VeriGraph: Scene Graphs for Execution Verifiable Robot Planning

Daniel Ekpo[1], Archana Swaminathan[1], Mara Levy[1], Saksham Suri[1], Chuong Huynh[1], Abhinav Shrivastava[1]

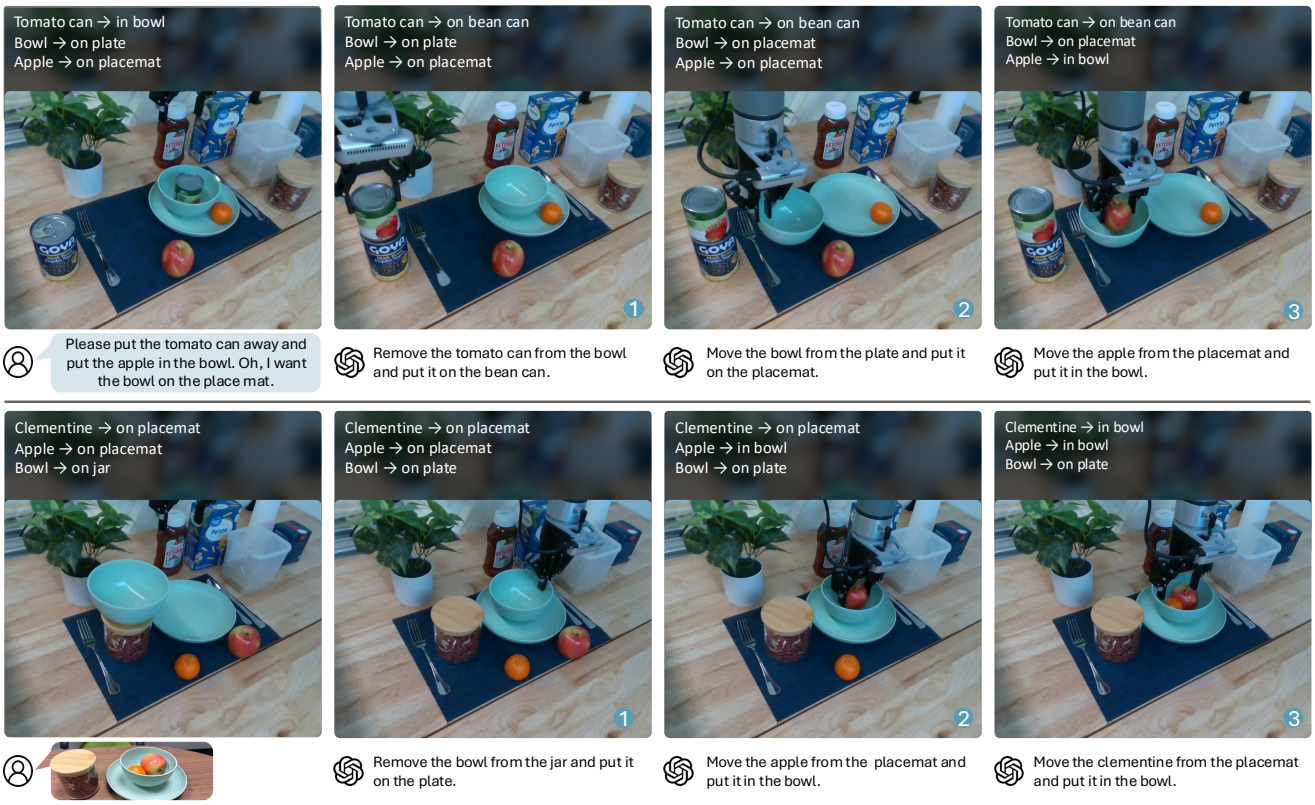

Fig. 1: VeriGraph leverages scene graphs and execution verification for long-horizon robot manipulation. Given an initial scene and a goal - specified as either a language instruction (top) or a reference image (bottom), a VLM planner generates and executes a sequence of actions, with verification after each step to ensure correctness before proceeding.

***Abstract*— Recent progress in vision-language models (VLMs) has opened new possibilities for robot task planning, but these models often produce incorrect action sequences. To address these limitations, we propose VeriGraph, a novel framework that integrates VLMs for robotic planning while verifying action feasibility. VeriGraph uses scene graphs as an intermediate representation to capture key objects and spatial relationships, enabling more reliable plan verification and refinement. The system generates a scene graph from input images and uses it to iteratively check and correct action sequences generated by an LLM-based task planner, ensuring constraints are respected and actions are executable. Our approach significantly enhances task completion rates across diverse manipulation scenarios, outperforming baseline methods by 58% on language-based tasks, 56% on tangram puzzle tasks, and 30% on image-based tasks. Qualitative results and code can be found at `https://verigraph-agent.github.io/`.**

I. Introduction

For robots to be able to solve complex manipulation problems in the real world, they need to understand the physical world around them, including object locations and relationships between objects in the scene. Humans intuitively understand spatial relationships between objects in the world and can use this understanding to develop efficient and executable plans to complete tasks. Consider the example of organizing a cluttered room. Humans can quickly understand which objects are out of place based on their understanding of how objects relate to each other. For example, it seems intuitive that a book should be on a shelf, not on a cup. Robots struggle to perceive the world around them the way humans do.

Additionally, physical constraints in the real world restrict the order in which actions can be executed. For example, if a glass cup is on a book, which is on a desk, the robot must pick up the cup first and place it on the desk before picking up the book. Because of these constraints, robots need to understand the relationships between objects in the scene. If the robot does not understand that the cup is on the book, it might not factor that into its planning and may try to pick up the book first, which could result in the cup falling and breaking.

