Schema-Guided Scene-Graph Reasoning Based on Multi-Agent Large Language Model System

Yiye Chen[1,3*], **Harpreet S. Sawhney**[2†], **Nicholas Gydé**[3], **Yanan Jian**[4†], **Jack Saunders**[5,3*], **Patricio Vela**[1], **Benjamin E Lundell**[6†]

[1]Georgia Institute of Technology
[2]Amazon Robotics
[3]Microsoft
[4]Nvidia
[5]University of Bath
[6]ARM

{yychen2019, pvela}@gatech.edu, hasawhne@amazon.com, gydenicholas@microsoft.com, yajian@nvidia.com, jrs68@bath.ac.uk, ben.lundell@arm.com

Abstract

Scene graphs have emerged as a structured and serializable environment representation for grounded spatial reasoning with Large Language Models (LLMs). In this work, we propose ***SG*^{2}**, an iterative **S**chema-**G**uided **S**cene-**G**raph reasoning framework based on multi-agent LLMs. The agents are grouped into two modules: a (1) *Reasoner* module for abstract task planning and graph information queries generation, and a (2) *Retriever* module for extracting corresponding graph information based on code-writing following the queries. Two modules collaborate iteratively, enabling sequential reasoning and adaptive attention to graph information. The scene graph schema, prompted to both modules, serves to not only streamline both reasoning and retrieval process, but also guide the cooperation between two modules. This eliminates the need to prompt LLMs with full graph data, reducing the chance of hallucination due to irrelevant information. Through experiments in multiple simulation environments, we show that our framework surpasses existing LLM-based approaches and baseline single-agent, tool-based Reason-while-Retrieve strategy in numerical Q&A and planning tasks.

Introduction

With Large Language Models (LLMs) showing remarkable prowess and versatile skills across a wide range of domains, recent research has increasingly focused on grounding the LLMs their reasoning in situated environments. Scene graphs have emerged as a scalable, high-level environment representation for LLM-based spatial reasoning and planning, showing effectiveness in both simulation-based (Yang et al. 2025b) and real-world applications (Gu et al. 2024; Ni et al. 2023; Cheng et al. 2024). Prior work has explored graphs-as-text as the LLM input for the single generation with various prompt guidance (Fatemi, Halcrow, and Perozzi 2024; Gu et al. 2024), categorized as *"Reason-only"* methods in Figure. 1. A more advanced strategy, *"Retrieve-then-Reason"* (Luo et al. 2024; Sun et al. 2023; Rana et al. 2023), improves upon this by using the LLM agent to trim the graph first for retaining only task-relevant subgraph before reasoning. Despite the effort, LLMs still frequently hallucinate wrong solutions (Wang et al. 2023; Rana et al. 2023), underscoring the need for continued research on the intersection of LLMs and scene graphs. We argue that this difficulty stems from a well-documented property of LLMs: that they are easily distracted by redundant information (Adlakha et al. 2024; Bruno et al. 2023). This limitation is problematic in spatial tasks, where the reasoning process involves sequential, step-wise attention shifts over the graph. It suggests that the majority of graph data could be irrelevant at any intermediate step of reasoning, which might degrade LLMs' performance.

A promising solution to the problem is the *Reason-while-Retrieve* strategy, which enables dynamic attention on the information by iteratively carrying out the two steps (Jiang et al. 2023; Press et al. 2022). We first explore an iterative solution based on ReAct (Yao et al. 2022), where scene graph API(s) are curated as graph data access "actions" to enable interleaved task solving and graph information retrieval. While this solution is effective, we observe that it is highly sensitive to the design of available APIs (or tools). In particular, fixed-capacity APIs severely constrain the access patterns on the graph. When the API set lacks expressiveness, the agent is forced to invoke more API calls, resulting in inefficient exploration on the graph. What's more, this inefficiency is amplified by the single-agent nature of ReAct, where the retrieval and reasoning is not decoupled. Since the entire task solving history is re-prompted back to the agent in a loop, redundant context accumulation can impair future reasoning or retrieval steps (Chiang and Lee 2024; Wu et al. 2024).

To mitigate this issue, we develop a **S**chema-**G**uided **S**cene-**G**raph reasoning approach based on multi-agent architecture, (dubbed ***SG*^{2}**). The framework is comprised of two modules: a *Reasoner* module that decomposes the task and generates information queries for subsequent steps; and a *Retriever*

[*]Work done while working as an intern at Microsoft.
[†]Work done while working at Microsoft.

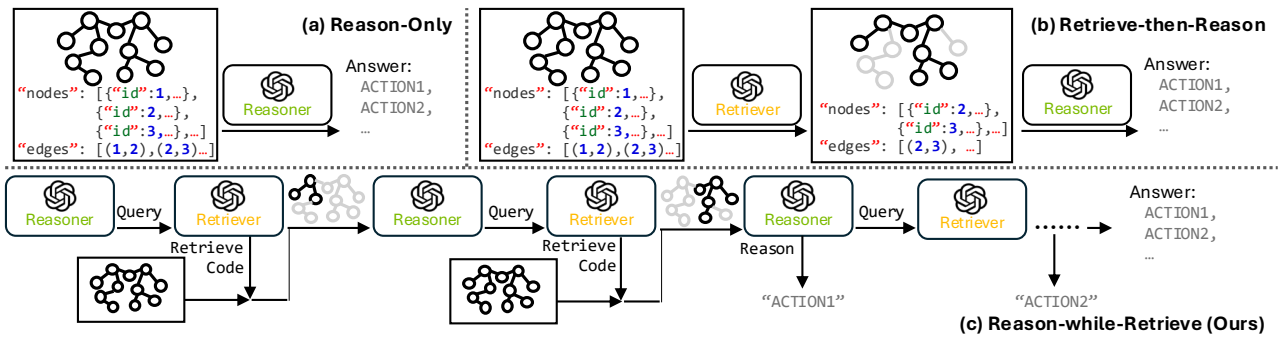


Figure 1: (a) Reason-Only: A Reasoner is directly prompted with a full textualized graph. (b) Retrieve-then-Reason: A Retriever filters out a task-related sub-graph followed by another Reasoner processing the remaining of the graph. (c) Reason-while-Retrieve (Ours): A Reasoner and a Retriever collaborate in solving a task by attending to the graph dynamically based on the progress in solving the task.

