LEARNING TO PLAN WITH NATURAL-LANGUAGE WORLD MODELS FROM HUMAN EGOCENTRIC VIDEOS

Anonymous authors
Paper under double-blind review

ABSTRACT

Human demonstrations provide examples of how tasks are performed, but embodied agents need reusable general knowledge of when actions apply, how they change the world, and how they can be recomposed into new plans. We present Planning in Abstract Worlds (PAW), a framework that learns natural-language abstract world models from human egocentric videos and fine-tunes a vision-language model (VLM) to plan with them in a closed loop. PAW represents tasks with natural-language predicates and high-level actions with explicit preconditions, effects, and bimanual coordination types. Instead of treating the induced abstract world model as an exact transition system, it uses the model to reconstruct state-action supervision from unlabeled video and as structured context for a learned planner. The planner maintains a belief state, supports information gathering under partial observability, and replans from new visual observations. Across offline planning and online human assistance experiments in three everyday manipulation domains, PAW outperforms VLM planner baselines and symbolic planning with induced models, generalizes to unseen action compositions, and remains effective with incomplete models. Robot demonstrations further illustrate transfer from human videos to physical execution. Together, these results support learning reusable abstract world models from human videos and learning to plan with them.

1 INTRODUCTION

Consider a robot or an AI assistant asked to complete a multi-step everyday task in an unfamiliar kitchen. It should produce a plan that fits the current situation rather than perform a fixed sequence of steps. Doing so requires understanding *transitions*: when an action is applicable and how it changes the world. Abstract world models capture exactly this structure by representing a task-relevant abstract state together with high-level actions defined by their preconditions and effects (Konidaris et al., 2018; Silver et al., 2021; Liang et al., 2025).

Recent work has substantially broadened abstract world models, both in what they can represent and in how they can be learned. Abstract world models can now capture richer dynamics such as exogenous processes, stochastic effects, and timing (Silver et al., 2021; Liang et al., 2026), while foundation models can propose predicates and operators (Liu et al., 2024; Liang et al., 2025; Athalye et al., 2026; Wang et al., 2025), extend models continually across tasks (Byrnes et al., 2025), or revise them from failures and feedback (Han et al., 2024; Zhu et al., 2025). Yet these advances have largely been studied in relatively constrained domains, for example with only a small number of steps or a limited set of strategies per goal.

Like other world models (Gao et al., 2026; Assran et al., 2025), abstract world models should benefit from broad coverage of *transitions*. Human videos offer such coverage naturally: different initial states, multiple strategies for the same goal, information-gathering behavior, and manipulation using one or two hands. However, this setting exposes two limitations of existing abstract world model methods. First, video-induced models are unlikely to be exact, whereas symbolic search assumes exact preconditions and effects (Silver et al., 2021; Athalye et al., 2026; Ye et al., 2025). Second, existing representations do not capture the full range of transitions present in human demonstrations, including information gathering and bimanual coordination.

We present Planning in Abstract Worlds (PAW) to address both challenges. PAW first induces a natural-language abstract world model from egocentric human videos. The state is represented using natural-language predicates whose values may be true, false, or unknown, and the action library

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

contains high-level actions with explicit preconditions and effects, including information-gathering actions that resolve unknown predicates and bimanual actions with coordination constraints. The model is induced through a hierarchical pipeline that segments hand-object contacts, groups them into semantic actions, and generates and consolidates an action schema for each. Second, rather than treating the induced model as an exact symbolic specification, we learn a planner to use it. The model serves a dual role: its action descriptions provide the basis for reconstructing state-action supervision from the videos, and it conditions the planner during both training and deployment. Using this supervision, we fine-tune a vision-language model (VLM) to interleave action selection with successor-belief prediction, given a goal, the current belief state, and the abstract world model.

Experiments show that PAW learns effectively from egocentric human videos without action labels, outperforming both symbolic-planning approaches and VLM planners trained on the same videos without an explicit world model. Ablations show that the action preconditions and effects supplied by the abstract world models, belief-state modeling, and state-action supervision each contribute to performance, while controlled experiments show graceful degradation as the induced models become less complete. In online human assistance, PAW gathers information when relevant state is unknown, adapts its plan to one- or two-handed execution in qualitative examples, and completes tasks without any intervention about 50% more often than the strongest baseline. Together, these results show that explicit abstract world models provide useful knowledge for planning beyond demonstrated procedures without requiring the induced model to be exact.

2 Related Work

Learning abstract world models and planning representations. Abstract world models describe states and how actions change them. Early methods learn symbolic operators from robot demonstrations using predefined predicates, or learn predicates within a fixed grammar (Silver et al., 2021; 2023; Kumar et al., 2023; Silver et al., 2022; Li & Silver, 2023; Li et al., 2025; Liu et al., 2026). More recently, foundation models have been used to generate predicates (Liang et al., 2025; Athalye et al., 2026; Wang et al., 2025) and operators (Liu et al., 2024; Ye et al., 2025), in some cases from human video (Athalye et al., 2026). In almost all of these methods, the learned model is passed to a symbolic planner, which relies on the preconditions and effects being correct; errors in the learned model can therefore cause planning failures. Language-based and code-based world models have also been explored, either by searching over synthesized programs (Hao et al., 2023; Tang et al., 2024) or by predicting state changes in language without explicit action semantics (Chen et al., 2025). PAW keeps explicit preconditions and effects and trains a planner to use them, which allows it to work with incomplete abstract world models. It additionally represents unknown predicate values, information-gathering actions, and bimanual coordination.

Learning procedural knowledge from instructional videos. Instructional-video datasets have been used to learn task structure (Zhukov et al., 2019; Tang et al., 2019) and to predict action sequences from start and goal observations or language goals (Chang et al., 2020; Zhao et al., 2022; Patel et al., 2023; Islam et al., 2024; Zhang et al., 2026; Chen et al., 2025). Recent methods also model state changes (Niu et al., 2024; Nagasinghe et al., 2024) or generate alternative action sequences (Wang et al., 2023b; Shi et al., 2025b; 2026). However, most of these methods are evaluated offline, commonly on three- or four-step plans (Wang et al., 2023b; Chen et al., 2025). PAW learns abstract world models for closed-loop planning over up to 15 high-level actions per episode and evaluates them through online human assistance.

Learning robotic manipulation from human videos. Human videos have been used to learn visual representations and reward functions (Nair et al., 2022; Ma et al., 2023; Chen et al., 2021), affordances (Bahl et al., 2023; Heidinger et al., 2025), manipulation priors (Mandikal & Grauman, 2021; Singh et al., 2025), policies from human motion (Bahl et al., 2022; Shaw et al., 2022; Shi et al., 2025a; Wang et al., 2023a; Zhu et al., 2024; Papagiannis et al., 2025), and robot world models (Gao et al., 2026). These methods mainly support skill execution. PAW instead learns planning representations that describe when actions apply and how they change the world, and that are independent of low-level control, so the same planner can guide a person or select robot skills.