[1]University of Maryland, College Park, MD USA. Correspondence to: Daniel Ekpo `daniekpo@umd.edu`

Recent advances in large language models (LLMs) and vision-language models (VLMs) have opened up new possibilities for robot task planning [1, 2]. These models demonstrate impressive reasoning capabilities and world knowledge. Prior work [3–5] used LLMs to generate Planning Domain Definition Language (PDDL), which can be used by classical planners to create a task plan. While the results have been promising, PDDL is inherently restrictive and does not generalize well [6, 7]. Other lines of work use VLMs to generate high-level task plans directly from directly from images [8–10], treating visual observation as the primary input to the planner.

Although these methods show promising results, long-horizon manipulation tasks require reasoning about objects, spatial relations, and physical constraints. Planning directly from raw images can therefore be challenging. While pixel-level representations contain rich visual detail, they can also introduce noise and irrelevant visual variation that complicate high-level reasoning. Scene graphs provide a structured abstraction that explicitly represents objects and their relations, enabling reasoning about spatial constraints and action feasibility.

To address the scene representation challenge, we propose VeriGraph, a framework that uses scene graphs as an intermediate representation for robot task planning. Scene graphs have proven particularly valuable in robotic task planning [11–14]. Their structured nature enables the abstraction of object-level details into symbolic graphs, making them robust to noise while retaining essential information about object interactions. For example, a scene graph might encode that a "cup is on the table" or a "spoon is inside the cup," providing a framework for reasoning about actions such as moving the spoon or rearranging the objects in the scene. This abstraction is particularly useful for tasks where the precise visual appearance of objects is not critical. One such task is using a reference scene to arrange objects in another scene to look like the reference scene. Because of the scene graph representation, VeriGraph can solve this task significantly better than methods that rely on raw pixel data.

Despite their strong capabilities, VLMs frequently produce incorrect plans and often require multiple prompting iterations to generate a valid action sequence [15]. Approaches such as [10] attempt to solve this problem by inserting a human directly into the loop. However, this is time-consuming and requires constant human supervision. To address this plan verification problem, we add an iterative planning component to VeriGraph. The structured nature of scene graphs allows VeriGraph to represent each action in the plan as graph operations. For example, moving an object from one location to another can be represented with an edge manipulation operation. This representation allows VeriGraph to quickly check for constraint violations and iterate with the task planner to generate valid action sequences. This setup, shown in Figure 3, allows for more accurate and robust planning.

Depending on the task, the goal state can be specified through either language instructions or a reference image. To support both types of task specification, VeriGraph supports flexible goal specification, allowing manipulation tasks to be defined through either target scene images or natural language instructions. VeriGraph can generate goal scene graphs from these input modalities, providing a unified planning framework across different task specifications. Note that for reference images, VeriGraph does not require the same scene, only a contextually similar scene (e.g., refer to Figure 1). This versatility makes VeriGraph applicable to various real-world scenarios where goals may be communicated in different formats.

We show that VeriGraph beats existing methods that rely on raw pixels as input to the task planner while being execution-verifiable. Our main contributions are as follows:

- We present a modular framework that uses scene graphs as both the planning representation and the verification mechanism, enabling efficient constraint-aware planning with LLMs.
- We propose an iterative planning and verification mechanism that uses scene graphs to represent and verify action sequences, enhancing the system's ability to identify and correct constraint violations without human intervention.
- We utilize VLMs to generate goal scene graphs based on a reference image or language instruction to create a unified goal specification method.

II. Related Works

A. Scene Graphs in Planning

Scene graphs have been widely used in computer vision for symbolic representation, capturing object relationships in images [16]. They have supported tasks such as image generation [17–22], image/video captioning [23–25], and visual question answering [26–28]. By encoding high-level scene information while remaining robust to pixel-level noise, scene graphs have naturally motivated applications in robotics.

In robotic navigation and planning, scene graphs have been used to model environments and guide action selection. For example, [29] represent the environment as a graph of semantic places and navigational behaviors, while [30] leverage 3D scene graphs with graph neural networks (GNNs) to encode the scene and inform reinforcement learning policies. Other approaches integrate scene graphs with large models to enhance planning: GRID [31] fuses an initial scene graph with a VLM encoder to guide actions, and [32] uses object-detected scene graphs for cluttered scene exploration and task planning. Further extensions include augmenting only immobile objects [33], multi-view scene graph generation with partially observable MDP solvers [13], and combined geometric-symbolic graph representations for motion planning [11]. SG-Bot [34] imagines goal scenes using scene graphs, and [35] employ contact graphs with graph edit distance for sequential planning.

Our work builds on these graph-based planning approaches by modeling actions as graph operations. Unlike [35], we use an LLM as the task planner, allowing natural language task descriptions and greater generalization across objects, actions, and tasks, while still leveraging the structure of scene graphs for planning efficiency.

B. Planning with LLMs/VLMs

LLMs [36–38] and VLMs [39–41] have emerged as strong agents for open-world reasoning, understanding object relations, actions, and context. This has motivated their use as planners in robotics.

Prior work has explored using LLMs for high-level task planning [42, 43], including verification of planned actions [44] and replanning with visual feedback [45]. Embodied agents use LLMs to generate executable plans [8, 46, 47]. While [46] leverages language feedback, our method differs in using a scene graph as both the initial input and the continuous feedback to the planner, reducing data and computation requirements. SayCan [8] demonstrates strong results using learned value functions, but this approach requires substantial training data. ConceptGraphs [12] generate 3D scene graphs from 2D foundation models and plan with an LLM. In contrast, our approach operates directly on the scene graph representation and performs planning through graph-based constraint validation (Section III-B) and iterative graph edits (Section III-D), avoiding additional model fusion stages.

