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

PSALM-V: Automating Symbolic Planning in Interactive Visual Environments with Large Language Models

Wang Bill Zhu Miaosen Chai Ishika Singh Robin Jia Jesse Thomason
Department of Computer Science
University of Southern California
†Corresponding author: wangzhu@usc.edu

***Abstract*— We propose PSALM-V, one of the first autonomous neuro-symbolic learning systems able to induce symbolic *action semantics (i.e., pre- and post-conditions)* in visual environments through interaction. PSALM-V bootstraps reliable symbolic planning without expert action definitions, using LLMs to generate heuristic plans and candidate symbolic semantics. Previous work has explored using large language models to generate action semantics for Planning Domain Definition Language (PDDL)-based symbolic planners. However, these approaches have primarily focused on text-based domains or relied on unrealistic assumptions, such as access to a predefined problem file, full observability, or explicit error messages. By contrast, PSALM-V dynamically infers PDDL problem files and domain action semantics by analyzing execution outcomes and synthesizing possible error explanations. The system iteratively generates and executes plans while maintaining a tree-structured belief over possible action semantics for each action, iteratively refining these beliefs until a goal state is reached. Simulated experiments of task completion in ALFRED demonstrate that PSALM-V increases the plan success rate from 37% (Claude-3.7) to 74% in partially observed setups. Results on two 2D game environments, RTFM and Overcooked-AI, show that PSALM-V improves step efficiency and succeeds in domain induction in multi-agent settings. PSALM-V correctly induces PDDL pre- and post-conditions for real-world robot BlocksWorld tasks, *despite low-level manipulation failures from the robot*. Videos and resources at https://psalmv.github.io/.**

I. Introduction

Planning in interactive, visual environments presents a fundamental challenge for embodied artificial intelligence. Symbolic planners have traditionally been a powerful tool for structured decision-making, generating sequences of actions to transition from an initial state to a goal state. These planners rely on expert-defined action semantics, typically encoded in domain-specific languages such as the Planning Domain Definition Language (PDDL; [1]). Constructing these logical action semantics requires extensive manual effort and domain knowledge, limiting scalability to new environments.

Recent LLM and robot research has explored leveraging large language models (LLMs) to alleviate the need for human-authored action semantics by generating PDDL-like action definitions [2], [3]. While promising, these existing approaches have strong and unrealistic assumptions, such as:

- **Focus on text-based domains**: Real-world scenarios often require visual interaction to understand environmental dynamics.
- **Reliance on full observability**: Many real-world scenarios involve partial observability, where agents must infer missing information from past interactions.
- **Predefined, fully observed symbolic initial state**: While PDDL problem files characterizing the initial state of the world are provided in some environments, generating them, particularly in visual environments, is complex.
- **Explicit error messages**: Error messages are usually not available in real-world exploration. Previous work by [2] highlights failures without explicit error messages.
- **List structure of pre- and post-conditions**: The assumption that action semantics are intersections of logical conditions joined by `and` limits the use of other logical operators like `or` and `when`.

To address these limitations, we introduce Predicting Semantics of Actions with Language Models for Vision and Robotics (PSALM-V), the first neuro-symbolic framework that tackles key challenges in automated planning through flexible semantic updates and dynamic trajectory sampling. Starting with a problem file initialized from visual input, PSALM-V explores partially observed environments, constructing semantic maps to aid problem file generation. The method iteratively samples and executes trajectories, predicts error messages, and updates action semantics to refine memory. It maintains a tree-structured belief over possible action semantics for each action, refining these beliefs until a valid plan is executed to achieve the goal state. A PDDL domain file, a symbolic representation of the rules of the world, is sampled based on the current memory of hypothesized action semantics, and a symbolic planner searches for a path to the goal state given those rules.

We evaluate PSALM-V in three visual simulation environments and in a physical manipulation task with a Franka Panda arm. In simulation, we explore ALFRED [4], RTFM [5], and Overcooked-AI [6]. In ALFRED, PSALM-V increases the planning success rate from 37% (Claude-3.7) to 74% under conditions of partial observability. In RTFM and Overcooked-AI, PSALM-V improves step efficiency and successfully induces action semantics in multi-agent settings. Real-world BlocksWorld robot experiment results show that PSALM-V successfully recovers the full domain (F1 of 100%) with

*Accepted by ICRA 2026.

an average of 9.3 environment steps and achieves a goal-conditioned completion rate of 66.7%.

II. Background and Related Works

We first introduce symbolic planning, and then compare PSALM-V's domain induction setup with previous LLM-augmented planning approaches, enumerated in Table I.

A. Classical and Symbolic Planning

Classical planning algorithms have been extensively applied in domains such as autonomous spacecraft control, military logistics, manufacturing, games, and robotics. The automated STRIPS planner was the first algorithm to operate the Shakey robot [15]. Classical planners assume a finite, deterministic environment with full state observability, enabling them to generate guaranteed plans when a path from the initial to the goal state exists. Other planning frameworks, such as PRODIGY [16] and HTNs [17], have also proven effective in robotic planning contexts. Symbolic planning languages, such as the Planning Domain Description Language (PDDL; [1]) and Answer Set Programming (ASP; [18], [19]), offer structured and expressive ways to represent planning problems.

We use PDDL for symbolic planning. We define a planning problem P as a tuple $\langle \mathcal{D}, s^i, \mathcal{S}^g \rangle$, where $\mathcal{D}$ denotes the domain, s^i the initial state, and $\mathcal{S}^g$ the goal specification, *i.e.*, the set of goal states satisfying the defined goal conditions. A solution to P is a T-step action sequence $a_{1..T}$ such that executing $a_{1..T}$ from s^i leads to a state in $\mathcal{S}^g$.

