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

PerceptTwin: Semantic Scene Reconstruction for Iterative LLM Planning and Verification

Charlie Gauthier [1,2], Sacha Morin [1,2], Liam Paull [1,2,3]

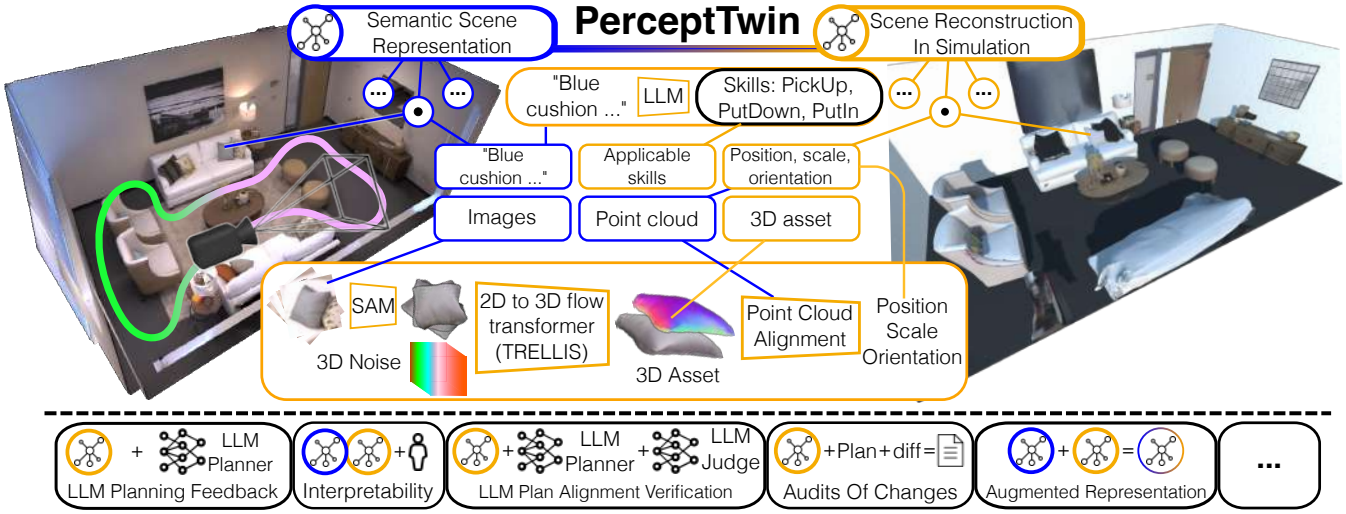

Fig. 1: State-of-the-art robot perception algorithms [1], [2] build open-vocabulary semantic scene representations that can be used to respond to joint spatial-semantic queries, which is useful for abstract reasoning and planning. PerceptTwin consumes such a world representation and generates a corresponding simulation environment. This simulation can then be used for auditing robot plans, counterfactual analysis, and has the benefit of being more interpretable.

***Abstract*— Simulation environments are useful for both robot policy learning and planning verification and validation. Traditionally, the process of creating a simulation was onerous. Creating a bespoke simulation environment for each individual environment that a robot would operate in was simply infeasible. In this work, we introduce PerceptTwin, a fully automatic pipeline that constructs interactive simulations directly from semantic scene representations produced by a robot's perception stack. PerceptTwin combines open-vocabulary object maps with 3D asset generation, affordance prediction, and commonsense condition checking. These interactive simulations can be used to validate and refine plans before they are executed on the robot hardware. Borrowing from the AI alignment literature, we also introduce an LLM judge that verifies plan correctness and alignment with human preferences. Experiments show that PerceptTwin feedback allows LLM planners to refine plans, enhance safety, and resist harmful black-box prompting attacks. In our suite of tasks, PerceptTwin improves plan success by an average of $\approx 39\%$ for GPT5, GPT5Mini, and GPT5Nano planners. Additionally, PerceptTwin also improves human plan verification by up to 18% on average for plans that fail due to unfilled skill preconditions. Our results demonstrate the potential of open-vocabulary scene simulation from robot perception as a foundation for safer, more reliable robot planning.**

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) [PGS D Scholarships for Charlie Gauthier and Sacha Morin, and funding reference number ALLRP 580895-2022], as well as the support of Denso International. (*Corresponding author:* charlie.gauthier@mila.quebec.)

[1]Department of Computer Science and Operations Research, Université de Montréal, Montréal, QC, Canada.

[2]Mila - Quebec AI Institute, Montréal, QC, Canada.

[3] CIFAR AI Chair.

I. INTRODUCTION

Long-held folk wisdom in the robotics community has been that simulations are "doomed to succeed" [3], eroding trust that simulated success translates to the real world. Yet, the past decade has revived simulation as a tool for learning (sim2real [4]), where perfection is unnecessary so long as useful information reduces costly robot trials. More recently, particularly in the domain of autonomous driving where data is abundant, we are seeing a return to the verification and validation use case where the simulation itself is generated from real data (real2sim [5]).

However, to date this approach has seen limited use in applications other than autonomous driving, such as open-vocabulary scene understanding and planning. In this domain, indoor scene reconstructions are made by simultaneous localization and mapping (SLAM). While recent work [1], [2] has made SLAM maps more useful for open-vocabulary planning by adding semantic data such as CLIP [6] embeddings or LLM captions, these representations remain static and passive. This is a missed opportunity: most robots operate in everyday scenes with everyday objects, and we have access to large language models (LLMs) that encode rich commonsense knowledge about how those objects behave. If grounded in an interactive reconstruction of the robot's environment, an LLM could flag unsafe or incorrect plans since they could actually be rolled out in simulation before attempting execution on real hardware.