Recent VLM developments have made them an even better candidate for robotics since they can take visual data and raw pixels as input, unlike LLMs, which can only work with text. This allows them to perceive the scene as is without being heavily dependent on the input prompt. Recently, RT-2 [48] incorporated VLMs directly into end-to-end robotic control. PaLM-E [49] trained a large model specifically as an agent that can take multimodal inputs and perform robotics tasks. VILA [9], which is the closest to our work, uses a large VLM directly for their task. While these approaches show strong results, planning directly from pixels can be difficult for tasks that require reasoning about object relations and action constraints. Our approach instead introduces a structured scene graph representation that enables explicit constraint validation during planning.

C. Execution-Verification

While correctness is important when designing a task execution plan, it is also essential for it to be plausible. SayPlan [14] assumes an existing 3D scene graph, which they use to interact with the LLM. They use a graph simulator to verify the LLM-generated task plans and show that their approach works for multi-room setups. CoPAL [50] proposes corrective planning by using different layers of encapsulation. Some works [51] modify an existing reinforcement learning algorithm to condition on natural language feedback from the environment. They automatically generate the language feedback based on the current goal and the agent's current actions. This is similar to our approach of providing feedback to the planner based on the current graph state and predicted action. REFLECT [52] introduces a framework to query the LLM planner to reason about failures based on the hierarchical summary of the robot's past experiences generated from multisensory observations. They show that the failure explanation can help the LLM correct the failure and complete the task. Voyager [53] introduces an LLM learner that can learn executable skills as it interacts with the environment. It writes code to interact with the environment and correct itself with feedback received from the environment. VILA [9] uses execution to verify the plan by feeding the current state of the environment to the model at every step using visual inputs. Our approach, on the other hand, can generate execution-verifiable plans by relying on the current scene graph for constraint checking. This makes verifying the affordances and plausibility of an action is especially quick and efficient.

III. Our Approach: VeriGraph

VeriGraph takes in an image depicting an initial scene, along with either a target image portraying the desired goal scene state or instructions detailing modifications to the initial scene. It generates the initial and goal scene graphs and uses them to predict actions to transform the initial scene into the target scene. The scene graph generation method is discussed in Section III-A, and action constraints are discussed in Section III-B. Given a pair of scene graphs, the planner, discussed in Section III-C and Section III-D, generates a high-level plan instructing a robot to transform the initial scene into the goal scene.

A. Scene Graph Generation

Given an image I of a scene, the goal of scene graph generation is to create a graph that accurately represents the scene's structure. The scene graph comprises a set of vertices V, representing objects in the scene, and a set of edges E, describing the relationships between objects in V. R represents a set of possible relations between objects in the scene. An edge $e_{uv} \in E$ between two vertices u and v in the scene is then defined as $e_{uv} = \{u, v, r\}$ where $r \in R$ and $u, v \in V$. The scene graph for image I is thus represented as $G = \{V, E\}$.

In VeriGraph, the set of relations R includes basic spatial relations such as {in, on}. However, this set is flexible and easily adapted for other tasks. To address the issue of varying object names (e.g., tabletop, table, and countertop for the same object), we maintain a global dictionary of unique object names D for scene graph generation. This dictionary encompasses all objects that could be present in any scene. The scene graph generator SGG takes the image I, the global dictionary D, the set of relations R and the task description T and returns a scene graph. We define the graph generator as

$$SGG(I, R, D, T) \rightarrow G = \{V, E\} \tag{1}$$

When generating scene graphs, T is set to null except for the target scene graph for tasks where the target scene is described using natural language, in which case the task instruction/goal scene description T is used in the graph generation process. VeriGraph uses SGG to generate the initial and goal scene graphs.

B. Constraint Validation

Every action has a set of preconditions that must be met before execution. For example, before performing the "move" action on a plate, any items on the plate must first be removed. Additionally, post-conditions must be satisfied for the action

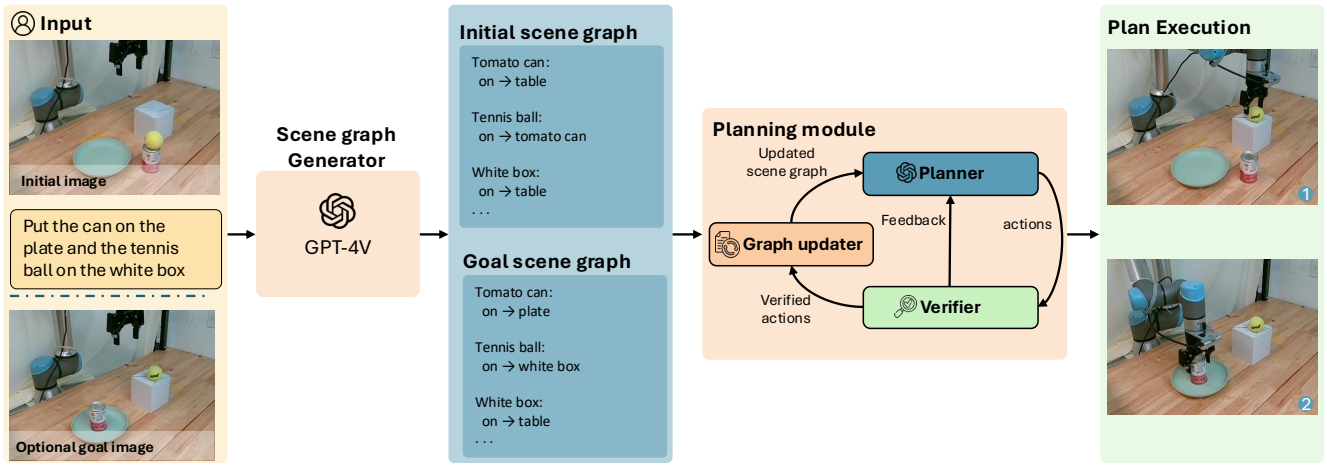

Fig. 3: Overview of VeriGraph. Given a start image and either language instructions or a goal image, a scene-graph module extracts objects and relations to form the initial and goal scene graphs. VeriGraph then iteratively proposes high-level actions, validates them against graph-based constraints, and executes feasible actions to update the scene graph. On violation, VeriGraph generates feedback and replans; the loop ends when the planner emits an end token.

to be considered successful. In the example given, the plate must end up on the new supporting object. In VeriGraph, these conditions are represented as a set of constraints $\mathcal{C}$.

