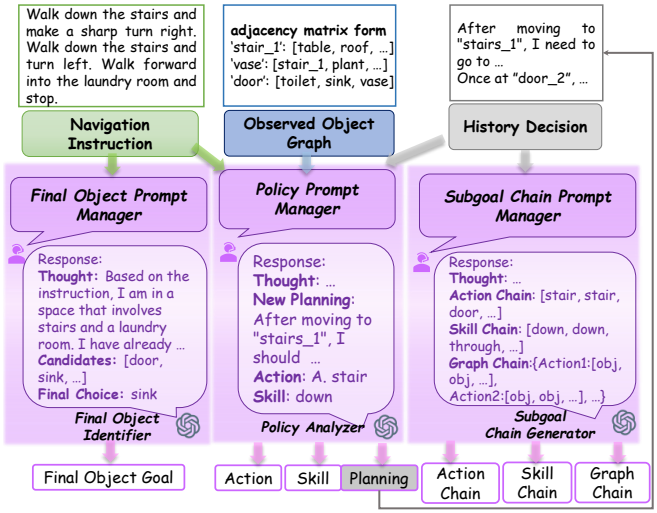

Fig. 3. Slow LLM Planner prompt structure and operation process.

As show in Fig. 3 The Slow LLM Planner operates as a sequential pipeline of three functional modules:

1) Final Object Identifier (f_{goal}): This initial module parses the user's command I to determine the final navigation target o_{goal}:

$$o_{\text{goal}} = f_{\text{goal}}(I). \tag{1}$$

The LLM is prompted to extract the object name where the navigation task concludes. This output defines the agent's global stopping condition. The task is successful if the agent's state s_T at the final timestep T satisfies distance$(s_T, o_{\text{goal}}) \le d_{\text{stop}}$, where d_{stop} is a predefined threshold (e.g., 3 m).

2) Policy Analyzer (f_{policy}): This module generates the agent's immediate subgoal and long-horizon strategic plan based on the current context. Let $H_{t-1} = \{o_1, o_2, \ldots, o_{t-1}\}$ denote the history of previously reached subgoal objects. The module is conditioned on the instruction I, history H_{t-1}, and current perceived object graph G_t^p:

$$(R_t, (o_t, sk_t)) = f_{\text{policy}}(I, H_{t-1}, G_t^p). \tag{2}$$

Here, o_t is the immediate target object and sk_t is the required skill to reach it. The reasoning trace R_t outlines the high-level plan in natural language, while the subgoal (o_t, sk_t) is an actionable command for execution in the next phase.

3) Subgoal Chain Generator (f_{chain}): This module translates the human-readable reasoning trace R_t into a structured, machine-executable subgoal chain $\mathscr{P}_t$:

$$\mathscr{P}_t = f_{\text{chain}}(R_t). \tag{3}$$

The plan $\mathscr{P}_t$ consists of N future subgoals:

$$\mathscr{P}_t = [(o_{t+j}, sk_{t+j}, G_{t+j}^i)]_{j=0}^{N-1}, \tag{4}$$

where o_{t+j} is the target object, sk_{t+j} is the required skill, and G_{t+j}^i is the LLM-generated imagined object graph, representing a structured semantic–spatial prior for the expected scene at subgoal o_{t+j}. The first tuple (o_t, sk_t, G_t^i) corresponds to the immediate subgoal to be executed by π_{fast} unless the Slow–Fast Bridge triggers Π_{slow} based on low confidence.

C. Fast Reactive Navigator: Object Graph Construction and Skill Planner

The fast reactive navigator is responsible for grounding the Slow Planner's (Π_{slow}) abstract commands into low-level actions. It operates in a tight perception–action loop with two primary functions: 1) constructing a real-time representation of the environment as a perceived object graph, and 2) executing the current LLM-defined subgoal based on this representation.