module that processes the queries and retrieves related graph information for the Reasoner. One key innovation in our approach is the incorporation of *scene graph schema*. Treating a scene graph as a specific instance of an abstract schema, we frame the role of the Retriever as *graph database query language generation*—that is, translating the natural language queries into executable graph programs. The query language is then executed on the scene graph to obtain the queried information thereby filtering out irrelevant data. Furthermore, the schema is also prompted to the Reasoner for support abstract, structure-aware reasoning, and to guide the generation of schema-aligned natural language queries for the Retriever. The multi-agent design is essential for the autonomous operation of our system: where the inter-module split ensures explicitly separation of the reasoning and retrieval, while intra-module agents collaborate to enhance the reliability of both reasoning and programming-based retrieval.

We evaluate our method with two simulation environments: BabyAI (Chevalier-Boisvert et al. 2018), a 2D grid world environment; and VirtualHome (Puig et al. 2018), a large-scale indoor multi-room environment. Our experiments on numerical Q&A and planning tasks show that SG^2 greatly improves the reasoning ability of LLMs on scene graphs, outperforming ReAct and graph prompting baselines in all evaluations. What's more, we show that even being constrained to the same static APIs with limited functionalities, our multi-agent framework still achieves better results compared to ReAct. In summary, our contributions include:

- Exploring Reason-while-Retrieve framework with reasoning-oriented information gathering mechanism for task solving on scene graphs.
- A schema-based multi-agent approach that enables dynamic information retrieval with LLM programming and decoupled reasoning and retrieval.
- Showing the efficacy of the proposed SG^2, which achieves superior performance in two distinct environments that encompass a wide range of tasks.

Related Literature

Language models for Task and Motion Planning Existing efforts harness the power of large language models for decision making (Chen et al. 2023; Liu et al. 2023) and robotic control (Dalal et al. 2024; Zhang et al. 2023; Lin et al. 2023; Chen et al. 2021). With rich built-in knowledge and in-context learning ability, LLMs are used for generating task-level plans (Raman et al. 2022; Gao et al. 2024), action selection (Ahn et al. 2022; Nasiriany et al. 2024), and processing environmental or human feedback (Skreta et al. 2023). To ground to the environment, recent studies have explored using LLMs for programmatic plan generation (Singh et al. 2023), combining knowledge from external perception tools (Liang et al. 2023; Huang et al. 2023), and value function generation (Yu et al. 2023). While proven effective, those methods are limited to small scale environments, and rely on expert perception models to extract task-related states from implicit spatial representation. In this work, we study using pretrained LLMs to process the the global representation of large environments with explicit structure.

Graph as the Scene Representation The scope of the solvable task is largely determined by the state representation. Compare to sensory representation such as images or point clouds, scene graphs are compact thus scalable to large environments (Greve et al. 2024), structured to represent spatial layout explicitly (Hughes, Chang, and Carlone 2022; Wu et al. 2021), and efficient in representing diverse states of the environment (Armeni et al. 2019). Therefore, they have been used in various manipulation or navigation tasks (Ravichandran et al. 2022; Zhu et al. 2021). In this paper, we exploit these favorable features of the scene graph representation to ground the reasoning process of LLMs to the environment.

LLMs for Reasoning on Graph Leveraging language models to reason with graphs is a growing area. While prior works trains to integrates graph and language knowledge (Ye et al. 2023; Ni et al. 2023), recent study explores serializing graph-structured data as prompts for pretrained LLMs (Wang et al. 2023; Fatemi, Halcrow, and Perozzi 2024). This strategy has been successfully used in knowledge-graph-enhanced

LLMs reasoning (Sun et al. 2023; Luo et al. 2024) and scene-graph-based robotic task planning (Gu et al. 2024). Closest to our work, SayPlan (Rana et al. 2023) prompts scene graphs to LLMs and designs a Retrieve-then-Reason framework for robotic planning. However, its room-by-room retrieval heuristic is only effective in the object search task. Instead, we design the *SG²* framework for general spatial reasoning with scene graphs.

Method

Problem Statement

Our problem setting involves a natural language task instruction I and a scene graph $\mathcal{G} = (V, E)$, where V and E denote vertices and edges, respectively. Each node V_i represents an item with its attributes, such as coordinates or colors, while each edge indicates a type of spatial relationship, such as inside or on top of. Additionally, we assume access to the *scene graph schema* $\mathcal{S}$, which is a textual description of types, formats, and the semantics of the graph vertices and edges. Our objective is to generate the solution of I using LLMs, based on the available information above, expressed as $\mathcal{A} = f(I, \mathcal{G}, \mathcal{S}; LLMs)$.

Overview of *SG²*

We explore grounding the task solving to scene graphs **based on the scene graph schema** $\mathcal{S}$. We develop *SG²*, a schema-guided multi-agent framework that iteratively reasons through the next steps and retrieves the necessary information from the graph. As shown in Figure 2, our method contains two multi-agent modules: a *Reasoner* and a *Retriever*. Both modules consist of two LLM agents: the Reasoner is comprised of a Task Planner and a Tool Caller, whereas the Retriever is comprised of a Code Writer and a Verifier. Given a task, the Reasoner determines the next substep to approach the task and identifies necessary scene graph information. It then raises a natural language query to the Retriever for this information. Upon receiving the query, the Retriever processes the scene graph through code-writing and sends the data back to Reasoner. By iteratively performing these steps, both modules collaborate to solve the task. Prompted with the schema $\mathcal{S}$, both modules in our approach are NOT contextualized with the graph data $\mathcal{G}$, which differs from prior graph reasoning methods. Formally, at each time step t:

$$a_t, q_t = Reasoner(\{a_0, q_0, \mathcal{G}'_0\}, \{a_1, q_1, \mathcal{G}'_1\}, \cdots; \boldsymbol{\mathcal{S}}) \quad (1)$$

$$h_t = Retriever(q_t; \boldsymbol{\mathcal{S}}) \quad (2)$$

$$\mathcal{G}'_t = h_t(\mathcal{G}) \quad (3)$$

where a denotes the Reasoner's internal analysis; q represents queries for the graph information; h denotes the retrieval program following the query; and $\mathcal{G}'$ refers to the retrieved information by executing the code on the scene graph $\mathcal{G}$.