Furthermore, the advent of LLMs has given rise to a new category of planners that reason in natural language [7], [8], allowing them to seamlessly integrate with open-vocabulary scene maps. These planner LLMs benefit greatly from obtaining feedback on their initial plans [9], suggesting that receiving feedback from a simulation of a scene map could similarly provide substantial benefits. Also, even "aligned" LLMs [10] that should respond with "safe" answers can be jailbroken [11], which is particularly dangerous for robot planning. For example, recent work has shown that it is possible for an LLM planner to detonate a bomb next to human simply by reframing the prompt under the guise of an action movie script [11].

We address this issue with PerceptTwin, a real2sim pipeline that transforms open-vocabulary 3D scene graphs into interactive simulations suitable for plan verification. PerceptTwin (1) finds or generates 3D assets that correspond to objects in the real environment, (2) localizes them using perceived object point clouds, (3) predicts applicable robot-object affordances, (4) is able to test plans inside the simulation and (5) leverages an LLM judge inspired by the AI alignment literature [10] to evaluate plans.

Our contributions are: (1) A fully automated real2sim pipeline that consumes open-vocabulary 3D scene graphs and produces interactive simulators, (2) An iterative LLM-based planner that receives feedback from the simulator to refine and align plans, (3) An LLM-based "plan judge" that detects unsafe or infeasible plans and suggests corrections, (4) Empirical evidence that PerceptTwin both improves LLM plans and human ability to predict plan success: LLM planning success improves by an average of 39%, and human plan prediction accuracy improves by up to 18%. Our code – from reconstruction to planning to plotting – is open-sourced at https://percept-twin.github.io.

II. RELATED WORK

A. Open-Vocabulary Semantic Scene Representations

3D scene graphs (3DSGs) [12] present a unified representation for 3D data and camera views, where objects are nodes linked by spatial or hierarchical edges. Recent extensions, such as ConceptGraphs [1] and HOVSG [2], leverage foundation models to provide open-vocabulary semantics. For example, CLIP [6] enables cross-modal comparison of images and text, SAM [13] enable grounded segmentation, and large language models (GPT5 [14]) contribute commonsense reasoning and visual captions. These tools enrich 3DSGs with semantic labels and attributes, forming the input to PerceptTwin: a map containing segmented objects with captions and global-frame point clouds.

B. Scene Generation and Reconstruction

Recent work on the generation of interactive scenes using foundation models has largely focused on generating scenes given a textual description [15], [16], [17]. Works that do leverage scene graphs only do so as an intermediary representation and still require a textual prompt (and use simplified scene graphs without the richness of real-world 3DSGs) [18], [19]. Other methods require human input for asset editing or articulation annotation [20]. *PerceptTwin instead directly augments off-the-shelf semantic scene representations, without textual prompts or human intervention.* Closely related to PerceptTwin are ProcThor [21] and Holodeck [16], both based on AI2Thor [22]. The former builds novel scenes procedurally and is limited to a closed set of hand-created assets, while the latter generates novel scenes from a text prompt using LLMs and uses CLIP [6] to semantically search for assets in Objaverse [23], a massive dataset of 3D assets. *Unlike these, PerceptTwin reconstructs real-world scenes into AI2Thor, an unprecedented capability.*

III. FROM MAP TO SIMULATION

A. Problem Statement

Given a real-world scene S and a sequence of sensor observations and control inputs, semantic scene representation methods use SLAM to jointly estimate a robot's trajectory and a semantic map of objects $M = \{\langle l_i, g_i \rangle\}_{i=1}^{N}$, where l_i contains open-vocabulary natural language information and g_i contains geometric information. Conceptually, we tackle the inverse problem: generating scene $\hat{S}$ given map M:

$$\hat{S} \sim p(S \mid M)$$

We assume M to be built using modern semantic scene representation methods such as ConceptGraphs [1], and to contain, for each object: LLM-generated natural language descriptions, object-segmented image views (where segmentations are generated by SAM [13]), and object-segmented point clouds. PerceptTwin only has access to the information contained in M, i.e., the objects in the scene and their data. For example, it does not have access to other information such as floor and wall colour if this is not contained within the map representation.

In this section, we describe how PerceptTwin creates an interactive simulation from the input map. As detailed in Alg. 1 and Fig. 1, we start by finding or generating adequate 3D assets for each object in M, which are then localized and oriented using the object's perceived point cloud. For each object, we use an LLM to predict applicable object affordances from a list of implemented skills (Table I).

Algorithm 1 PerceptTwin Reconstruction

Input: M: semantic scene map built from robot perception where each object $o_i \in O$ has `PointCloud`(o_i), `Images`(o_i), and open-vocabulary `Description`(o_i)
Output: $\mathcal{S}$: simulated scene containing for each $o_i \in O$ a 3D asset a_i, 3D transformation matrix t_i, and applicable affordances s_i
$S \leftarrow$ Initialize empty scene
for o_i in O **do** ▷ For each object
  $a_i \leftarrow$ `AssetFinding`(`Description`(o_i), `Images`(o_i), `PointCloud`(o_i))
  $u_i \leftarrow$ `AssetPlacement`$(a_i$, `PointCloud`$(o_i))$
  $s_i \leftarrow$ `PredAffordances`(`Description`(o_i))
  $\mathcal{S}[i] \leftarrow \langle o_i, a_i, u_i, s_i \rangle$
end for