3 Problem Formulation

We consider goal-conditioned planning for everyday manipulation tasks under partial observability. The environment is a partially observable Markov decision process $\langle \mathcal{X}, \mathcal{U}, \mathcal{T}, \mathcal{O}, \Omega, g_\ell \rangle$ with raw

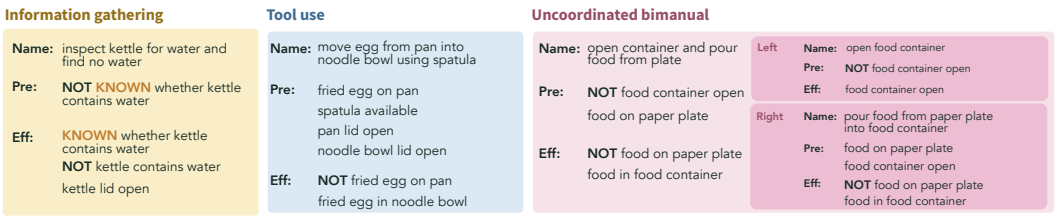


Figure 1: **Natural-language abstract world models.** Representative action definitions illustrate information gathering, tool use, and uncoordinated bimanual manipulation. Preconditions (Pre) specify applicability, and effects (Eff) describe changes in predicate values. Orange text highlights a predicate with uncertainty. Left and Right identify the left- and right-hand subactions, whose definitions require the container to be open before food is poured into it.

state space $\mathcal{X}$, low-level action space $\mathcal{U}$, an unknown transition function $\mathcal{T}$, observation space $\mathcal{O}$, an unknown observation model $\Omega : \mathcal{X} \to \mathcal{O}$, and a goal-satisfaction function $g_\ell : \mathcal{X} \to \{T, F\}$ induced by a natural-language goal ℓ. Following the options framework (Sutton et al., 1999), we plan over a space $\mathcal{A}$ of temporally extended high-level actions such as *fill kettle with water*, each realized as a sequence of low-level actions by an executor, a person or a robot. At each planning step the agent receives an image observation $o_t \in \mathcal{O}$ and selects a high-level action $a_{t+1} \in \mathcal{A}$.

As the agent observes the world only through images and acts one step at a time, what it must learn is a policy $\pi_\theta(a_{t+1} \mid h_t, \ell)$ that maps the history $h_t = (o_0, a_1, o_1, \ldots, a_t, o_t)$ and the goal to the next action. Since the state is not directly observed, we follow belief-space planning (Kaelbling et al., 1998; Kaelbling & Lozano-Pérez, 2013; Curtis et al., 2024) and let planning condition on the history through a belief b_t rather than on the latest observation o_t alone. The planner realizes π_θ by generating a plan at every step and executing its first action.

We learn from a dataset $\mathcal{D} = \{(\ell^i, o^i_{0:T_i})\}_{i=1}^N$ of egocentric human videos, each an observation sequence paired only with a language description ℓ^i of the goal it achieves. Neither the low-level actions in $\mathcal{U}$ nor the high-level actions in $\mathcal{A}$ are labeled, and $\mathcal{A}$ itself must be recovered from the videos. The objective is a policy π_θ that, given a possibly novel goal ℓ and the observation history of a new episode, selects high-level actions whose execution reaches a state satisfying g_ℓ.

4 Method

PAW learns a natural-language abstract world model from human egocentric videos and learns to plan with it. From the dataset $\mathcal{D}$ we learn the abstract world model and a planner that uses it, where the planner is trained on supervision that the abstract world model reconstructs from the videos. We discuss the representation in Section 4.1, how it is induced from video in Section 4.2, how it is used to construct supervision to train a planner in Section 4.3, and finally closed-loop planning in Section 4.4.

4.1 Natural-language abstract world models

An abstract world model in PAW is a triple $(\mathcal{P}, \mathcal{A}, \tau)$ consisting of a vocabulary of natural-language predicates $\mathcal{P}$, a library of high-level actions $\mathcal{A}$, and a transition model τ that specifies, for each action, its preconditions and effects over $\mathcal{P}$. Predicates and actions are natural-language statements that name the objects involved, such as the predicate *kettle contains water* and the action *fill kettle from faucet*. We use natural language rather than a fixed symbolic vocabulary so that predicates and actions induced from different videos can be matched by meaning, and so that an LLM planner can reason over them directly. Figure 1 shows example action definitions. Three representational choices, described next, capture transitions that everyday tasks require.

Three-valued predicates. Everyday tasks depend on state that cannot be observed directly, such as whether a closed kettle contains water. We therefore allow two kinds of predicates. A binary predicate takes a value in $\{T, F\}$. A three-valued predicate additionally takes the value U, denoting unknown, following belief-space planning under partial observability (Kaelbling et al., 1998; Bonet & Geffner, 2011). The planner's belief b_t is the vector of current values of all predicates in $\mathcal{P}$. Keeping U distinct from F lets the planner recognize that a goal depends on a value it has not observed and choose an action that resolves it.

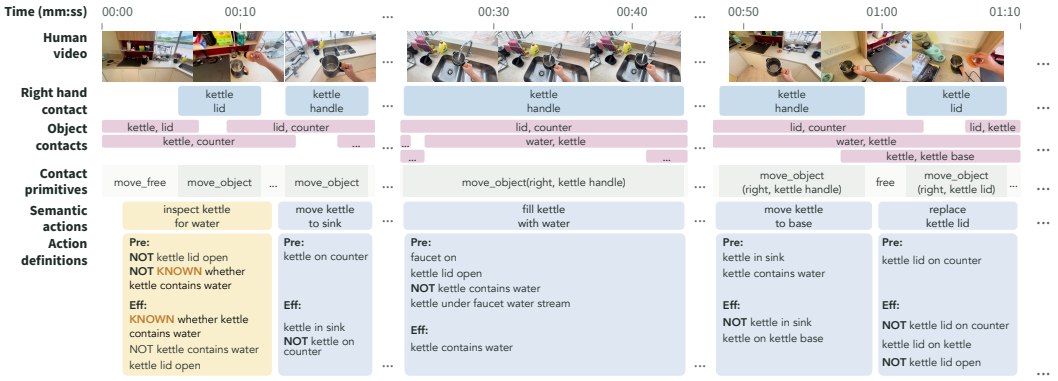


Figure 2: **Hierarchical induction of the abstract world model.** Low-level contact segmentation extracts contact primitives from a human demonstration video and its hand-object and object-object contact annotations. High-level semantic segmentation groups these primitives into semantic actions using video context. Action-definition generation uses each action and its neighboring actions to generate its name, preconditions, effects, and bimanual coordination constraints where applicable.

Information-gathering actions. We distinguish actions by their effect on the belief. A manipulation action rewrites the known value of a predicate, binary or three-valued, but cannot resolve U. An information-gathering action, such as opening a closed microwave, resolves a three-valued predicate from U to a known value. Since the outcome is not known in advance, each information-gathering action has two definitions, one whose effect resolves the predicate to T and one to F. During planning, the planner may proceed under one assumed outcome, but the outcome is determined by observing the environment after the action executes, for example by looking inside the kettle once its lid is open.

