Seeing is Believing: Planning to Perceive with Foundation Models and Act Under Uncertainty

Linfeng Zhao[¶], Willie McClinton[*§], Aidan Curtis[*§], Nishanth Kumar[§],
Tom Silver[‡], Leslie Pack Kaelbling[§], Lawson L.S. Wong[¶]
[*]Equal Contribution, [¶]Northeastern University, [§]MIT, [‡]Princeton University

***Abstract*—Generalizable robotic mobile manipulation in open-world environments poses significant challenges due to long horizons, complex goals, and partial observability. A promising approach to address these challenges involves planning with a library of parameterized skills, where a task planner sequences these skills to achieve goals specified in structured languages, such as logical expressions over symbolic facts. While vision-language models (VLMs) can be used to ground these expressions, they often assume full observability, leading to suboptimal behavior when the agent lacks sufficient information to evaluate facts with certainty. This paper introduces a novel framework that leverages VLMs as a perception module to estimate uncertainty and facilitate symbolic grounding. Our approach constructs a symbolic belief representation with ternary logic and uses a belief-space planner to generate uncertainty-aware plans that incorporate strategic information gathering. We demonstrate our system on a range of challenging real-world tasks that require reasoning in partially observable environments. Simulated evaluations show that our approach outperforms both vanilla VLM-based end-to-end planning and VLM-based state estimation baselines, by planning for—and executing—strategic information gathering. This work highlights the potential of VLMs to construct belief-space symbolic scene representations, enabling downstream tasks such as uncertainty-aware planning.**

***Index Terms*—Partial Observability, Long-Horizon, Mobile Manipulation, Task & Motion Planning, Vision-Language Models**

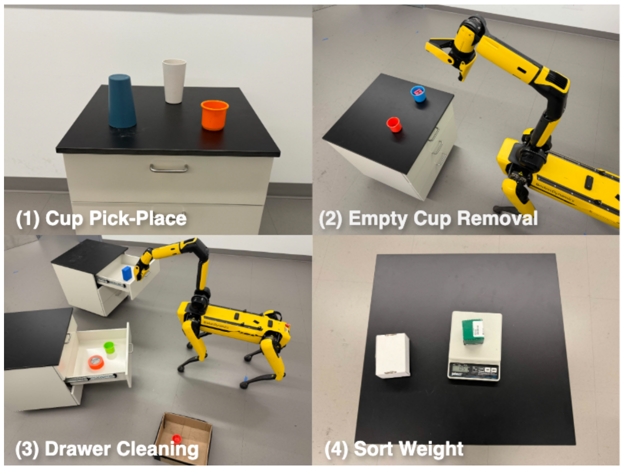

Fig. 1: **Example tasks demonstrating various uncertainty levels.** (1) Cup Pick-Place: a fully observable tabletop manipulation task with multiple cups. (2) Empty Cup Removal: requires inspecting cups from above to determine if they are empty before removal. (3) Drawer Cleaning: involves opening drawers to discover and remove objects inside. (4) Sort Weight: requires weighing sealed boxes on a scale to identify and dispose of empty ones. These tasks demonstrate increasing complexity in information gathering, from fully observable scenarios to those requiring several strategic information-gathering steps.

I. Introduction

Task-level planning [1] is critical to achieving long-horizon mobile manipulation tasks in complex environments [2]. However, many planning methods typically assume complete knowledge of the environment, including object properties and relations, rendering them inapplicable to partially observable settings where robots must dynamically discover, perceive, and interact with objects while resolving uncertainties. Some domains require integrating strategic information gathering with manipulation planning.

We address the challenge of operating in partially observable environments characterized by (1) uncertain object properties, (2) unknown object instances (number, type, location), and (3) goals specified via ungrounded natural language. In such domains, information gathering becomes crucial (Fig 1).

Handling partial observability on real robots, especially for long-horizon mobile manipulation tasks, is notoriously difficult . These problems are often modeled as Partially Observable Markov Decision Processes (POMDPs) [3]. Compared to tabletop setups with fixed cameras offering near-complete views, mobile robots face inherently greater uncertainty about the environment state and limitations from onboard, viewpoint-dependent perception. One major strategy for tackling POMDPs involves planning in *belief space*, aggregating information into an explicit belief state representing the agent's knowledge and uncertainty. Our work follows this strategy.

While excellent solutions for belief-space planning exist [4–7], they often require sophisticated algorithms, complex belief representations (e.g., full probability distributions), and carefully hand-engineered perception pipelines. An alternative strategy involves learning history-conditioned policies directly [8]. The recent foundation model approaches (VLMs, LLMs) typically fall into this category but often struggle with systematic planning under uncertainty and strategic information gathering, especially with limited real-robot data [9, 10].

In this paper, we aim to illustrate that the core ideas of belief-space planning can be applied relatively straightforwardly to mobile manipulation tasks by leveraging modern,

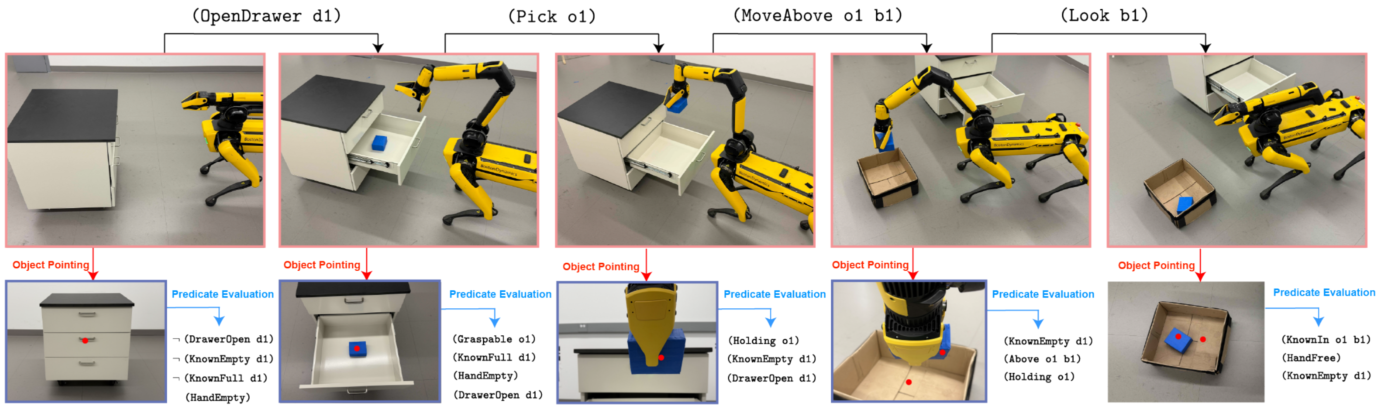

Fig. 2: **Example plan.** A task to put any object in the drawer into a paper bin. Because the drawer is closed, the robot needs to maintain uncertainty of the environment and plan under uncertainty to achieve the belief goal: `KEmpty+(drawer)` and `Inside(block, box)`. The sequence shows: (1) initial reach to the closed drawer (without knowing if the drawer is empty or not), (2) opening the drawer to reveal a blue block inside and update belief, (3) grasping the block from the drawer, (4) moving the block over the paper bin, and (5) successfully placing the block into the bin. This demonstrates how the robot handles uncertainty through interleaved information gathering (opening drawer to check contents) and manipulation actions (grasping and placing the block).

off-the-shelf components: classical planners (adapted via determinization, §III-D) and Vision-Language Models (VLMs) for evaluating perceptual predicates (§IV-A).

To develop a tractable approach for belief-space planning, we make simplifying assumptions: we treat predicate values as either known true, known false, or unknown; observations, while local, are assumed perfect; world-changing actions have deterministic effects; and there is no information loss over time or due to exogenous events (see §III-A for details). These assumptions allow for a lightweight belief representation and planning strategy.

Following Bonet and Geffner [11], we represent the agent's belief using three-valued logic (`True`/`False`/`Unknown`) for predicates, explicitly tracking known vs. unknown states via two binary predicates (*K-fluents* [11]). We formulate planning as finding a sequence of actions to reach a goal belief state (e.g., knowing an object property). Information-gathering actions are modeled systematically. To handle the resulting belief non-determinism tractably, we employ an *all-outcomes determinization* and online *replanning* strategy [11]. Our integrated pipeline leverages VLMs within this framework as flexible perception modules to evaluate predicate truth values (updating K-fluents) and to ground natural language goals.

We demonstrate our approach through experiments on mobile-manipulation tasks in simulated synthetic environments (constructed by real images) and on a physical Spot robot. These tasks require discovering object properties during execution. Our results show the system effectively manages partial observability, dynamically adapts plans, and leverages strategic information gathering more efficiently than end-to-end VLM baselines, particularly in simulation.