The PDDL domain file defines the symbolic action set $\mathcal{A}$ available in the environment, along with predicates representing object properties. Each action is specified by a string identifier (*e.g.*, `Putdown`), a list of parameters (*e.g.*, `?ob`), and its semantics. The semantics Φ_a of an action $a \in \mathcal{A}$ include its preconditions (conditions under which the action can be executed) and postconditions (effects describing state changes resulting from the action).

B. Domain Induction

LLMs can generate plans directly from given initial and goal states [20], [21]. However, such stochastic, generative approaches sacrifice the success guarantees provided by symbolic planners. To improve plan correctness, several methods employ learned latent spaces [22] or LLMs [23]–[25] to generate executable programs, thereby introducing symbolic structure and constraints. These approaches can achieve up to 90% plan success in simple domains such as Grippers [26], but their performance degrades in more complex domains [10], even when combined with symbolic planner verifiers [27], [28].

Domain induction, which uses LLMs to predict domain files followed by symbolic solving to generate the final plan, has been shown to be effective in text-based domains [2]. This problem has been explored under various setups (Table I). PAM [11] introduced several formulations, including predicting preconditions and postconditions from valid plans or reusable plan fragments. Ada [3] and LASP [29] focused on completing missing preconditions and effects within a domain file, leveraging existing action semantics from the same or similar domains. Given a provided problem file, NL2PDDL [13] demonstrated single-round action semantics generation using LLMs, while PSALM [2] achieved domain file reconstruction with provided domain predicates in text-based domains by learning from environment feedback. Unlike LLM+P [7], which generates PDDL problem files from in-context examples, we infer action semantics directly in domain files in a zero-shot manner.

More recently, InterPreT [14] incorporated natural language feedback from humans during embodied interaction to infer domain predicates and action semantics. Their work addressed complex visual and robotics environments, enabling robots to follow human-specified goals such as "`stack red block on coaster`" with human feedback. NSPs [30] predict both predicates and action semantics with an online algorithm, but are limited to predicting one action at a time. In contrast, PSALM-V introduces a fully autonomous system for domain induction in visual and robotics settings, without relying on human evaluation or feedback.

III. Method

PSALM-V takes as input a natural language task description and a list of domain action signatures—a string name for the action and the type and number of arguments it takes—and predicts both the problem file and the action semantics, *i.e.*, the preconditions and postconditions of actions in the domain file. The full pipeline of PSALM-V is illustrated in Figure 2, comprising six main steps.

Step 0: Initialize the problem file using an LLM based on initial, visual observation(s).

Step 1: Given the inferred domain and problem files, sample a trajectory (*i.e.*, a plan) using an LLM. If a partial trajectory is available from the symbolic planner, use it to guide the LLM generation.

Step 2: Execute the sampled trajectory in a simulated or real environment.

Step 3: Correct the problem file based on the execution, or generate execution error message.

Step 4: Predict action semantics from the valid actions and observations and update the memory;

Step 5: Sample a domain file as the current belief and validate it by using the symbolic planner to search for a path from the inferred initial state to the specified goal state; if validation fails, or if the resulting plan fails during execution, return to **Step 1** with the updated domain file.

To improve efficiency and ensure that the execution trajectory is informative, for **Step 1** and **3**, we perform *prospection* for $k = 5$ steps on LLM output to ensure the first few actions are valid under the inferred domain and problem file. We use GPT-4o as the default LLM unless otherwise specified.

a) Problem file initialization.: A PDDL problem file comprises three core components: objects, initial state, and goal state. We generate these components in two stages due to their differing dependencies. Objects and the initial state depend solely on the environment configuration, while the

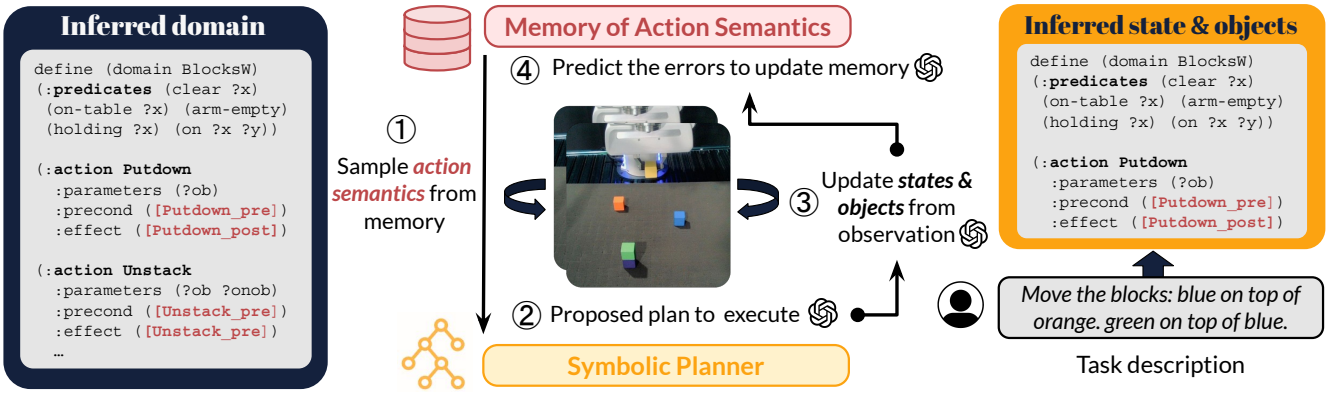

Fig. 1. LLMs can propose plans and generate action semantics, but struggle with state tracking. PDDL symbolic planners leverage specialized search algorithms, but require predefined action semantics for the environment. PSALM-V integrates the strengths of both for tasks on partially observed, visual environments.