Importantly, our method differs from previous iterative methods such as ReAct (Yao et al. 2022) in the following ways: (1) Our method is conditioned on the schema input instead of API annotations; (2) Our method programs to retrieve information instead of relying on provided APIs, which is more flexible and efficient; (3) Our method, powered by multi-agent designs, separates the reasoning the graph exploration processes. As we show in the experiment section, these designs improve the efficacy of our approach and robustness against limited API capacity.

The remaining of the section describes the multi-agent workflow as well as the roles of each agent.

Reasoner Module

The **Task Planner** is the central agent steering the task-solving iterations based on the scene graph schema $\mathcal{S}$. It takes as input the task I and schema $\mathcal{S}$ and initiates the problem solving process. Initially, without any knowledge about the graph data, Task Planner analyzes I and $\mathcal{S}$, and sends out the first associated query q_0 to the Retriever. At the t^{th} round of conversation, it consumes past analyses, queries, and retrieved information, and then generates one of the three types of responses, each of which is sent to a different recipient agent: (1) QUERY: querying for more information from the Retriever. This response is sent to the Retriever-side Code Writer; (2) TOOL-CALL: calling a reasoning tool to process collected information. This response is sent to Tool Caller; (3) SOLUTION: generating the solution, which terminates the task solving process. The TOOL-CALL invokes provided reasoning tools to solve complex spatial sub-problems. This is motivated by previous literature revealing the inability of to reliably solve quantitative problems (Ahn et al. 2024). To circumvent the deficiency, we follow prior work (Schick et al. 2024; Paranjape et al. 2023) to enable tool-use by providing Task Planner with annotations of programmatic functions, such as `traverse_room` for solving navigation problem in Figure. 2, so that it is able to suggest suitable tools and corresponding arguments to address atomic problems critical to the given task family.

Concretely, we prompt Task Planner to generate the following outputs at each step:

Explanation: Summarize the reasoning process and justify the generation of the current response.

Mode: The type of the current response.

Content: The detailed message in the current response, such as the desired graph information for QUERY; tool name and arguments for TOOL-CALL; or the final answer for SOLUTION.

We filter out only the **Content** message for the recipient agent, reducing the interference by the redundant reasoning process behind the request. Few-shot examples are prompted to Task Planner to enhance the performance in task solving and enforce the output format.

The schema prompt $\mathcal{S}$ is critical by serving two purposes. First, it leads the Task Planner to reason the task *abstractly*, reducing hallucination due to task-irrelevant graph information (Wang et al. 2023). What's more, it streamlines the Reasoner-Retriever collaboration by guiding the generation of query message, ensuring that it is parsable by the Retriever.

The **Tool Caller** is prompted with the reasoning tool annotations, and is responsible for translating the tool-calling messages from Task Planner into executable python programs. We observe that Task Planner along might not be able to invoke tools with the correct format. Hence we split the burden

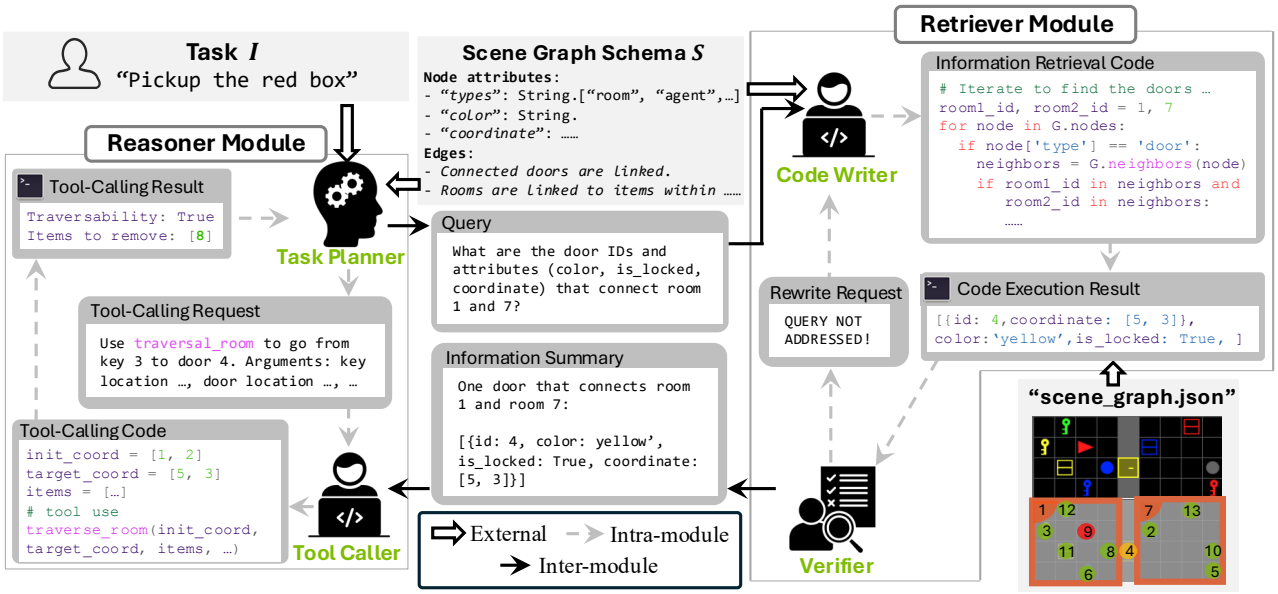


Figure 2: SG^2 solves tasks on scene graphs through the iterative collaboration of two LLM-based multi-agent modules: the Reasoner and the Retriever. The *Scene Graph Schema* lies at the core of the approach, guiding the Reasoner to break down the problem and formula queries, and enabling the Retriever to generate code that efficiently processes the scene graph and fulfills those queries. LLM agents are color coded.

and use a separate Tool Caller agent for formatting. While other structured output techniques exist (Liu et al. 2024), we find adding another agent to be most flexible and sufficient for our application.