II. Related Work

A. Foundation Models for Planning

Recent advances using LLMs and VLMs enable approximate robot planning in diverse environments [9, 12–14], but evaluations highlight limitations in long-horizon or combinatorial problems when lacking an explicit planning model [15, 16]. These foundation models can be used in a "zero shot" way to directly generate plans from natural language goals and visual inputs, but they perform poorly with systematic planning, maintaining long-term context, handling uncertainty, and strategic information gathering [9, 17]. These shortcomings are particularly visible in partially observable scenarios requiring multi-step planning and deliberate information gathering.

B. Belief-Space Planning

Belief-space planning effectively handles uncertainty from partial observability [18–24]. It models the robot's knowledge as a belief state and plans to achieve belief goals (e.g., "know the lights are off") using both world-changing and information-gathering actions. This is crucial for mobile manipulation in partially known environments with unknown objects or states.

Belief-based systems typically involve a state estimator `SE` for belief updates and a policy π mapping beliefs to actions. Computing a complete policy offline is often intractable since only a tiny subset of possible beliefs will ever be reached. We instead follow an alternative approach that constructs partial plans online and replans if unanticipated observations are obtained [18]. Beliefs can be modeled as probability distributions or, alternatively, as sets of possible worlds represented compactly using three-valued logic (true, false, unknown) [25–29], which efficiently extends deterministic models to belief space [11].

Manual specification of planning domains is laborious; predicate and operator invention methods automate learning symbolic representations from data [30, 31]. Our framework reduces belief-space planning to replanning in a determinized belief state constructed from standard planning operators (Section III). This suggests that techniques for learning standard planning operators might be adaptable to learn the belief-space operators to reduce manual effort.

C. Object Uncertainty, Search, and Grounding

A key challenge in applying planning methods to partially-observed scenarios is handling uncertainty related to objects, including their existence, properties, and relevance to the task. Typical planning systems often assume a pre-specified domain of objects and properties, limiting their applicability when the set of objects is unknown or properties need active discovery [19].

One aspect of handling object existence uncertainty is *object search*, where the goal is to locate objects, potentially requiring exploration or manipulation [32, 33]. Much existing work focuses specifically on finding objects of a particular category, often using special-purpose strategies. However, many tasks require gathering information about object *properties* beyond just their location or category (e.g., determining if a container is empty). Classical systems have explored planning to perceive specific properties [34–36], but integrating general property verification with manipulation planning remains challenging. Our work aims for a more general mechanism for gathering critical property information, integrated within belief-space planning.

Furthermore, interpreting goals and understanding scenes becomes complex when objects are initially unknown. While foundation models like LLMs and VLMs have been used for goal parsing and scene understanding in structured and fully observable environments [16], partially observable settings demand active perception to dynamically discover objects and evaluate their properties [9]. While Sun et al. [37] explored LLMs for POMDP planning, their approach relied on leveraging VLM-based perception to implicitly interpret belief-updates and their effects on planning, instead of having an explicit belief space within a belief-space task planning framework. With an explicit representation, one common approach to handling the object uncertainty not addressed in Sun et al. [37] is using lifted representations (e.g., first-order logic goals like "all empty cups"), which allows generalization of objects of similar type. Each object has a grounded representation (referring to specific object instances like ‘cup1‘), which require mechanisms for grounding goals and actions as new objects are discovered during execution. Since integrating active perception with belief-space planning is crucial for enabling robots to iteratively gather information and refine their understanding, we chose an explicit representation over relying on VLM to do both belief-state estimation and planning.

III. Formulation: Modeling with Uncertainty

The problem of sequential decision-making under uncertainty in partially observable environments, such as mobile manipulation tasks, is typically modeled as a *partially observable Markov decision process (POMDP)* [3].

A. Modeling the Environment with Uncertainty

We approach this POMDP by converting it to a planning problem in an associated *belief-space MDP*. Our approach involves: (1) explicitly representing the agent's *belief state* b about the true world state, and (2) performing *online approximate planning* within this belief space, utilizing replanning to handle unexpected observations or outcomes resulting from the discrepancy between the planning model and the execution environment.

We model the partially observable planning problem in terms of: (1) a object-centric belief state b, discussed in §III-B; (2) a set of object-parameterized *actions* A that can be executed by the agent, modifying the belief state, discussed in §III-D; and (3) a *goal* G, specified as a first-order logical expression over relations that must be `True` in the terminal belief, discussed in §III-C.

To make this belief representation and the online planning approach tractable, we adopt the following core simplifying assumptions in our formulation:

- *Perfect Observations:* While observations are partial (local), the information obtained about the observed part of the state is assumed to be accurate and noise-free.
- *Deterministic World Dynamics:* Actions that intentionally change the physical state of the world (world-changing actions, A_w) are assumed to have deterministic effects on the underlying world state, as defined by their models.
- *Static Environment (No Exogenous Changes):* The state of the world only changes due to the robot's actions; there are no external events modifying the environment. Furthermore, we assume no loss of information over time (e.g., forgetting known facts).
- *Separable Action Effects:* We model actions such that their effects can be separated into either changing the physical world state or purely gathering information (updating belief), not both simultaneously.

These assumptions, particularly perfect observations and deterministic dynamics, significantly simplify the belief update and planning process. Unexpected observations and exogenous changes to object locations and properties, which violate these assumptions, are handled at *runtime* via replanning (with updated belief state; see §IV). Relaxing these assumptions typically requires more complex probabilistic belief representations and planning algorithms [19, 22].

B. Belief State Representation

Conceptually, the agent's belief state b represents a set of possible world states consistent with its observations and action history. We explicitly structure this belief state using the following components:

- A set of conjectured *objects* $\mathbb{O}$ currently believed to exist.
- A set of symbolic *predicates* $\mathcal{P}$ (e.g., `Empty(cup)`). They are object-parameterized functions using $\mathbb{O}$, and their values produce the symbolic *state variables*.

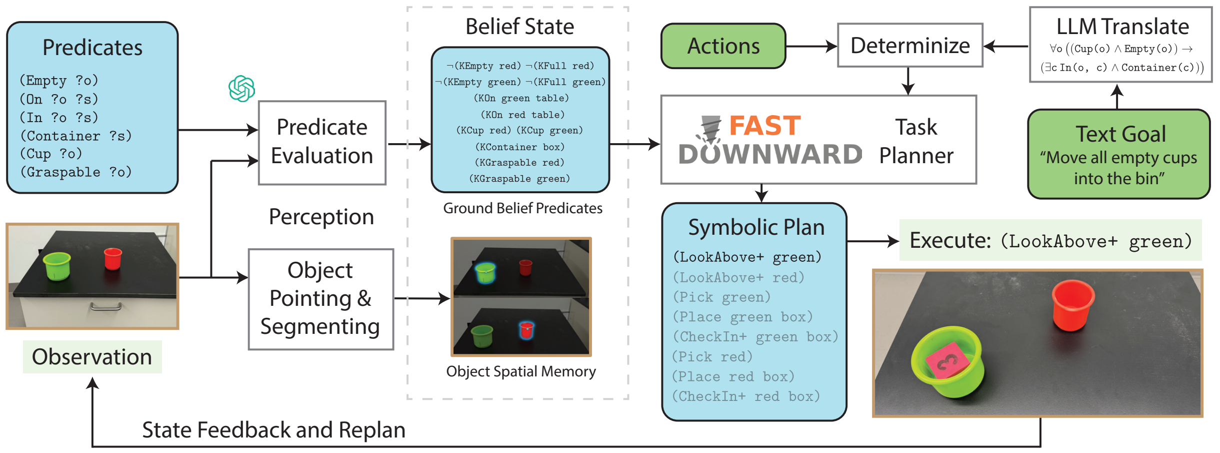

Fig. 3: **Pipeline overview.** Our system integrates perception, belief-state update, and planning. The example shows a task of moving empty cups to a bin, where the system must evaluate cup properties and plan appropriate manipulation actions. Before runtime, a text goal is first translated into symbolic specifications, which along with actions are determinized for the task planner. During a step of belief state update at runtime, given an observation (images and sensor inputs from a robot), the system performs two parallel processes for: (1) object pointing and segmenting to maintain a spatial memory of objects, and (2) predicate evaluation to ground belief predicates (e.g., Empty, On). The planner generates a symbolic plan based on the symbolic belief state, and the first action is executed to generate a new observation. The belief state is updated based on the new observation, and the process repeats until the goal is satisfied.

- Physical information, such as object memory (e.g., object positions) and robot configuration.

