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

A Robot BlocksWorld Tasks

A.1 Task description

We modified the BlocksWorld IPC tasks [1] and adapted the tasks to robot domain. Instead of using symbols `b1`, `b2` to represent blocks, we used colors, such as `green` and `yellow`, to represent blocks to avoid confusion in object detection. We selected three representative tasks, with the number of blocks to no more than 5, as we have only 5 different colored blocks. We describe the three robot tasks, their initial states, goal descriptions, and plans generated by GPT-4o.

Task 1: Figure 1 show the initial state, the goal description is *Your goal is to move the blocks. yellow should be on top of orange. green should be on top of blue.* The ground truth plan is

```
(unstack orange green)
(putdown orange)
(unstack green yellow)
(stack green blue)
(pickup yellow)
(stack yellow orange).
```

And the GPT-4o generated plan is

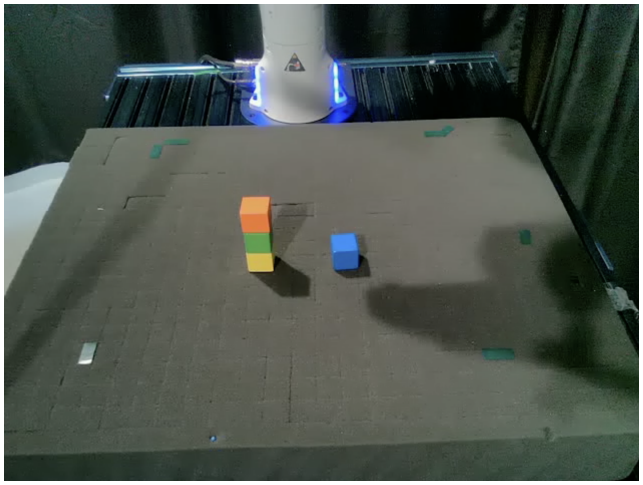

Figure 1: The initial state of robot BlockWorld task 1 on 4 blocks

```
(unstack orange green)
(putdown orange)
(unstack green yellow)
(stack green blue)
(pickup yellow)
(stack yellow orange).
```

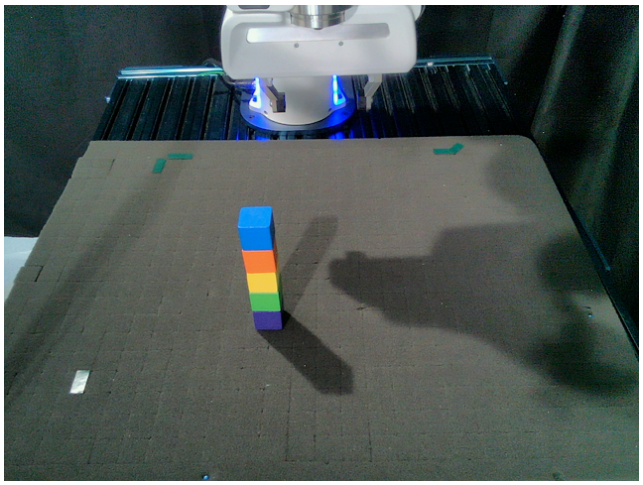

Figure 2: The initial state of robot BlockWorld task 2 on 5 blocks

Task 2: Figure 2 show the initial state, the goal description is *Your goal is to move the blocks. orange should be on top of green. green should be on top of purple.* The ground truth plan is

```
(unstack blue orange)
(putdown blue)
(unstack orange yellow)
(putdown orange)
(unstack yellow green)
(putdown yellow)
(pickup orange)
(stack orange green).
```

And the GPT-4o generated plan is

```
(unstack blue orange)
(putdown blue)
(unstack orange yellow)
(putdown orange)
(unstack yellow green)
(stack yellow blue)
(pickup orange)
(stack orange green).
```