TABLE I

PSALM-V DOMAIN INDUCTION TASK SETUP *vs.* REPRESENTATIVE RELATED WORKS IN LLM AND LLM-MODULO PLANNING; HERE, "AS" IS SHORT FOR ACTION SEMANTICS, "PF" IS SHORT FOR PROBLEM FILE, "FULL STATE" MEANS FULL STATE INFO IS GIVEN.

	Objective	**Independent of**					**Visual & Robot**
		Partial AS	Valid plans	Human eval	PF	Full state	
LLM+P [7]	PF	✗	✓	✓	✗	✗	✗
PDDLEGO [8]	PF	✓	✓	✓	✗	✓	✗
CLMASP [9]	Plan	✓	✓	✗	-	✗	✗
Generalized PDDL [10]	Program	✗	✓	✓	✓	✗	✗
PAM [11]	AS	✓	✗	✓	-	✗	✗
Ada [3]	AS	✗	✓	✓	✓	✗	✗
LLM-DM [12]	AS	✓	✓	✗	✓	✗	✗
NL2PDDL [13]	AS	✓	✓	✓	✗	✗	✗
PSALM [2]	AS	✓	✓	✓	✗	✗	✗
InterPreT [14]	AS	✓	✓	✗	✓	✗	✓
PSALM-V	AS	✓	✓	✓	✓	✓	✓

goal state depends on both the environment and the specified language task.

First, we initialize the objects and initial state based on the environment's observability. For fully observed environments, these are generated directly from the initial observation (o_0) using a single prompt. For partially observed environments, such as ALFRED [4] (illustrated in Figure 3), we first perform exploration to build a semantic map using FILM [31]. Subsequently, a Vision-Language Model (VLM) processes this map to generate the PDDL objects, the initial state predicates, and a location map. This location map provides a mapping from abstract location identifiers used in PDDL back to specific coordinates within the interactive visual environment.

Then, we generate the goal state. An LLM determines the goal conditions (*e.g.*, one of the goal conditions: `(exists (?o - object ?a - agent) (and (objectType ?o BookType) (holds ?a ?o)))`) based on the language instruction (*e.g.*, *Take the math book to the night stand*), the previously inferred objects, and the initial state. Once generated, this goal state remains fixed. Here `?o` and `?a` are arguments required by the To enhance the fidelity of goal generation, especially when a training dataset is available, we employ retrieval-augmented generation (RAG). We use BERTScore [32] to retrieve the top-2 most similar language instructions from the training set as in-context examples during goal generation.

b) Problem file editing.: During the execution of a sampled trajectory, we monitor for action failures using environmental feedback, such as observing no change in state or detecting a game restart. When a failure occurs, say at time t, execution halts. The halt signifies that actions $a_{0...t-1}$ were successful, while action a_t failed. At this point, an LLM problem file checker analyzes the situation. It takes the current observation o_t, the attempted action sequence $a_{0...t}$, and the inferred PDDL domain and problem files as input to determine if the failure is due to inaccuracies in the problem file. If the checker identifies errors (*e.g.*, missing objects), it proposes corrections.

Because attempting to correct the domain induction based on failures caused by an erroneous problem file is unreliable, we incorporate the LLM's proposed revisions into the problem file and return to **Step 1** to re-initiate trajectory sampling with the corrected file.