Retriever Module

The **Code Writer** processes the query q from Reasoner and, conditioned on the graph schema prompt $\mathcal{S}$, generates the code h to process scene graphs programmatically. The schema and the query language enable the Code Writer to compose low-level APIs, introduce control-flow, and generally convert the natural language queries into executable program to run on the scene graph. This code-writing strategy offers significant advantages over traditional API-calling methods. By enabling efficient graph traversal for query-oriented information filtering, the irrelevant parts of the graph never enter the retriever's context window and so retrieved information is well-aligned with the reasoning demands.

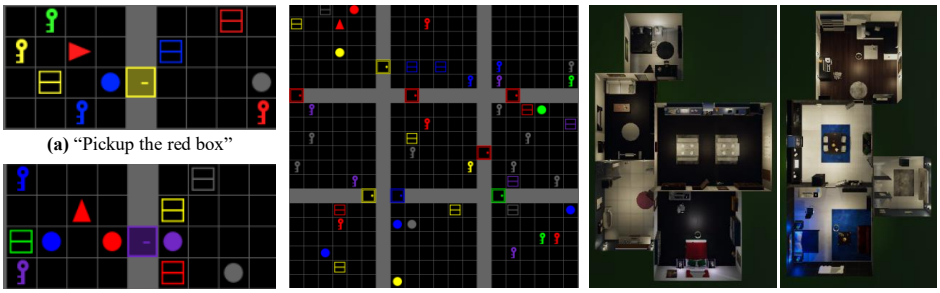
Following prior work (Chen et al. 2024), we incorporate a self-debugging mechanism to address possibility that LLMs may generate unexecutable code even with sufficient context. Specifically, we iteratively re-prompt the past code-writing attempts and execution errors back to Code Writer until successful execution is achieved.

Even with error prevention mechanism, the final code execution might not produce valid results for the query due to multiple possibilities, such as missing result output or scattered graph information output along the multiple code-rewriting attempts. To mitigate the issue, we introduce the **Verifier** agent to evaluate the code execution results. It takes as input the information retrieval query as well as all past code execution results, and determines if the query is addressed. If the query is deemed addressed, it prompts the Code Writer to re-write the code. Otherwise, it summarizes the result and sends it back to the Reasoner.

Note unlike the single-agent ReAct method, our multi-agent pipeline naturally filters the context exploiting the conditional independence structure inherent in the task solving process. For example, Code Writer generates the program h solely based on the query q without full reasoning history, and the Task Planner receives only the retrieved graph data $\mathcal{G}'$ without being exposed to code generation and correction details. This ensures that each agent operates strictly within their designated responsibility, free from irrelevant distractions that could impair their response (Yoran et al. 2023).

Experimental Settings

We evaluate our method on a series of numerical Q&A (NumQ&A) and planning tasks, which require both global and local spatial reasoning, in the BabyAI (Chevalier-Boisvert et al. 2018; Chevalier-Boisvert et al. 2023) and VirtualHome (VH) (Puig et al. 2018) environments. For each environment, we provide an unified scene graph schema consistent across epoches with distinct scene graphs. Each task requires reasoning on both the spatial structure and the semantic information encoded in the graph. For evaluation metric, we use the **success rate**, defined as the ratio of the trials where the task solving is successful. The success is defined as either providing the correct answer for the Q&A tasks or achieving the desired outcome for the planning tasks. Note that all experiments in this paper are conducted in the static setting, where the tested methods generate solutions solely



(a) "Pickup the red box"

(b) "Pickup the yellow box"

(c) "Find the color of the ball in a room next to the room with 4 grey keys"

(d) "Put the soap in the bathroomcabinet"

Figure 3: Evaluation environments and tasks. **(a)** BabyAI Trv-1 task with single-side door obstacle; **(b)** BabyAI Trv-2 task with double-side door obstacles; **(c)** BabyAI Numerical Q&A task; **(d)** Two VirtualHome household environments (left: VH-1; right: VH-2) and an examplar task.

based on the initial scene graph without interacting with the environment or modifying the graph.

Unless otherwise specified, we use GPT-4o as the backbone LLM for all methods. We implement SG^2 with AutoGen (Wu et al. 2023). For all methods, we set both temperature and random seed to 0.

Baselines Following NLGraph (Wang et al. 2023), we compare our approach with several direct graph prompting methods including: **zero-shot prompting** (ZERO-SHOT), **Zero-Shot Chain-of-Thought** (0-COT) (Kojima et al. 2022), **Least-to-Most** (LTM) (Zhou et al. 2022), **Chain-of-Thought** (COT) (Wei et al. 2022), **Build-a-Graph** (BAG) (Wang et al. 2023), **Algorithmic Prompting** (ALGORITHM) (Wang et al. 2023). In addition to the few-shot examples, ALGORITHM also requires a language description of the task solving method. We also compare against **SayPlan** (Rana et al. 2023), a retrieve-then-reason baseline specifically designed for the scene graphs, and **ReAct** (Yao et al. 2022), a generic iterative reasoning and acting approach that invokes database APIs to aggregate information. Since SayPlan does not release the source code, we evaluate with our implementation of the method. For ReAct, we curate a graph traversal tool `expand(nodeID)` to retrieve attributes of a specified node plus the IDs and attributes of all its neighbor nodes. We also provide it with any reasoning tools available to SG^2 depending on the task. We annotate few-shot examples involving detailed task solving process for both SayPlan and ReAct following their format.

Following subsections summarize the environment and task designs.

2D Grid World Numerical Q&A

Our first experiment is on a numerical Q&A task in a customized 9-room 2D BabyAI (Chevalier-Boisvert et al. 2018) environment, as shown in Figure 3(c). We generate scene graph representation of the environment following the hierachical graph design from 3DSG (Armeni et al. 2019), illustrated in Figure 4. Specifically, the graph represents the spatial scene layout through three levels: root, rooms, and objects, with additional door nodes connecting room pairs.