1) Perceived Object Graph Construction: At each timestep t, the navigator builds a dynamic, attributed graph $G_t^p = (V_t, E_t)$, to represent its local understanding of the scene, as illustrated in Fig. 4. This graph includes the agent itself as a special reference node. The set of nodes V_t consists of a timestep node v_{timestep} representing the agent, and a node v_j for each object detected in the current field of view. Each object node v_j is attributed with its semantic label (e.g., chair), its estimated 3D position relative to the agent, and the timestamp of its last observation. On the other hand, the edges E_t represent spatial relationships between nodes. For simplicity and efficiency, we construct a star-topology graph where edges connect the timestep node v_{timestep} to each observed object node v_j. Each edge $e_{\text{timestep},j} \in E_t$ is attributed with the relative distance and bearing from the agent to the object.

Different from other zero-shot VLN methods using large-scale VLM for all navigable viewpoint description, our object graph is constructed by only one-time lightweight object detection for the current view of the agent, thus reducing token and time consumption in the visual perception process.

2) Subgoal Execution Planner: The planner's goal is to execute the current subgoal (o_t, sk_t, G_t^i) provided by the Slow Planner Π_{slow}. It implements a reactive policy π_{fast} that maps the current perceived graph G_t^p, the target object o_t and its corresponding skill sk_t to a low-level control action a_t. For instance, the subgoal execution planner can adopt skills such as approach, through or go up or go down to the target objects, which improves the navigator execution accuracy and efficiency.

D. Slow-Fast Bridge: Attributed Object Graph Alignment

The slow–fast bridge adopts a lightweight, asynchronous triggering mechanism that governs the slow–fast collaboration for superior efficiency. This is achieved by computing an advanced structured, attributed graph [17] matching probability as the navigator confidence C_t. This confidence score is based on the alignment of the perceived object graph G_t^p with the imagined object graph chain G_t^i.

The vertices in the structured attributed graphs represent viewpoint timesteps as users $\mathscr{V}_{\text{u}}$ and objects as attributes $\mathscr{V}_{\text{a}}$. To describe the probabilistic model for edges, we first consider the set of object–object vertex pairs $\mathscr{E}_{\text{u}} \triangleq \mathscr{V}_{\text{u}} \times \mathscr{V}_{\text{u}}$ and the set of object–viewpoint vertex pairs $\mathscr{E}_{\text{a}} \triangleq \mathscr{V}_{\text{u}} \times \mathscr{V}_{\text{a}}$. For any vertex pair $e \in \mathscr{E} \triangleq \mathscr{E}_{\text{u}} \cup \mathscr{E}_{\text{a}}$, we write $G_t^p(e) = 1$ if an edge exists and $G_t^p(e) = 0$ otherwise (and similarly for $G_t^i(e)$). The joint observation of an edge pair is denoted $(G_t^p, G_t^i)(e)$. The edges are generated according to a correlated model.

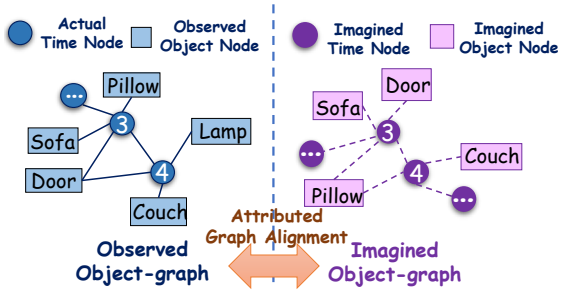

Fig. 4. Observed and imagined attributed graph structure.

Algorithm 1: P, Q Computation for Graph Alignment

Input : Perceived Graph $G^p_{1:t}$, Imagined Graph $G^i_{1:t}$
Output: Probability matrices P, Q

$\mathscr{T} \leftarrow \text{time\_node}(G^p_{1:t}) \cup \text{time\_node}(G^i_{1:t})$;
$\mathscr{O} \leftarrow \text{objects}(G^p_{1:t}) \cup \text{objects}(G^i_{1:t})$;
$Q = \begin{bmatrix} q_{00} & q_{01} \\ q_{10} & q_{11} \end{bmatrix} \leftarrow \mathbf{0}_{2\times 2} \quad P = \begin{bmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{bmatrix} \leftarrow \mathbf{0}_{2\times 2}$;
for $(t,o) \in \mathscr{T} \times \mathscr{O}$ **do**
  $i \leftarrow (t \in G^p_{1:t} \wedge o \in G^p_t)$;
  $j \leftarrow (t \in G^i_{1:t} \wedge o \in G^i_t)$;
  $q_{ij} \leftarrow q_{ij} + 1$;
for $t_m, t_n \in \mathscr{T}, m < n$ **do**
  $i \leftarrow \left(\exists o' \in \mathscr{O} : o' \in G^p_{t_m} \wedge o' \in G^p_{t_n}\right)$;
  $j \leftarrow \left(\exists o' \in \mathscr{O} : o' \in G^i_{t_m} \wedge o' \in G^i_{t_n}\right)$;
  $p_{ij} \leftarrow p_{ij} + 1$;
$P \leftarrow Norm_1(P)$, $Q \leftarrow Norm_1(Q)$;
return P, Q;