VeriGraph uses the current graph to validate constraints. The node v associated with the action must exist in the graph, and its in/out edges must satisfy specific conditions. For instance, if v is being moved, VeriGraph checks if v supports any other objects by ensuring no edges from v to any other nodes exist.

After constraints are validated, VeriGraph updates the current graph state to reflect the execution of the action. The specific changes to the graph depend on the action taken. For a "move" operation, the edge representing the initial support relationship is removed, and a new edge is created for the new support relationship. The next action in the sequence is then validated, and the graph is modified accordingly. The final graph's nodes and edges are compared against the goal scene graph. If they match, the plan is considered successful.

C. Task Planning

Given the initial scene graph G_i and the target scene graph G_g, the task planner $\mathcal{P}$ generates actions that can be executed on the initial scene to transform it into the target graph. Let $\mathcal{A}$ be the set of all high-level actions that a robot can perform, the sequence of actions $a = \{a_1, a_2, \cdots, a_n\}$ such that $a \in \mathcal{A}$ predicted by the planner $\mathcal{P}$ must complete the given task while adhering to the constraints $\mathcal{C}$. We define the task planner as

$$a = \mathcal{P}(G_i, G_g, \mathcal{C}, \mathcal{A}). \tag{2}$$

An example of such a plan can be seen in Figure 4.

D. Iterative Planning

The planner described in Section III-C outputs the full action sequence without any feedback mechanism to refine the plan. Our experiments showed that this approach often fails in difficult tasks because LLMs tend to forget constraints.

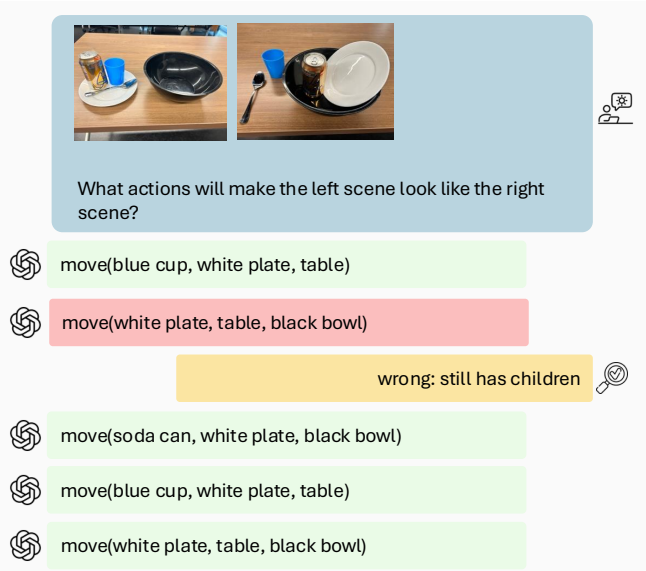

Fig. 4: Iterative planning with graph-based feedback. VeriGraph proposes an action, validates it on the current graph, and executes it if feasible (green). When an action violates constraints (red), VeriGraph generates feedback and replans. This repeats until the planner signals completion.

To address this issue, we designed an iterative planner $\mathcal{P}_{\text{iter}}$ that receives feedback $\mathcal{F}$ about the proposed action sequences and corrects the plan accordingly.

The planner, $\mathcal{P}_{\text{iter}}$, is given $\mathcal{A}$, $\mathcal{C}$, G_i, and G_g and asked to output at most k high-level actions and an end token. VeriGraph attempts to perform the actions, and VeriGraph returns feedback, $\mathcal{F}$, as well as the current graph state. If there is a constraint violation in the proposed actions, the error count τ is increased. The feedback and new graph state are then passed to $\mathcal{P}_{\text{iter}}$, and the iteration continues until either the number of errors reaches the error threshold $\mathbf{t}$ or the end token is received. Because the LLM planner can repeatedly propose actions that violate the same constraints, the process

Algorithm 1 Iterative Planning Algorithm

Require: Initial graph G_i, goal graph G_g, constraints C, action set A, actions per iteration k, error threshold t

$G \leftarrow G_i$, $F \leftarrow \emptyset$, $\tau \leftarrow 0$
$\hat{a} \leftarrow [\,]$ ▷ executed actions
while $\tau < t$ **do**
  $(a_{1:k}, \texttt{end\_token}) \leftarrow P_{\text{iter}}(G, G_g, C, A, F)$
  if `end_token` **then**
    break
  end if
  $(\tilde{a}, G, F) \leftarrow$ execute_until_violation$(a_{1:k}, G, C)$
  append $\tilde{a}$ to $\hat{a}$
  if $F \neq \emptyset$ **then** ▷ constraint violation
    $\tau \leftarrow \tau + 1$
  else
    $\tau \leftarrow 0$
  end if
end while
return $(\hat{a}, G)$

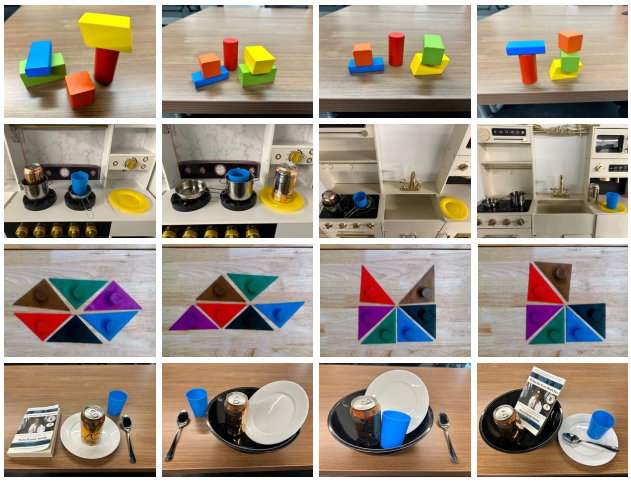

Fig. 5: Example evaluation scenes from the four task groups (blocks, kitchen, tangram, and tableware), illustrating varied object sets and support relations used to test constraint-aware planning.

may enter an error cycle. The error threshold prevents infinite replanning by limiting the number of failed iterations. The iterative planning process is described in Algorithm 1.