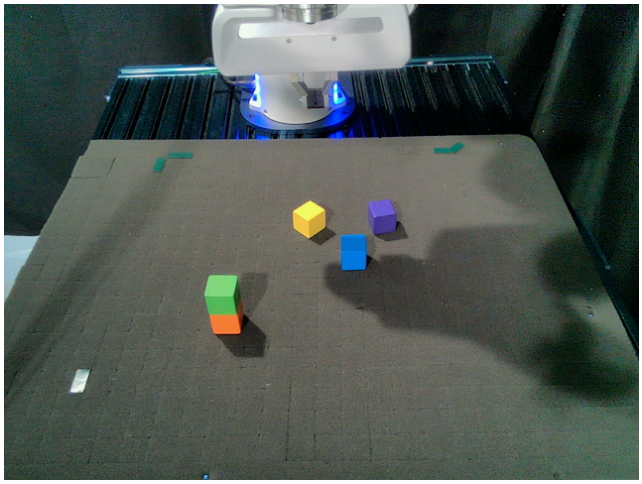

Figure 3: The initial state of robot BlockWorld task 3 on 5 blocks

Task 3: Figure 3 show the initial state, the goal description is *Your goal is to move the blocks. green should be on top of purple. blue should be on top of green.* The ground truth plan is

```
(unstack purple yellow)
(putdown purple)
(unstack yellow green)
(putdown yellow)
(unstack green orange)
(stack green purple)
(pickup blue)
(stack blue green).
```

And the GPT-4o generated plan is

```
(unstack purple yellow)
(putdown purple)
(unstack yellow green)
(putdown yellow)
(pickup green)
(stack green purple)
(pickup blue)
(stack blue green) .
```

Note that as we limit the number of blocks to 5, this task is not a hard planning task, GPT-4o can sometimes generate correct plans, but they cannot learn action semantics, and cannot recover from low-level failures.

Figure 4: Simulated envs: (left) ALFRED [4], (middle) RTFM [5], (right) Overcooked-AI [6].

A.2 Experiment details

We adapt the PSALM-V pipeline to robot tasks by adding a reset-free setup and a step-wise error detector.

Reset-free setup: Before calling trajectory sampler or symbolic planner, we use GPT-4o to read the current state from image and generate the problem file for the current state. As the modified BlocksWorld task has no more than 5 blocks, the problem file generation by GPT-4o is accurate from manual inspection.

Step-wise error detector: Robot experiment can fail in any action due to low-level manipulation error. Instead of using the LLM error predictor, for each step, we check the action description (*e.g.*, `(stack green blue)`), and the state transition from the initial state of the action and the end state of the action (*i.e.*, two images) to verify if the state transition matches the action description. If not, we stop the execution and perform action semantic generation.

B Robot Learning and Setup Details

Our robot policy [2] 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 [3] for Franka Emika Panda arm. We use a single front-facing camera. We omit the details on how the policy works.

C Simulation Environment Details

C.1 Simulation Environments

In Figure 4, we show the visualization of three environments.

ALFRED. The ALFRED benchmark is a large-scale dataset for evaluating embodied agents on instruction-following tasks in interactive environments. It consists of expert demonstrations paired with natural language directives, divided into distinct training, validation, and test sets. The dataset is carefully split to test generalization, with both seen and unseen splits in the validation and test sets.

The training set contains 21,023 annotated trajectories spanning 108 distinct scenes. The validation set includes 820 annotations over 88 scenes for the seen split and 821 annotations over 4 scenes for the unseen split. The test set similarly includes 1,533 seen examples from 107 scenes and 1,529 unseen examples from 8 scenes.

ALFRED has a total of 10 actions in the domain file (we remove the unnecessary actions in the domain file, such as `look` and `help`): `SliceObject`, `ToggleObject`, `CoolObject`, `HeatObject`,

`CleanObject`, `PutObject`, `PickupObject`, `CloseObject`, `OpenObject` and `GotoLocation`. As the testing data has no ground truth domain file to compare the F1 score, for a fair comparison with the previous works we use the average score on the validation unseen data for evaluation. The evaluation measure in ALFRED is the success rate (SR) and the goal-conditioned completion (GC).

RTFM. The RTFM benchmark is designed to test an agent's ability to jointly reason over a structured goal, a symbolic document describing environment-specific rules, and first-person environment observations. Each episode provides a unique setting with a natural language document that defines relationships among monsters, item types, and their interactions. The agent must parse this document, identify relevant entities in the environment, and select appropriate actions to achieve the goal.

We use the "rock_paper_scissors" task, which has 4 actions `up`, `down`, `left` and `right`. The original `stop` action is not meaningful as we force the environment to be static for our experiment. 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.