For each object–object pair $e \in \mathscr{E}_{\text{u}}$, the edge probabilities $P = (p_{11}, p_{10}; p_{01}, p_{00})$ are given by:

$$(G^p_t, G^i_t)(e) = \begin{cases} (1,1) & \text{w.p. } p_{11} \text{ (shared edge)} \\ (1,0) & \text{w.p. } p_{10} \text{ (disagreement)} \\ (0,1) & \text{w.p. } p_{01} \text{ (disagreement)} \\ (0,0) & \text{w.p. } p_{00} \text{ (shared non-edge)} \end{cases} \tag{5}$$

where $p_{11} + p_{10} + p_{01} + p_{00} = 1$. For each object–viewpoint pair $e \in \mathscr{E}_{\text{a}}$, a similar distribution is defined with probabilities $Q = (q_{11}, q_{10}; q_{01}, q_{00})$. Given the perceived graph G^p_t and the imagined graph G^i_t, the P, Q can be approximated by Algorithm 1. Then we define:

$$\psi_{\text{u}} \triangleq \left(\sqrt{p_{11}p_{00}} - \sqrt{p_{10}p_{01}}\right)^2, \tag{6}$$

$$\psi_{\text{a}} \triangleq \left(\sqrt{q_{11}q_{00}} - \sqrt{q_{10}q_{01}}\right)^2. \tag{7}$$

To make the navigator effective, we need to measure the confidence in the alignment between the perceived graph G^p_t and the imagined graph G^i_t. We define:

- An alignment is a mapping, or permutation, between the n objects in G^p_t and the m objects in G^i_t. We denote a specific alignment by π.
- $\mathscr{S}_n$ is the set of all $n!$ possible alignments.
- π_{id} is the identity alignment, which represents the true, correct alignment where every object in G^p_t is matched to its corresponding object in G^i_t.
- $\delta_\pi(G^p_t, G^i_t)$ is the alignment difference. This score measures how much more likely the true alignment π_{id} is compared to an incorrect alignment π. If $\delta_\pi > 0$, the previous alignment is a better fit.

A matching error occurs if there exists any incorrect alignment $\pi \in \mathscr{S}_n \setminus \{\pi_{\text{id}}\}$ that looks as good as, or better than, the true one. This happens when $\delta_\pi(G^p_t, G^i_t) \leq 0$.

Therefore the probability of ambiguous alignment is defined as

$$\text{P(A)} = \text{P}\left(\exists \pi \in \mathscr{S}_n \setminus \{\pi_{\text{id}}\}, \delta_\pi\left(G^p_t, G^i_t\right) \leq 0\right). \tag{8}$$

Calculating this probability exactly is often intractable. Instead, we rely on an upper bound from the latest graph alignment theory [17], which limits the maximum possible value of this error probability. Given $\left(G^p_t, G^i_t\right)$ with n and m nodes, the upper bound can be computed by

$$\text{P(A)} \leq e^{-2\log n + 2m\psi_{\text{a}} + 2np_{11}}. \tag{9}$$

With this upper bound on the error, the navigator confidence C_t is given by

$$C_t = 1 - \text{P(A)} \geq 1 - e^{-2\log n + 2m\psi_{\text{a}} + 2np_{11}}. \tag{10}$$

If $C_t > \tau_C$, the alignment between G^p_t and G^i_t is considered reliable with probability C_t, and the agent proceeds without invoking the costly slow system Π_{slow}, thereby achieving superior efficiency. The operation process of our SFCo-Nav is demonstrated in Algorithm 2.