Bimanual actions. Following the taxonomy in bimanual manipulation (Krebs & Asfour, 2022), we design our abstract world models to represent unimanual actions, coordinated bimanual actions, such as holding a plate while scrubbing it, and uncoordinated bimanual actions that perform two unimanual actions in parallel. The first two have preconditions and effects and cannot be decomposed; an uncoordinated bimanual action has preconditions and effects for each hand. Our planner can compose two unimanual actions for parallel execution, or decompose an uncoordinated bimanual action into two unimanual actions when only one hand is available (e.g., a single-arm robot).

4.2 Inducing abstract world models from egocentric video

We induce the abstract world model $(\mathcal{P}, \mathcal{A}, \tau)$ from a dataset $\mathcal{D}$ of human egocentric videos, each paired with a language description of its goal. Compared with the robot demonstrations used in prior work (Silver et al., 2021; Liu et al., 2024; Ye et al., 2025), these videos capture a broader range of strategies and initial conditions, so the induced abstract world model covers a wider range of transitions.

As shown in Figure 2, we introduce an automated pipeline that constructs the abstract world model hierarchically. A video large language model first segments each video temporally into contact primitives, where a hand makes or breaks contact with an object, and semantic actions, which group consecutive primitives of one or both hands into a unit such as *fill kettle with water*. A large language model (LLM) then generates a definition for each semantic action given its neighboring actions as context. Each definition consists of an action name, preconditions, effects, and a coordination type; in this step the LLM also identifies information-gathering actions and, through them, which predicates are three-valued. Finally, an LLM consolidates the per-video definitions into one abstract world model by collapsing surface variants of predicates, grouping predicates with the same meaning, and merging actions whose preconditions and effects largely agree. The segmentation stage can additionally take manually annotated hand-object contact intervals for improved object detection and temporal segmentation. However, we show in our experiment that abstract world models generated without any annotations still yield strong planning performance.

Across all videos, the action names form $\mathcal{A}$, the predicates that appear in preconditions and effects form $\mathcal{P}$, and the preconditions and effects form τ. The resulting abstract world model is not a clean specification (e.g., predicates may still have duplicates and preconditions may omit dependencies). A

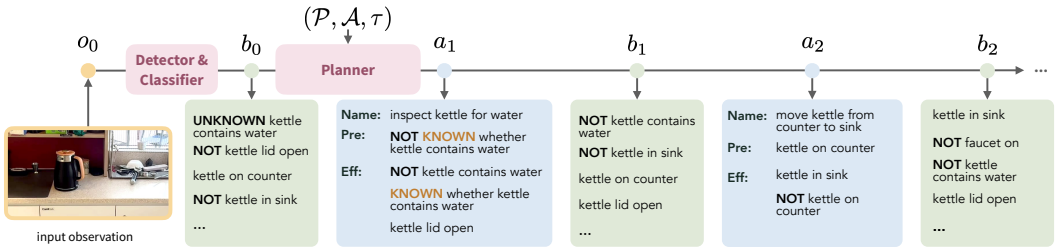


Figure 3: **Learning to plan with the abstract world model.** The same abstract world model guides reconstruction of belief–action supervision from human videos and conditions the planner during training and inference. A detector and predicate classifier ground the initial observation into b_0, from which the planner interleaves goal-conditioned action selection with successor-belief prediction.

symbolic planner that treats $(\mathcal{P}, \mathcal{A}, \tau)$ as exact propagates each of these errors into unreachable goals or invalid plans. PAW instead uses $(\mathcal{P}, \mathcal{A}, \tau)$ as an input to a learned planner, as we discuss next.

4.3 Learning to plan with the abstract world model

PAW uses a fine-tuned VLM to plan with the abstract world model. Symbolic planners treat generated preconditions and effects as exact transition rules, so errors in the model lead to planning failures; end-to-end VLM planners receive only the action vocabulary and must learn task structure implicitly. Our planner instead learns to apply preconditions and effects while tolerating their imperfections. The abstract world model serves as test-time structure for planning and as training supervision for the planner, reconstructed from the unlabeled videos (Figure 3).

Learned planner. The planner $\mathcal{M}_\theta$ is a VLM that, given a language goal ℓ, the current belief b_x, and the full model $(\mathcal{P}, \mathcal{A}, \tau)$, generates a plan as an interleaved sequence of actions and resulting beliefs $\sigma = (a_{x+1}, b_{x+1}, \ldots, a_y, b_y)$. It is fine-tuned by maximum likelihood on reconstructed plans described next, where p_θ is the distribution over plans defined by $\mathcal{M}_\theta$,

$$\theta^* = \arg\max_\theta \sum_{(\ell, b_x, \sigma) \in \tilde{\mathcal{D}}} \log p_\theta(\sigma \mid b_x, \ell, \mathcal{P}, \mathcal{A}, \tau). \tag{1}$$

Predicting each successor belief, rather than actions alone, forces the planner to apply the effects of each action and check the preconditions of the next.

Constructing supervision from videos. We generate training examples for the planner directly from the videos. Our hierarchical pipeline already turns each video into $(o_0, a_1, o_1, \ldots, a_K, o_K)$, which supplies the actions but not the beliefs the planner must predict. We recover the beliefs in two steps. First, we ground each frame o_k into an observed state s_k by prompting a pretrained vision-language model with the frame, the predicates in $\mathcal{P}$ over the objects detected in it, and the effects of the preceding action. Second, we compute a belief with memory, $b_k = \mathrm{Update}(b_{k-1}, s_k)$. Update merges the observed state into the previous belief. Every predicate in s_k takes its observed value, overriding b_{k-1} where the two conflict, and every predicate not in s_k keeps its value from b_{k-1}. The initial belief b_0 is s_0 with unobserved three-valued predicates set to unknown. This yields $(b_0, a_1, b_1, \ldots, a_K, b_K)$ for each video. We then generate training examples to form $\tilde{\mathcal{D}}$ by enumerating subsequences $(b_x, a_{x+1}, \ldots, a_y, b_y)$ and pairing each with a goal ℓ that a language model summarizes from its actions and final belief. Complete method details and worked examples are provided in Appendix A.3.

4.4 Closed-loop planning

At deployment the planner selects high-level actions from the goal, its belief, and new observations, realizing the goal-conditioned policy as $\pi_\theta(a_{t+1} \mid h_t, \ell) = \mathcal{M}_\theta(b_t, \ell, (\mathcal{P}, \mathcal{A}, \tau))_1$, the first action of the generated plan. At each step, an object detector finds the relevant objects in o_t and a VLM classifies the values of their corresponding predicates, giving the observed state s_t. The belief is updated as $b_t = \mathrm{Update}(b_{t-1}, s_t)$, and the planner generates a new plan from b_t. Replanning at every step lets an information-gathering action lead to observation of an initially unknown predicate and the plan adapt to it. In human-assistance deployment the selected action is shown to a person as a natural-language instruction, with classification running on the camera input; a robot executor implements the same action as a sequence of low-level actions in $\mathcal{U}$ via learned manipulation policies.



Figure 4: **Overview of three task domains.**

5 EXPERIMENTS

Our experiments answer four questions: 1) does PAW plan better than existing approaches, 2) which design decisions account for its performance, 3) how robust is it to noisy or incomplete abstract world models and to less training data, and 4) does it help human users and robots complete everyday tasks.