Overcooked-AI. : Overcooked-AI is a multi-agent planning and collaboration environment designed to evaluate cooperative human-AI task performance. Inspired by the popular video game *Overcooked*, the benchmark involves two agents working together in a shared kitchen environment to prepare and deliver soups. Each soup requires placing up to three ingredients into a pot, allowing it to cook, and then serving it. Agents must coordinate actions such as ingredient collection, pot monitoring, and delivery to maximize reward within a limited time frame.

To enforce multi-agent evaluation of PSALM-V, we evaluation 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.

C.2 Details on prospection

Even though the LLM is prompted with the current beliefs about action preconditions, it can still produce action sequences that violate those rules. To address this, we introduce trajectory prospection — a mechanism that allows the system to simulate ahead using its symbolic understanding of the world, without yet interacting with the actual environment.

Given a proposed trajectory $\tau = a_{1:T}$, we perform a symbolic check on the first k steps to ensure they are consistent with the current action semantics. If any action among $a_{1:k}$ violates its preconditions according to the current belief, we reject that action and repeatedly sample a new one until we find a valid alternative. This process continues until we obtain a prefix of k valid steps. If all of $a_{1:k}$ are valid from the start, we keep the original trajectory $a_{1:T}$ as-is. In practice, we set $k = 5$ to ensure that the trajectory begins with a reliable sequence of five valid actions before execution.

D Additional Experiments

D.1 Definition of evaluation measures

Let $\hat{\Phi}_a$ denotes the predicted action semantics of action a and Φ_a denotes the ground truth action semantics of action a. The F1 used in our experiment is defined as

$$
\begin{aligned}
&precision = \sum_{a \in \mathcal{A}} |\hat{\Phi}_a \cap \Phi_a| / \sum_{a \in \mathcal{A}} |\Phi_a| \\
&recall = \sum_{a \in \mathcal{A}} |\hat{\Phi}_a \cap \Phi_a| / \sum_{a \in \mathcal{A}} |\hat{\Phi}_a| \\
&\text{F1} = 2 \cdot precision \cdot recall / (precision + recall)
\end{aligned}
$$

Model	ALFRED			RTFM			Overcooked-AI		
	F1	NR	NES	F1	NR	NES	F1	NR	NES
PSALM-V	91	24	158	100	10	135	100	13	119
w/o prospection	76	29	151	100	59	306	100	72	463
w/o error message	64	37	241	91	>100	1097	100	78	672
w/o PF checking	65	39	287	91	>100	928	100	13	165
w/o RAG goal	82	32	205	-	-	-	-	-	-

Table 1: Removing components such as prospection, error message prediction, or problem file checking increases revision steps and lowers F1. RAG goal is only applied on ALFRED as it has a training set.

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

Table 2: 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.

D.2 Exact number of PSALM-V ablations

We show the exact number of PSALM-V ablations in Table 1.

Each PSALM-V component enhances accuracy and step efficiency. Table 1 illustrates that removing any component from PSALM-V results in lower F1 scores and higher execution costs, as indicated by the Number of Resets (NR) and Number of Executed Steps (NES). For instance, disabling problem file checking causes NES to surge from 158 to 287 in ALFRED and from 135 to 928 in RTFM, indicating inefficient exploration due to unresolved problem file issues.

Prospection and error feedback are essential for effective failure recovery. Eliminating predicted error messages forces the model to depend solely on environment feedback, leading to performance drops (F1 falls from 91 to 64 in ALFRED) and inefficient retries (NES exceeds 1000 in RTFM). Similarly, without prospection, the model lacks foresight in planning, resulting in more frequent corrections.

RAG-based goal inference improves both planning precision and efficiency. Removing retrieval-augmented goal construction leads to declines in both F1 and step efficiency, highlighting the value of retrieved examples in guiding goal inference, especially under ambiguity or limited task specification.

D.3 Extension to public models

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 [7]. We list the results in Table 2. 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, especially the problem file checker. We demonstrate that public LLMs still face limitations in structured planning tasks.

Trajectory sampling with Qwen-72B preserves planning accuracy, but increases execution cost. When replacing only the trajectory sampler (TS) with Qwen-72B, the F1 score remains high across all environments (100 in RTFM and Overcooked-AI, and 76 in ALFRED). However, NES increases significantly, for example, from 179 to 245 in ALFRED and from 135 to 279 in RTFM, indicating that the sampled plans are less step-efficient, possibly due to reduced commonsense priors in trajectory generation. These results suggest that while Qwen-72B can propose reasonable action sequences, it lacks GPT-4o's ability to optimize for minimal correction and retry cycles.