To handle uncertainty, the truth value for any predicate $P \in \mathcal{P}$ applied to objects $\mathbf{o} \subseteq \mathbb{O}$ can be True (T), False (F), or Unknown (U). Following Bonet and Geffner [11], we represent the agent's knowledge about these three-valued predicates using two binary *K-fluents*. For each belief predicate $P(\mathbf{o})$, we define $K_P(\mathbf{o})$ (known true) and $K_{\neg P}(\mathbf{o})$ (known false). The values regarding $P(\mathbf{o})$ is encoded as: (1) T if $K_P(\mathbf{o})$ is true, (2) F if $K_{\neg P}(\mathbf{o})$ is true, and (3) U if both $K_P(\mathbf{o})$ and $K_{\neg P}(\mathbf{o})$ are false. This allows explicit but determinsitic reasoning about knowns and unknowns. A consequence of our core assumptions (perfect observations, static world, no information loss) is *monotonic knowledge acquisition*: known facts (T or F) remain known unless explicitly changed by a world-changing action.

C. Goal Representation

A task goal G is specified as an *achievable* First-Order Logic (FOL) formula over belief-state predicates, typically requiring certain K-fluents to be true (e.g., $K_{\text{Empty}}(\text{cup1})$). Goals can be (1) *grounded*, referring to specific known objects in $\mathbb{O}$, or (2) *lifted*, quantifying over objects possibly including those yet undiscovered, which is crucial for open-world tasks.

The scope of achievable goals is limited by the predicate set $\mathcal{P}$: (1) Goals must be expressible using the defined predicates (or their K-fluents). (2) Goals involving specific unknown objects or properties (e.g., finding a green_cup before achieving Inside(green_cup)) may implicitly require prerequisite information gathering actions.

D. Action Modeling and Planning via Determinization

Actions A available to the agent are parameterized operators defined by preconditions and effects on the belief state b. Following our core assumptions (§III-A), we separate actions into two types based on their primary effect:

World-Changing Actions (A_w): These actions modify the physical world state deterministically. Their effects on the belief state b update K-fluents to reflect the known outcome. For example, picking up an object ?o:

```
(:action Pick
 :parameters (?o - object)
 :precondition (and (HandEmpty) (CanGrasp ?o))
 :effects (and (¬HandEmpty) (Holding ?o)))
```

Information-Gathering Actions (A_i): These actions aim to reduce uncertainty about a predicate $P(\mathbf{o})$ whose value is Unknown ($\neg K_P(\mathbf{o}) \wedge \neg K_{\neg P}(\mathbf{o})$). They only update the belief state by revealing information (making the predicate known True or known False), without directly changing the physical world state. For example, ObserveEmptiness(?cup) reveals whether a cup is empty.

Planning Model via Determinization: Information-gathering actions (A_i) introduce non-determinism into the belief state transition, as the observation outcome (True or False) is unknown beforehand. This complicates planning, potentially requiring complex conditional plans. To enable the use of efficient classical planners, we employ an *all-outcomes determinization* approach [11]. This involves (1) *compiling* each information-gathering action A_i (like ObserveEmptiness) into two deterministic PDDL actions, A_i^+ and A_i^-, one for each possible observation outcome (e.g., observing True or observing False), and (2) *optimistic planning*, choosing the outcome action (A_i^+ or A_i^-) that appears most beneficial for reaching the goal according to the deterministic model. For instance, ObserveEmptiness(?o) is split into two actions:

Algorithm 1 Belief-Space Planning and Execution (§IV)

```
Require: Initial belief state $b_0$ with initial objects $\mathbb{O}_0$, text goal $g_{\text{text}}$, predicates $\mathcal{P}$, actions $\mathcal{A}$
 1: $g \leftarrow$ Translate($g_{\text{text}}, \mathcal{P}, \mathbb{O}_0$)          ▷ Translation (§IV)
 2: $b \leftarrow b_0$
 3: while $\neg$Satisfied($g, b$) do
 4:     $p \leftarrow$ Plan($b, g, \mathcal{A}$)          ▷ Generate plan (§IV-D)
 5:     if $p =$ None then return False          ▷ Goal is infeasible.
 6:     for $a$ in $p$ do
 7:         $o \leftarrow$ Execute($a$)          ▷ Get observation (§IV-A)
 8:         $b \leftarrow$ BeliefUpdate($b, a, o$)          ▷ Update (§IV-B)
 9:         if $\neg$ExpectedEffects($a, b$) then
10:             break          ▷ Replan with updated belief (§IV-D)
11:         end if
12:     end for
13: end while
14: return True          ▷ Goal achieved
```

```
(:action ObserveEmptiness+ ; Outcome is True
 :parameters (?o - object)
 :precondition (and (CanObserve ?o)
                    (not (KEmpty+ ?o)) (not (
                       KEmpty- ?o)))
 :effect (KEmpty+ ?o)) ; Effect: Known True

(:action ObserveEmptiness- ; Outcome is False
 :parameters (?o - object)
 :precondition (and (CanObserve ?o)
                    (not (KEmpty+ ?o)) (not (
                       KEmpty- ?o)))
 :effect (KEmpty- ?o)) ; Effect: Known False
```

Here, KEmpty+ and KEmpty- represent K_{Empty} (known true) and $K_{\neg\text{Empty}}$ (known false), respectively. The precondition (not (KEmpty+ ?o)) (not (KEmpty- ?o)) ensures the action is only applicable when the emptiness status of ?o is Unknown. This determinization allows plan generation using a classical planner, but the optimism means that discrepancies between the planned outcome and the actual observation during execution necessitate runtime *replanning* to correct the course of action (§IV).

Executing Symbolic Plans: We assume that each symbolic action in a generated plan corresponds to an executable skill or program. These skills are responsible for achieving the action's specified effects in the physical world. Internally, they might employ methods like motion planning or utilize learned policies to realize the desired outcome, while their failures will be handled via replanning, similar to [38].

IV. Implementation: Planning under Uncertainty

In this section, we detail the runtime pipeline for planning under uncertainty on a mobile-manipulation robot, building upon the formulation described in §III. We term our system BKLVA: ***B**elief-space planning with **K**-fluents, **L**anguage-based goal-grounding, **V**LM-based perception and estimation, and information-gathering **A**ctions*. The execution pipeline takes as input (see Algorithm 1): (1) a PDDL domain definition corresponding to the belief-space actions $\mathcal{A}$, (2) a natural language goal g_{text}, (3) pretrained foundation models for perception (detailed in §IV-A), and (4) an initial belief state b_0.

The execution then proceeds in an observe-update-plan-execute cycle, as outlined in Algorithm 1:

(1) Perception and (2) Belief Update (Observe, BeliefUpdate) The system processes sensor observations using VLMs to identify objects and evaluate relevant predicates (detailed in §IV-A). This information is then integrated into the current belief state b through data association and predicate updates, potentially discovering new objects (detailed in §IV-B).

(3) Planning (Plan) Given the current belief state b and goal G, the symbolic planner (Fast Downward [39]) generates a plan p using the determinized PDDL action descriptions derived from our formulation (§III). This high-level plan assumes downward refinability into executable skills.

(4) Execution (Execute) The first action $a = p[0]$ of the plan is executed using the corresponding low-level skill. After execution and subsequent observation/belief update, the system checks if the outcome matches the planner's optimistic assumption. If not, replanning is triggered (detailed in §IV-D).

This cycle continues until either the belief state satisfies the goal ($b \models G$) or the goal is determined to be infeasible. Replanning occurs from the current, updated belief state whenever expectations are violated.

A. Perception via Vision-Language Models

Our perception system utilizes Vision-Language Models (VLMs) to process visual observations and extract both object detections and predicate evaluations, corresponding to the Observe step in Algorithm 1. The perception pipeline consists of two main components:

1) Object Detection and Localization: For object detection, we utilize MOLMO [40], a pre-trained model with a pointing feature that enables object localization. Given an observation (e.g., an image $\bar{o}$) and a set of textual prompts describing potential objects (slightly tuned for the objects in tasks), MOLMO identifies objects of interest $\mathbb{O}'$, providing their pixel locations in the image. These detections are then processed using the Segment Anything Model (SAM) [41] to generate segmentation masks. Combined with depth information, these masks yield the center of mass for each object in a map coordinate frame (maintained by an underlying SLAM-based odometry system).