V. EXPERIMENTS

In this section, SFCo-Nav is evaluated against some state-of-the-art zero-shot VLN systems. First, we introduce the experimental setup, evaluation metrics, and VLN environments used for testing. Then, experiments conducted on open datasets is presented, focusing on two key aspects: 1) the overall task success performance, and 2) the token/temporal computational efficiency of SFCo-Nav.

A. Experimental Setup

1) Evaluation Metrics: The VLN systems are evaluated on three levels: task effectiveness, token efficiency, and temporal efficiency.

Task-Level Metrics evaluate navigation effectiveness and path efficiency using standard VLN metrics [1]. Success Rate (SR) measures task completion. Success rate weighted by Path Length (SPL) measures success and efficiency. Navigation Error (NE) measures final distance to goal. Oracle Success Rate (OSR) measures trajectory quality. Trajectory Length (TL) measures path length.

Token-Level Metrics quantify the computational cost to validate efficiency gains.

- Language Tokens per Path (L-Tok): Average number of LLM tokens processed per episode, measuring slow-system cost.
- Vision Tokens per Path[1] (V-Tok): Average number of

[1]Visual token counts are measured based on GPT-4o tokenizer.

Algorithm 2: SFCo-Nav: Slow–Fast Collaborative Zero-Shot VLN

Input: Navigation instruction I
Output: Action sequence $\mathbf{A} = (a_0, a_1, \ldots, a_T)$

Initialize: $G_0^p \leftarrow \emptyset$, $H_0 \leftarrow \emptyset$;
Call Π_{slow} to identify final target o_{goal} via Eq. (1);
Generate initial plan $\mathscr{P}_0 \leftarrow f_{\text{chain}}(f_{\text{policy}}(I, H_0, G_0^p))$;
for $t \leftarrow 1$ **to** T **do**
  1. Perception & Local Graph Update:
    Build perceived object graph G_t^p from onboard sensors;
  2. Confidence Evaluation:
    Retrieve current imagined graph G_t^i from $\mathscr{P}_{t-1}$;
    Compute C_t via Eq. (10) for G_t^p vs. G_t^i;
  if $C_t \leq \tau_C$ **then**
    3a. Slow LLM Trigger:
      Append reached subgoals to H_{t-1};
      $(R_t, (o_t, sk_t)) \leftarrow f_{\text{policy}}(I, H_{t-1}, G_t^p)$;
      $\mathscr{P}_t \leftarrow f_{\text{chain}}(R_t)$;
  else
    3b. Fast Navigator Execution:
      Keep $\mathscr{P}_{t-1}$ without triggering LLM;
  4. Reactive Navigation:
    $a_t \leftarrow \pi_{\text{fast}}(G_t^p, o_t, sk_t)$; Execute a_t
    Pop the first executed item of $\mathscr{P}_{t-1}$;
  if Reached o_{goal} within d_{stop} **then**
    Terminate and return $\{a_k\}_{k=0}^t$; **break**;

vision tokens processed per episode, measuring visual perception load.

- Unified Tokens per Path (U-Tok): A combined metric representing overall token throughput, calculated as

$$\text{U-Tok} = \text{L-Tok} + \lambda \times \text{V-Tok}, \tag{11}$$

where λ is a cost coefficient that weights vision tokens relative to language tokens. Following OpenAI's reported inference costs[2], we set $\lambda = 4$ to reflect that each vision token is approximately four times more expensive to process than a text token.

Timing-Level Metrics measure wall-clock latency for real-world deployment.

- LLM Time per Path (L-Time): Total wall-clock time (s) for all LLM inference calls per episode, capturing slow-system latency.
- Vision Time per Path[3] (V-Time): Total wall-clock time (s) for all visual processing per episode, capturing visual perception latency.
- Total Time per Path (T-Time): A unified metric for the overall episode latency, calculated as

$$\text{T-Time} = \text{L-Time} + \text{V-Time}. \tag{12}$$

2) Development of Experimental Datasets: To demonstrate the effectiveness of SFCo-Nav, experiments are conducted on public visual language navigation datasets and real-world suite environment.

R2R [1]: The standard benchmark for VLN, requiring agents to follow detailed path-based instructions in photo-realistic indoor environments.

REVERIE [18]: A more challenging object-grounding benchmark where agents follow high-level, goal-oriented instructions to find a specific target object.