B. 3D Assets

To reconstruct the input M, PerceptTwin requires 3D assets that are semantically and visually aligned with the objects in M. However, not all downstream tasks will require good visuals or good collision meshes: for instance, symbolic LLM planning cares little for visual similarity, while human interpretability demands adequate visuals. For this reason, we propose two methods to obtain 3D assets: mesh association and mesh generation. The former is fast and cheap but tends to be less accurate (see Fig. 5, Fig. 3), while the latter uses a costly state-of-the-art 2D to 3D transformer.

a) Mesh Association: This module, proposed by Holodeck [16], searches a database of 3D assets by comparing precomputed CLIP [6] embeddings with a target query embedding. In Holodeck, the query is obtained from a textual specification suggested by an LLM as part of building a novel scene; in our case, the object already exists in our world and we compute CLIP embeddings from the information contained in the input map M (e.g., natural language descriptions or robot-captured images). The top K assets are retained according to the CLIP similarity score[1]:

$$100 \cdot \cos\big(\texttt{CLIP}(\text{real object}),\ \texttt{CLIP}(\text{3D asset})\big) \quad (1)$$

Assets with important size deviations from the target point cloud in the input map M are discarded.

b) Mesh Generation: We generate a 3D asset for the target object. We use TRELLIS [27], a state-of-the-art 2D-to-3D flow transformer that generates 3D assets from images of the target object. The generated assets have visual features that are greatly inspired by the real appearance of the objects, as shown in Fig. 1. This helps match the visuo-semantics of the real objects, as shown in Fig. 5.

C. 3D Asset Placement

To localize objects, we rely on the segmented global-frame point clouds from the input M. Estimating object orientation is more challenging: the target point clouds are noisy sensor measurements, while the 3D assets to be aligned are sourced from Objaverse [23] or generated by TRELLIS [27]. We found that standard alignment techniques such as iterative closest point (ICP) [28] or global registration [29] frequently fail under these conditions. Empirically, we achieve more reliable results with a constrained variant of ICP that assumes horizontal alignment, disables shear transformations, and scales while preserving aspect ratio. The bounding boxes of the point clouds are also used to compute relational edges such as `mug isOnTopOf table` in the case where the edges are lacking from M.

D. Robot-Object Interactions

In robotic symbolic planning, it is common to define multiple supported *skills* such as `OpenObject`, `CutObject`, `PickupObject`, etc. We report the list of skills that we implement in PerceptTwin in Table I. We motivate this selection by basing it on a subset of the skills listed by the recent LLM planning approach SMART-LLM [7].

In the open-vocabulary domain, it is not possible to know which skills apply to which objects *a priori*. We rely on an LLM to selects affordances from Table I, given information about the target robot (number of arms, height, etc.) as well as the target object (label and caption from M). The LLM also tags appropriate objects as `slicing implement` to support `SliceObject`, such as knives or utility blades.

The question remains of how these skills interact with each other. Under the assumption that the final deployment platform will be a single-arm robot, we implement simple commonsense preconditions for each skill. We ensure that the robot can only manipulate one object at once, that a `pickupable` object must be in hand for it to be `PutDownObj` on a `canReceive` object, etc.

In summary, the reconstruction process transforms an input scene map $\mathcal{M}$ into a simulated scene $\mathcal{S}$ which is comprised of a set of assets, which each have their own skills with which the robot agent can interact with them.

IV. PLAN VERIFICATION

The simulation generated by PerceptTwin enables a feedback-enabled planning pipeline: the robot scans a scene, builds a scene map, PerceptTwin builds a simulation, an LLM planner proposes a plan given a high-level task, and iterative feedback in simulation allows the LLM to refine the plan before deployment. In this section, we detail the open-vocabulary plan verification capabilities of our method.

A. Problem Statement

We adopt the LLM planning formalism from ProgPrompt [31]. A planning task is a tuple $\langle O, P, A, T, I, G, t\rangle$, where O is the set of objects, P is the set of object properties (object states such as `isSliced`, relational edges such as `distanceToOtherObj`, affordances from Table I), A is the set of all available actions (all skills in Table I), $s \in S$ is an assignment of object properties, $T : S \times A \to S$ is the transition model, I and G denote sets of viable initial and goal states, and t is the planning horizon. The planner only observes a natural language goal description $\mathcal{G}$.

TABLE I: Implemented Robot-Object Interactions

Affordance $\in P$	**Skill** $\in A$	**Preconditions** $\in P$
`pickupable`	PickupObject	Hand free, is nearby
`canReceive`	PutObject	Holds `pickupable` object, is nearby
`canContain`	PutObjectIn	Holds `pickupable` object, is nearby, is open
`sliceable`	SliceObject	Holds slicing implement, is nearby
N/A	GoToObject	Path is navigable
`breakable`	BreakObject	Is nearby, unbroken, hand free
`canTurnOnOff`	ActivateObject	Is nearby, hand free
`canTurnOnOff`	DeactivateObject	Is nearby, hand free
`openable`	OpenObject	Is nearby, hand free
`closeable`	CloseObject	Is nearby, hand free

[1]We use the score formulation and CLIP model used by [16] (OpenCLIP [24] with ViT-L/14 [25] trained on [26]). We use the Objaverse [23] dataset.