2) Predicate Evaluation: For each detected object in $\mathbb{O}'$, we query a VLM (e.g., GPT-4o [42]) to evaluate the truth values of relevant predicates $\mathcal{P}$. For example, given an image containing a drawer, we might ask "Is the drawer empty?" to evaluate EmptyContainer(drawer1). The VLM acts as a belief-space grounding classifier, determining whether each predicate is known-true (K_P), known-false ($K_{\neg P}$), or remains unknown ($\neg K_P \wedge \neg K_{\neg P}$). This evaluation is done in a batched manner for all relevant predicates across all camera views at the current time step for efficiency, potentially using a single query to evaluate multiple predicates simultaneously. Details on VLM prompting strategies, batching implementation, and cost/timing considerations are provided in Appendix. While VLM inference can be computationally intensive, calls are typically limited within the execution loop (e.g., once per observation cycle or per skill execution for pre/post-condition checks), making the approach practical.

B. Belief State Update

The belief-state update process (`BeliefUpdate` in Algorithm 1) integrates the structured observation o (containing $\mathbb{O}'$, L, S, and predicate evaluations from §IV-A) with the existing belief state b, maintaining consistency in object tracking and predicate evaluations:

1) Object List Update: When new observations yield object detections $\mathbb{O}'$, we merge them with the existing object set $\mathbb{O}$ in the belief b. This involves data association, primarily based on object unique identifiers (represented in language) across all observations, to match observed objects $\mathbb{O}'$ with known objects $\mathbb{O}$. Unmatched and verified novel detections are added to $\mathbb{O}$. For example, if we detect "cup4" near a previously known "drawer1", and it doesn't match any existing object based on location, we add it to our object set $\mathbb{O}$. This process ensures consistent object tracking while allowing for the discovery of new objects.

2) Predicate Value Update: For all objects visible in the current observation, we update the truth values (K-fluents) of associated predicates in the belief b based on the VLM's evaluations from §IV-A2. The belief update mechanism follows a monotonic knowledge acquisition principle, consistent with our formulation (§III): unknown predicates ($\neg K_P \wedge \neg K_{\neg P}$) can transition to known states (K_P or $K_{\neg P}$) based on confident VLM outputs, but known predicates never revert to unknown states. This reflects the assumption that observations are perfect (§III).

C. Planning in Belief Space

In the planning process, corresponding to the `Plan` step in Algorithm 1, we follow a procedure similar to [38] where we are given a task and the robot will plan to generate actions that are likely to achieve the goal from the current state. Planning is decomposed into two levels: skill sequencing and continuous parameter selection. Skill sequencing consists of generating a *skeleton*, e.g., (`MoveTo(box,` θ_1`)`, `Pick(box, floor,` θ_2`)`, `MoveTo(table,` θ_3`)`, `Place(box, table,` θ_4`)`). Given a skeleton, we select continuous parameters using $\theta \sim$ *parameter policies* defined for each skill.

We sample and execute each skills *greedily*. If the skill terminates and does not meet its success condition (as defined by the operator effects), we replan. During the evaluation, the robot continues to plan and execute until a maximum number of actions is reached. We refer the reader to other references [39, 43] for a more formal description of the planning techniques we use in this work.

D. Skill Plan Execution

This section details the `Execute` step and the replanning logic within our observe-update-plan-execute cycle (§IV, Algorithm 1).

1) Execution: The `Execute` step takes the first symbolic action $a = p[0]$ from the current plan (e.g., `PlaceOn(cup1, bin0)`) and executes the corresponding parameterized *skill*. Skills encapsulate the low-level control logic required to perform the action. Some skills may involve internal computations or closed-loop control for robustness, similar to approaches like Kumar et al. [38]. The low-level aspects, including grounding symbolic object parameters (like 'cup1') to continuous geometric values (e.g., poses, trajectories) and handling physical constraints via motion planning, are managed by the skill execution module. We assume a skill is a function that can execute and may fail, and their failure will be handled via replanning (§III-A). Further details on the skill execution framework, parameter grounding methods, and low-level implementation specifics are provided in Appendix.

2) Replanning: Replanning is triggered in the following scenarios: (1) when an observation action yields an unexpected outcome (e.g., finding a drawer non-empty when assumed empty by the planner's optimistic determinization), or (2) when new objects are incidentally discovered during perception (§IV-B1). When replanning is triggered, we update the belief state with the new information and generate a new plan from the current state by re-invoking the `Plan` step.

V. Experiments

Through our experiments, we aim to answer several key questions: (i) Does our structured approach (using VLMs for belief state estimation for symbolic planning) effectively handle uncertainty? (ii) How efficient is our belief-space planning strategy compared to alternatives? (iii) Can this pipeline extend to a real-world robot scenario with real perception and control?

Environments. We evaluate the approaches on (1) a synthetic environment with real images for mobile manipulation, and (2) a Spot real-robot mobile-manipulation environment. A synthetic task is defined within a fully-observable transition graph that accepts symbolic actions and returns both images and other non-visual predicates (e.g,. whether the agent is holding an object or reachable to an object), though the images may not reveal the entire system state (for example, a drawer's contents remain hidden from view until it is opened). Consequently, similar to a real-robot scenario, the agent maintains a partially observable belief model to plan effectively. We build synthetic domains using real-world images (taken by phones or robots), where each node in the environment's transition graph represents a distinct viewpoint (represented as images and other non-visual predicates). When the agent selects an action, the environment presents images of the resulting state, allowing the agent to update its belief, which initially may contain uncertain or incomplete knowledge of object properties. This symbolic environment with real images (1) systematically evaluates symbolic belief-space planning that integrates perception and task reasoning, (2) reduces the

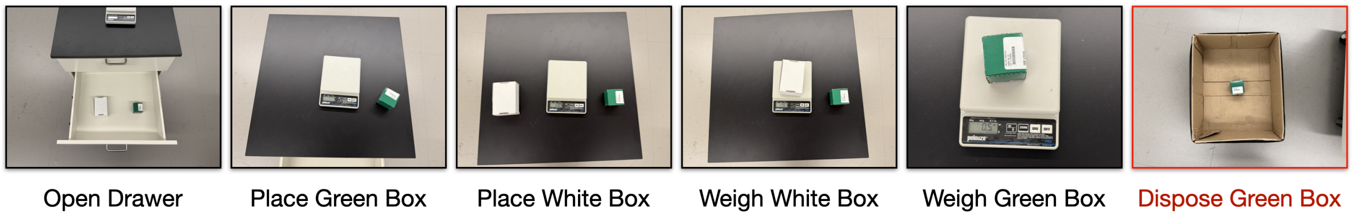

Fig. 4: *Sort Weight*. An example task in our synthetic environment with real images. The agent needs to open the drawer and retrieve sealed boxes to weigh them. The boxes cannot be opened by the agent but can only be measured indirectly by a scale. The goal is to find empty boxes and remove it to a bin, and a few notable states are shown in the figure. The optimal path takes 14 steps.

Methods \ Tasks	Cup Pick-Place		Drawer Cleaning		Sort Weight	
	Success	SPL	Success	SPL	Success	SPL
Random	0%	0.00 ± 0.00	0%	0.00 ± 0.00	0%	0.00 ± 0.00
VLM End-to-End	30%	0.15 ± 0.24	0%	0.00 ± 0.00	0%	0.00 ± 0.00
VLM (Captioning) + LLM	**100%**	0.49 ± 0.07	10%	0.04 ± 0.11	0%	0.00 ± 0.00
VLM (Labeling) + LLM	90%	0.69 ± 0.28	0%	0.00 ± 0.00	0%	0.00 ± 0.00
BKLVA (Ours)	**100%**	$\mathbf{1.00 \pm 0.00}$	**80%**	$\mathbf{0.32 \pm 0.16}$	**70%**	$\mathbf{0.46 \pm 0.32}$

TABLE I: Performance comparison across synthetic tasks. Success indicates success rate (%) and SPL indicates average success rate weighted by (normalized inverse) path length (between 0 to 1, with 1 being optimal).

complexity of physical control, and (3) manages randomness when comparing against multiple baselines. These features enable us to explore long-horizon belief-space planning tasks.

We evaluate on the following tasks:

- *Cup Pick-Place (Synthetic)*. This is a fully-observable task that tests the agent's ability to rearrange some cups on a table into a box. A set of information-gathering actions is provided to verify if an agent understands whether uncertainty is present and needed to be reduced.
- *Drawer Cleaning (Synthetic)*. In this task, one or more drawers of cabinets may contain various objects. Initially, the robot does not know which drawers hold items. It must open each drawer to observe its contents, dynamically update its belief regarding newly found objects, and then remove those objects to a box. An illustration with one drawer and one block is shown in Figure 2. The synthetic task features 2 objects and 1 drawer.
- *Sort Weight (Synthetic)*. The agent needs to remove closed boxes by using a scale to measure the weight of the boxes. The boxes are hidden inside cabinets, so the agent needs to find the cabinets and measure their weight. After finding the empty boxes, the agent needs to remove them to a bin.
- *Empty Cup Removal (Robot)*. A Spot robot is used to execute this task, where three cups are placed on two tables, while the robot does not know if cups contain contents or are empty. From a normal front-facing view, the robot cannot distinguish whether a cup is empty or not. It must navigate closer, take a camera perspective from above, and inspect the contents. Once the robot identifies an empty cup, it removes it to a bin. We include this setup as a real-robot demo of the system, integrated with real-world object detection, segmentation, and belief-state update. See Figure 3 and the supplementary video.