Real-world Suite: Deployment on a physical legged robot in an indoor hotel suite setting to test the system's robustness and practical performance under real-world conditions.

3) Compared Algorithms: SFCo-Nav is compared against latest (partial) zero-shot VLM-LLM based visual language navigation methods as follows:

- NavGPT [8]: A foundational approach that uses an LLM as a zero-shot planner, making decisions based entirely on textual descriptions of visual observations obtained by VLM.
- MapGPT [9]: This method augments the LLM planner with an explicit spatial memory by constructing a real-time topological map and feeding it to the LLM in a textual format obtained by VLM for navigation.
- NavCoT [14]: This approach utilizes Chain-of-Thought (CoT) prompting to improve reasoning by instructing the LLM to generate an explicit thought process before selecting an action, which is based on zero-shot VLM and fine-tuned LLM.
- SF-Nav: Our proposed slow–fast framework variant without the slow–fast bridge for asynchronous LLM triggering. In this setting, the slow LLM planner is invoked at every navigation step.

B. Experimental Results and Discussions

1) R2R Dataset: On the R2R benchmark, SFCo-Nav achieves a strong balance between accuracy and efficiency, as shown in Table I. It achieves competitive task success metrics that closely match or surpass prior zero-shot VLN baselines, while drastically reducing computational cost. Compared to NavGPT, MapGPT, NavCoT, SF-Nav, SFCo-Nav achieves the lowest unified token usage, thanks to its asynchronous slow–fast triggering mechanism. This reduction directly translates into the fastest total inference time, more than 4× faster than MapGPT and 33% faster than SF-Nav, without sacrificing navigation quality. These results confirm that SFCo-Nav delivers near state-of-the-art accuracy with unmatched token and time efficiency.

2) REVERIE Dataset: On the REVERIE dataset, SFCo-Nav achieves an NE of 7.16 meters and SR of 31.33%, ranking second only to SF-Nav, while outperforming all other baselines, as shown in Table II. In terms of efficiency, SFCo-Nav reduces LLM token consumption to 9.44k, a

[2]The token price of OpenAI may refer to `https://platform.openai.com/docs/pricing#image-tokens`.

[3]Visual perception time for our method is measured using Grounding-DINOv2 [24], whereas VLM-based baselines use BLIP-2.

TABLE I

COMPARISON ON THE R2R DATASET.

Method	Task Success Metric					Token Efficiency Metric			Time Efficiency Metric[s]		
	TL	NE	OSR	SR	SPL	V-Tok	L-Tok	U-Tok	V-Time	L-Time	T-Time
NavGPT (GPT-4)	11.45	6.46	42.0	34.0	29.0	108k	64.62k	496.62k	1440	48.19	1488.19
MapGPT (GPT-4)	–	6.29	**57.6**	38.8	25.8	4.95k	9.46k	29.26k	66	41.01	107.01
NavCoT (Tuned* Llama)	**9.83**	6.67	44.0	36.4	33.17	4.50k	**1.42k**	19.42k	60	**16.42**	76.42
SF-Nav (GPT-4o)	10.89	**5.70**	53.8	**41.3**	**36.35**	**0.9k**	13.10k	16.10k	**3.36**	36.0	39.36
SFCo-Nav (GPT-4o)	10.80	6.04	50.5	38.2	32.54	**0.9k**	9.94k	**13.54k**	**3.36**	22.8	**26.16**

TABLE II

COMPARISON ON THE REVERIE DATASET.

Method	Task Success Metric					Token Efficiency Metric			Time Efficiency Metric[s]		
	TL	NE	OSR	SR	SPL	V-Tok	L-Tok	U-Tok	V-Time	L-Time	T-Time
NavGPT (GPT-4)	–	–	28.3	19.2	14.6	108.00k	57.35k	489.35k	1440	49.74	1489.74
MapGPT (GPT-4)	–	–	42.6	28.4	14.5	4.95k	7.56k	27.36k	66	26.57	92.57
NavCoT (Tuned* Llama)	12.36	–	14.20	9.20	7.18	4.50k	**1.44k**	19.44k	60	**11.40**	71.40
SF-Nav (GPT-4o)	**10.05**	7.87	**46.63**	**35.87**	**30.62**	**0.9k**	14.04k	17.64k	**3.36**	31.4	34.76
SFCo-Nav (GPT-4o)	10.32	**7.16**	42.36	31.33	27.01	**0.9k**	9.44k	**13.04k**	**3.36**	21.6	**24.96**