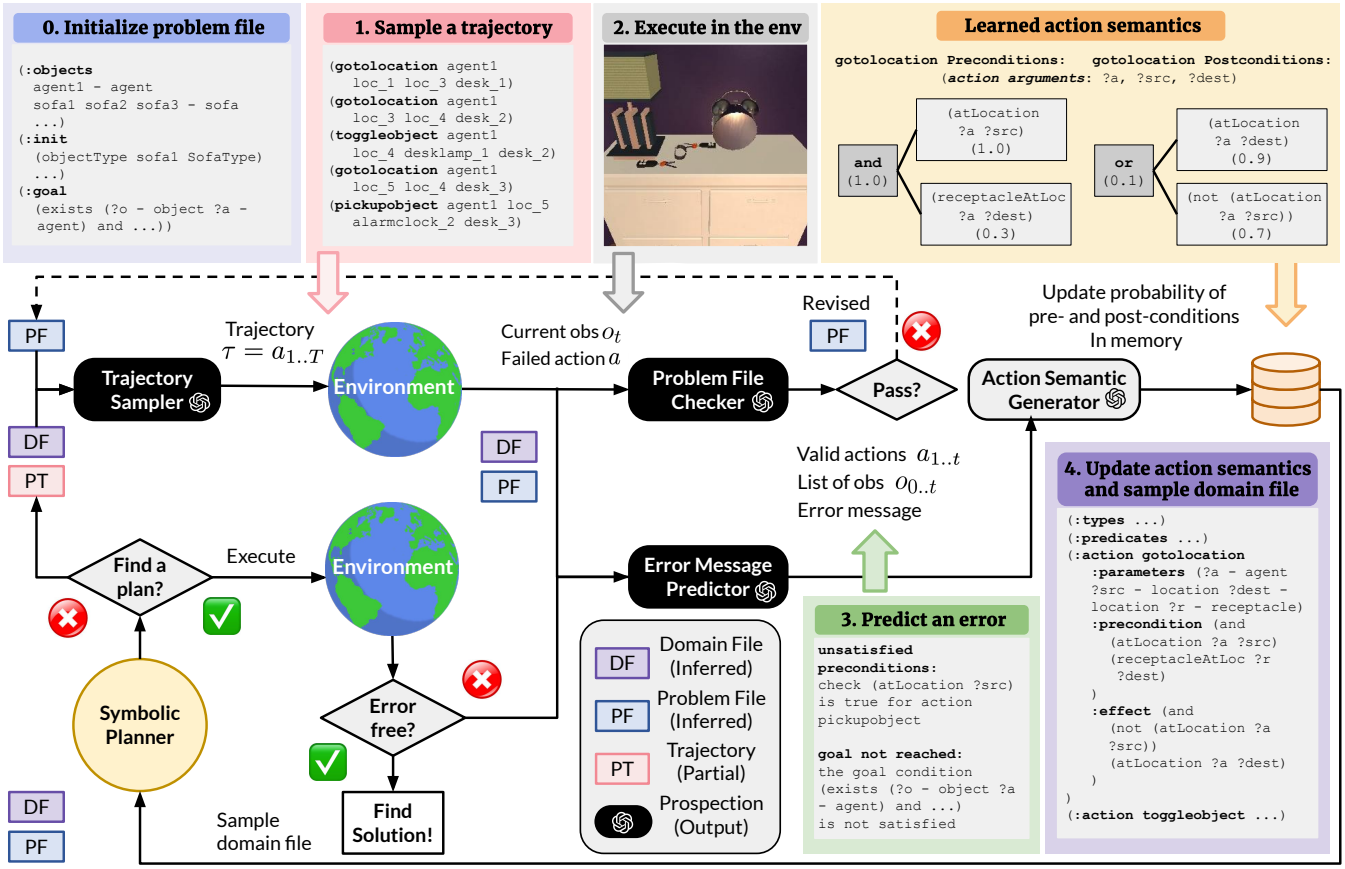

Fig. 2. Starting with a problem file initialized from visual input, PSALM-V iteratively samples and executes trajectories, predicts error messages, and updates action semantics to refine memory. We maintain a tree-structured belief over possible action semantics for each action, refining these beliefs until a valid plan is found by the symbolic planner that successfully reaches a goal state on execution.

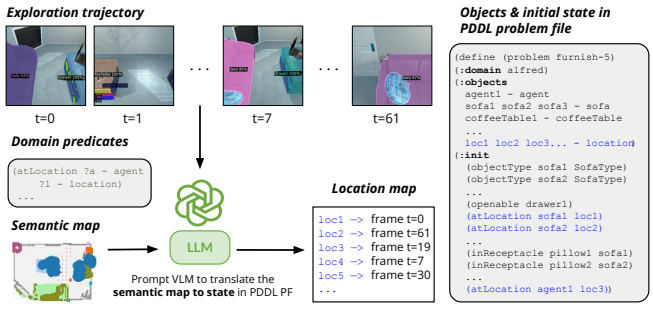

Fig. 3. Problem file initialization in the partially observed environment. We first explore the environment to build the semantic map, then use a VLM to translate it to a symbolic initial state.

c) Action semantics generation.: Error messages are important for inferring action semantics [2]. We first predict error messages for execution failures, and then use them to generate the action semantics.

An LLM is employed to predict two primary types of errors: 1) Unsatisfied preconditions, identified when an attempted action fails during execution; and 2) Unreached goals, identified when a sampled trajectory terminates without achieving the desired goal state. To provide informative feedback, particularly when explicit environmental error messages are sparse, the LLM also generates detailed explanations specifying which preconditions were violated or which goal conditions remained unmet.

The list of observations $o_{0...t}$, the sequence of attempted actions $a_{1...t}$, along with the predicted error message and its explanation, serve as input to a separate LLM-based action semantics generator. For cost efficiency, this generator produces the semantics (*e.g.*, pre- and post-conditions) for all domain actions in a single inference call. A key distinction from prior work [2], which restricted logical connectors to only `and`, is our support for multiple common connectors: `and`, `or`, and `when`. The LLM output is subsequently parsed to extract the formalized action semantics.

d) Tree-structured memory update.: We keep a memory of the hypothesized action semantics. For each action a, we store two trees: one for preconditions and one for postconditions. Each tree encodes predicted logical connector nodes and statement nodes. Each statement node ϕ is associated with a belief $p(\phi|a)$, representing the probability that ϕ belongs to the semantics of action a. This belief is used as a binary sampling probability during each iteration to construct concrete action semantics as input to the symbolic solver. We define the negation node $\neg\phi$ of a node ϕ as a sibling node with the opposite statement, *e.g.*, the negation

TABLE II

PSALM-V ACHIEVES STRONG DOMAIN INDUCTION (F1) AND OUTPERFORMS OTHER BASELINES ON ENVIRONMENT-DEFINED SUCCESS MEASURES. DIRECT LLM PLANNING HAS *NO* DOMAIN INDUCTION F1 SCORE.

Model	ALFRED			RTFM		Overcooked-AI	
	F1 (↑)	SR (↑)	GC (↑)	F1 (↑)	Win (↑)	F1 (↑)	Win (↑)
– Plan w/o PDDL							
GPT-4o	-	31	45	-	**100**	-	**100**
Claude-3.7	-	37	57	-	**100**	-	60
RoboGPT	-	60	71	-	-	-	-
– Plan after domain induction							
GPT-4o	71	0	0	85	30	68	0
Claude-3.7	76	0	0	88	34	75	0
PSALM-V	**91**	**74**	**75**	**100**	**100**	**100**	**100**

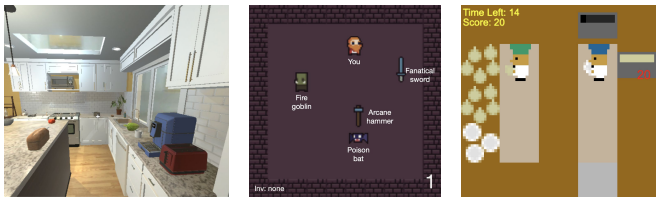

Fig. 4. Simulated envs: (left) **ALFRED** [4], (middle) **RTFM** [5], (right) **Overcooked-AI** [6]. ALFRED is under partially observed setup, while RTFM and Overcooked-AI are under fully observed setup.

of "`(not (arm-empty))`" is "`(arm-empty)`". When a node is predicted, the belief of its negation node is reduced.