Approaches. We evaluate the performance of our approach in comparison to several baselines across simulation and real-robot environments. The environments vary in terms of observability and task complexity, including both fully observable and partially observable settings, as well as short-horizon and long-horizon tasks. The approaches are summarized here, where more details are provided in the appendix:

- *Random* planner that selects object-parameterized skills and valid object parameters uniformly at random.
- *VLM End-to-end Planning* uses VLMs for end-to-end planning without explicit handling of uncertainty. It has been explored in tabletop manipulation tasks and web agent literature.
- *VLM State Captioning + LLM Planning* uses VLM to generate a text caption of the history of observations (with images and other non-visual predicates). An LLM then outputs a sequence of actions using the caption.
- *VLM Predicate Labeling + LLM Planning* uses VLM to perceive predicate values and LLM for planning, but does not handle uncertainty explicitly either.
- *BKLVA (Ours)*: An approach that uses belief-space operators integrated with VLM-based belief-state estimation to plan to gather information in order to achieve the goal.

For VLM-based planner or VLM-based state captioning, we provide the history of visual observations to them for handling partial observability. Replanning is used and needed when the expected outcome is not achieved, particularly in partially observable environments where belief-space operators are determinized to an optimistic outcome. See the appendix for more details.

Experimental Setup. For each synthetic task, we run 10 random seeds and report the average results, controlling for variance in the perception system, foundation model calls, planning, low-level control, and other factors. For all LLM- and VLM-based approaches, we use `GPT-4o` with zero-shot prompting, with available objects, operators parameterized by objects, operators' preconditions and effects, current state (LLM-based), and observation (VLM-based) or history (partially observable variants) provided in the context window. We use the following metrics to evaluate the approaches for performance and efficiency in handling uncertainty in completing tasks: (1) task success rate, (2) average number of symbolic actions task plan length required for successful runs of a task weighted by the success rate, also referred as "SPL" (Success rate weighted by Path Length) in visual navigation. The agent succeeds when it reaches the goal state and fails when it reaches the max number of steps, transitions to an illegal state (e.g., pick up full cup), or gets stuck in a state. For *real-robot* experiments, we use the Spot robot and only demonstrate with our approach. We use the same set of symbolic actions (operators) as the synthetic tasks, including the information-gathering and manipulation actions. Each action is additionally associated with a skill that executes on the robot. More details are provided in the appendix.

Results and Discussion. We first evaluate whether our structured approach, which integrates VLM-based predicate estimation with symbolic planning, effectively handles uncertainty (**Q1**). As shown in Table I, our method consistently outperforms baselines on partially observable tasks like *Drawer Cleaning* and *Sort Weight*. Unlike end-to-end VLM methods or caption-based state representations, our system selectively gathers information only when needed (e.g., opening drawers or weighing boxes if relevant), thereby avoiding redundant actions.

We also compare the efficiency of our belief-space planner against alternatives on gathering information (**Q2**). We notice a few patterns that baselines do not handle well and result in lower success rates and longer SPL: (1) it relies on commonsense knowledge to plan, which may take extra steps based on its commonsense; (2) it does not capture some subtle diffences in state, causing failure or inefficiency on the task; (3) it does not necessarily understand the history and may retry the same actions at same or similar states that occurred before; (4) it does not output correct format of the state. As an example, the *Cup Pick-Place* task provides information-gathering actions while all information is provided, all baselines tend to take extra steps and result in lower SPL, because it does not understand the actions can be directly executed without additional information needed. A hypothetical approach that separates information gathering and manipulation stages by exhaustively removing uncertainty (e.g., opening all drawers first) would be prohibitively time-consuming in large or intricate environments. By focusing on only those uncertainty necessary to fulfill the task goals, our method achieves higher success rates with fewer actions.

Finally, we test whether the proposed pipeline extends to real-robot scenarios on the *Empty Cup Removal* task (**Q3**), with a video provided in the supplementary material. A Spot robot inspects three cups placed on different tables to determine which are empty before disposing of them. The same VLM-based predicate evaluation prompts are used as in synthetic tasks on images. Our demonstration shows that the pipeline of VLM-based belief state estimation and symbolic planning in BKLVA transfers well to real-world perception and control despite additional noise and uncertainty, as seen in the accompanying video. Thus, the real-robot trials confirm that our framework remains effective in physical settings, suggesting strong potential for more complex mobile-manipulation tasks beyond laboratory conditions.

Overall, these results validate each of our three research questions: (1) explicit predicate-based reasoning and belief maintenance of BKLVA enables robust handling of uncertainty, (2) belief-space planning improves efficiency compared to naive or exhaustive approaches, and (3) the BKLVA pipeline scales to a real robot with minimal modification (besides additional real-world perception and control), demonstrating its viability for real-world partial observability.

VI. Limitations and Discussion

To make belief-space planning tractable and demonstrate its core benefits, our framework incorporates several simplifying assumptions, which also highlight avenues for future enhancement.

First, we simplify perception by assuming *perfect observations* from the VLM and represent belief using *ternary logic* (True/False/Unknown). This facilitates integration but ignores perceptual noise and graded belief. Future work could increase robustness by adopting *probability-valued fluents* and *Bayesian belief updates* to handle imperfect VLM outputs more naturally. Second, for planning efficiency, we employ *optimistic determinization* of information-gathering actions. This allows the use of classical planners but is a simplification compared to fully *probabilistic planning frameworks* that explicitly reason about stochastic outcomes, which could yield more robust plans at higher computational cost. Third, we assume a *static world* between actions and *monotonic knowledge acquisition* (no information loss). This simplification is reasonable for many short-horizon tasks but limits applicability in dynamic environments or over longer periods. Modeling *exogenous effects* and information decay, at least in the belief update, would extend the framework's realism.

Further simplifications enabling our current focus include: (1) relying on *manually defined action operators*, whereas learning or extracting these could increase adaptability; (2) addressing *perception scope* primarily through VLM predicate grounding, leaving complex object search and long-term data association for future work; and (3) abstracting away low-level *skill execution details*, where tighter integration with planning methods is needed for robust physical deployment.

While these simplifications enabled us to demonstrate the value of VLM-integrated belief-space planning, relaxing them, particularly towards more probabilistic representations and reasoning, constitutes important next steps.

VII. Conclusion

In this paper, we presented a novel approach to belief-space robot planning that effectively addresses the challenges of partial observability and uncertainty in partially observable environments on mobile-manipulation robots. By integrating belief-space planning with information-gathering actions and leveraging vision-language models (VLMs) as belief-state estimators, our method demonstrated superior performance in handling long-horizon tasks in diverse, partially observable settings. The ability to reason about information gathering using ternary belief-space predicates enabled the robot to systematically reduce uncertainty, leading to higher task-success rates and improved efficiency compared to baseline approaches.

Our results highlight the importance of combining high-level belief-space reasoning with robust perception systems in robotic planning frameworks. Unlike baselines that lacked explicit uncertainty handling or efficient planning for information gathering, our approach demonstrated the advantage of integrating symbolic planning with modern perception systems, particularly VLMs. By automating the process of belief-state estimation with VLMs, we made belief-space planning more feasible in real-world scenarios, addressing the complexity of manually designing predicates for dynamic, unstructured environments.

This work serves as a foundation for future research in belief-space planning. While we focused on leveraging VLMs for belief-state estimation, future extensions could explore learning belief-space operators, improving sampling strategies, and integrating end-to-end learning to bridge the gap between planning and execution. These advancements could further enhance the practicality of belief-space planning and enable more adaptive and capable robotic systems in partially observed settings.

References

[1] Tomás Lozano-Pérez, Joseph L. Jones, Emmanuel Mazer, and Patrick A. O'Donnell. Task-level planning of pick-and-place robot motions. *Computer*, 22(3):21–29, 1989.

[2] Sriram Yenamandra, Arun Ramachandran, Mukul Khanna, et al. Towards open-world mobile manipulation in homes: Lessons from the neurips 2023 homerobot open vocabulary mobile manipulation challenge. In *NeurIPS Competition Track*, 2023.

[3] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998.