Replacing the action semantics generator leads to semantic misalignment. Substituting the action semantics generator (ASG) with Qwen-72B causes performance drops in both accuracy and step efficiency. The F1 score decreases to 70 in ALFRED and 100 to 92 in Overcooked-AI. NES also increases substantially (*e.g.*, 306 in ALFRED and 513 in RTFM), indicating that the inferred semantics become noisier or less aligned with the environment's execution logic. This results in higher failure rates and more corrective actions.

Problem file checking (PFC) is essential for reliable exploration. The most significant degradation occurs when replacing the problem file checker (PFC). In ALFRED, F1 drops to 34 and NES surges to 688, while in RTFM and Overcooked-AI, NES exceeds 800 and 580 respectively. More importantly, Qwen-72B frequently makes unnecessary edits to the problem file when the problem file is correct, which induces additional complexity to the domain induction process. The phenomenon occurs less in less complex and smaller environment, such as Overcooked-AI (5x5 grid). This results demonstrate the critical role of robust problem validation in reducing resets and guiding the planner away from invalid trajectories.

Error message prediction supports efficient learning from failures. Replacing the error message predictor (EMP) also negatively affects performance. While the F1 scores remain relatively moderate (e.g., 61 in ALFRED and 100 in RTFM), the number of executed steps increases significantlyâĂŤNES rises to 299 in ALFRED and over 300 in RTFM. This implies that without accurate failure feedback, the system must rely solely on environment-level signals, which are often sparse or delayed, resulting in inefficient recovery and increased exploratory retries.

Combined replacement of multiple modules compounds errors. When replacing multiple GPT-4o modules with Qwen-72B simultaneously, the degradation becomes more severe. For instance, replacing TS & ASG results in an F1 of 70 in ALFRED but increases NES to 371; further adding EMP causes F1 to drop to 59 and NES to spike to 576. In RTFM and Overcooked-AI, NES consistently exceeds 700 with F1 dropping below 90. These compounding effects confirm that while Qwen-72B can fulfill individual roles moderately well, its performance suffers dramatically when expected to coordinate multiple interdependent reasoning tasks.

E Prompt Templates

Here, we list the prompt templates for PDDL problem file generation (Figure 5), PDDL goal generation (Figure 6) trajectory sampling (Figure 7), action semantics generator (Figure 8), problem file checker (Figure 9).

F Limitation

While PSALM-V offers a significant step toward automated planning in visual domains, several limitations remain:

Problem File Generation

[[domain_nl]]

The environment is defined with the Planning Domain Definition Language (PDDL) as below, while the precondtions and postconditions of actions are unknown.

[[domain_acthead_pddl]]

An example planning problem is:

[[image]]

[[context_nl]]

The PDDL problem file for the example problem is:

[[context_pl]]

Now I have a new planning problem and its description. Can you provide the PDDL problem file for this problem?

The new planning problem is:

[[image]]

[[task_nl]]

Please only provide the PDDL problem file in the format "

(define (problem ...)

(:domain blocksworld-4ops)

(:objects ...)

(:init ...)

(:goal ...)

)

"

Please output a clean, complete list of objects and initial states.

Do NOT use the word repeat in the response.

Do NOT output any extra reasoning steps or additional information.

The PDDL problem file for the new planning problem is:

Figure 5: Prompt for PDDL problem file generation.

Dependency on domain specification. PSALM-V requires a set of predefined action names, object types, and predicates to induce action semantics. This partial dependency on prior domain knowledge limits full generalization to novel domains where such abstractions are unavailable. Future research could explore using LLMs or pretrained vision-language models to automatically extract candidate predicates and type hierarchies from demonstration data or affordance priors.

Instability on public LLMs. The system's performance depends on accurate error explanation and problem file generation, which can be unstable across less powerful public VLMs (*e.g.*, Qwen-2.5-

Problem File - Goal Generation

[[domain_nl]]

The environment is defined with the Planning Domain Definition Language (PDDL) as below, while the preconditions and postconditions of actions are unknown.

[[domain_acthead_pddl]]

{

An example planning problem is:

[[image]]

[[context_nl]]

The initial state of the task in PDDL is:

[[task_init]]

The goal state of the task in PDDL is:

[[task_goal]]

} * [[RAG_count]]

We have a new planning problem:

[[image]]

[[task_nl]]

The initial state of the new task in PDDL is:

[[task_init]]

Can you provide the goal in PDDL for problem?