When a node ϕ is first predicted as part of a's semantics, its belief is initialized to 1 (unless its negation is also predicted). Subsequent updates follow an exponential forgetting rule. Let $\hat{\Phi}_{a,t}$ denote the predicted semantics of a at time step t. The update rule for $p_{t+1}(\phi \mid a)$ is:

$$p_{t+1}(\phi|a) = \begin{cases} \min(\mathbb{1}[\phi \in \hat{\Phi}_{a,t+1}] - \alpha \mathbb{1}[\neg\phi \in \hat{\Phi}_{a,t+1}], 0) \\ \qquad\qquad \text{if } p_t(\phi|a) = 0, \\ \min(\beta p_t(\phi|a) + (1-\beta)\mathbb{1}[\phi \in \hat{\Phi}_{a,t+1}] - \\ \alpha(1-\beta)\mathbb{1}[\neg\phi \in \hat{\Phi}_{a,t+1}], 0) \qquad \text{otherwise.} \end{cases}$$

Here, $\alpha = 0.7$ is the contradiction penalty, and $\beta = 0.8$ is the forgetting factor. This update is applied to all nodes ϕ predicted in the current step or previously observed, i.e., $\phi \in \bigcup_{t' \in \{1 \ldots t+1\}} \hat{\Phi}_{a,t'}$.

e) Symbolic planner verification.: We use the FAST-DOWNWARD [33] planner to verify whether a valid plan can be generated from the inferred domain and problem files, with a search time limit of $W = 30$ seconds during each PSALM-V loop. If FAST-DOWNWARD fails to produce a complete plan but returns partial trajectories, these are also passed to the trajectory sampler as candidate trajectories.

IV. SIMULATION EXPERIMENTS

We evaluate PSALM-V across three simulated visual environments in Figure 4: (1) **ALFRED**: An interactive 3D simulation environment where agents must perform complex tasks based on human instructions. We conduct RAG on the training set of ALFRED for goal state generation, and evaluate on the unseen validation set. ALFRED has a total of 10 relevant actions in the domain file: texttttSliceObject, `ToggleObject`, `CoolObject`, `HeatObject`, `CleanObject`, `PutObject`, `PickupObject`, `CloseObject`, `OpenObject` and `GotoLocation`. The evaluation measure in ALFRED is the success rate (SR) and the goal-conditioned completion (GC). (2) **RTFM**: The RTFM environment offers a text-based 2D game setting requiring the agent to read textual instructions and act to defeat enemies. We use the "rock_paper_scissors" task, which has 4 actions `up`, `down`, `left` and `right`. To facilitate symbolic planning, we disable the movement of enemies and force the environment to be static. The evaluation measure in RTFM is the win rate (Win), which is originally the rate of winning rate over multiple trails. We incorporate the efficiency measure in it and redefine the win rate as the winning score over optimal winning score in 100 steps. (3) **Overcooked-AI**: Overcooked-AI simulates a multi-agent kitchen scenario where cooperation and real-time decision-making are essential. To enforce multi-agent evaluation of PSALM-V, we evaluate on the "Force Coordination" layout, with 6 actions `up`, `down`, `left`, `right`, `pick-up` and `put-down`. Overcooked-AI is evaluated on the win rate, similar as in RTFM.

A. Baselines and Evaluation Metrics

We compare PSALM-V to (1) *closed-loop plan generation* and (2) *single-round domain induction* on proprietary models GPT-4o [34] and Claude-3.7-Sonnet [35], and the current state-of-the-art method RoboGPT [36] on ALFRED. We also apply Predicting Semantics of Actions with Language Models for Vision and Robotics on Qwen2.5-72B [37] to show which components in the pipeline are generalizable to public models.

We evaluate the domain induction with three metrics. (1) **F1** score: For each predicted action semantics tree, we list paths from the root to each statement node. *E.g.*, `Putdown_pre-and-(arm-empty)` in the precondition tree of action `Putdown` from the root to the logical node `and` down to the leaf `(arm-empty)`. We compute the F1 score between the predicted and ground truth path lists, with high scores indicating close match. (2) Number of Resets (**NR**): how many times the environment returns to s_i because

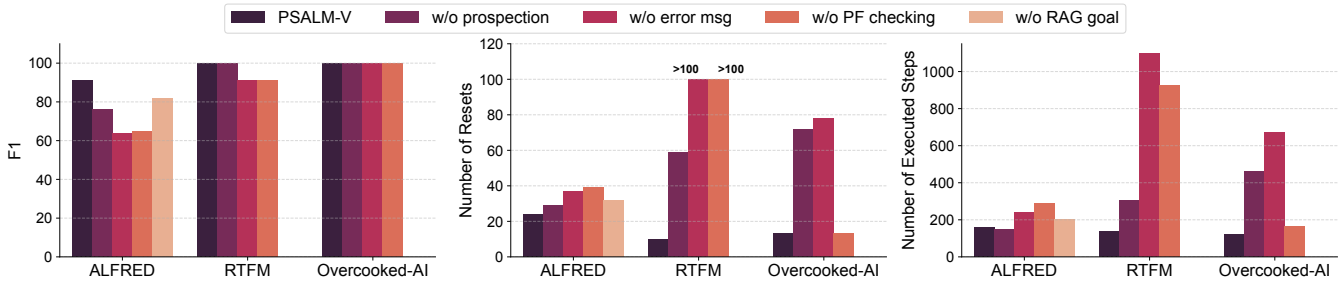

Fig. 5. Removing PSALM-V components such as prospection, error message prediction, and problem file checking increases reset steps and lowers domain induction accuracy (F1). Note that RAG of goal states is only applied on **ALFRED**.