[4] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated robot task and motion planning in belief space. 2012. MIT DSpace, Accepted July 2012.

[5] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 2021.

[6] Hanna Kurniawati, David Hsu, and Wee Sun Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and systems*, volume 2008. Citeseer, 2008.

[7] Robert Platt Jr, Russ Tedrake, Leslie Kaelbling, and Tomas Lozano-Perez. Belief space planning assuming maximum likelihood observations. In *Robotics: Science and systems*, 2010.

[8] Sushmita Bhattacharya, Sahil Badyal, Thomas Wheeler, Stephanie Gil, and Dimitri Bertsekas. Reinforcement learning for pomdp: Partitioned rollout and policy iteration with application to autonomous sequential repair problems. *IEEE Robotics and Automation Letters*, 5(3):3967–3974, 2020. doi: 10.1109/LRA.2020.2978451.

[9] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Proceedings of The 6th Conference on Robot Learning*, 2023.

[10] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. In *Proceedings of the 6th Conference on Robot Learning (CoRL)*, 2022.

[11] Blai Bonet and Hector Geffner. Planning under partial observability by classical replanning: Theory and experiments. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

[12] Rishi Hazra, Pedro Zuidberg Dos Martires, and Luc De Raedt. Saycanpay: Heuristic planning with large language models using learnable domain knowledge. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI)*, 2024.

[13] Aidan Curtis, Nishanth Kumar, Jing Cao, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Trust the proc3s: Solving long-horizon robotics problems with llms and constraint satisfaction. In *Proceedings of the 8th Conference on Robot Learning (CoRL)*, 2024.

[14] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, et al. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2023.

[15] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning (ICML)*, 2022.

[16] Tom Silver, Varun Hariprasad, Reece S. Shuttleworth, Nishanth Kumar, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Pddl planning with pretrained large language models. In *NeurIPS Workshop on Foundation Models for Decision Making*, 2022.

[17] Karthik Valmeekam, Sarath Sreedharan, Matthew Marquez, Alberto Olmo, and Subbarao Kambhampati. On the planning abilities of large language models: A critical investigation with a proposed benchmark. In *Proceedings of The 37th Conference on Neural Information Processing Systems*, 2023.

[18] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Unifying perception, estimation and action for mobile manipulation via belief space planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2952–2959. IEEE, 2012.

[19] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, 32(9-10):1194–1227, 2013.

[20] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Pre-image backchaining in belief space for mobile manipulation. In *Robotics Research: The 15th International Symposium ISRR*, pages 383–400. Springer, 2017.

[21] Caelan Reed Garrett, Chris Paxton, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Dieter Fox. Online replanning in belief space for partially observable task and motion problems. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[22] Aidan Curtis, George Matheos, Nishad Gothoskar, Vikash Mansinghka, Joshua Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Partially observable task and motion

planning with uncertainty and risk awareness. In *Robotics: Science and Systems (RSS)*, 2024.

[23] Aidan Curtis, Xiaolin Fang, Leslie Pack Kaelbling, Tomás Lozano-Pérez, and Caelan Reed Garrett. Long-horizon manipulation of unknown objects via task and motion planning with estimated affordances. In *Proceedings of the 2022 IEEE International Conference on Robotics and Automation (ICRA)*, 2022.

[24] Leslie Pack Kaelbling, Alex LaGrassa, and Tomás Lozano-Pérez. Specifying and achieving goals in open uncertain robot-manipulation domains. In *Robotics: Science and Systems Workshop on Models and Representations for Human-Robot Communication*, 2021.

[25] Matthew L. Ginsberg. Multivalued logics: A uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4(3):256–316, 1988.

[26] Stephen Cole Kleene. *Introduction to Metamathematics*. North-Holland Publishing Company, 1952.

[27] Edgar F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4):397–434, 1979.

[28] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8(3): 338–353, 1965.

[29] Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. Using abstraction for generalized planning. In *Proceedings of the International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2008.

[30] Tom Silver, Ashay Athalye, Joshua B. Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Learning neuro-symbolic skills for bilevel planning. In *Proceedings of the 6th Annual Conference on Robot Learning (CoRL)*, 2022.

[31] Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomás Lozano-Pérez, Leslie Kaelbling, and Joshua B Tenenbaum. Predicate invention for bilevel planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12120–12129, 2023.

[32] Lawson L. S. Wong, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Manipulation-based active search for occluded objects. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[33] Lawson L. S. Wong, Thanard Kurutach, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Object-based world modeling in semi-static environments with dependent dirichlet process mixtures. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[34] Mohan Sridharan, Jeremy Wyatt, and Richard Dearden. Planning to see: A hierarchical approach to planning visual actions on a robot using pomdps. *Artificial Intelligence*, 174(11):704–725, 2010.

[35] Siddharth Srivastava, Stuart J. Russell, Paul Ruan, and Xiang Cheng. First-order open-universe pomdps. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence (UAI)*, 2014.

[36] Xinkun Nie, Lawson L. S. Wong, and Leslie Pack Kaelbling. Searching for physical objects in partially known environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[37] Lingfeng Sun, Devesh K. Jha, Chiori Hori, Siddarth Jain, Radu Corcodel, Xinghao Zhu, Masayoshi Tomizuka, and Diego Romeres. Interactive planning using large language models for partially observable robotic tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[38] Nishanth Kumar, Tom Silver, Willie McClinton, Linfeng Zhao, Stephen Proulx, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Jennifer Barry. Practice makes perfect: Planning to learn skill parameter policies. In *Proceedings of Robotics: Science and Systems (RSS)*, 2024.

[39] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[40] Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, et al. Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models, 2024.

[41] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

[42] OpenAI. Gpt-4 technical report. *arXiv preprint 2303.08774*, 2023.

[43] Maria Fox and Derek Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)*, 20:61–124, 2003.

VIII. Extended Setup Information

A. Environment-Agent Interface for Spot

We employ the Boston Dynamics Spot robot, which is equipped with six RGBD cameras. One camera is mounted on the manipulator arm, providing a close-up view for fine manipulation tasks. The remaining five are body cameras (two front-facing, one left-facing, one right-facing, and one rear-facing), enabling a 360° view for object detection, tracking, and situational awareness.

The Python SDK for Spot is used to send velocity and motion commands to the robot, receive feedback on joint states and robot pose, and retrieve RGB and depth images from all six cameras. By leveraging this SDK, we can synchronize robot actions with sensor data retrieval, ensuring that each execution step is accompanied by timely, high-quality visual feedback.

B. Parameterized Skill Execution

Our approach separates planning decisions from low-level control. A classical symbolic planner outlines sequences of actions (like 'pick' or 'place'), but the details of each action—such as determining grasp points or collision-free paths—come from parameterized robot skills.

For instance, when the robot needs to pick an object, we first verify that its gripper is free. Then, we figure out where to grab the object, how much force to apply, and what collision checks are needed. These skill parameters are fed into low-level controllers that oversee individual movements and sensor checks.

We find that splitting tasks in this manner keeps the planner's high-level reasoning clean and avoids entangling it with control-specific intricacies [38**?**]. In future work, we plan to detail the underlying motion planners and control loops used for these skill executions, as well as refine thresholds like approach velocity or force limits.

C. Simulated Environment Setup

We develop a real-image interactive symbolic environment (RISE) to replicate tasks without running on the real robot. This environment is partially observable and uses pre-captured images—one image per state—to provide synthetic RGBD inputs for our planner and visual pipeline. Each state is annotated with which objects are currently visible, whether the robot's gripper is free or holding an object, and other relevant properties. By linking these states with "transitions" that correspond to high-level actions (e.g., `PickObjectFromTop` or `MoveToHandViewObject`), we can define which actions can succeed or fail from a given state.

The primary data structure keeps track of:

1) Unique State IDs
2) Associated camera images (for simulating "views" of the environment)
3) A record of which objects appear in that view and which, if any, the robot is currently holding
4) A set of transitions connecting one state to another, each labeled with the operator name and whether the action succeeds or fails

For instance, if the planner issues a `PickObjectFromTop` action in the "initial" state, the mock environment checks whether there is a corresponding transition from "initial" under `PickObjectFromTop`. If so, it moves us to the new state (e.g., "holding_block") and returns updated mock sensor data. Because transitions are explicitly defined, we can also encode negative outcomes—like failing to pick an object—by directing the environment to a "pick_failed" state.

To better test tasks involving uncertainty, we add "belief operators." These can require or modify belief-related predicates (e.g., `Unknown_ContainerEmpty`) instead of purely physical ones. For example, an `ObserveContainerContent` operator can transition the system from an unknown content state to a known one, based on an imagined "camera check."