5.1 EXPERIMENTAL SETUP

Task domains. We design three task domains to evaluate planning from visual observations: *Boil Water*, *Make Instant Noodles*, and *Clean Up Kitchen* (Figure 4). Each domain supports multiple distinct strategies for accomplishing the task. For example, in *Boil Water*, water can be heated using a microwave, stove, or electric kettle, and can be obtained from a faucet, an already-filled kettle, or a water bottle. Each strategy in turn yields several action sequences, since the initial state of the environment determines which steps are needed. For each action sequence, we record multiple egocentric videos of a human completing the task while varying object layouts. *Boil Water* and *Make Instant Noodles* use unimanual actions; *Clean Up Kitchen* includes both unimanual and bimanual actions. Additional domain details and statistics are provided in Appendix A.7.

Evaluation setup. We split recorded videos into training and test sets under two generalization settings. *New Action Sequence* randomly partitions the unique action sequences between training and testing. For *New Strategy*, test tasks require combining strategies observed separately during training in novel ways. For example, a planner may need to heat water poured from a water bottle into a mug using a microwave, even if this particular combination was never observed during training. Abstract world models are only constructed based on the training videos. Each domain has a separate model.

We use offline evaluation to assess plan quality and online evaluation to test physical task execution. In offline evaluation, we compare predicted action sequences with ground-truth actions from recorded test videos, using *Boil Water* for all baseline comparisons, ablations, and controlled experiments. Following prior work (Park et al., 2025), we use an LLM-as-judge to rate plans from 1–5 on Presence (coverage of required steps), No Extra Steps (absence of unnecessary steps), and Ordering (correct relative ordering of required steps). In online evaluation, we compare PAW with the best-performing baseline from the offline comparison in all three domains, under both generalization settings. The planner guides a human user through held-out tasks, with the environment initialized according to test videos; the user follows the predicted actions and intervenes when an action is infeasible. We report the intervention rate (interventions divided by the length of the ground-truth action sequence) and, inspired by SPL (Anderson et al., 2018), success weighted by the optimal plan length. Offline evaluation include 360 test plans, and online evaluation include 40 trials per method.

Baselines. We compare against four classes of methods that span the dominant approaches to language-conditioned long-horizon planning. *VLM-as-Planner* uses a pre-trained VLM to directly perceive the visual state and output a complete action sequence (Hu et al., 2023; Zhi et al., 2025), and we instantiate this class with both a proprietary VLM and an open-weight VLM. *Finetuned VLM-as-Planner* additionally fine-tunes a VLM on in-domain data to predict high-level action sequences (Lin et al., 2025; Physical Intelligence et al., 2025; Chen et al., 2025). Finally, *LLM-as-Domain-Generator* uses an LLM to generate a planning domain definition that is then consumed by a symbolic task planner (Liu et al., 2024; Wong et al., 2024; Ye et al., 2025).

Implementation. We use Gemini-3.5 Flash for segmentation and GPT-5.6 Sol for abstract model construction and as the LLM judge, and fine-tune Qwen3-VL-8B with LoRA as the planner. Baselines use the same models and training budget. See Appendices A.1, A.5, and A.6 for details.

Table 1: **Baseline comparison.** PAW outperforms baselines in all settings.

Method	Model	*New Action Sequence*			*New Strategy*			Mean ↑
		Pres. ↑	No Extra ↑	Order. ↑	Pres. ↑	No Extra ↑	Order. ↑	
VLM-as-Planner	GPT-5.6 Sol	4.22 ± 0.91	4.37 ± 0.81	4.41 ± 0.79	4.25 ± 0.88	4.39 ± 0.78	4.36 ± 0.84	4.33
VLM-as-Planner	Qwen3-VL	3.48 ± 1.13	4.20 ± 0.90	4.16 ± 0.96	3.46 ± 1.16	4.21 ± 0.93	4.27 ± 0.97	3.96
Finetuned VLM-as-Planner	Qwen3-VL	<u>4.39</u> ± 0.83	4.48 ± 0.73	<u>4.43</u> ± 0.80	<u>4.32</u> ± 0.89	4.45 ± 0.79	<u>4.49</u> ± 0.78	<u>4.43</u>
LLM-as-Domain-Generator	GPT-5.6 Sol	2.55 ± 1.63	<u>4.64</u> ± 0.68	2.69 ± 1.79	2.63 ± 1.69	<u>4.60</u> ± 0.73	2.74 ± 1.83	3.31
PAW (Ours)	Qwen3-VL	**4.63** ± 0.56	**4.79** ± 0.47	**4.88** ± 0.37	**4.61** ± 0.59	**4.76** ± 0.50	**4.80** ± 0.44	**4.75**

Table 2: **Ablation study.** Removing individual components reduces planning performance.

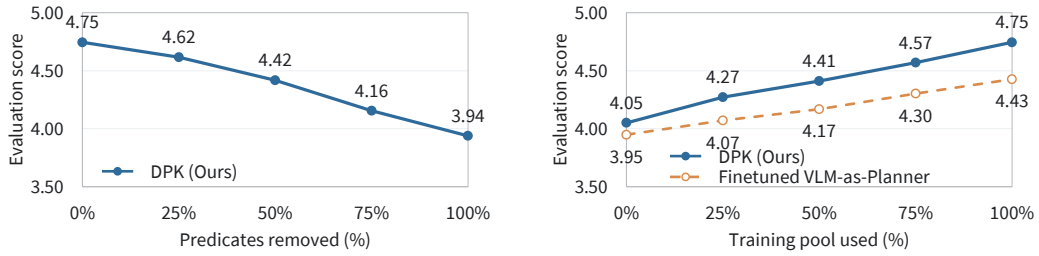
Method	*New Action Sequence*			*New Strategy*			Mean ↑
	Pres. ↑	No Extra ↑	Order. ↑	Pres. ↑	No Extra ↑	Order. ↑	
PAW (Ours)	**4.63** ± 0.56	**4.79** ± 0.47	**4.88** ± 0.37	**4.61** ± 0.59	**4.76** ± 0.50	**4.80** ± 0.44	**4.75**
w/o τ as input	3.84 ± 1.08	3.93 ± 0.96	4.11 ± 0.85	3.82 ± 1.12	3.88 ± 1.00	4.05 ± 0.92	3.94
w/o σ as prediction target	4.23 ± 0.88	4.18 ± 0.93	4.15 ± 1.03	4.19 ± 0.92	4.20 ± 0.90	4.08 ± 1.07	4.17
w/o belief update	4.18 ± 0.90	4.21 ± 0.86	4.30 ± 0.82	4.15 ± 0.94	4.17 ± 0.89	4.26 ± 0.86	4.21
w/o belief update & state classification	3.72 ± 1.05	4.08 ± 0.92	4.10 ± 0.94	3.69 ± 1.09	4.03 ± 0.96	4.14 ± 0.91	3.96
w/o finetuning on $\tilde{\mathcal{D}}$	3.64 ± 0.96	4.29 ± 0.77	4.21 ± 0.82	3.68 ± 0.98	4.23 ± 0.81	4.25 ± 0.85	4.05
w/o contact annotation in domain generation	4.49 ± 0.74	4.61 ± 0.63	4.67 ± 0.69	4.45 ± 0.79	4.58 ± 0.68	4.60 ± 0.73	4.57

5.2 Results

Task planning performance. As shown in Table 1, PAW outperforms all baselines under both generalization settings. Among training-free planners, the open-weight Qwen3-VL base model trails the proprietary GPT-5.6 Sol by a wide margin. Fine-tuning Qwen3-VL on the training videos closes most of that gap, and PAW improves further on the same backbone and training budget. LLM-as-Domain-Generator scores lowest on Presence and Ordering. It has the second-highest No Extra Steps score because the symbolic planner frequently returns an empty plan when the generated abstract world model is incomplete or inconsistent. This contrast supports our design of learning to plan with extracted abstract world models rather than treating their action semantics as exact symbolic transition rules. Finally, every method scores similarly under New Action Sequence and New Strategy. We attribute this to the fact that a new action sequence already requires composing actions in an unseen order, so both settings demand the same kind of compositional generalization.