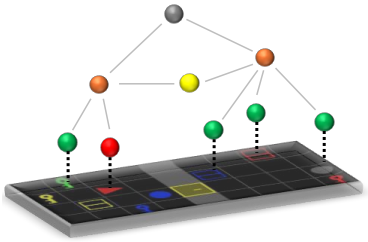


Figure 4: BabyAI Scene Graph Representation. Graph nodes represent items, agents, rooms, and doors. Edges indicate items or agents located inside a room, or doors that connect rooms. Room nodes are connected to a root node.

Following SayPlan (Rana et al. 2023), we design the following question template: `find the color of the {TARGET_OBJECT} in a room next to the room with {NUM_IDENTIFIER} {COLOR_IDENTIFIER} {IDENTIFIER_OBJECT}`, where contents in curley brackets are populated based on each environment instance. The environment and question pairs are designed to ensure that there is only one answer.

We test each method in 100 task instances. We manually annotate the few-shot demonstrations for few-shot methods. We also annotate the task solving process for SayPlan, ReAct, and SG^2 following their respective response formats.

2D Grid World Traversal Planning

We also test on the traversal planning in BabyAI, where the task is to generate a sequence of node-centric actions to pick up a target item. We design three atomic actions, including (1) `pickup(nodeID)`: Walk to and pickup an object by the node ID; (2) `remove(nodeID)`: Walk to and remove an object by the node ID; (3) `open(nodeID)`: Walk to and open a door by the node ID.

As shown in Figure 3(a)(b), the traversal planning task is tested in two related double-room environments, both of which require the agent to pick up the key of the correct color to unlock the door, remove any obstacle that blocks the

door, open the door, and pick up the target. The difference is that the first environment, dubbed **Trv1**, contains only the agent-side obstacle, whereas the second environment, dubbed **Trv2**, contains another target-side obstacle. We generate the in-context examples *only in Trv1* , and test if the methods can extrapolate to Trv2. As before, we evaluate each method in 100 times in different instance of both types of the environment. For *SG²* and ReAct, we provide the reasoning tool `traversal_room` programmed based on the A^* algorithm, which identifies the item to remove in order to reach from an initial to a desired location within the same room.

Household Task Planning

The last evaluation is in two VirtualHome (VH) (Puig et al. 2018) environments shown in Figure 3(d), denoted as **VH-1** and **VH-2**, respectively. We use the built-in environmental graph as the scene graph. Compared to BabyAI, VH environments have larger state space and action space, containing 115 object instances, 8 relationship types, and multiple object properties and states. Hence, VH environments are more challenging with richer information in the graphs. For each environment, we adopt the 10 household tasks from ProgPrompt (Singh et al. 2023), such as "`put the soap in the bathroom cabinet`", and query each method for the action sequence in the VH action format to accomplish the task. We use two of the tasks, together with the ground truth actions, as the few-shot examples, and test with the other eight. We follow CoELA (Zhang et al. 2024) to specify the task as the desired states. For example, the task of above is specified as `soap INSIDE bathroomcabinet`. Due to the unavailability of reasoning process annotation behind the solution, we do not include it in the few-shot prompt for *SG²*, and do not test ReAct and SayPlan as they do not work well without the demonstrations.

Results and Analysis

Experiment Results

Numerical Q&A Resutls The results are tabulated in Table 1. For baselines, few-shot CoT underperforms even compared to the zero-shot counterpart 0-CoT, suggesting that few-shot prompts do not show consistent effect in this task. We observe that although LLMs can imitate the reasoning trace, solving atomic spatial tasks, such as counting the item or locating the neighboring rooms, is not straightforward for LLMs when processing large graphs as text. On the other hand, ReAct outperforms other graph prompting baselines by at least 19%, showing the effectiveness of Reason-while-Retrieve strategy in addressing the aforementioned issue. However, ReAct is still limited by the API-based information retrieval, requiring multiple calls for simple sub-problem such as "finding a room with 4 green balls". In contrast, the program-based information retrieval in our method is able to solve the sub-problem, and the multi-agent framework ensures that the reasoning is not misdirected by the long program generated by the Retriever. Both factors attributes to the 12% performance advantage of our method *SG²* agains ReAct.

2D Traversal Results Table 1 also reports the success rate in the traversal task. The few-shot prompts demonstrate more advantage in this task, showing as CoT outperforms 0-CoT by 4% and BAG achieving the best performance compared to other graph prompting methods under the in-domain Trv-1. However, the advantage does not extrapolate with slight domain change. In Trv-2, those few-shot methods all underperform compared to zero-shot methods, with CoT, BAG, and SayPlan even dropping to 0%. This again suggest that graph prompting is not an effective solution for scene graph reasoning. On the other hand, ReAct and our method again show better capacity, achieving more than 20% lead in success rate compared to other methods. While our method still outperforms ReAct, the gap is small, with only 1% or 3%.

Household Task Planning Results The planning success rate on the 8 tasks in the 2 VH environments are shown in Table 2. We observe that all baselines consistently fail to address the precondition of the planned action. For example, all of them failed to generate `[open] <garbagecan> (ID)` before `[putin] <plum> (ID) <garbagecan> (ID)`, forgetting that the state of the garbage can is `state:{CLOSED}` from the extensive graph input. On the other hand, *SG²* doesn't process the entire graph. Instead, it queries for the specific object information, which helps to better determine the action parameter and examine the action preconditions.