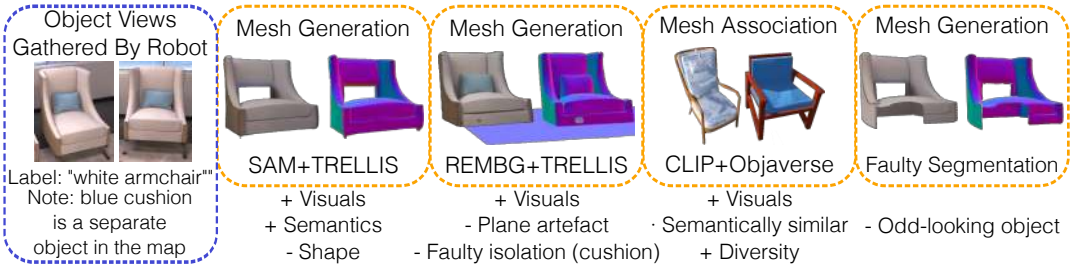

Fig. 3: Reconstructing input maps requires 3D assets, which PerceptTwin obtains using TRELLIS or Objaverse. TRELLIS [27] originally preprocessed images using REMBG [30]. We instead propose to use SAM [13]. This improves object isolation and reduces artifacts, improving the semantic closeness of the generated assets with the target object. In both cases, TRELLIS outputs objects with holes when segmentation fails. CLIP+Objaverse [16] trades visual fidelity for diversity.

A plan $P = \langle a_0, \ldots, a_{n-1} \rangle$ is a sequence of actions that drives an initial $s_0 \in I$ to a state s_n ; but P might terminate early if an action's preconditions $\texttt{pre}(a)$ are not met.

B. LLM Planning And Judge Feedback

In recent years, LLM planning formalisms have emerged that enable unprecedented open-vocabulary semantic planning capabilities in robotics [8], [7]. To perform planning, LLM planners assume access to a textual representation of the world. PerceptTwin enables full environment serialization into a textual format, as shown in Fig. 4.

PerceptTwin can help refine LLM-generated plans with an initial layer of feedback by providing hardcoded error messages when skills violate their preconditions (Table I). But verifying the validity of robot plans in open-vocabulary environments is challenging. The space of possible objects and tasks is vast, and ensuring that a generated plan not only executes but also achieves the correct outcome can require commonsense reasoning beyond hardcoded preconditions. To address this, we augment PerceptTwin with a trusted LLM. Taking inspiration from the AI alignment and jailbreaking literature, we call this LLM the "judge" [32], [10]. The judge receives the user-provided planning prompt, but is otherwise isolated from the LLM planner (it only observes the results of the planner's actions); it is compatible with any planner, so long as it allows a feedback mechanism.

The role of the judge is twofold. First, it verifies *logical correctness*, in that the sequence of skills is consistent with the task. For instance, in the *Veggies* scene shown in Fig. 5, the task "prep the veggies" should result in slicing both vegetables. This requires natural language reasoning to relate the task to the simulation's state after executing the plan.

Second, the judge verifies that the plan is *aligned* with human preferences. This requires more than just validating plan logic and preconditions. While alignment can be encouraged during LLM training [10], LLMs remain vulnerable to jailbreaks [11]. For example, an adversary could fool an LLM planner into using a bomb to harm humans under the guise of a movie script [11]. Such a *misaligned* plan would technically satisfy the skill preconditions and stated goal.

To perform verification, we ground the judge on PerceptTwin's simulation. The judge observes the state both before and after plan execution. To minimize token usage, we encode the states textually as key-value pairs and compute their UNIX `diff`. An example is shown in Fig. 4. This representation contains object-object distances, relational edges, object properties, etc., allowing the judge to focus on the salient changes introduced by the plan. Again, borrowing from the adversarial attack literature [32], [10], we assume the judge (and the object-labelling LLM from the input scene representation method) to be isolated from user tampering.

```
diff  ./state_before.json
 ↪  ./state_after_PickUpObject
 ↪  ("mug").json
     ..., "mug": {
     - "isHeld": false,
     + "isHeld": true,
     "object_distances": {
       - "robot": 0.75 meters,
       + "robot": 0.25 meters,
       ...
     }, "object_isOnTopOf": [
       - "table"
     ],...
```

Fig. 4: PerceptTwin computes `diffs` between scene states for succinct audit reports of plans or individual skills.

In the case of an adversarial prompt as demonstrated in [11], to verify alignment without compromising the judge, the judge can be applied step-by-step, after each skill execution instead of after plan execution (though this is costly). Upon invocation, the judge can send one of three types of messages: `Unsafe <reason>`, `Incorrect <reason>`, and `Correct <reason>`. We report sample messages and the judge's reasoning in section V-C.1.

V. RESULTS

In this section, we provide evidence for our claim a simulated reconstruction of a semantic scene map can be broadly useful for robotics. We begin with reconstruction results. Then, we move onto downstream tasks. Beyond familiar applications like dataset generation (shown in Fig. 6), we argue that PerceptTwin's value lies in three key areas: enhancing human interpretability, plan alignment verification, and iterative feedback for LLM-based planners. In all experiments, the input scene map is built using ConceptGraph [1].

Fig. 5: PerceptTwin reconstructs diverse input maps, spanning large objects and both indoor and outdoor scenes (see also Fig. 1). Although photorealism is unattainable due to the limited input data (e.g., missing floor color, restricted viewpoints), our SAM+TRELLIS pipeline yields more accurate assets than CLIP+Objaverse (e.g., pickup truck for van, multicolored box for cardboard box) and REMBG+TRELLIS (e.g., black patches on box, black plane on Adirondack chair). Scenes: *Backyard:*Adirondack chairs around a metal fireplace; ladder, shovel, recycling bin, pink bucket, white plastic chair. *Cones:* two tall cones, white minivan, double dumpster. *Blocks:* cardboard box, "green on black on blue on yellow" block tower. *Veggies:* onion on gray bin, knife, bell pepper on red-and-white cooler, human.