Ablation. Table 2 shows that both the explicit abstract world model and the planner's learned state-transition prediction contribute to PAW's performance. Removing preconditions and effects from the input (w/o τ) causes the largest drop, from 4.75 to 3.94, and replacing the state-action target σ with action-only supervision lowers the mean to 4.17. Belief tracking also matters for planning in the real world. Removing the belief update lowers the mean to 4.21, and additionally removing state classification lowers it to 3.96. Finally, prompting the pretrained VLM with the same abstract world models but no fine-tuning reaches only 4.05, demonstrating the benefit of learning to use these models rather than prompting alone.

Planning with imperfect abstract world models. We test PAW's robustness to imperfect abstract world models in two settings. First, we remove the contact annotations from the generation pipeline. Without them, temporal segmentation sometimes oversegments an action into smaller steps or merges neighboring actions. As shown in Table 2, PAW still outperforms the strongest baseline without contact annotations. Second, we randomly remove varying fractions of predicates from action preconditions and effects to mimic omissions in generated abstract world models. For each removal level, the corresponding incomplete abstract world model is used during both planner training and evaluation. As shown in Figure 5a, PAW stays ahead of the strongest baseline through 25% removal



(a) Incomplete Abstract World Model **(b) Data Efficiency**

Figure 5: **Controlled experiments.** (a) Varying the fraction of predicates removed from action preconditions and effects. (b) Varying the amount of fine-tuning data. Curves show the unweighted mean across three metrics and two generalization settings; 0% in (b) denotes no fine-tuning.

Table 3: **Online human assistance.** Closed-loop comparison of PAW and Finetuned VLM-as-Planner, with a human executing the predicted actions.

Method	***Boil Water***		***Make Instant Noodles***		***Clean Up Kitchen***	
	SPL ↑	Int. ↓	SPL ↑	Int. ↓	SPL ↑	Int. ↓
New Action Sequence						
Finetuned VLM-as-Planner	0.861 ± 0.113	0.178 ± 0.147	0.873 ± 0.087	0.156 ± 0.124	0.886 ± 0.080	0.135 ± 0.097
PAW (Ours)	**0.976** ± 0.058	**0.028** ± 0.068	**0.948** ± 0.096	**0.065** ± 0.123	**0.967** ± 0.067	**0.038** ± 0.077
New Strategy						
Finetuned VLM-as-Planner	0.848 ± 0.102	0.194 ± 0.145	0.834 ± 0.049	0.203 ± 0.071	0.899 ± 0.053	0.116 ± 0.061
PAW (Ours)	**0.948** ± 0.057	**0.058** ± 0.064	**0.942** ± 0.084	**0.069** ± 0.102	**0.958** ± 0.066	**0.049** ± 0.076

and nearly matches it at 50%. Beyond 50%, performance drops more steeply. We hypothesize that extensive predicate removal obscures how actions change states and enable subsequent actions, making these dependencies harder for the planner to learn. These results show that PAW can learn to use incomplete abstract world models effectively, while more complete action semantics further improve planning.

Learning efficiency. We compare the sample efficiency of PAW with the strongest baseline and observe that PAW achieves similar performance to the fully trained baseline using only 50% of the planner fine-tuning data (Figure 5b). The trends further show that our method benefits more from additional data. Together with the state-action supervision ablation, these results support using abstract world models to construct supervision that improves the planner's data efficiency.

Online human assistance. Table 3 shows results from online evaluation, where the planner serves as an assistant. This evaluation differs from the offline evaluation in two ways. Methods are no longer scored against a ground-truth action sequence, of which there may be several for open-ended tasks, and the planner is allowed to re-observe the environment and replan after each step. The results show that our method's improvement over the baseline translates into fewer interventions and plans closer in length to the optimal plan. On average, our planner guides human users to complete the task without any intervention in 78% of episodes under New Action Sequence and 60% under New Strategy, compared with 24% and 11% for the baseline. These results show that learning to plan with video-derived abstract world models improves closed-loop task planning.

Qualitative analysis. Figure 6 illustrates qualitative improvements over the baseline. Episode (b) additionally tests a constraint absent from training, in which the planner must complete the task using a single hand. PAW completes all episodes without intervention, versus two interventions per episode for Finetuned VLM-as-Planner. In (a), PAW inspects the closed microwave to resolve uncertainty about its contents. In (b), PAW executes uncoordinated action pairs sequentially, although training demonstrations show only bimanual execution of these pairs. These examples support our main claim that abstract world models acquired from human videos enable planning beyond observed procedures. We show more examples in Appendix B.