SG² v.s. ReAct

Despite the same iterative Reason-while-Retrieve strategy, our method differs from ReAct in program-based graph interaction, schema contextualization, and the multi-agent design that separates reason and retrieve contexts. In this section, we justify our designs through ablation on the following variants:

- **ReAct-limit**: This variant is the ReAct with weaker graph traversal APIs. We weaken the `expand(nodeID)` API to `get_neighbors(nodeID)` and `get_attrs(nodeID)`, which obtains only the IDs of neighbor nodes and attributes of a specific node, respectively. In this way, each function obtains less information, requiring more API calls to aggregate information for reasoning. This variant examines ReAct's sensitivity to the API capacity, which determines its compatibility with different graph databases. In our case, the `get_neighbors` and `get_attrs` are directly provided by the NetworkX library, whereas `expand` requires manual curation.
- ***SG²*-limit**: It is a variant of *SG²* without programming-based information retrieval. Instead, the Retriever is redesigned as a *ReAct-limit* agent, relying only on graph APIs of limited capacity. Differs ReAct-limit in keeping the multi-agent design, this test verifies the efficacy of the Reason-Retrieve separation without programming.

All variants are tested in BabyAI tasks.

Results The results are collected in Table 3. Compared to ReAct, the performance of ReAct-limit drops significantly. The success rates of all three tasks are lowered by more than 36% solely due to the breakdown of the API function. This result verifies the drawback of ReAct, which is its over-reliance on the API quality. In the case where the information gathered from APIs has larger semantic gap to the reasoning demand,

	Zero-Shot			Few-Shot					
Task	ZeroShot	0-CoT	LTM	CoT	BAG	Alg	SayPlan	ReAct	*SG*^{2}
NumQ&A	29%	60%	52%	56%	49%	67%	35%	86%	98%
Trv-1	20%	50%	63%	54%	71%	45%	18%	94%	97%
Trv-2	13%	16%	20%	0%	0%	11%	0%	95%	96%

Table 1: BabyAI evaluation results. *SG*^{2} outperforms all baseline methods, showing efficacy of our design.

Method	VH-1	VH-2
ZeroShot	7/8	6/8
0-CoT	7/8	6/8
LTM	7/8	5/8
CoT	7/8	6/8
BAG	7/8	5/8
***SG*^{2}**	**8/8**	**8/8**

Table 2: VirtualHome evaluation results. The number of accompolished tasks out of 8. The superior performance of *SG*^{2} shows its practicality in realistic environments.

increased number of API calls are necessary. Without separating the reasoning and retrieval history, the context for both stages build up with more iteration steps, leading to higher chance of hallucination. On the other hand, even with the same set of graph APIs, *SG*^{2}-limit still outperforms ReAct-limit on all three tasks, with more 35% gap on both traversal tasks. This validates the importance of distributing reasoning and retrieval to multiple agents, which effectively filter the context based on the functionality of each component.

Small Language Models Performance

We conduct studies on the choice of LLM backbone, especially of the open-source Small Language Models (SLMs). Specifically, we test *SG*^{2}, together with ZeroShot, 0-CoT, LTM, CoT, BAG, Alg, and ReAct baselines, with Phi4-14B (Abdin et al. 2024), Qwen3-14B (Yang et al. 2025a), and DeepSeek-7B (Bi et al. 2024) models on the BabyAI NumQ&A task. Each method is tested with 20 trials and the success rate is collected.

Results The results are illustrated in the Figure. 5. The performance of all baseline models drop significantly with SLMs, with the best success rate being only 30% with the Phi4-14B model and less than 20% with the Qwen3-14B or deepseek-7B, suggesting the weak ability of SLMs to comprehend graph structure with textual inputs. On the other hand, despite the equally poor performance of our method with Qwen3-14B or deepseek-7B, *SG*^{2} achieves 60% success rate with Phi-14B, which doubles compared to even the best-performing baseline. This suggests that reasoning abstractly with graph schema might be a simpler compared to comprehending the entire textualized graph for SLMs, which shows the potential of utilizing multi-agent and code-writing in graph reasoning tasks.

Method	Num Q&A	Trv-1	Trv-2
ReAct-limit	40%	58%	11%
ReAct	86%	94%	95%
***SG*^{2}-limit**	47%	93%	70%
***SG*^{2}**	98%	97%	96%

Table 3: ReAct vs. *SG*^{2}. *SG*^{2}-limit outperforms ReAct-limit, showing the benefit of multi-agent system. *SG*^{2} further improves the performance, validating the design of graph processing with schema-guided abstract programs.

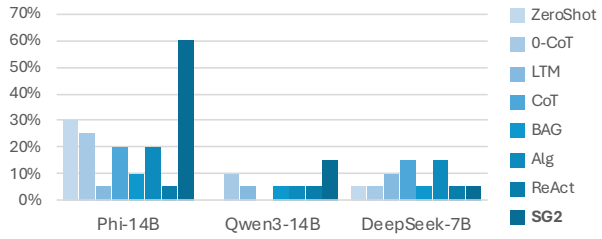


Figure 5: Small Language Models (SLMs) results in the BabyAI NumQ&A task.

Conclusion

In this work, we propose *SG*^{2}, a schema-guided, multi-agent framework performs iterative reason-while-retrive on scene graphs. The use of schema induces the idea of separating logic from the data, allowing the Reasoner to perform task planning abstractly, and the Retriever to write symbolic code for accessing necessary data while filtering irrelevent information. The multi-agent design facilitates the realization of the above process, by separating the reason and retrieve process along with their contextual inputs. Our experiments show that our method achieves the best results on all tested benchmarks. What's more, we verify the efficacy of both the schema-guided programming as well as the multi-agent designs through ablation studies.

Future work could explore the flexibility of *SG*^{2} framework to seamlessly integrate additional agents with new specialties, such as modality agent to process richer information. Reasoning trace optimization could also be explored, as the conversation rounds scale with task difficulty and agent numbers.

References

Abdin, M.; Aneja, J.; Behl, H.; Bubeck, S.; Eldan, R.; Gunasekar, S.; Harrison, M.; Hewett, R. J.; Javaheripi, M.; Kauffmann, P.; et al. 2024. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*.

Adlakha, V.; BehnamGhader, P.; Lu, X. H.; Meade, N.; and Reddy, S. 2024. Evaluating correctness and faithfulness of instruction-following models for question answering. *Transactions of the Association for Computational Linguistics*, 12: 681–699.