TABLE III

REPLACING GPT-4O MODULES IN PSALM-V WITH A PUBLIC MODEL QWEN-2.5-72B: RESULTS SHOW PERFORMANCE DEGRADATION IN F1 AND STEP EFFICIENCY (NR, NES) UNDER PARTIAL REPLACEMENTS, ESPECIALLY THE PROBLEM FILE CHECKER. TS: TRAJECTORY SAMPLER; ASG: ACTION SEMANTICS GENERATOR; PFC: PROBLEM FILE CHECKER; EMP: ERROR MESSAGE PREDICTOR.

Public Module	**ALFRED**			**RTFM**			**Overcooked-AI**		
Qwen-72B	F1 (↑)	NR (↓)	NES (↓)	F1 (↑)	NR (↓)	NES (↓)	F1 (↑)	NR (↓)	NES (↓)
None (GPT-4o only)	**83**	**26**	**179**	**100**	**10**	**135**	**100**	**13**	**119**
TS	76	37	245	100	21	279	100	26	283
ASG	70	42	306	100	52	513	100	32	341
PFC	34	89	688	91	**>100**	832	100	55	589
EMP	61	41	299	100	35	320	100	29	303
TS & ASG	70	55	371	100	53	524	100	31	330
TS & ASG & EMP	59	87	576	87	**>100**	892	95	**>100**	723

planned actions did not reach $\mathcal{S}^g$. (3) Number of Executed Steps (**NES**): how many total actions are executed in learning.

B. Results and Analysis

PSALM-V successfully recovers the domain and outperforms other baselines on environment-defined success measures. All components of PSALM-V contribute to accuracy and step efficiency.

a) PSALM-V is robust on domain induction, while LLMs fail.: PSALM-V consistently recovers accurate action semantics and problem representations across partially observed and multi-agent environments (Table II). In contrast, LLMs like GPT-4o and Claude-3.7 never produce valid domain files. This finding highlights that one-shot domain inference by LLMs is brittle, whereas PSALM-V's iterative refinement and validation enable reliable planning from noisy inputs.

b) PSALM-V enables strong planning after domain induction.: Table II shows that LLMs alone struggle with partially observed planning tasks. PSALM-V outperforms the best public model, RoboGPT, on ALFRED by 14% in success rate (SR) and 4% in goal-conditioned completion (GC). Compared to Claude-3.7 and GPT-4o, PSALM-V more than doubles the SR on ALFRED and achieves consistently robust performance—maintaining a 100% win rate—on both RTFM and Overcooked-AI.

c) All components of PSALM-V contribute to accuracy and step efficiency.: Figure 5 shows that removing components of PSALM-V leads to reduced F1 scores and increased execution cost, as measured by Number of Resets (NR) and Number of Executed Steps (NES). For example, disabling problem file checking causes NES to spike from 158 to 287 (nearly 2x) in ALFRED and from 135 to 928 (nearly 7x) in RTFM, reflecting inefficient exploration due to undetected problem file errors.

d) Error messages and prospection are critical for learning from failure.: Without access to predicted error messages, the model must rely solely on environmental signals, resulting in degraded performance (F1 drops from 91 to 64 in ALFRED) and inefficient recovery (NES increases to over 1000 in RTFM). Without prospection, planning becomes less guided, requiring more corrections.

e) Retrieval-augmented goal inference boosts planning accuracy.: Ablating RAG-based goal construction reduces both F1 and step efficiency, indicating that retrieved examples help ground goal inference in ambiguous or underspecified tasks.

f) Public LLMs still face limitations in structured planning: From initial evaluation, we noticed that replacing GPT-4o with a public vision-language model for the whole PSALM-V system will break the system and result in a poor (<30) F1 score, even with careful parsing strategy to avoid syntactical errors. To systematically study the generalizability to public models, we replace the GPT-4o LLM modules in PSALM-V by a powerful public vision-language model, Qwen-2.5-72B [37]. We list the results in Table III. Note that for time efficiency we evaluate on 10% of the validation unseen data in ALFRED. The results show performance degradation in F1 and step efficiency (NR, NES) under partial replacements,

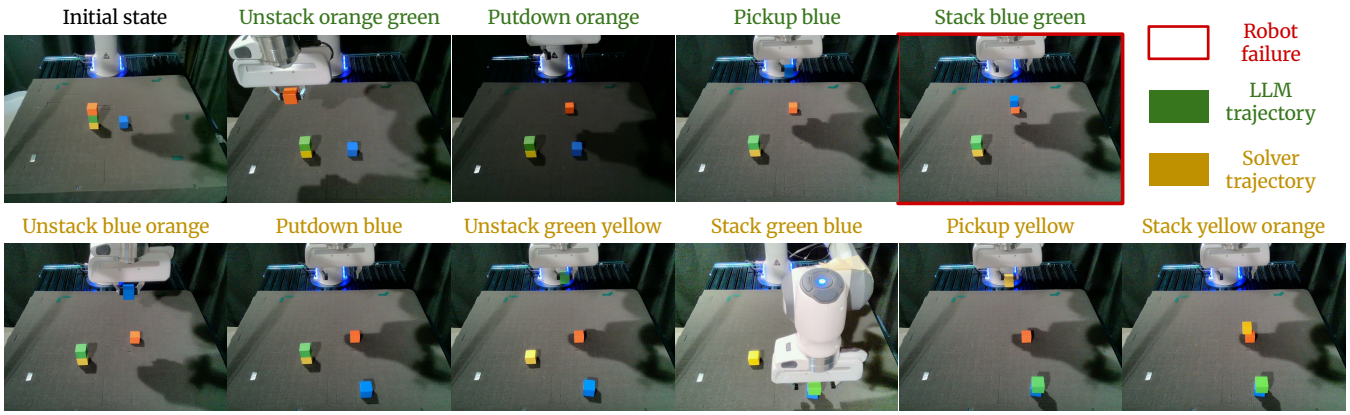

Fig. 6. PSALM-V remains robust even in the presence of low-level execution failures.