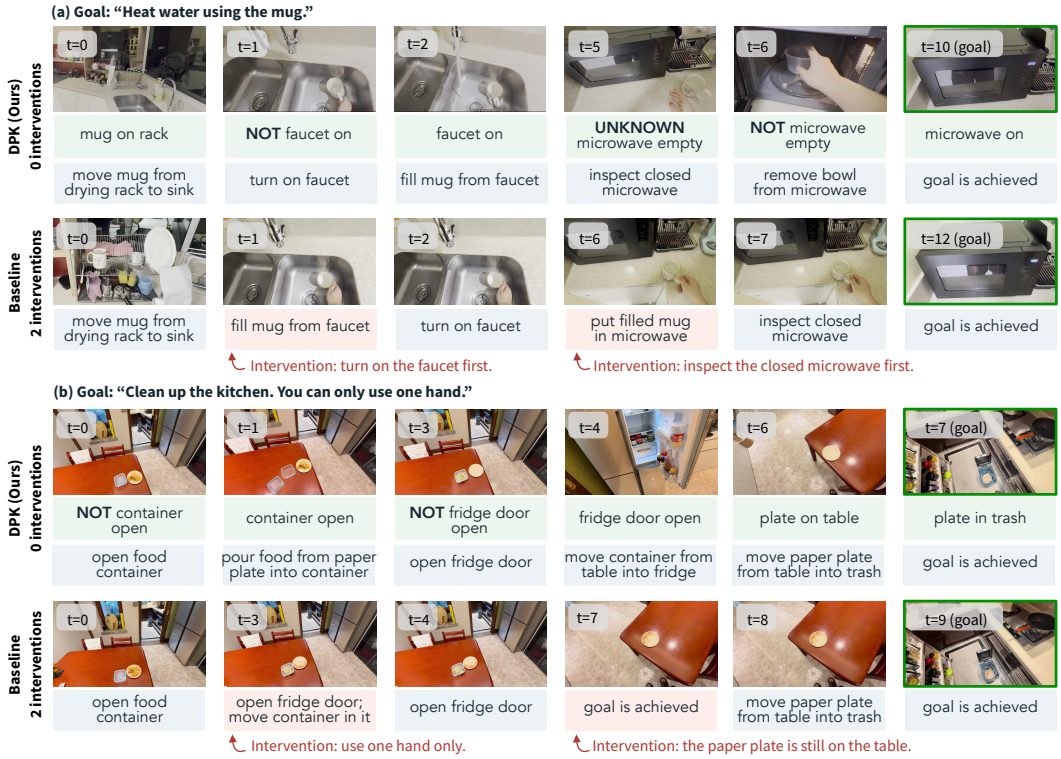


Figure 6: **Qualitative online human assistance.** Selected steps illustrate that PAW gathers information to resolve uncertainty (a) and decomposes uncoordinated bimanual actions for one-handed execution (b). PAW completes both episodes without human intervention, compared with multiple interventions for Finetuned VLM-as-Planner. Red text denotes interventions.

Transfer to a real robot. Finally, we test whether our planner, trained entirely on human egocentric videos, can guide a Franka robot through multi-step tasks. The planner observes the scene through the robot's camera and commands high-level actions, which are executed by low-level robot skills learned from teleoperated demonstrations, similar to (Nie et al., 2026). As shown in Figure 7, the planner guides the robot to complete the *Boil Water* task fully autonomously and adapts to the initial conditions, choosing between the electric kettle (a, c) and the pot (b) depending on which objects are available, and skipping the filling step when the kettle already contains water (a).



Figure 7: **Demonstration on a robot.** The planner, trained only on human egocentric videos, guides a robot through the *Boil Water* task under different initial conditions and with different strategies.

6 Conclusion

We presented PAW, which induces a natural-language abstract world model from egocentric human videos and fine-tunes a vision-language model to plan with it, using the abstract world model both to reconstruct training supervision and to provide test-time structure. This outperforms symbolic planning over generated domain models and VLM planners trained on the same videos. We see this as a paradigm for planning with large language models that keeps the structure of an explicit abstract world model without the brittleness of treating it as exact. A main limitation of this work is the reliance on other grounding models for object detection and state classification. Future work includes combining the structured belief with visual memory, predicting low-level actions alongside high-level ones, and inducing abstract world models from larger video corpora.

AI USAGE STATEMENT

In this work, we used generative AI tools, including Codex, Claude Code, ChatGPT, and Gemini, to assist with selected engineering prototypes and supporting code, and to provide feedback on research methodology, experiments, and our interpretation of results. Separately, pretrained language and vision-language models are components of the DPK pipeline for inducing abstract world models and constructing planner-training supervision from recorded human demonstrations. These methodological uses are described in Sections 4.2 and 4.3 and Appendices A.1 and A.3. We did not use AI assistants to develop theoretical models or conceptual frameworks, propose new research hypotheses, translate the manuscript, clean or reformat datasets, or conduct qualitative or thematic data analysis. Assistance with mathematical claims and proofs is not applicable to this work. Additionally, we used generative AI tools to assist with manuscript drafting and language polishing, literature retrieval and discovery, and visualization code. We have reviewed all AI-assisted work. The authors determined the research direction and high-level designs. AI-assisted coding was limited to selected prototypes, supporting engineering code, and visualization code; the remaining research code was written by the authors. We reviewed, iteratively revised, and tested AI-generated code, and repeatedly tested the resulting system through human-assistance and robot experiments. We checked AI-assisted writing and technical claims, verified suggested references against the original publications, and checked visualizations against the underlying data. We take responsibility for the final content of this work, including all AI-assisted text, claims, code, and artifacts.

REFERENCES

Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir R. Zamir. On evaluation of embodied navigation agents. *arXiv:1807.06757*, 2018. 6

Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Mojtaba Komeili, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, Sergio Arnaud, Abha Gejji, Ada Martin, Francois Robert Hogan, Daniel Dugas, Piotr Bojanowski, Vasil Khalidov, Patrick Labatut, Francisco Massa, Marc Szafraniec, Kapil Krishnakumar, Yong Li, Xiaodong Ma, Sarath Chandar, Franziska Meier, Yann LeCun, Michael Rabbat, and Nicolas Ballas. V-JEPA 2: Self-supervised video models enable understanding, prediction and planning. *arXiv:2506.09985*, 2025. 1

Ashay Athalye, Nishanth Kumar, Tom Silver, Yichao Liang, Jiuguang Wang, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. From pixels to predicates: Learning symbolic world models via pretrained VLMs. *IEEE Robotics and Automation Letters*, 11(4):4002–4009, 2026. doi: 10.1109/LRA.2026.3662533. 1, 2

Shikhar Bahl, Abhinav Gupta, and Deepak Pathak. Human-to-robot imitation in the wild. In *Robotics: Science and Systems (RSS)*, 2022. 2

Shikhar Bahl, Russell Mendonca, Lili Chen, Unnat Jain, and Deepak Pathak. Affordances from human videos as a versatile representation for robotics. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 2

Blai Bonet and Hector Geffner. Planning under partial observability by classical replanning: Theory and experiments. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2011. 3

Walker Byrnes, Miroslav Bogdanovic, Avi Balakirsky, Stephen Balakirsky, and Animesh Garg. CLIMB: Language-guided continual learning for task planning with iterative model building. In *International Conference on Robotics and Automation (ICRA)*, 2025. 1

Chien-Yi Chang, De-An Huang, Danfei Xu, Ehsan Adeli, Li Fei-Fei, and Juan Carlos Niebles. Procedure planning in instructional videos. In *European Conference on Computer Vision (ECCV)*, 2020. 2

Annie S. Chen, Suraj Nair, and Chelsea Finn. Learning generalizable robotic reward functions from "in-the-wild" human videos. In *Robotics: Science and Systems (RSS)*, 2021. 2

Delong Chen, Theo Moutakanni, Willy Chung, Yejin Bang, Ziwei Ji, Allen Bolourchi, and Pascale Fung. Planning with reasoning using vision language world model. *arXiv:2509.02722*, 2025. 2, 6

Aidan Curtis, George Matheos, Nishad Gothoskar, Vikash Mansinghka, Joshua B. 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. 3

Shenyuan Gao, William Liang, Kaiyuan Zheng, Ayaan Malik, Seonghyeon Ye, Sihyun Yu, Wei-Cheng Tseng, Yuzhu Dong, Kaichun Mo, Chen-Hsuan Lin, Qianli Ma, Seungjun Nah, Loic Magne, Jiannan Xiang, Yuqi Xie, Ruijie Zheng, Dantong Niu, You Liang Tan, K. R. Zentner, George Kurian, Suneel Indupuru, Pooya Jannaty, Jinwei Gu, Jun Zhang, Jitendra Malik, Pieter Abbeel, Ming-Yu Liu, Yuke Zhu, Joel Jang, and Linxi Fan. DreamDojo: A generalist robot world model from large-scale human videos. *arXiv:2602.06949*, 2026. 1, 2