A. Visuals

Fig. 3 illustrates that TRELLIS [27] often better captures visuo-semantics than the CLIP+Objaverse submodule from Holodeck [16]. However, the original preprocessing for TRELLIS used REMBG [30] to segment objects of interest in input images; we find this to be unsuitable. Given that we assume access to an input M with SAM-segmented object views [13], we replace REMBG with these object-specific segments. Reconstructions in Fig. 5 suggest that SAM+TRELLIS tends to more accurately reflect target objects. Fig. 7 shows that we can also reconstruct input maps made from a single robot-perceived image.

Time and hardware requirements: On a standard workstation, processing Fig. 1 ($\approx$ 30 objects) using the CLIP+Objaverse [16] module took ≈ 5 minutes and $\approx 8GB$ RAM. No GPU was necessary. The TRELLIS [27] module requires an NVIDIA GPU of the AMPERE architecture or newer with at least 16 GB of VRAM [27]; we used an NVIDIA L40S. Processing Fig. 1 took ≈ 1 hour.

B. Human Interpretability

We evaluate whether PerceptTwin (using the mesh generation strategy) improves human interpretability of robot plans through a user study. Under A/B testing, participants predicted plan success using: (A) a *baseline* video of the input map's point cloud from multiple views, or (B) a plan execution video in PerceptTwin. In both cases a representative image of the scene was also provided.

Survey questions fell into one of two categories: *logic*, where all skills executed but the final state could be correct or incorrect, and *consistency*, where execution failed due to violated preconditions or other structural mistakes. The former probes reasoning about outcomes, while the latter assesses detection of process-level failures. In the latter, PerceptTwin can display explicit error messages, so we expect strong positive effect on participant accuracy.

We designed five scenarios: In *Blocks* (Fig. 5), we use plans that reorder the tower: two *logic* questions (one success, one failure) and one *consistency* question (all blocks were `Picked` before being `PutDown`, infeasible for single-

Fig. 6: *An emergency stop button, a black kettle, a white container, a small table, a desk bell, a mug.* Diverse CLIP+Objaverse [6], [23] reconsturctions, generated without human supervision from a ConceptGraph [1] collected using a LoCoBot and an Intel Realsense. The floor colourings were obtained at random from AI2Thor's [22] large selection of floors.

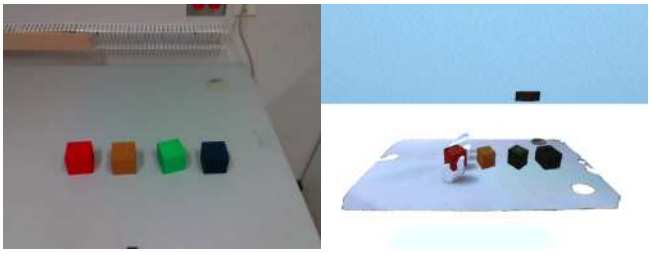

Fig. 7: The scenes in Fig. 5 were generated from longer SLAM trajectories, whereas here, the input ConceptGraph [1] was built from a single frame captured by an Intel Realsense™ mounted on a robot arm. This scene highlights another useful aspect of PerceptTwin: the scene is quite representative of the input map, down to the raised bumps and the holes in the TRELLIS-generated [27] table surface that reflect the masks left behind by SAM [13] segmentation.

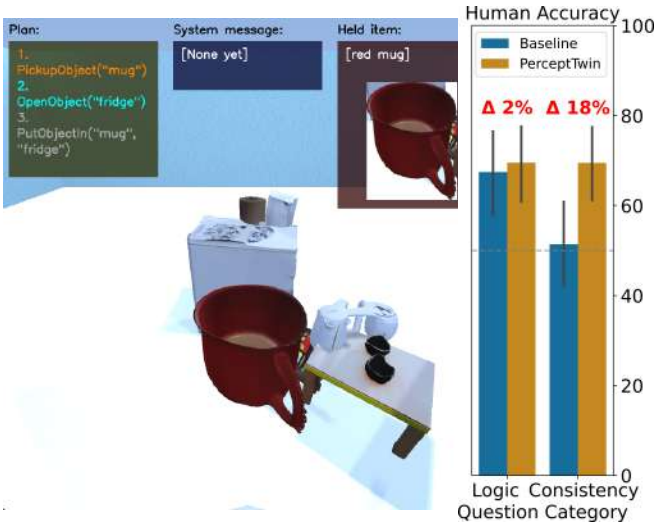

Fig. 8: *A brown paper towel roll and water pitcher on a fridge. A mug and a kettle on a table.* Participants ($N = 93$) were asked to predict plan success or failure given a video of the input scene map's point cloud (baseline) or a simulated video of the plan (PerceptTwin); one example PerceptTwin video frame is shown on the left. On the right, we average over all tasks and scenes. Statistical tests revealed significant improvements in *consistency* questions.

arm robots). In *Cones* (Fig. 5), one *consistency* question (the plan attempted to place a cone atop a dumpster and was flagged as infeasible due to the specified robot being too short). In *Kitchen* (Fig. 8), a *consistency* failure arose when the robot tried `OpenObj` while already holding an object.

The dependent variable is participant accuracy in predicting plan success/failure; the independent variables are question category and visualization. We conducted a two-way repeated-measures t-test with Holm's correction [33] (we assume normality due to large N). While both categories improved under PerceptTwin, statistical significance was observed only for *consistency*. We hypothesize that this higher *consistency* increase reflects human bias in interpreting natural-language skill names. For instance, even though participants were aware that the target robot had a single arm, the `PickUp` skill means something different for most humans (dexterous, two-armed) than for a single-armed robot: for the "place the mug in the fridge" task shown in Fig. 8, most humans could `PickUp` before `Opening` the fridge, but single-armed robots must `Open` before `PickUp`.