TABLE IV

PSALM-V CAN RECOVER BLOCKSWORLD ACTION SEMANTICS COMPLETELY WITH AN AVERAGE OF 9.3 STEPS, WHILE GPT-4O IS ONLY ABLE TO RECOVER IT WITH AN AVERAGE F1 OF 87.5%.

Model	Task 1: 4 Blocks			Task 2: 5 Blocks			Task 3: 5 Blocks			Overall		
	F1 (↑)	GC (↑)	NES (↓)	F1 (↑)	GC (↑)	NES (↓)	F1 (↑)	GC (↑)	NES (↓)			
GPT-4o - *Plan*	-	0	4	-	**50**	8	-	0	5	-	16.7	5.7
GPT-4o - *Domain*	91.8	-	-	93.9	-	-	76.9	-	-	87.5	-	-
PSALM-V	100.0	**100**	10	100.0	**50**	9	100	**50**	9	**100.0**	**66.7**	9.3

especially the problem file checker, implying that public LLMs still face limitations in structured planning tasks.

V. REAL ROBOT EXPERIMENTS

We conduct the 4-action BlocksWorld task [26], where the robot reorganizes a collection of block piles arranged on a table into a specified configuration. The robot has an arm that can hold one block at a time, operating four actions: `pickup`, `putdown`, `stack`, and `unstack`. We evaluate on 3 BlocksWorld tasks, each with at most 5 colored blocks (See Figure 6 for an example). Because the task environment is fully-observed and the problem file prediction is accurate, we change to a reset-free setup for the new problem file for the next iteration (*i.e.*, before calling trajectory sampler or symbolic planner) of PSALM-V, introducing an additional inference challenge.

We use pretrained skill policies [38] for each of the above actions. The policy predicts two keyframes for each action. The first keyframe predicts block grasp or release location based on the instruction, and the second keyframe moves the arm to neutral pose above the BlocksWorld environment for the next action execution. The keyframes are achieved via an IK motion planner from Deoxys real-time controller [39] for Franka Emika Panda arm. We use a single front-facing camera.[1]

Table IV shows that PSALM-V successfully recovers the full domain (F1 of 100%) with an average of 9.3 environment steps and achieves a goal-conditioned completion rate of 66.7%. In contrast, GPT-4o achieves only partial recovery, with an average F1 score of 87.5% and a goal-conditioned completion rate of 16.7%. Notably, PSALM-V remains robust even in the presence of low-level execution failures, such as mismanipulating the target object, as illustrated in Figure 6, where the robot mistakenly stacks the blue cube on the orange cube.

[1]See `https://psalmv.github.io/` for details on experiments.

VI. DISCUSSION

We presented PSALM-V, a neuro-symbolic framework for planning in partially observed, visually grounded environments without requiring predefined problem files or full observability. By combining LLM-based inference, trajectory execution, and symbolic validation, PSALM-V iteratively induces domain semantics and recovers valid PDDL representations. Our experiments demonstrate strong performance across simulated and real-world settings, showing substantial gains in planning success and semantic induction. These results suggest that integrating language models with symbolic reasoning offers a scalable path toward robust planning in complex environments. Future work includes scaling to longer-horizon tasks, improving generalization to unseen domains, and incorporating richer feedback modalities beyond language and observation.

REFERENCES

[1] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL - the planning domain definition language," *Technical Report, Tech. Rep.*, 1998.

[2] W. Zhu, I. Singh, R. Jia, and J. Thomason, "Language models can infer action semantics for symbolic planners from environment feedback," in *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 2024.

[3] L. S. Wong, J. Mao, P. Sharma, Z. S. Siegel, J. Feng, N. Korneev, J. B. Tenenbaum, and J. Andreas, "Learning adaptive planning representations with natural language guidance," in *The International Conference on Learning Representations (ICLR)*, 2023.

[4] M. Shridhar, J. Thomason, D. Gordon, Y. Bisk, W. Han, R. Mottaghi, L. Zettlemoyer, and D. Fox, "Alfred: A benchmark for interpreting grounded instructions for everyday tasks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10 740–10 749.

[5] V. Zhong, T. RocktÃd̈schel, and E. Grefenstette, "Rtfm: Generalising to novel environment dynamics via reading," *arXiv*, 2021. [Online]. Available: https://arxiv.org/abs/1910.08210

[6] M. Carroll, R. Shah, M. K. Ho, T. Griffiths, S. Seshia, P. Abbeel, and A. Dragan, "On the utility of learning about humans for human-ai coordination," *Advances in neural information processing systems*, vol. 32, 2019.

[7] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone, "LLM+P: Empowering large language models with optimal planning proficiency," *ArXiv preprint*, 2023.

[8] L. Zhang, P. Jansen, T. Zhang, P. Clark, C. Callison-Burch, and N. Tandon, "PDDLEGO: Iterative planning in textual environments," in *Proceedings of the 13th Joint Conference on Lexical and Computational Semantics (*SEM 2024)*, D. Bollegala and V. Shwartz, Eds. Mexico City, Mexico: Association for Computational Linguistics, June 2024, pp. 212–221. [Online]. Available: https://aclanthology.org/2024.starsem-1.17

[9] X. Lin, Y. Wu, H. Yang, Y. Zhang, Y. Zhang, and J. Ji, "CLMASP: Coupling large language models with answer set programming for robotic task planning," *ArXiv*, 2024.

[10] T. Silver, S. Dan, K. Srinivas, J. B. Tenenbaum, L. P. Kaelbling, and M. Katz, "Generalized planning in pddl domains with pretrained large language models," in *AAAI Conference on Artificial Intelligence*, 2023.