Ahn, J.; Verma, R.; Lou, R.; Liu, D.; Zhang, R.; and Yin, W. 2024. Large language models for mathematical reasoning: Progresses and challenges. *arXiv preprint arXiv:2402.00157*.

Ahn, M.; Brohan, A.; Brown, N.; Chebotar, Y.; Cortes, O.; David, B.; Finn, C.; Fu, C.; Gopalakrishnan, K.; Hausman, K.; et al. 2022. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*.

Armeni, I.; He, Z.-Y.; Gwak, J.; Zamir, A. R.; Fischer, M.; Malik, J.; and Savarese, S. 2019. 3d scene graph: A structure for unified semantics, 3d space, and camera. In *Proceedings of the IEEE/CVF international conference on computer vision*, 5664–5673.

Bi, X.; Chen, D.; Chen, G.; Chen, S.; Dai, D.; Deng, C.; Ding, H.; Dong, K.; Du, Q.; Fu, Z.; et al. 2024. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*.

Bruno, A.; Mazzeo, P. L.; Chetouani, A.; Tliba, M.; and Kerkouri, M. A. 2023. Insights into Classifying and Mitigating LLMs' Hallucinations. *arXiv preprint arXiv:2311.08117*.

Chen, H.; Du, Y.; Chen, Y.; Tenenbaum, J.; and Vela, P. A. 2023. Planning with sequence models through iterative energy minimization. *arXiv preprint arXiv:2303.16189*.

Chen, X.; Lin, M.; Schärli, N.; and Zhou, D. 2024. Teaching Large Language Models to Self-Debug. In *The Twelfth International Conference on Learning Representations*.

Chen, Y.; Xu, R.; Lin, Y.; and Vela, P. A. 2021. A joint network for grasp detection conditioned on natural language commands. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 4576–4582. IEEE.

Cheng, A.-C.; Yin, H.; Fu, Y.; Guo, Q.; Yang, R.; Kautz, J.; Wang, X.; and Liu, S. 2024. SpatialRGPT: Grounded Spatial Reasoning in Vision Language Model. *arXiv preprint arXiv:2406.01584*.

Chevalier-Boisvert, M.; Bahdanau, D.; Lahlou, S.; Willems, L.; Saharia, C.; Nguyen, T. H.; and Bengio, Y. 2018. Babyai: A platform to study the sample efficiency of grounded language learning. *arXiv preprint arXiv:1810.08272*.

Chevalier-Boisvert, M.; Dai, B.; Towers, M.; Perez-Vicente, R.; Willems, L.; Lahlou, S.; Pal, S.; Castro, P. S.; and Terry, J. 2023. Minigrid & Miniworld: Modular & Customizable Reinforcement Learning Environments for Goal-Oriented Tasks. In *Advances in Neural Information Processing Systems 36, New Orleans, LA, USA*.

Chiang, C.-H.; and Lee, H.-Y. 2024. Over-Reasoning and Redundant Calculation of Large Language Models. arXiv:2401.11467.

Dalal, M.; Chiruvolu, T.; Chaplot, D.; and Salakhutdinov, R. 2024. Plan-seq-learn: Language model guided rl for solving long horizon robotics tasks. *arXiv preprint arXiv:2405.01534*.

Fatemi, B.; Halcrow, J.; and Perozzi, B. 2024. Talk like a Graph: Encoding Graphs for Large Language Models. In *International Conference on Learning Representations*.

Gao, J.; Sarkar, B.; Xia, F.; Xiao, T.; Wu, J.; Ichter, B.; Majumdar, A.; and Sadigh, D. 2024. Physically grounded vision-language models for robotic manipulation. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 12462–12469. IEEE.

Greve, E.; Büchner, M.; Vödisch, N.; Burgard, W.; and Valada, A. 2024. Collaborative dynamic 3d scene graphs for automated driving. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 11118–11124. IEEE.

Gu, Q.; Kuwajerwala, A.; Morin, S.; Jatavallabhula, K. M.; Sen, B.; Agarwal, A.; Rivera, C.; Paul, W.; Ellis, K.; Chellappa, R.; et al. 2024. Conceptgraphs: Open-vocabulary 3d scene graphs for perception and planning. In *International Conference on Robotics and Automation (ICRA)*, 5021–5028. IEEE.

Huang, W.; Wang, C.; Zhang, R.; Li, Y.; Wu, J.; and Fei-Fei, L. 2023. VoxPoser: Composable 3D Value Maps for Robotic Manipulation with Language Models. In *Conference on Robot Learning*, 540–562. PMLR.

Hughes, N.; Chang, Y.; and Carlone, L. 2022. Hydra: A real-time spatial perception system for 3D scene graph construction and optimization. *arXiv preprint arXiv:2201.13360*.

Jiang, Z.; Xu, F. F.; Gao, L.; Sun, Z.; Liu, Q.; Dwivedi-Yu, J.; Yang, Y.; Callan, J.; and Neubig, G. 2023. Active retrieval augmented generation. *arXiv preprint arXiv:2305.06983*.

Kojima, T.; Gu, S. S.; Reid, M.; Matsuo, Y.; and Iwasawa, Y. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213.

Liang, J.; Huang, W.; Xia, F.; Xu, P.; Hausman, K.; Ichter, B.; Florence, P.; and Zeng, A. 2023. Code as policies: Language model programs for embodied control. In *International Conference on Robotics and Automation (ICRA)*, 9493–9500. IEEE.

Lin, K.; Agia, C.; Migimatsu, T.; Pavone, M.; and Bohg, J. 2023. Text2motion: From natural language instructions to feasible plans. *Autonomous Robots*, 47(8): 1345–1365.

Liu, B.; Jiang, Y.; Zhang, X.; Liu, Q.; Zhang, S.; Biswas, J.; and Stone, P. 2023. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.

Liu, Y.; Li, D.; Wang, K.; Xiong, Z.; Shi, F.; Wang, J.; Li, B.; and Hang, B. 2024. Are LLMs good at structured outputs? A benchmark for evaluating structured output capabilities in LLMs. *Information Processing & Management*, 61(5): 103809.