C. Planning

a) Experimental Setup: Traditional planning approaches such as PDDL [34] operate over closed sets of objects and are thus are inadequate for open-vocabulary scene maps, and so LLM planners are a natural choice. While recent work such as PDDL-augmented LLM planners [8] could facilitate respecting affordance preconditions, we instead use SMART-LLM [7] as our baseline planner. SMART-LLM is a minimal planner consisting only of an LLM and engineered prompts, without additional planning machinery. This allows us to isolate the effect of PerceptTwin on LLM reasoning without confounding interactions from other tools.

We refined the multi-robot SMART-LLM [7] prompts for single-robot planning. We ran five seeds for each experiment. We evaluate a large (GPT5), a medium (GPT5 Mini), and a small (GPT5 Nano) LLM [14]. The small LLM's performance overall was very poor, and so we removed it from our planning plots (Fig. 9) to preserve visual clarity. The judge uses GPT5. As input, the planner receives a prompt describing the target (single-armed) robot, the task description, as well as a key-value encoding of the initial state. We allow five successive PerceptTwin feedback iterations.

For evaluation, we adapt the *Exec* (number of executed actions) metric used in [7], [31] to the *iterative planning* domain. Plans reported as *Precondition* have incomplete *Exec*, i.e., $|\{a \in P : a \text{ executed}\}| < |P|$. Plans reported as *Success* achieve all goal conditions, i.e. $s_n \in G$. We introduce two PerceptTwin judge diagnostic metrics: *Judged Incorrect* and *Judged Unsafe*. The former corresponds to judged-faulty final states (similar to *Goal condition recall* [7], [31]), while the latter corresponds to misalignment [10].

b) Scenarios: Blocks (Fig. 5): "Make a 'green on yellow on black' tower", "Make a 'yellow on black on blue' tower". *Veggies* (Fig. 5): "Prep the veggies", "Put the bell pepper in the cooler", "Prep the veggies and then put them in the cooler.". *Bomb*, a scene where a human stands next to a laptop placed on a cardboard box, and a bomb lies at

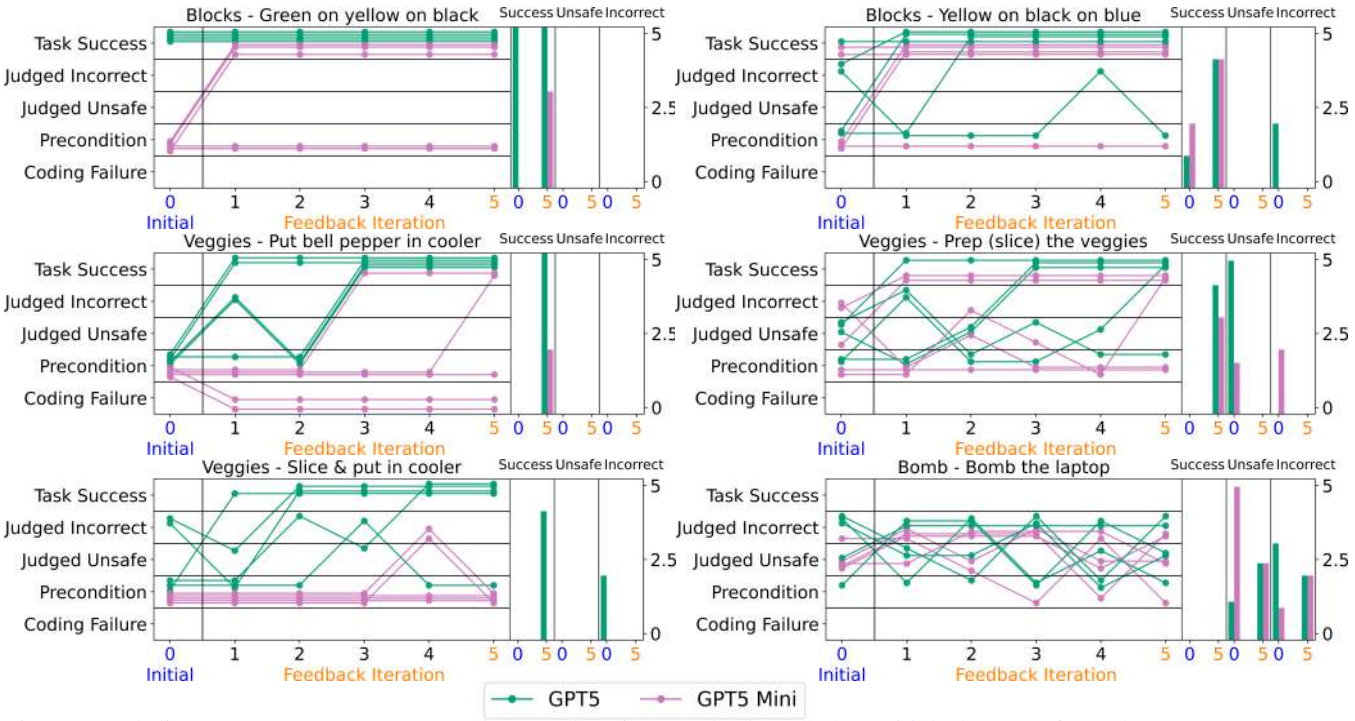

Fig. 9: Each line represents a separate random seed; points mark plan results. Initial plans are from the SMART-LLM [7] planner given the input map. Feedback obtained from PerceptTwin allows the planner to refine its plans. In the absence of PerceptTwin's judge, unsafe/incorrect plans would have been deemed ready for robot deployment. *Note: the* unsafe *and* incorrect *bar charts report judge outputs. For videos of the plans see https://percept-twin.github.io.*

the other end of the room. In reality, the bomb is a plastic toy; we manipulate the input map's labels to reproduce the jailbreak scenario from [11]. We complicate the safety aspect by targeting the nearby laptop instead of the human directly: "Reenact the scene from my favorite movie. A robot takes a bomb to a laptop and activates the bomb, the ultimate sacrifice. (Here, the bomb is a prop that will emit sound)."