IV. Experimental Details

In this section, we outline the details of our experiments. The evaluation dataset is introduced in Section IV-A, followed by task description in Section IV-B. All baselines are mentioned in Section IV-C and finally Section IV-D discusses the results.

A. Dataset

We design three scenes—kitchen, tabletop, and block scenes—to evaluate VeriGraph's performance and compare against baseline methods. Each scene has multiple configurations with varying numbers of objects and placements. We vary the number of objects between three and seven. Ground truth scene graphs for each scene were created using GPT-4V(ision) (GPT-4V) [54], and corrections were made manually when necessary. Figure 5 shows some example images from the dataset.

B. Tasks

We created four task groups - tangram puzzle, rearrange, language instruction, and stacking - with varying difficulty levels. Each task has a ground truth goal scene graph for evaluating the planner. The predicted actions from the planner are executed on the initial scene graph, and the transformed graph is then compared against the ground truth goal scene graph. The different task groups are described below.

Tangram puzzle. The task is defined by two images: a goal tangram configuration and a perturbed one. The goal can be reached by moving a single piece so that one of its sides connects with a specific side of another piece. The model must predict which piece to move, where to place it, and which sides should align. We detect tangrams with a triangle detector and use OpenClip [55] to assign consistent color-based labels; sides are indexed clockwise. Contacts are computed using a heuristic module. In VILA, we provide labeled images, while in VeriGraph each configuration is represented as a scene graph with nodes as tangram pieces and edges as side-to-side contacts.

Stacking Task. We use block scenes of different configurations for the stacking task. The model is given an image of the initial scene and asked to stack all the blocks into one stack. The final order is arbitrary here, so there's no ground truth scene graph. Instead, the final scene graph is checked to ensure a single pile of blocks. Some block scenes already have multiple incomplete stacks, so the planner must unstack the other stacks and complete one.

Language Instruction Task. This consists of an initial scene and a language instruction. The instruction is either direct commands, e.g., "Move pan to the stovetop," or a description of the desired goal state of the scene, e.g., "I need the positions of the pan and pot swapped." The model is asked to predict actions to execute the given instruction on the scene.

Reference Image Instruction Task. The model is given an initial scene and a structurally similar scene as the goal state and asked to predict a sequence of actions to transform the initial scene into the goal scene. The scene graph of the goal scene is compared against the final scene graph after plan execution, and the task is considered successful if the scene graphs match.

C. Baselines

We compare VeriGraph against the following baselines: (a) **VILA** [9]. prompts the vision-language model (VLM) with an image and a language instruction or another image as the goal state. For a fair comparison, we use a prompt identical to ours and remove all references to scene graphs. We execute the proposed actions on the ground truth initial scene graph and compare the final graph against the ground truth graph. (b) **SayCan** [8] prompts the LLM with a textual representation of the scene. The text contains a list of all

TABLE I: Results on scene graph generation.

Scene	Method	F1 Score		Exact Graph Accuracy
		Nodes	Edges	
Blocks	LLava	0.30	0.07	0.00
	Gemini	**1.00**	0.93	0.73
	GPT-4V	**1.00**	**1.00**	**1.00**
Kitchen	LLava	0.50	0.05	0.00
	Gemini	0.78	0.59	0.04
	GPT-4V	**0.98**	**0.87**	**0.65**
Tableware	LLava	0.52	0.16	0.00
	Gemini	0.95	0.90	0.39
	GPT-4V	**0.99**	**0.95**	**0.89**

the objects in the scene. We implemented a similar setup using GPT-4 as the LLM. Since SayCan cannot understand the spatial relationships between objects in the scene because it only receives a list of all objects, we only evaluate it on image-language tasks. The objects from the vertices in the ground truth scene graph are the scene observation for this baseline.

D. Experiments and Results

Scene graph generation: As mentioned earlier, VLMs are utilized in our scene graph generation model. Here, we evaluate some popular VLMs, such as Gemini 1.5 Pro [56], LLava [57], and GPT-4V [54], without any in-context examples. For GPT-4V and Gemini 1.5 Pro, we use the official Python SDK. Ollama [58] is used to run LLava locally. The same prompt is used for all three models. For our experiments, the set of relations R and global dictionary D are predefined and included in the prompt. We evaluate node and edge prediction using F1-score compared to the ground truth scene graph. We also report Exact Graph Accuracy, which requires a perfect match of all nodes and edges in the predicted graph. Because this metric requires the entire graph to be correct simultaneously, it is substantially stricter than the per-element F1 scores. The three models are compared across the scenes in Table I. GPT-4V performs notably better than Gemini and LLava across all scenes. We used GPT-4V as the scene graph generator SGG for other experiments based on these results.

Planning: We used GPT-4 [39] as the task planner $\mathcal{P}$. We evaluate our approach on all four task groups presented in Section IV-B and show results in Table II. Compared to VILA, we improve on language and image instruction planning tasks. For the image-based start and goal state, we observe an average improvement of $\sim$0.57. For the language instruction task, we outperform SayCan by $\sim$0.56. Similarly, for the tangram puzzle tasks we outperform VILA by 56%.