Luo, L.; Li, Y.-F.; Haf, R.; and Pan, S. 2024. Reasoning on Graphs: Faithful and Interpretable Large Language Model Reasoning. In *International Conference on Learning Representations*.

Nasiriany, S.; Xia, F.; Yu, W.; Xiao, T.; Liang, J.; Dasgupta, I.; Xie, A.; Driess, D.; Wahid, A.; Xu, Z.; et al. 2024. Pivot: Iterative visual prompting elicits actionable knowledge for vlms. *arXiv preprint arXiv:2402.07872*.

Ni, Z.; Deng, X.; Tai, C.; Zhu, X.; Xie, Q.; Huang, W.; Wu, X.; and Zeng, L. 2023. Grid: Scene-graph-based instruction-driven robotic task planning. *arXiv preprint arXiv:2309.07726*.

Paranjape, B.; Lundberg, S.; Singh, S.; Hajishirzi, H.; Zettlemoyer, L.; and Ribeiro, M. T. 2023. Art: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*.

Press, O.; Zhang, M.; Min, S.; Schmidt, L.; Smith, N. A.; and Lewis, M. 2022. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*.

Puig, X.; Ra, K.; Boben, M.; Li, J.; Wang, T.; Fidler, S.; and Torralba, A. 2018. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 8494–8502.

Raman, S. S.; Cohen, V.; Rosen, E.; Idrees, I.; Paulius, D.; and Tellex, S. 2022. Planning with large language models via corrective re-prompting. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

Rana, K.; Haviland, J.; Garg, S.; Abou-Chakra, J.; Reid, I.; and Suenderhauf, N. 2023. SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Task Planning. In *7th Annual Conference on Robot Learning*.

Ravichandran, Z.; Peng, L.; Hughes, N.; Griffith, J. D.; and Carlone, L. 2022. Hierarchical representations and explicit memory: Learning effective navigation policies on 3d scene graphs using graph neural networks. In *International Conference on Robotics and Automation (ICRA)*, 9272–9279. IEEE.

Schick, T.; Dwivedi-Yu, J.; Dessì, R.; Raileanu, R.; Lomeli, M.; Hambro, E.; Zettlemoyer, L.; Cancedda, N.; and Scialom, T. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

Singh, I.; Blukis, V.; Mousavian, A.; Goyal, A.; Xu, D.; Tremblay, J.; Fox, D.; Thomason, J.; and Garg, A. 2023. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 11523–11530. IEEE.

Skreta, M.; Yoshikawa, N.; Arellano-Rubach, S.; Ji, Z.; Kristensen, L. B.; Darvish, K.; Aspuru-Guzik, A.; Shkurti, F.; and Garg, A. 2023. Errors are useful prompts: Instruction guided task programming with verifier-assisted iterative prompting. *arXiv preprint arXiv:2303.14100*.

Sun, J.; Xu, C.; Tang, L.; Wang, S.; Lin, C.; Gong, Y.; Shum, H.-Y.; and Guo, J. 2023. Think-on-Graph: Deep and Responsible Reasoning of Large Language Model with Knowledge Graph. arXiv:2307.07697.

Wang, H.; Feng, S.; He, T.; Tan, Z.; Han, X.; and Tsvetkov, Y. 2023. Can Language Models Solve Graph Problems in Natural Language? In *Thirty-seventh Conference on Neural Information Processing Systems*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35: 24824–24837.

Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Zhang, S.; Zhu, E.; Li, B.; Jiang, L.; Zhang, X.; and Wang, C. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Wu, S.; Xie, J.; Chen, J.; Zhu, T.; Zhang, K.; and Xiao, Y. 2024. How Easily do Irrelevant Inputs Skew the Responses of Large Language Models? In *First Conference on Language Modeling*.

Wu, S.-C.; Wald, J.; Tateno, K.; Navab, N.; and Tombari, F. 2021. Scenegraphfusion: Incremental 3d scene graph prediction from rgb-d sequences. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7515–7525.

Yang, A.; Li, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Gao, C.; Huang, C.; Lv, C.; et al. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Yang, J.; Dong, Y.; Liu, S.; Li, B.; Wang, Z.; Tan, H.; Jiang, C.; Kang, J.; Zhang, Y.; Zhou, K.; et al. 2025b. Octopus: Embodied vision-language programmer from environmental feedback. In *European Conference on Computer Vision*, 20–38. Springer.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; and Cao, Y. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Ye, R.; Zhang, C.; Wang, R.; Xu, S.; Zhang, Y.; et al. 2023. Natural language is all a graph needs. *arXiv preprint arXiv:2308.07134*, 4(5): 7.

Yoran, O.; Wolfson, T.; Ram, O.; and Berant, J. 2023. Making retrieval-augmented language models robust to irrelevant context. *arXiv preprint arXiv:2310.01558*.

Yu, W.; Gileadi, N.; Fu, C.; Kirmani, S.; Lee, K.-H.; Arenas, M. G.; Chiang, H.-T. L.; Erez, T.; Hasenclever, L.; Humplik, J.; et al. 2023. Language to Rewards for Robotic Skill Synthesis. In *Conference on Robot Learning*, 374–404. PMLR.

Zhang, H.; Du, W.; Shan, J.; Zhou, Q.; Du, Y.; Tenenbaum, J. B.; Shu, T.; and Gan, C. 2024. Building Cooperative Embodied Agents Modularly with Large Language Models. In *International Conference on Learning Representations*.

Zhang, J.; Zhang, J.; Pertsch, K.; Liu, Z.; Ren, X.; Chang, M.; Sun, S.-H.; and Lim, J. J. 2023. Bootstrap your own skills: Learning to solve new tasks with large language model guidance. *arXiv preprint arXiv:2310.10021*.

Zhou, D.; Schärli, N.; Hou, L.; Wei, J.; Scales, N.; Wang, X.; Schuurmans, D.; Cui, C.; Bousquet, O.; Le, Q.; et al. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.

Zhu, Y.; Tremblay, J.; Birchfield, S.; and Zhu, Y. 2021. Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 6541–6548. IEEE.