Please only provide the PDDL problem file in the format (:goal ...)

Please output a clean, complete list of goal conditions..

Do NOT use the word repeat in the response.

Do NOT output any extra reasoning steps or additional information.

The PDDL goal for the new planning problem is:

Figure 6: Prompt for PDDL goal generation.

72B). These models often hallucinate symbolic elements or fail to capture nuanced state transitions, especially under ambiguous feedback. Future studies could incorporate more systematic verification mechanisms, such as constrained decoding checkers or multi-run tests, to filter or revise outputs from the LLMs, improving reliability across model variants.

Limitations of PDDL formalism. PDDL, while expressive for discrete planning, has known limitations in modeling continuous or dynamic environments, such as those involving time, probability, or numeric reasoning. In such settings, symbolic representations may fail to capture fine-grained dynamics (e.g., grasp force, trajectory optimization). A promising direction is to hybridize PDDL

Trajectory Sampling

[[domain_nl]]

The environment is defined with the Planning Domain Definition Language (PDDL) as

[[domain_type_pred]]

Here are the actions you can perform in the environment:

[[action_description]]

An example planning problem is:

[[image]]

[[context_nl]]

which means the initial state is:

[[context_init]]

A plan for the example problem is:

[[context_sol]]

Now I have a new planning problem and its description. Can you provide a executable plan, in the way of a sequence of behaviors, to solve the problem?

The new planning problem is:

[[image]]

[[task_nl]]

which means the initial state is:

[[task_init]]

[[failed_traj]]Please provide the list of actions in the format as described in the example.

Feel free to use try other actions not in the example when their preconditions are satisfied.

Please output a clean, complete list of actions, including the option and the starting actions you choose, to solve the problem.

Don't use the word repeat in the response.

A plan for the new planning problem is:

Figure 7: Prompt for trajectory sampling.

with continuous control modules or extend the symbolic formalism to support numeric fluents and temporal constraints, enabling planning in richer real-world domains.

Action Semantics Generator

[[domain_nl]]

The current guess of the preconditions and effects for actions are:

[[action_description]]

Please infer missing or correct wrong [[postfix]] for action [[action]] in CNF, given the PDDL types and predicates:

[[domain_type_pred]]

To help you predict the [[postfix]], we provide a running example in this domain. For the problem:

[[image]]

[[task_nl]]

The initial state of the task in PDDL is:

[[task_init]]

A plan (list of actions) runs in the environment like:

[[trajectory]]

Please focus on the action semantics of the plan, ignoring failures caused by robot instability, and output the CNF in a clean format of "(and (...) (...))", where each parentheses contains some predicates.

Note that you should not specify types, e.g., "- object", in preconditions and effects.

The [[postfix]] for [[action]] are:

Figure 8: Prompt for action semantics generator.

References

[1] P. Smirnov, F. Joublin, A. Ceravola, and M. Gienger. Generating consistent pddl domains with large language models. *ArXiv preprint*, 2024.

[2] 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.

[3] 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. doi:10.48550/arXiv.2210.11339.

[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*, pages 10740–10749, 2020.

[5] V. Zhong, T. RocktÃd'schel, and E. Grefenstette. Rtfm: Generalising to novel environment dynamics via reading. *arXiv*, 2021. URL https://arxiv.org/abs/1910.08210.

Problem File Checker

[[domain_nl]]

Please check if the problem file is correct, given the PDDL types and predicates below:

[[domain_type_pred]]

The current guess of the preconditions and effects for actions are:

[[action_description]]

The current belief of the problem file (task objects and initial state) is:

[[task_init]]

We provide a running example in this domain. For the problem:

[[image]]

[[task_nl]]

A plan (list of actions) runs in the environment like:

[[trajectory]]

Output format (in JSON): {

"Correctness": "yes" or "no",

"Edits": [":init from (arm-empty) to (not (arm-empty))", ":objects add (desk1 - object)"]

}

If the problem file is correct, output "yes" in the Correctness field, otherwise output "no" and provide edits. The edits can be on initial states or objects with three types of changes, modification ("from ... to ..."), add ("add ...") or delete ("delete ...")

Please check if the problem file is correct, output (in JSON):

Figure 9: Prompt for problem file checker.

[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*, 32, 2019.

[7] Q. Team. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.