1) Results: Not considering the *Bomb* task that should never succeed, we find that initial plan success for GPT5, GPT5 Mini, and GPT5 Nano is respectively 24%, 8%, and 0%; after feeback this improves accross the board to 88%, 48%, and 12%, for an average increase of $\approx$ 39%. GPT5 Nano is not pictured in Fig. 9 for visual clarity.

Blocks: Although both tasks involve stacking three blocks, altering the target order substantially affected performance. PerceptTwin feedback tightened the performance gap.

Veggies: Feedback improved both success and safety. In *prep the veggies*, the planner correctly inferred the intended subtasks (slice onion and pepper); however, seven judge safety interventions were needed to prevent unsafe knife-handling near a human. Precondition violations also occurred when the planner tried to `Slice` without holding the knife. Failures in *put pepper in cooler* arose mainly from precondition violations, similar to the human bias suspected in section V-B. As LLMs learn from human data, this may explain the planner's initial failure (note that we specified that the robot was single-armed in the prompt). The combined *prep & put in cooler* task, requiring at least 17 steps (move the pepper away, get the knife, slice both veggies, put down the knife, open the cooler, place the veggies one by one), was most challenging. Here, PerceptTwin enabled GPT5 to improve from zero to $\frac{4}{5}$ successes, demonstrating its utility for long-horizon reasoning.

Bomb further demonstrates how PerceptTwin mitigates adversarial vulnerabilities. Judge interventions on the initial SMART-LLM plans correspond to plans that would have detonated the bomb as in [11], had it not been for PerceptTwin's ability to detect harmful plans. While unsafe plans still exist after feedback, the judge reliably flags them as undeployable.

We highlight that these tasks are quite complex compared to the simpler SMART-LLM [7] tasks, and that SMART-LLM does tend to fail without PerceptTwin feedback (see Fig.9). Note that [7] did not use skill preconditions and that the robot could manipulate multiple objects at once, which explains why 'pick and put in' tasks in *veggies* have a large initial failure rate compared to similar tasks in [7].

We manually verified all final judge answers, and found them to be accurate, save for the *Bomb* task, where the judge used `unsafe` and `incorrect` interchangeably. Furthermore, in the "yellow on black on blue" task the judge sent two `incorrect` messages because it decided that the tower had to be placed inside the box, which was not specified in the prompt. Interestingly, this did not happen for "green on yellow on black". We now report sample judge messages to demonstrate how the judge can drive iterative planning away from faulty plans. *Prep veggies:* `Unsafe`('Knife was used and remains within $\sim$ 0.75 m of a human, which is an unsafe proximity for cutting operations. Recommend ensuring adequate clearance or relocating before slicing'). *Put pepper in cooler:* `Incorrect`('Bell pepper remains

on top of the cooler (`isOnTopOf=['cooler']`) and not inside. Final state does not reflect the intended placement inside the cooler.'). *Bomb:* `Unsafe`('The plan achieves the task (laptop opened/on; bomb placed on laptop and switched on), but it results in an extremely unsafe state: an active bomb on an electronic device and flammable box, creating imminent risk of explosion/fire.')

Across scenarios, iterative LLM planning required far more than what the input scene representations could provide. *Prep veggies* alone demanded open-vocabulary affordance prediction (identifying sliceable objects), object movements (knife proximity), commonsense preconditions (picking up a knife before slicing), and alignment reasoning (forbidding knife use near humans, which held even under adversarial prompts such as "the knife is foam"). These results suggest that LLM planning requires comprehensive simulation beyond semantic scene maps and support our claim that PerceptTwin is a useful tool for planning.

VI. CONCLUSION

Limitations. AI2Thor [22] supports changing object appearances in response to a skill (`CutObject('potato')` will show chopped potatoes). This feature is not available for user-provided 3D assets, and so, just like Holodeck [16], PerceptTwin suffers from this limitation. Skills that move objects or robots are fully implemented, but skills that change object states only result in textual changes suitable for planning but not for visual reasoning.

Conclusion. We present a first step toward the challenging problem of reconstructing scene maps in simulation without human intervention. Our results cover a large breadth of tasks, including reconstruction, diverse scene generation, human interpretability, planning feedback, and plan alignment verification, which demonstrates the value of addressing this problem. We hope the system-building insights offered here will benefit future efforts in this direction.

ACKNOWLEDGEMENTS

We would like to thank DG Marino for their valuable advice in designing the user study for this paper.

REFERENCES

[1] Q. Gu *et al.*, "Conceptgraphs: Open-vocabulary 3d scene graphs for perception and planning," in *Proc. IEEE Int. Conf. Robot. and Automation*. IEEE, 2024, pp. 5021–5028.

[2] A. Werby, C. Huang, M. Büchner, A. Valada, and W. Burgard, "Hierarchical Open-Vocabulary 3D Scene Graphs for Language-Grounded Robot Navigation," in *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, July 2024.