Because this environment structure stays close to the real system's domain model, it uncovers logical flaws before deployment. We also minimize discrepancies by matching coordinate frames and rough sensor noise levels wherever possible. In doing so, we ensure that end-to-end tests in this mock environment—involving both symbolic planning and VLM-based perception—translate smoothly onto the physical Spot robot.

IX. Methods and Baselines Details

A. Comparison of Approaches

To assess the performance of our system, we compare it with several baselines. End-to-end policy learning attempts to map raw pixels directly to actions without explicit symbolic inference; while flexible, it often struggles with task generalization and replanning. Classical planners without visual reasoning rely entirely on predetermined symbols and have difficulty adjusting when environmental assumptions become invalid. Reactive execution systems use local triggers without a global notion of a task-level plan, leading to limited adaptability.

In Table II, we summarize several approaches in terms of how they address "current-state uncertainty" (object existence, object property classification, etc.) and "future-state uncertainty" (belief-space planning, long-horizon tasks). For instance, BHPN [19] models partial observability and can handle unknown object states through belief-space planning, but it relies primarily on hand-coded perception and does not natively ground goals from language. EES [38] leverages a symbolic planner for complicated tasks but does not explicitly deal with object existence or property uncertainty and requires an enumerated domain.

VLM End-to-End approaches (e.g., [9]) can parse language goals but generally do not perform systematic belief updating or fully handle uncertainty about concealed or unknown objects. Similarly, VLM(Text)+LLM Plan can handle language-based goals and partially reason about objects but typically lacks explicit belief updates and relies

Components	Current-State Uncertainty: Perception				Future-State U.: Planning	
	Goal Grounding	O. Existance	O. Property	Belief Classifier	Belief-Space	Long-Horizon
BHPN [19]	?	✓	✓	?	✓	✓
EES [38]	×	×	×	×	×	✓
VLM End-to-End	✓	?	×	×	?	×
VLM (Text) + LLM Plan	✓	?	?	?	?	×
VLM (Predicate) + LLM Plan	✓	?	✓	✓	?	×
Ours	✓	?	✓	✓	✓	✓

TABLE II: Comparison of different approaches based on uncertainty handling and planning capabilities. O. stands for object.

on ad-hoc textual prompts to track state. A stronger variant, VLM(Predicate)+LLM Plan, uses predicate-level queries on images but lacks robust planning under unknown states or objects.

Our method (bottom in the table) combines all these features: it (1) accepts language-derived goals and newly discovered objects, (2) systematically updates beliefs about object existence or properties using VLM-based classifiers, and (3) runs a full belief-space planner for strategic information gathering and long-horizon tasks. This unified approach gives it comprehensive coverage of both current-state uncertainties (like unknown container contents) and future-state uncertainties (planning how and when to gather information).

X. Additional Implementation Details

A. Visual Predicates Evaluated by VLM

As part of our system, we define a set of predicates that are evaluated using the Visual Language Model (VLM). Each predicate has a name, a list of argument types (written as ?movable, ?container, etc.), and a specific prompt text. The prompt guides the VLM in determining whether the predicate is true based on a given image or scene description. Below is a bullet list of the main VLM-based predicates we use:

- **On**(`?movable`, `?base`)
  Prompt: "This predicate describes when a movable object is on a flat surface. It conflicts with the object being Inside a container. Please check the image and confirm the object is on the surface. If it's truly on top (e.g., on a table or floor) and not inside something else, answer yes. Otherwise, answer no."
- **Inside**(`?movable`, `?container`)
  Prompt: "Use this predicate when an object is inside a container and not just resting on a surface. If you see the object's shape overlapping the container's interior, answer yes. If it's merely on top or partially overlapping, answer no."
- **Blocking**(`?base`, `?base`)
  Prompt: "Check if one object is blocking the robot from accessing or viewing another. If so, answer yes; if not, answer no."
- **NotBlocked**(`?base`)
  Prompt: "Confirm that no object is blocking the given object. If you see no obstruction, answer yes. Otherwise, answer no."
- **NotInsideAnyContainer**(`?movable`)
  Prompt: "This predicate is true if the object is not inside any container. If it is inside something, answer no."
- **InHandViewFromTop**(`?robot`, `?movable`)
  Prompt: "Answer yes if the robot's camera is positioned above the movable object to see into it (e.g., looking inside a cup). If unsure or the view is angled, answer no."
- **Unknown_ContainerEmpty**(`?container`), **Known_ContainerEmpty**(`?container`), **BelieveTrue_ContainerEmpty**(`?container`), **BelieveFalse_ContainerEmpty**(`?container`)
  Prompts: "[Answer: yes/no only] (1) *Unknown_ContainerEmpty*: You do not have enough information to decide if the container is empty. (2) *Known_ContainerEmpty*: You are confident whether it is empty or not. (3) *BelieveTrue_ContainerEmpty*: You believe the container is empty (e.g., you see only a single color inside). (4) *BelieveFalse_ContainerEmpty*: You believe the container has contents (multiple colors or visible items)."
- **Unknown_Inside**(`?movable`, `?container`), **Known_Inside**(`?movable`, `?container`), **BelieveTrue_Inside**(`?movable`, `?container`), **BelieveFalse_Inside**(`?movable`, `?container`)
  Prompts: "[Answer: yes/no only] (1) *Unknown_Inside*: You are uncertain whether the object is inside the container. (2) *Known_Inside*: You can confidently tell if it is inside or not. (3) *BelieveTrue_Inside*: You believe the object is fully inside the container. (4) *BelieveFalse_Inside*: You believe the object is not inside (e.g., it is on top or separate)."

We also group certain predicates into belief categories or container-related predicates, but the fundamental structure remains the same: each predicate is evaluated by the VLM in response to a carefully written prompt that captures how to determine truth from the image.

B. World-State and Belief-Space Operators

Below, we present a representative set of operators in a PDDL-style pseudocode. Each operator includes parameters, preconditions, and effects, with line breaks and indentation for readability.

Note that these operators are shared for the synthetic RISE tasks and for the real-robot tasks.

```
(:action MoveToReachObject
 :parameters (?robot - Robot ?object -
     BaseObject)
 :precondition (and
   (NotBlocked ?object)
   (NotHolding ?robot ?object)
 )
 :effect (and
   (Reachable ?robot ?object)
))

(:action MoveToHandViewObject
 :parameters (?robot - Robot ?object - Movable
     )
 :precondition (and
   (NotBlocked ?object)
   (HandEmpty ?robot)
 )
 :effect (and
   (InHandView ?robot ?object)
))

(:action MoveToBodyViewObject
 :parameters (?robot - Robot ?object - Movable
     )
 :precondition (and
   (NotBlocked ?object)
   (NotHolding ?robot ?object)
 )
 :effect (and
   (InView ?robot ?object)
))

(:action PickObjectFromTop
 :parameters
   (?robot - Robot ?object - Movable ?surface -
       Immovable)
 :precondition (and
   (On ?object ?surface)
   (HandEmpty ?robot)
   (InHandView ?robot ?object)
   (NotInsideAnyContainer ?object)
   (IsPlaceable ?object)
   (HasFlatTopSurface ?surface)
 )
 :effect (and
   (Holding ?robot ?object)
   (not (On ?object ?surface))
   (not (HandEmpty ?robot))
   (not (InHandView ?robot ?object))
   (not (NotHolding ?robot ?object))
))

(:action PlaceObjectOnTop
 :parameters
   (?robot - Robot ?held - Movable ?surface -
      Immovable)
 :precondition (and
   (Holding ?robot ?held)
   (Reachable ?robot ?surface)
   (NEq ?held ?surface)
   (IsPlaceable ?held)
   (HasFlatTopSurface ?surface)
   (FitsInXY ?held ?surface)
 )
 :effect (and
   (On ?held ?surface)
   (HandEmpty ?robot)
   (NotHolding ?robot ?held)
   (not (Holding ?robot ?held))
))

(:action MoveToHandViewObjectFromTop
 :parameters (?robot - Robot ?object - Movable
     )
 :precondition (and
   (NotBlocked ?object)
   (HandEmpty ?robot)
 )
 :effect (and
   (InHandViewFromTop ?robot ?object)
   (InHandView ?robot ?object) ; derived from
       the top view
))

(:action ObserveCupContentFindNotEmpty
 :parameters (?robot - Robot ?cup - Container
     ?surface - Immovable)
 :precondition (and
   (On ?cup ?surface)
   (InHandViewFromTop ?robot ?cup)
   (HandEmpty ?robot)
   (NotHolding ?robot ?cup)
   (Unknown_ContainerEmpty ?cup)
 )
 :effect (and
   (Known_ContainerEmpty ?cup)
   (BelieveFalse_ContainerEmpty ?cup)
   (not (Unknown_ContainerEmpty ?cup))
))

(:action ObserveCupContentFindEmpty
 :parameters (?robot - Robot ?cup - Container
     ?surface - Immovable)
 :precondition (and
   (On ?cup ?surface)
   (InHandViewFromTop ?robot ?cup)
   (HandEmpty ?robot)
   (NotHolding ?robot ?cup)
   (Unknown_ContainerEmpty ?cup)
 )
 :effect (and
   (Known_ContainerEmpty ?cup)
   (BelieveTrue_ContainerEmpty ?cup)
   (not (Unknown_ContainerEmpty ?cup))
))

(:action ObserveDrawerEmpty
 :parameters (?robot - Robot ?container -
     Container)
 :precondition (and
   (Unknown_ContainerEmpty ?container)
   (DrawerOpen ?container)
   (Reachable ?robot ?container)
 )
 :effect (and
   (Known_ContainerEmpty ?container)
   (BelieveTrue_ContainerEmpty ?container)
   (not (Unknown_ContainerEmpty ?container))
))

(:action ObserveDrawerNotEmpty
 :parameters (?robot - Robot ?container -
     Container)
 :precondition (and
   (Unknown_ContainerEmpty ?container)
   (DrawerOpen ?container)
   (Reachable ?robot ?container)
 )
 :effect (and
   (Known_ContainerEmpty ?container)
   (BelieveFalse_ContainerEmpty ?container)
   (not (Unknown_ContainerEmpty ?container))
))
```