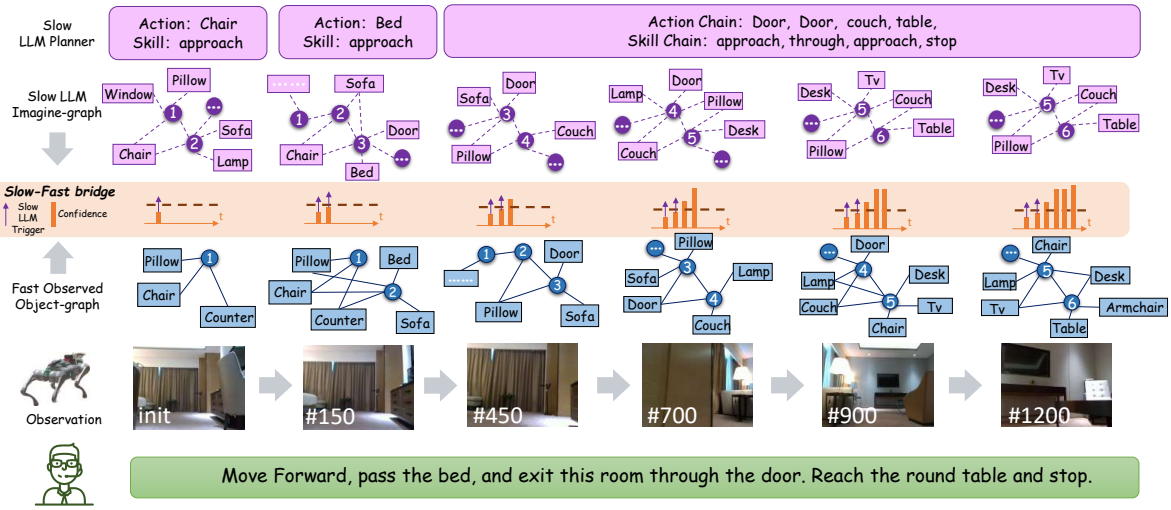

Fig. 5. Real-world hotel suite deployment of SFCo-Nav. Early in navigation, sparse observations yield low match probability, triggering the slow planner. As observations grow, confidence exceeds the threshold, enabling fast, LLM-free execution. This slow-fast collaboration preserves success while improving time efficiency and reducing token usage.

TABLE III

COMPARISON OF VARIOUS CONFIDENCE LEVEL ON R2R SUBSET.

SFCo-Nav-Confidence Threshold	Task Success Metric			Efficiency Metric	
	SR	OSR	SPL	U-Tok	T-Time[s]
SFCo-Nav-1.0	**39.67**	49	**37.08**	12.58k	28.08
SFCo-Nav-0.95	36.67	**50.33**	34.10	10.55k	22.5
SFCo-Nav-0.85	36.67	**50.33**	34.03	10.42k	22.2
SFCo-Nav-0.6	33.33	48.33	31.22	9.77k	20.34
SFCo-Nav-0.4	29.67	45	27.39	**9.51k**	**19.64**

TABLE IV

ABLATION STUDY OF SFCO-NAV ON R2R SUBSETS.

LLMs	Chain Decision	Object-Skill Pair	SR	U-Tok	T-Time[s]
GPT-4o	×	×	32	13.29k	32.90
	✓	×	31.33	10.13k	28.73
	×	✓	45.67	13.06k	31.05
	✓	✓	37	9.9k	27.47
Deepseek-V3.1	×	×	32.67	15.07k	99
	✓	×	31	12.87k	96.84
	×	✓	41	15.91k	102.24
	✓	✓	34	12.52k	97.56

33% drop compared with SF-Nav, yielding the lowest unified token usage among all methods. This compact token footprint translates into the fastest total inference time at 24.96 seconds, over 3.7× faster than MapGPT and more than 59× faster than NavGPT. These results confirm that SFCo-Nav retains competitive task performance while achieving substantial gains in computational and time efficiency.