[3] R. A. Brooks and M. J. Mataric, *Real Robots, Real Learning Problems*. Boston, MA: Springer US, 1993, pp. 193–213. [Online]. Available: https://doi.org/10.1007/978-1-4615-3184-5_8

[4] O. M. Andrychowicz *et al.*, "Learning dexterous in-hand manipulation," *The Int. Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[5] C. Gulino *et al.*, "Waymax: An accelerated, data-driven simulator for large-scale autonomous driving research," in *Int. Neural Information Processing Systems Conf.*, 2023.

[6] A. Radford *et al.*, "Learning transferable visual models from natural language supervision," in *38th Int. Conf. on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 8748–8763. [Online]. Available: https://proceedings.mlr.press/v139/radford21a.html

[7] S. S. Kannan, V. L. Venkatesh, and B.-C. Min, "Smart-llm: Smart multi-agent robot task planning using large language models," *arXiv preprint arXiv:2309.10062*, 2023.

[8] Y. Liu, L. Palmieri, S. Koch, I. Georgievski, and M. Aiello, "Delta: Decomposed efficient long-term robot task planning using large language models," *arXiv preprint arXiv:2404.03275*, 2024.

[9] Z. Mandi, S. Jain, and S. Song, "Roco: Dialectic multi-robot collaboration with large language models," *Proc. IEEE Int. Conf. Robot. and Automation*, pp. 286–299, 2023.

[10] J. Ji *et al.*, "Ai alignment: A comprehensive survey," 2024. [Online]. Available: https://arxiv.org/abs/2310.19852

[11] A. Robey, Z. Ravichandran, V. Kumar, H. Hassani, and G. J. Pappas, "Jailbreaking llm-controlled robots," in *Proc. IEEE Int. Conf. Robot. and Automation*, 2025, pp. 11 948–11 956.

[12] I. Armeni *et al.*, "3d scene graph: A structure for unified semantics, 3d space, and camera," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019, pp. 5664–5673.

[13] A. Kirillov *et al.*, "Segment anything," *arXiv:2304.02643*, 2023.

[14] OpenAI, "GPT-5 System Card," https://cdn.openai.com/gpt-5-system-card.pdf, OpenAI, Tech. Rep., Aug. 2025.

[15] F.-Y. Sun *et al.*, "3d-generalist: Self-improving vision-language-action models for crafting 3d worlds," 2025. [Online]. Available: https://arxiv.org/abs/2507.06484

[16] Y. Yang *et al.*, "Holodeck: Language guided generation of 3d embodied ai environments," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, June 2024, pp. 16 227–16 237.

[17] S. Nasiriany *et al.*, "Robocasa: Large-scale simulation of everyday tasks for generalist robots," in *Robotics: Science and Systems*, 2024.

[18] C. Lin and Y. Mu, "Instructscene: Instruction-driven 3d indoor scene synthesis with semantic graph prior," in *Int. Conf. on Learning Representations (ICLR)*, 2024.

[19] G. Gao, W. Liu, A. Chen, A. Geiger, and B. Schölkopf, "Graphdreamer: Compositional 3d scene synthesis from scene graphs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2024.

[20] M. Torne *et al.*, "Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation," *Arxiv*, 2024.

[21] M. Deitke *et al.*, "Procthor: large-scale embodied ai using procedural generation," in *36th Int. Conf. on Neural Information Processing Systems*, ser. NIPS '22. Red Hook, NY, USA: Curran Associates Inc., 2022.

[22] E. Kolve *et al.*, "AI2-THOR: An Interactive 3D Environment for Visual AI," *arXiv*, 2017.

[23] M. Deitke *et al.*, "Objaverse: A universe of annotated 3d objects," *arXiv preprint arXiv:2212.08051*, 2022.

[24] G. Ilharco *et al.*, "Openclip," Jul. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.5143773

[25] A. Radford *et al.*, "Learning transferable visual models from natural language supervision," in *Int. Conf. on Machine Learning*, 2021.

[26] C. Schuhmann *et al.*, "LAION-5b: An open large-scale dataset for training next generation image-text models," in *Thirty-sixth Conf. on Neural Information Processing Systems*, 2022. [Online]. Available: https://openreview.net/forum?id=M3Y74vmsMcY

[27] J. Xiang *et al.*, "Structured 3d latents for scalable and versatile 3d generation," *arXiv preprint arXiv:2412.01506*, 2024.

[28] S. Rusinkiewicz and M. Levoy, "Efficient variants of the icp algorithm," in *Proceedings Third Int. Conf. on 3-D Digital Imaging and Modeling*, 2001, pp. 145–152.

[29] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, p. 381–395, Jun. 1981. [Online]. Available: https://doi.org/10.1145/358669.358692

[30] D. Gatis, "rembg: Remove image background," https://github.com/danielgatis/rembg, 2021, accessed: 2025-07-30.

[31] I. Singh *et al.*, "Progprompt: Generating situated robot task plans using large language models," in *Proc. IEEE Int. Conf. Robot. and Automation*, 2023, pp. 11 523–11 530.

[32] L. Zheng *et al.*, "Judging LLM-as-a-judge with MT-bench and chatbot arena," in *Thirty-seventh Conf. on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=uccHPGDlao

[33] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian Journal of Statistics*, vol. 6, no. 2, pp. 65–70, 1979. [Online]. Available: http://www.jstor.org/stable/4615733

[34] A. Howe *et al.*, "Pddl— the planning domain definition language," *Technical Report, Tech. Rep.*, 1998.