[11] A. Arora, H. Fiorino, D. Pellier, M. M. Â. Etivier, and S. Pesty, "A Review of Learning Planning Action Models," *Knowledge Engineering Review*, vol. 33, 2018.

[12] L. Guan, K. Valmeekam, S. Sreedharan, and S. Kambhampati, "Leveraging pre-trained large language models to construct and utilize world models for model-based task planning," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=zDbsSscmuj

[13] J. Oswald, K. Srinivas, H. Kokel, J. Lee, M. Katz, and S. Sohrabi, "Large language models as planning domain generators," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 34, 2024, pp. 423–431.

[14] M. Han, Y. Zhu, S.-C. Zhu, Y. N. Wu, and Y. Zhu, "Interpret: Interactive predicate learning from language feedback for generalizable task planning," in *Robotics: Science and Systems (RSS)*, 2024.

[15] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, 1971.

[16] J. Carbonell, O. Etzioni, Y. Gil, R. Joseph, C. Knoblock, S. Minton, and M. Veloso, "Prodigy: An integrated architecture for planning and learning," *SIGART Bull.*, 1991.

[17] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman, "Shop2: An htn planning system," *J. Artif. Intell. Res. (JAIR)*, 2003.

[18] G. Brewka, T. Eiter, and M. Truszczyński, "Answer set programming at a glance," *Commun. ACM*, 2011.

[19] V. Lifschitz, "Answer set programming and plan generation," *Artificial Intelligence*, 2002.

[20] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, Eds., 2022.

[21] B. Ichter, A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, D. Kalashnikov, S. Levine, Y. Lu, C. Parada, K. Rao, P. Sermanet, A. T. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, M. Yan, N. Brown, M. Ahn, O. Cortes, N. Sievers, C. Tan, S. Xu, D. Reyes, J. Rettinghouse, J. Quiambao, P. Pastor, L. Luu, K.-H. Lee, Y. Kuang, S. Jesmonth, K. Jeffrey, R. J. Ruano, J. Hsu, K. Gopalakrishnan, B. David, A. Zeng, and C. K. Fu, "Do as i can, not as i say: Grounding language in robotic affordances," in *6th Annual Conference on Robot Learning*, 2022.

[22] D. Trivedi, J. Zhang, S. Sun, and J. J. Lim, "Learning to synthesize programs as interpretable and generalizable policies," in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021.

[23] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "Progprompt: Generating situated robot task plans using large language models," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[24] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as policies: Language model programs for embodied control," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[25] Z. Hu, F. Lucchetti, C. Schlesinger, Y. Saxena, A. Freeman, S. Modak, A. Guha, and J. Biswas, "Deploying and evaluating llms to program service mobile robots," *IEEE Robotics and Automation Letters*, vol. 9, no. 3, pp. 2853–2860, 2024.

[26] J. Seipp, Á. Torralba, and J. Hoffmann, "PDDL generators," 2022.

[27] Z. Zhao, W. S. Lee, and D. Hsu, "Large language models as commonsense knowledge for large-scale task planning," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 31 967–31 987.

[28] R. Hazra, P. Z. Dos Martires, and L. De Raedt, "Saycanpay: Heuristic planning with large language models using learnable domain knowledge," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, 2024, pp. 20 123–20 133.

[29] G. Chen, L. Yang, R. Jia, Z. Hu, Y. Chen, W. Zhang, W. Wang, and J. Pan, "Language-augmented symbolic planner for open-world task planning," in *Robotics: Science and Systems Conference (RSS)*, 2024.

[30] Y. Liang, N. Kumar, H. Tang, A. Weller, J. B. Tenenbaum, T. Silver, J. F. Henriques, and K. Ellis, "Visualpredicator: Learning abstract world models with neuro-symbolic predicates for robot planning," in *The Thirteenth International Conference on Learning Representations*, 2025.

[31] S. Y. Min, D. S. Chaplot, P. Ravikumar, Y. Bisk, and R. Salakhutdinov, "Film: Following instructions in language with modular methods," *arXiv*, 2021.

[32] T. Zhang*, V. Kishore*, F. Wu*, K. Q. Weinberger, and Y. Artzi, "Bertscore: Evaluating text generation with bert," in *International Conference on Learning Representations*, 2020.

[33] M. Helmert, "The fast downward planning system," *J. Artif. Int. Res.*, vol. 26, no. 1, p. 191âĂŞ246, July 2006.

[34] OpenAI, "Introducing gpt-4o. https://openai.com/index/hello-gpt-4o/," 2024. [Online]. Available: https://openai.com/index/hello-gpt-4o/

[35] Anthropic, "Introducing claude 3.7 sonnet. https://www.anthropic.com/news/claude-3-7-sonnet," 2025. [Online]. Available: https://www.anthropic.com/news/claude-3-7-sonnet

[36] Y. Chen, W. Cui, Y. Chen, M. Tan, X. Zhang, D. Zhao, and H. Wang, "Robogpt: an intelligent agent of making embodied long-term decisions for daily instruction tasks," *arXiv preprint arXiv:2311.15649*, 2023.

[37] Q. Team, "Qwen2.5: A party of foundation models," September 2024. [Online]. Available: https://qwenlm.github.io/blog/qwen2.5/

[38] I. Singh, A. Goyal, S. Birchfield, D. Fox, A. Garg, and V. Blukis, "Og-vla: 3d-aware vision language action model via orthographic image generation," *arXiv preprint*, 2025.

[39] Y. Zhu, A. Joshi, P. Stone, and Y. Zhu, "Viola: Imitation learning for vision-based manipulation with object proposal priors," *arXiv preprint arXiv:2210.11339*, 2022.