3) Ablation Study: Ablation study is conducted on two main aspects: a) Impact of the confidence threshold. b) Imagined Chain and Object-Skill pair module ablation with different pre-trained LLM. To reduce evaluation cost, a subset of R2R including 300 instruction trajectories is used for this ablation study.

a) Impact of the confidence threshold: We investigate the impact of the confidence threshold on SFCo-Nav's performance by varying the trigger level for invoking the slow LLM planner from 1.0 to 0.4. As shown in Table III, higher thresholds generally yield stronger task success metrics but incur higher token and time costs. Specifically, SFCo-Nav-1.0 achieves the highest SR and SPL but requires 12.58k total tokens and 28.08 seconds per trajectory. Conversely, SFCo-Nav-0.4 minimizes computation, with the smallest total to-

kens and fastest runtime, but SR drops to 29.67% and SPL to 27.39%. Intermediate thresholds 0.85–0.95 offer a better balance, reducing runtime by 20% while maintaining SR within 3% of the maximum. These results highlight a clear accuracy–efficiency trade-off governed by the confidence threshold.

b) Ablation on Imagined Chain and Object–Skill Pair Modules with different LLM: We assess the impact of the Imagined Chain and Object–Skill Pair modules in SFCo-Nav using GPT-4o and Deepseek-V3.1 backbones, as shown in Table IV. Across both LLMs, the Object–Skill Pair consistently boosts SR, while the Imagined Chain mainly reduces token usage and runtime with minor SR changes. Combining both yields a balanced trade-off, for GPT-4o with the lowest token usage and fastest runtime; for Deepseek, tokens drop from 15.07k to 12.52k with only a slight SR decrease. These results indicate that the Object–Skill Pair improves navigation quality, while the Imagined Chain enhances efficiency.

4) Real-world Suite Case Study: To assess real-world deployability, we implemented SFCo-Nav on a legged robot navigating a furnished hotel suite, as depicted in Fig. 5. In early steps, few observed objects yield low match probability and confidence, triggering the slow LLM planner for planning and multi-step imagination. As observations grow, confidence surpasses the threshold. This enables the fast module to execute without additional LLM calls. This asynchronous slow–fast triggering mechanism preserves navigation success while improving time efficiency and reducing token use. It demonstrates SFCo-Nav's capability for efficient and accurate real-world navigation.

VI. CONCLUSIONS

In this work, we introduce SFCo-Nav, a slow–fast collaborative framework for zero-shot visual language navigation that couples a slow LLM-based planner with a fast reactive navigator via an asynchronous confidence-triggered bridge. Experiments on R2R and REVERIE show that SFCo-Nav achieves competitive or superior success rates with significantly reduced inference cost. A real-world deployment on a legged robot in a furnished hotel suite demonstrates its ability to adaptively invoke LLM reasoning only when necessary, maintaining navigation accuracy while improving time efficiency and lowering token usage. These results highlight SFCo-Nav's potential for practical, efficient embodied AI in real-world environments.

REFERENCES

[1] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. V. D. Hengel, "Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 3674–3683.

[2] Y. Liu, L. Liu, Y. Zheng, Y. Liu, F. Dang, N. Li, and K. Ma, "Embodied navigation," *Sci. China Inf. Sci.*, vol. 68, no. 4, p. 141101, 2025.

[3] Z. Yang, L. Li, K. Lin, J. Wang, C.-C. Lin, Z. Liu, and L. Wang, "The dawn of lmms: Preliminary explorations with gpt-4v(ision)," *arXiv preprint arXiv:2309.17421*, 2023.

[4] DeepSeek-AI, "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437*, 2025.

[5] OpenAI, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2024.

[6] Y. Qiao, W. Lyu, H. Wang, Z. Wang, Z. Li, Y. Zhang, M. Tan, and Q. Wu, "Open-nav: Exploring zero-shot vision-and-language navigation in continuous environment with open-source llms," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2025.

[7] C. Huang, O. Mees, A. Zeng, and W. Burgard, "Visual language maps for robot navigation," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2023.

[8] G. Zhou, Y. Hong, and Q. Wu, "NavGPT: Explicit reasoning in vision-and-language navigation with large language models," in *Proc. AAAI Conf. Artif. Intell. (AAAI)*. AAAI Press, 2024, pp. 849–857.