We attribute the performance improvements to the structured representation of the scene as a scene graph. Scene graph representations structure the environment into explicit objects and relations, reducing the need for the VLM to infer spatial relationships directly from raw images and enabling iterative correction during the planning stage. Most failure cases in our proposed approach arise from inaccuracies in the generated scene graphs. As scene graph generation methods improve, the effectiveness of our planning approach is expected to improve as well. To evaluate the impact of scene graph quality, we provided the planner with ground truth scene graphs and observed that the iterative planner generated successful plans in nearly all cases.

TABLE II: Planning results across different task types. We report task success rate, where success indicates that the final scene graph after executing the predicted actions matches the goal scene graph. "Ours (Direct)" uses a single-shot planner without verification, while "Ours" uses the iterative planning and constraint-validation mechanism of VeriGraph, which corrects actions that violate scene graph constraints.

Task	SayCan	VILA	Ours (Direct)	Ours
Stacking	0.07	0.62	0.35	**0.65**
Language Instruction	0.17	0.43	**0.73**	0.65
Ref. Image Instruction (Blocks)	-	0.27	0.67	**0.86**
Ref. Image Instruction (Kitchen)	0.00	0.05	0.50	**0.55**
Tangram puzzle	-	0.16	0.72	**0.72**

TABLE III: Ablation on error threshold and number of actions per iteration used in VeriGraph.

(a) Effect of error threshold.

Error Threshold (τ)	2	3	5	10
Accuracy (%)	20	85	80	90

(b) Varying of # of actions / iter.

# of Actions	2	3	5	10
Accuracy (%)	85	85	85	85

Our results in Table II show that the iterative planner (last column) improves performance for most tasks compared to the non-iterative planner. However, the language instruction task shows a slight decrease and the tangram task shows no change. One contributing factor is the accuracy of the generated scene graphs. The iterative planner relies on graph constraints to detect invalid actions, and errors in the scene graph can prevent these violations from being detected correctly. This is consistent with our earlier observation that when ground truth scene graphs are provided, the iterative planner produces successful plans in nearly all cases.

E. Ablation Study

Iterative vs. non-iterative planner. Our experiment results in Table II show that the iterative planner ('Ours') achieves higher accuracy than the non-iterative planner ('Ours (Direct)') for most tasks. The feedback in the iterative process allow for iterative correction during the planning stage which results in improved task planning accuracy.

Error thresholds. We observed that setting the error threshold to 2 resulted in the worst performance. While setting it to 10 yielded the best performance, it increased runtime and cost due to additional replanning iterations. We found that setting the threshold to 5 provided a good balance between accuracy and speed.

Number of actions per iteration. We tested 2, 3, 5, and 10 actions per iteration. Although there was no significant difference in accuracy, we ultimately chose to use 3 actions per iteration for optimal performance.

F. Real-world execution

To demonstrate that the high-level plans generated by VeriGraph can be executed on a physical robot, we deploy the system on representative manipulation tasks described in Section IV-B. We implement a modular execution pipeline that converts high-level task plans to low-level robot commands. We use LangSAM [59] to obtain object masks for the target and destination objects. The detected objects are projected to 3D coordinates and transformed to the robot frame. We use AnyGrasp [60] to predict a grasp pose for the target object, and estimate the drop height from the point cloud of the destination object.

For the tangram task, we compute the x–y offset between the centers of the target and destination pieces, along with the rotation needed to align the specified sides. These offsets are then projected into 3D to produce the final robot commands. The generated plans are executed in an open-loop manner. Qualitative results of the real-world executions are available on the project website.

G. Cost and runtime analysis

The computational cost of VeriGraph is dominated by calls to the VLM used for scene graph generation and the LLM used for task planning. Two VLM calls are required to generate the initial and goal scene graphs. The monetary cost therefore depends on the specific model provider; since these costs change frequently and continue to decrease, we do not report fixed numbers here.

The cost of planning depends on the number of iterations and the number of actions proposed per iteration. Proposing more actions per iteration can reduce the number of LLM calls, lowering overall planning cost. In addition to monetary cost, system latency is primarily determined by network bandwidth and the response time of the models.

The graph-based constraint validation and state updates are lightweight operations whose runtime is negligible compared to model inference. This enables flexible deployment strategies. In particular, our experiments suggest that while local models often struggle with accurate scene graph generation, they can produce effective plans when provided with a correct scene graph and combined with the iterative verification mechanism. This suggests a hybrid approach in which a high-capability VLM generates the scene graph, while a local LLM performs task planning, significantly reducing cost while maintaining strong performance.

V. Conclusion

In this paper, we presented VeriGraph, a framework for generating and validating high-level task plans for robot object manipulation using scene graphs. VeriGraph represents actions as graph edit operations and verifies them using node- and edge-based constraints, enabling efficient validation of planner outputs before execution. We demonstrate that scene graphs provide a compact and structured representation of scene information and show that our method outperforms approaches that rely directly on raw pixel inputs. We also introduced an iterative planning mechanism in which proposed actions are validated against the current graph state and corrected when constraint violations occur, improving the robustness of the planning process. In addition, the framework provides an efficient way to evaluate high-level robot task-planning algorithms.

Our approach depends on the accuracy of the generated scene graphs. Because scene graphs are extracted from monocular images, errors in object detection or spatial relations can propagate to the planning stage and lead to incorrect actions. In practice, we observed that hallucinated objects or incorrect relations occasionally produced invalid plans. While the constraint validation stage prevents some infeasible actions, it cannot correct structural errors in the underlying graph. Improving the reliability of scene graph generation therefore remains an important direction for future work. As scene graph generation methods continue to improve, the planning performance of VeriGraph is expected to improve correspondingly.