C. Prompts to Pretrained Models

Below are example prompts used to query our vision-language and language models:

VLM Predicate Evaluation Prompt

```
Your goal is to answer questions related to
    object relationships in the
given image(s) from the cameras of a Spot
    robot. Each question is independent
while all questions rely on the same set of
    Spot images at a certain moment.

We will use the following predicate-style
    descriptions to ask questions:
   Inside(object1, container)
   Blocking(object1, object2)
   On(object, surface)

Some predicates may include 'KnownAsTrue' or '
    KnownAsFalse'.
You should respond 'Yes' or 'No' but never '
    Unknown'.
If you don't know the answer for 'KnownAsTrue'
     or 'KnownAsFalse' predicates,
say 'No'.

Here are VLM predicates we have, note that
    they are defined
over typed variables. Example: (<
    predicate-name> <obj1-variable>:<obj1-type
    > ...)

VLM Predicates (separated by line or newline
    character):
{vlm_predicates}

Examples (separated by line or newline
    character):
Do these predicates hold in the following
    images?
1. Inside(apple:object, bowl:container)
2. On(apple:object, table:surface)
3. Blocking(apple:object, orange:object)
4. Blocking(apple:object, apple:object)
5. On(apple:object, apple:object)
6. On(apple:object, bowl:container)
7. EmptyKnownTrue(bowl:container)
8. EmptyKnownFalse(bowl:container)
9. Inside(bowl:container, bowl:container)

Answer with explanation and Yes/No for each
    question. Keep each explanation
and answer in a single line, with no empty
    lines between responses:
1. I can see the apple is clearly contained
    within the bowl's interior. [Yes]
2. The apple appears to be floating above the
    table, not making contact. [No]
3. The apple is positioned directly in front
    of the orange, preventing access. [Yes]
4. ... [No]
5. ... [No]
6. ... [Yes]
7. ... [Yes]
8. ... [No]
9. ... [No]

Actual questions (separated by line or newline
     character):
Do these predicates hold in the following
    images?
{question}

Answer with explanation and Yes/No for each
    question. Keep each explanation and
answer in a single line, with no empty lines
    between responses:
```

LLM Planner Prompt

```
You are highly skilled in robotic task
    planning, breaking down intricate and
    long-term tasks into distinct primitive
    actions.
Consider the following skills a robotic agent
    can perform. Note that each of these
    skills takes the form of a `
    ParameterizedOption` and may have both
    discrete arguments (indicated by the `
    types` field, referring to objects of
    particular types),
as well as continuous arguments (indicated by
    `params_space` field, which is formatted
    as `Box([<param1_lower_bound>, <
    param2_lower_bound>, ...], [<
    param1_upper_bound>, <param2_upper_bound>,
     ...], (<number_of_params>,), <
    datatype_of_all_params>)`).

{options}

Preconditions indicate the properties of the
    scene that must be true for you to execute
     an action. The effects are what will
    happen to the scene when you execute the
    actions.
You are only allowed to use the provided
    skills. It's essential to stick to the
    format of these basic skills. When
    creating a plan, replace
the arguments of each skill with specific
    items or continuous parameters. You can
    first describe the provided scene and what
     it indicates about the provided
task objects to help you come up with a plan.

Here is a list of objects present in this
    scene for this task, along with their type
     (formatted as <object_name>:<type_name>):
{typed_objects}
```

```
And here are the available types (formatted in
     PDDL style as `<type_name1> <type_name2
    >... - <parent_type_name>`). You can infer
     a hierarchy of types via this:
{type_hierarchy}

Here is the current state of the scene:
{state_str}

Finally, here is an expression corresponding
    to the current task goal that must be
    achieved:
{goal_str}

Here is the history of actions executed so far
     (if any):
{action_history}

Please return a plan that achieves the
    provided goal from the current state.
Please provide your output in the following
    format:
1. First write "Explanation of scene + your
    reasoning" followed by your explanation
2. Then write "Plan:" on a new line
3. Then write each action on a new line in
    EXACTLY this format (no numbers, no code
    blocks, no extra formatting):
skill_name(object1:type1, object2:type2)[
    param1, param2]

For example:
Explanation of scene + your reasoning
This is a simple pick and place task where we
    need to...

Plan:
MoveToObject(robot:robot, cup:movable_object)
    []
PickObject(robot:robot, cup:movable_object)[]
MoveToLocation(robot:robot, table:surface)[]
PlaceObject(robot:robot, cup:movable_object,
    table:surface)[]
OpenDrawer(robot:robot, drawer:container)[]

Do not include any numbers, bullet points,
    code blocks, or other formatting. Just
    write the plan exactly as shown above.
...
```

VLM Planner Prompt

```
You are highly skilled in robotic task
    planning, breaking down intricate and
    long-term tasks into distinct primitive
    actions.
Consider the following skills a robotic agent
    can perform. Note that each of these
    skills takes the form of a `
    ParameterizedOption` and may have both
    discrete arguments (indicated by the `
    types` field, referring to objects of
    particular types),
as well as continuous arguments (indicated by
    `params_space` field, which is formatted
    as `Box([<param1_lower_bound>, <
    param2_lower_bound>, ...], [<
    param1_upper_bound>, <param2_upper_bound>,
     ...], (<number_of_params>,), <
    datatype_of_all_params>)`).

{options}

You are only allowed to use the provided
    skills. It's essential to stick to the
    format of these basic skills. When
    creating a plan, replace
the arguments of each skill with specific
    items or continuous parameters. You can
    first describe the provided scene and what
     it indicates about the provided
task objects to help you come up with a plan.

Here is a list of objects present in this
    scene for this task, along with their type
     (formatted as <object_name>: <type_name>)
    :
{typed_objects}

And here are the available types (formatted in
     PDDL style as `<type_name1> <type_name2
    >... - <parent_type_name>`). You can infer
     a hierarchy of types via this:
{type_hierarchy}

Finally, here is an expression corresponding
    to the current task goal that must be
    achieved:
{goal_str}

Here is the history of actions executed so far
     (if any):
{action_history}

Please return a plan that achieves the
    provided goal from an initial state
    depicted by the image(s) below.
IMPORTANT: You must follow this EXACT format (
    including exact spacing and newlines):

Explanation of scene + your reasoning
<your explanation here>

Plan:
<action1>
<action2>
...

Each action must be in this exact format with
    no extra spaces or formatting:
skill_name(object1:type1, object2:type2)[
    param1, param2]

Example output:
Explanation of scene + your reasoning
The robot needs to pick up a cup from the
    table and place it on the shelf.

Plan:
MoveToObject(robot:robot, cup:movable_object)
    []
PickObject(robot:robot, cup:movable_object)[]
MoveToLocation(robot:robot, shelf:surface)[]
PlaceObject(robot:robot, cup:movable_object,
    shelf:surface)[]
```

```
CRITICAL:
- Do not add any numbers, bullet points,
    asterisks, or code blocks
- Do not add any extra newlines between
    actions
- Write "Plan:" exactly like that, with the
    colon and one newline after
- Each action must be on its own line with no
    extra formatting
```