[9] J. Chen, B. Lin, R. Xu, Z. Chai, X. Liang, and K.-Y. Wong, "MapGPT: Map-guided prompting with adaptive path planning for vision-and-language navigation," in *Proc. Annu. Meet. Assoc. Comput. Linguist. (ACL)*. Bangkok, Thailand: Assoc. Comput. Linguist., 2024, pp. 9796–9810.

[10] W. Zhang, C. Gao, S. Yu, R. Peng, B. Zhao, Q. Zhang, J. Cui, X. Chen, and Y. Li, "CityNavAgent: Aerial vision-and-language navigation with hierarchical semantic planning and global memory," in *Proc. Annu. Meet. Assoc. Comput. Linguist. (ACL)*. Vienna, Austria: Assoc. Comput. Linguist., 2025, pp. 31 292–31 309.

[11] A.-C. Cheng, Y. Ji, Z. Yang, X. Zou, J. Kautz, E. Biyik, H. Yin, S. Liu, and X. Wang, "NaVILA: Legged robot vision-language-action model for navigation," in *Proc. Robot. Sci. Syst. (RSS)*, 2025.

[12] S. Chen, P.-L. Guhur, M. Tapaswi, C. Schmid, and I. Laptev, "Think global, act local: Dual-scale graph transformer for vision-and-language navigation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2022, pp. 16 537–16 547.

[13] S. Chen, P.-L. Guhur, C. Schmid, and I. Laptev, "History aware multimodal transformer for vision-and-language navigation," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2021.

[14] B. Lin, Y. Nie, Z. Wei, J. Chen, S. Ma, J. Han, H. Xu, X. Chang, and X. Liang, "Navcot: Boosting llm-based vision-and-language navigation via learning disentangled reasoning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 47, no. 7, pp. 5945–5957, 2025.

[15] J. Li, D. Li, S. Savarese, and S. Hoi, "Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models," *arXiv preprint arXiv:2301.12597*, 2023.

[16] D. Kahneman, "Thinking, fast and slow," *Farrar, Straus and Giroux*, 2011.

[17] N. Zhang, Z. Wang, W. Wang, and L. Wang, "Attributed graph alignment," *IEEE Trans. Inf. Theory*, vol. 70, no. 8, pp. 5910–5934, 2024.

[18] Y. Qi, Q. Wu, P. Anderson, X. Wang, W. Y. Wang, C. Shen, and A. van den Hengel, "Reverie: Remote embodied visual referring expression in real indoor environments," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 9982–9991.

[19] Z. Wang, J. Li, Y. Hong, Y. Wang, Q. Wu, M. Bansal, S. Gould, H. Tan, and Y. Qiao, "Scaling data generation in vision-and-language navigation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, 2023.

[20] P. Chen, X. Sun, H. Zhi, R. Zeng, T. H. Li, G. Liu, M. Tan, and C. Gan, "A^2 nav: Action-aware zero-shot robot navigation by exploiting vision-and-language ability of foundation models," in *Proc. NeurIPS Workshops*, 2023, arXiv:2308.07997.

[21] B. Lin, Y. Nie, Z. Wei, Y. Zhu, H. Xu, S. Ma, J. Liu, and X. Liang, "Correctable landmark discovery via large models for vision-language navigation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 12, pp. 8534–8548, 2024.

[22] H. Xue, J. Ren, W. Chen, G. Zhang, Y. Fang, G. Gu, H. Xu, and C. Lu, "Reactive diffusion policy: Slow-fast visual-tactile policy learning for contact-rich manipulation," in *Proc. Robotics: Sci. Syst. (RSS)*, 2025.

[23] Z. Zhan, L. Yu, S. Yu, and G. Tan, "Mc-gpt: Empowering vision-and-language navigation with memory map and reasoning chains," *arXiv preprint arXiv:2405.10620*, 2024.

[24] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, Q. Jiang, C. Li, J. Yang, H. Su, J. Zhu, and L. Zhang, "Grounding dino: Marrying dino with grounded pre-training for open-set object detection," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*. Springer Nature Switzerland, 2024, pp. 38–55.