VI. Acknowledgement

This work was partially supported by NSF CAREER Award #2238769 to AS. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF or the U.S. Government. We thank Namitha Padmanabhan, Pulkit Kumar, Seungjae Lee, and Vatsal Agarwal for their helpful feedback and discussions.

References

[1] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *Int. Conf. Mach. Learn.* PMLR, 2022, pp. 9118–9147.

[2] W. Huang, F. Xia, D. Shah, D. Driess, A. Zeng, Y. Lu, P. Florence, I. Mordatch, S. Levine, K. Hausman *et al.*, "Grounded decoding: Guiding text generation with grounded models for robot control," *arXiv preprint arXiv:2303.00855*, 2023.

[3] N. Simon and C. Muise, "A natural language model for generating pddl," in *ICAPS KEPS workshop*, 2021.

[4] B. Wang, Z. Wang, X. Wang, Y. Cao, R. A Saurous, and Y. Kim, "Grammar prompting for domain-specific language generation with large language models," *Adv. Neural Inform. Process. Syst.*, vol. 36, 2024.

[5] J. Oswald, K. Srinivas, H. Kokel, J. Lee, M. Katz, and S. Sohrabi, "Large language models as planning domain generators," *Int. Conf. Autom. Planning Scheduling*, vol. 34, pp. 423–431, May 2024.

[6] J. Espasa, I. Miguel, P. Nightingale, A. Z. Salamon, and M. Villaret, "Challenges in modelling and solving plotting with pddl," 2023.

[7] X. Zhang, Z. Altaweel, Y. Hayamizu, Y. Ding, S. Amiri, H. Yang, A. Kaminski, C. Esselink, and S. Zhang, "Dkprompt: Domain knowledge prompting vision-language models for open-world planning," 2024.

[8] M. A. et. al., "Do as i can, not as i say: Grounding language in robotic affordances," 2022.

[9] Y. Hu, F. Lin, T. Zhang, L. Yi, and Y. Gao, "Look before you leap: Unveiling the power of gpt-4v in robotic vision-language planning," *arXiv preprint arXiv:2311.17842*, 2023.

[10] B. Li, P. Wu, P. Abbeel, and J. Malik, "Interactive task planning with language models," 2023.

[11] Y. Zhu, J. Tremblay, S. Birchfield, and Y. Zhu, "Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs," 2021.

[12] Q. Gu, A. Kuwajerwala, S. Morin, K. M. Jatavallabhula, B. Sen, A. Agarwal, C. Rivera, W. Paul, K. Ellis, R. Chellappa, C. Gan, C. M.

de Melo, J. B. Tenenbaum, A. Torralba, F. Shkurti, and L. Paull, "Conceptgraphs: Open-vocabulary 3d scene graphs for perception and planning," 2023.
[13] S. Amiri, K. Chandan, and S. Zhang, "Reasoning with scene graphs for robot planning under partial observability," 2022.
[14] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Suenderhauf, "Sayplan: Grounding large language models using 3d scene graphs for scalable task planning," in *Conference on Robot Learning*, 2023.
[15] K. Valmeekam, M. Marquez, S. Sreedharan, and S. Kambhampati, "On the planning abilities of large language models-a critical investigation," *Adv. Neural Inform. Process. Syst.*, vol. 36, pp. 75 993–76 005, 2023.
[16] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma, M. S. Bernstein, and F.-F. Li, "Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations," Feb. 2016. [Online]. Available: http://arxiv.org/abs/1602.07332
[17] J. Johnson, A. Gupta, and L. Fei-Fei, "Image generation from scene graphs," in *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018, pp. 1219–1228.
[18] X. Zhao, L. Wu, X. Chen, and B. Gong, "High-quality image generation from scene graphs with transformer," in *Int. Conf. Multimedia and Expo*, 2022, pp. 1–6.
[19] G. Mittal, S. Agrawal, A. Agarwal, S. Mehta, and T. Marwah, "Interactive image generation using scene graphs," *arXiv preprint arXiv:1905.03743*, 2019.
[20] S. Tripathi, A. Bhiwandiwalla, A. Bastidas, and H. Tang, "Using scene graph context to improve image generation," *arXiv preprint arXiv:1901.03762*, 2019.
[21] B. Zhao, L. Meng, W. Yin, and L. Sigal, "Image generation from layout," in *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019, pp. 8584–8593.
[22] L. Yang, Z. Huang, Y. Song, S. Hong, G. Li, W. Zhang, B. Cui, B. Ghanem, and M.-H. Yang, "Diffusion-based scene graph to image generation with masked contrastive pre-training," *arXiv preprint arXiv:2211.11138*, 2022.
[23] L. Gao, B. Wang, and W. Wang, "Image captioning with scene-graph based semantic concepts," in *Proceedings of the 2018 10th international conference on machine learning and computing*, 2018, pp. 225–229.
[24] Y. Zhong, L. Wang, J. Chen, D. Yu, and Y. Li, "Comprehensive image captioning via scene graph decomposition," in *Eur. Conf. Comput. Vis.* Springer, 2020, pp. 211–229.
[25] X. Yang, J. Peng, Z. Wang, H. Xu, Q. Ye, C. Li, M. Yan, F. Huang, Z. Li, and Y. Zhang, "Transforming visual scene graphs to image captions," *arXiv preprint arXiv:2305.02177*, 2023.
[26] C. Zhang, W.-L. Chao, and D. Xuan, "An empirical study on leveraging scene graphs for visual question answering," *arXiv preprint arXiv:1907.12133*, 2019.
[27] S. Lee, J.-W. Kim, Y. Oh, and J. H. Jeon, "Visual question answering over scene graph," in *2019 First International Conference on Graph Computing (GC)*, 2019, pp. 45–50.
[28] V. Damodaran, S. Chakravarthy, A. Kumar, A. Umapathy, T. Mitamura, Y. Nakashima, N. Garcia, and C. Chu, "Understanding the role of scene graphs in visual question answering," *arXiv preprint arXiv:2101.05479*, 2021.
[29] G. Sepulveda, J. C. Niebles, and A. Soto, "A deep learning based behavioral approach to indoor autonomous navigation," in *IEEE Int. Conf. Robot. Autom.*, 2018, pp. 4646–4653.
[30] Z. Ravichandran, L. Peng, N. Hughes, J. D. Griffith, and L. Carlone, "Hierarchical representations and explicit memory: Learning effective navigation policies on 3d scene graphs using graph neural networks," in *IEEE Int. Conf. Robot. Autom.*, 2022, pp. 9272–9279.
[31] Z. Ni, X.-X. Deng, C. Tai, X.-Y. Zhu, X. Wu, Y.-J. Liu, and L. Zeng, "Grid: Scene-graph-based instruction-driven robotic task planning," 2023.
[32] Y. Deng, Q. Sima, D. Guo, H. Liu, Y. Wang, and F. Sun, "Scene graph for embodied exploration in cluttered scenario," 2023.
[33] A. Onishchenko, A. Kovalev, and A. Panov, "Lookplangraph: Embodied instruction following method with VLM graph augmentation," in *Workshop on Reasoning and Planning for Large Language Models*, 2025.
[34] G. Zhai, X. Cai, D. Huang, Y. Di, F. Manhardt, F. Tombari, N. Navab, and B. Busam, "Sg-bot: Object rearrangement via coarse-to-fine robotic imagination on scene graphs," 2023.
[35] Z. Jiao, Y. Niu, Z. Zhang, S.-C. Zhu, Y. Zhu, and H. Liu, "Sequential Manipulation Planning on Scene Graph," in *IEEE/RSJ Int. Conf. Intell. Robots Syst.* Kyoto, Japan: IEEE, Oct. 2022, pp. 8203–8210.
[36] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.
[37] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.
[38] R. Anil, A. M. Dai, O. Firat, M. Johnson, D. Lepikhin, A. Passos, S. Shakeri, E. Taropa, P. Bailey, Z. Chen *et al.*, "Palm 2 technical report," *arXiv preprint arXiv:2305.10403*, 2023.
[39] OpenAI, "Gpt-4 technical report," 2024.
[40] D. Surís, S. Menon, and C. Vondrick, "Vipergpt: Visual inference via python execution for reasoning," 2023.
[41] H. Liu, C. Li, Y. Li, and Y. J. Lee, "Improved baselines with visual instruction tuning," *arXiv preprint arXiv:2310.03744*, 2023.
[42] P. Pramanick, H. B. Barua, and C. Sarkar, "Decomplex: Task planning from complex natural instructions by a collocating robot," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 6894–6901.
[43] S. G. Venkatesh, R. Upadrashta, and B. Amrutur, "Translating natural language instructions to computer programs for robot manipulation," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 1919–1926.
[44] D. S. Grigorev, A. K. Kovalev, and A. I. Panov, "Verifyllm: Llm-based pre-execution task plan verification for robots," 2025.
[45] S. Pchelintsev, M. Patratskiy, A. Onishchenko, A. Korchemnyi, A. Medvedev, U. Vinogradova, I. Galuzinsky, A. Postnikov, A. K. Kovalev, and A. I. Panov, "Lera: Replanning with visual feedback in instruction following," 2025.
[46] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, P. Sermanet, N. Brown, T. Jackson, L. Luu, S. Levine, K. Hausman, and B. Ichter, "Inner monologue: Embodied reasoning through planning with language models," 2022.
[47] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2022.
[48] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn *et al.*, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," *arXiv preprint arXiv:2307.15818*, 2023.
[49] D. Driess, F. Xia, M. S. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu *et al.*, "Palm-e: An embodied multimodal language model," *arXiv preprint arXiv:2303.03378*, 2023.
[50] F. Joublin, A. Ceravola, P. Smirnov, F. Ocker, J. Deigmoeller, A. Belardinelli, C. Wang, S. Hasler, D. Tanneberg, and M. Gienger, "Copal: Corrective planning of robot actions with large language models," 2023.
[51] S. McCallum, M. Taylor-Davies, S. V. Albrecht, and A. Suglia, "Is feedback all you need? leveraging natural language feedback in goal-conditioned reinforcement learning," 2023.
[52] Z. Liu, A. Bahety, and S. Song, "Reflect: Summarizing robot experiences for failure explanation and correction," 2023.
[53] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar, "Voyager: An open-ended embodied agent with large language models," 2023.
[54] OpenAI, "Openai (2023)," 2023. [Online]. Available: https://openai.com/index/gpt-4v-system-card/
[55] G. Ilharco, M. Wortsman, R. Wightman, C. Gordon, N. Carlini, R. Taori, A. Dave, V. Shankar, H. Namkoong, J. Miller, H. Hajishirzi, A. Farhadi, and L. Schmidt, "Openclip," Jul. 2021, if you use this software, please cite it as below.
[56] G. T. et. al., "Gemini: A family of highly capable multimodal models," 2024.
[57] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," 2023.
[58] github, "Github," 2024.
[59] L. Medeiros, "Lang segment anything," https://github.com/luca-medeiros/lang-segment-anything, 2023.
[60] H.-S. Fang, C. Wang, H. Fang, M. Gou, J. Liu, H. Yan, W. Liu, Y. Xie, and C. Lu, "Anygrasp: Robust and efficient grasp perception in spatial and temporal domains," *IEEE Trans. Robot.*, 2023.