Muzhi Han, Yifeng Zhu, Song-Chun Zhu, Ying Nian Wu, and Yuke Zhu. InterPreT: Interactive predicate learning from language feedback for generalizable task planning. In *Robotics: Science and Systems (RSS)*, 2024. 1

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023. 2

Marvin Heidinger, Snehal Jauhri, Vignesh Prasad, and Georgia Chalvatzaki. 2HandedAfforder: Learning precise actionable bimanual affordances from human videos. In *International Conference on Computer Vision (ICCV)*, 2025. 2

Yingdong Hu, Fanqi Lin, Tong Zhang, Li Yi, and Yang Gao. Look before you leap: Unveiling the power of GPT-4V in robotic vision-language planning. *arXiv:2311.17842*, 2023. 6

Md Mohaiminul Islam, Tushar Nagarajan, Huiyu Wang, Fu-Jen Chu, Kris Kitani, Gedas Bertasius, and Xitong Yang. Propose, assess, search: Harnessing LLMs for goal-oriented planning in instructional videos. In *European Conference on Computer Vision (ECCV)*, 2024. 2

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. 3

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998. 3

George Konidaris, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018. 1

Franziska Krebs and Tamim Asfour. A bimanual manipulation taxonomy. *IEEE Robotics and Automation Letters*, 7(4):11031–11038, 2022. 4

Nishanth Kumar, Willie McClinton, Rohan Chitnis, Tom Silver, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Learning efficient abstract planning models that choose what to predict. In *Conference on Robot Learning (CoRL)*, 2023. 2

Amber Li and Tom Silver. Embodied active learning of relational state abstractions for bilevel planning. In *Conference on Lifelong Learning Agents*, 2023. 2

Bowen Li, Tom Silver, Sebastian Scherer, and Alexander G. Gray. Bilevel learning for bilevel planning. In *Robotics: Science and Systems (RSS)*, 2025. 2

Yichao Liang, Nishanth Kumar, Hao Tang, Adrian Weller, Joshua B. Tenenbaum, Tom Silver, João F. Henriques, and Kevin Ellis. VisualPredicator: Learning abstract world models with neuro-symbolic predicates for robot planning. In *International Conference on Learning Representations (ICLR)*, 2025. 1, 2

Yichao Liang, Dat Nguyen, Cambridge Yang, Tianyang Li, Joshua B. Tenenbaum, Carl Edward Rasmussen, Adrian Weller, Zenna Tavares, Tom Silver, and Kevin Ellis. ExoPredicator: Learning abstract models of dynamic worlds for robot planning. In *International Conference on Learning Representations (ICLR)*, 2026. 1

Fanqi Lin, Ruiqian Nai, Yingdong Hu, Jiacheng You, Junming Zhao, and Yang Gao. OneTwoVLA: A unified vision-language-action model with adaptive reasoning. *arXiv:2505.11917*, 2025. 6

Weiyu Liu, Neil Nie, Ruohan Zhang, Jiayuan Mao, and Jiajun Wu. Learning compositional behaviors from demonstration and language. In *Conference on Robot Learning (CoRL)*, 2024. 1, 2, 4, 6

Y. Isabel Liu, Bowen Li, Benjamin Eysenbach, and Tom Silver. SLAP: Shortcut learning for abstract planning. In *International Conference on Learning Representations (ICLR)*, 2026. 2

Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. VIP: Towards universal visual reward and representation via value-implicit pre-training. In *International Conference on Learning Representations (ICLR)*, 2023. 2

Priyanka Mandikal and Kristen Grauman. DexVIP: Learning dexterous grasping with human hand pose priors from video. In *Conference on Robot Learning (CoRL)*, 2021. 2

Kumaranage Ravindu Yasas Nagasinghe, Honglu Zhou, Malitha Gunawardhana, Martin Renqiang Min, Daniel Harari, and Muhammad Haris Khan. Why not use your textbook? knowledge-enhanced procedure planning of instructional videos. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. 2

Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3M: A universal visual representation for robot manipulation. In *Conference on Robot Learning (CoRL)*, 2022. 2

Neil Nie, Wenlong Huang, Jiayuan Mao, Li Fei-Fei, Weiyu Liu, and Jiajun Wu. Learning composable skills by discovering spatial and temporal structure with foundation models. In *International Conference on Robotics and Automation (ICRA)*, 2026. 9

Yulei Niu, Wenliang Guo, Long Chen, Xudong Lin, and Shih-Fu Chang. SCHEMA: State changes matter for procedure planning in instructional videos. In *International Conference on Learning Representations (ICLR)*, 2024. 2

Georgios Papagiannis, Norman Di Palo, Pietro Vitiello, and Edward Johns. R+X: Retrieval and execution from everyday human videos. In *International Conference on Robotics and Automation (ICRA)*, 2025. 2

Chan Young Park, Jillian Fisher, Marius Memmel, Dipika Khullar, Seoho Yun, Abhishek Gupta, and Yejin Choi. Making vlms more robot-friendly: Self-critical distillation of low-level procedural reasoning. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025. 6

Dhruvesh Patel, Hamid Eghbalzadeh, Nitin Kamra, Michael Louis Iuzzolino, Unnat Jain, and Ruta Desai. Pretrained language models as visual planners for human assistance. In *IEEE/CVF International Conference on Computer Vision (CVPR)*, 2023. 2

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv:2504.16054*, 2025. 6

Tianhe Ren, Yihao Chen, Qing Jiang, Zhaoyang Zeng, Yuda Xiong, Wenlong Liu, Zhengyu Ma, Junyi Shen, Yuan Gao, Xiaoke Jiang, et al. Dino-x: A unified vision model for open-world object detection and understanding. *arXiv:2411.14347*, 2024. 47

Kenneth Shaw, Shikhar Bahl, and Deepak Pathak. VideoDex: Learning dexterity from internet videos. In *Conference on Robot Learning (CoRL)*, 2022. 2

Junyao Shi, Zhuolun Zhao, Tianyou Wang, Ian Pedroza, Amy Luo, Jie Wang, Jason Ma, and Dinesh Jayaraman. ZeroMimic: Distilling robotic manipulation skills from web videos. In *International Conference on Robotics and Automation (ICRA)*, 2025a. 2

Lei Shi, Paul-Christian Bürkner, and Andreas Bulling. ActionDiffusion: An action-aware diffusion model for procedure planning in instructional videos. In *Winter Conference on Applications of Computer Vision (WACV)*, 2025b. 2

Lei Shi, Victor Aregbede, Andreas Persson, Martin Längkvist, Amy Loutfi, and Stephanie Lowry. LAP: A language-aware planning model for procedure planning in instructional videos. *arXiv:2603.09743*, 2026. 2

Tom Silver, Rohan Chitnis, Joshua Tenenbaum, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning symbolic operators for task and motion planning. In *International Conference on Intelligent Robots and Systems (IROS)*, 2021. 1, 2, 4

Tom Silver, Ashay Athalye, Joshua B. Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Learning neuro-symbolic skills for bilevel planning. In *Conference on Robot Learning (CoRL)*, 2022. 2

Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomás Lozano-Pérez, Leslie Kaelbling, and Joshua B. Tenenbaum. Predicate invention for bilevel planning. In *AAAI Conference on Artificial Intelligence*, 2023. 2

Himanshu Gaurav Singh, Antonio Loquercio, Carmelo Sferrazza, Jane Wu, Haozhi Qi, Pieter Abbeel, and Jitendra Malik. Hand-object interaction pretraining from videos. In *International Conference on Robotics and Automation (ICRA)*, 2025. 2

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, 1999. 3

Hao Tang, Darren Key, and Kevin Ellis. WorldCoder, a model-based LLM agent: Building world models by writing code and interacting with the environment. In *Advances in Neural Information Processing Systems*, 2024. 2

Yansong Tang, Dajun Ding, Yongming Rao, Yu Zheng, Danyang Zhang, Lili Zhao, Jiwen Lu, and Jie Zhou. COIN: A large-scale dataset for comprehensive instructional video analysis. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

Chen Wang, Linxi Fan, Jiankai Sun, Ruohan Zhang, Li Fei-Fei, Danfei Xu, Yuke Zhu, and Anima Anandkumar. MimicPlay: Long-horizon imitation learning by watching human play. In *Conference on Robot Learning (CoRL)*, 2023a. 2

Hanlin Wang, Yilu Wu, Sheng Guo, and Limin Wang. PDPP: Projected diffusion for procedure planning in instructional videos. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023b. 2

Qianwei Wang, Bowen Li, Zhanpeng Luo, Yifan Xu, Alexander Gray, Tom Silver, Sebastian Scherer, Katia Sycara, and Yaqi Xie. Unifying deep predicate invention with pre-trained foundation models. *arXiv:2512.17992*, 2025. 1, 2

Lionel Wong, Jiayuan Mao, Pratyusha Sharma, Zachary S. Siegel, Jiahai Feng, Noa Korneev, Joshua B. Tenenbaum, and Jacob Andreas. Learning adaptive planning representations with natural language guidance. In *International Conference on Learning Representations (ICLR)*, 2024. 6

Haoming Ye, Yunxiao Xiao, Cewu Lu, and Panpan Cai. UniDomain: Pretraining a unified PDDL domain from real-world demonstrations for generalizable robot task planning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2025. 1, 2, 4, 6

Ce Zhang, Yale Song, Ruta Desai, Michael Louis Iuzzolino, Joseph Tighe, Gedas Bertasius, and Satwik Kottur. Enhancing visual planning with auxiliary tasks and multi-token prediction. In *Winter Conference on Applications of Computer Vision*, 2026. 2

He Zhao, Isma Hadji, Nikita Dvornik, Konstantinos G. Derpanis, Richard P. Wildes, and Allan D. Jepson. P3IV: Probabilistic procedure planning from instructional videos with weak supervision. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 2

Peiyuan Zhi, Zhiyuan Zhang, Yu Zhao, Muzhi Han, Zeyu Zhang, Zhitian Li, Ziyuan Jiao, Baoxiong Jia, and Siyuan Huang. Closed-loop open-vocabulary mobile manipulation with GPT-4V. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2025. 6

Wang Bill Zhu, Ishika Singh, Robin Jia, and Jesse Thomason. Language models can infer action semantics for symbolic planners from environment feedback. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, 2025. 1

Yifeng Zhu, Arisrei Lim, Peter Stone, and Yuke Zhu. Vision-based manipulation from single human video with open-world object graphs. *arXiv:2405.20321*, 2024. 2

Dimitri Zhukov, Jean-Baptiste Alayrac, Ramazan Gokberk Cinbis, David Fouhey, Ivan Laptev, and Josef Sivic. Cross-task weakly supervised learning from instructional videos. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

Supplementary Material

A Implementation Details and Complete Worked Examples

We describe abstract world model induction with a running action-level example (Appendix A.1), provide complete abstract world model catalogs (Appendix A.2), and illustrate state–action trajectory reconstruction for learning to plan (Appendix A.3). We then detail planner inference (Appendix A.4), baseline implementations and prompts (Appendix A.5), and planner training (Appendix A.6). Finally, we present the high-level action sequences and evaluation splits for all three task domains (Appendix A.7).

A.1 Abstract World Model Induction from Human Egocentric Videos

The induction pipeline converts observed executions into explicit language-based abstract world models: predicates describe task-relevant states, and each high-level action is specified by its applicability, effects, and execution constraints. The pipeline first resolves contact-level motion into semantic actions, then generates and refines the abstract world model.

Models and prompt presentation. Both temporal-segmentation stages use Gemini-3.5-Flash, with minimal thinking. Abstract world model generation and the subsequent refinement stages use GPT-5.6 Sol with high reasoning. The following listings present each module's prompt and illustrate its input and output representations.

Constructing an example abstract world model. To illustrate the construction of a language-based abstract world model, we follow a single action throughout the induction pipeline: opening the refrigerator door and moving a food container from the countertop into the refrigerator. This uncoordinated bimanual action comes from a Clean Up Kitchen demonstration. Its semantic interval is 00:25.975–00:43.625 on the 20-fps annotation time base. The same occurrence connects contact segmentation, semantic segmentation, abstract world model generation, and predicate refinement. We show only this action and the annotations needed to illustrate it, rather than every action in the video. The annotation time base specifies these timestamps, not the frame-sampling rate used to process video observations.

A.1.1 Low-Level Contact Segmentation

The first stage takes the video, hand–object and object–object contact annotations, and the object vocabulary derived from those annotations. Annotations constrain contact identities and temporal intervals; video evidence supplies the visible motion and interpretation of primitive phases. The stage produces separate left- and right-hand primitive sequences. `make_contact` and `break_contact` denote instantaneous boundaries, `move_object` denotes a contact interval, and `move_free` retains the gaps between episodes. Tool primitives add the contacted target when both the hand–tool and tool–target relations are supported. Timestamps preserve fractional seconds; the two streams use the same video time base.

Listing 1 shows three hand–object and four object–object annotation records for the example. Their original intervals are retained; an object contact can begin before or persist after this action. Listing 2 shows the corresponding primitive types and timestamps, omitting verbose motion descriptions. Adjacent hand-free intervals retain their original boundaries; semantic segmentation subsequently assigns the action's approach and withdrawal boundaries within them.

See Listing 24 for low-level contact-segmentation prompt.

See Listing 1 for selected contact annotations for the refrigerator-storage action.

See Listing 2 for contact primitives associated with the example action.

A.1.2 High-Level Semantic Segmentation

This stage uses the video together with the contact annotations and validated primitives from the preceding stage. Complete hand-free-to-hand-free contact episodes define the units of semantic

Listing 1: Selected contact annotations for the refrigerator-storage action.


```
[
  {"start": "00:27.65", "end": "00:39.15", "type": "hand-object", "objects": ["food
     container"],
   "hand": "left"},
  {"start": "00:27.65", "end": "00:39.15", "type": "hand-object",
   "objects": ["food container lid"], "hand": "left"},
  {"start": "00:32.35", "end": "00:39.4", "type": "hand-object", "objects": ["fridge door
     "],
   "hand": "right"},
  {"start": "00:00.05", "end": "00:29.1", "type": "object-object",
   "objects": ["food container", "countertop"]},
  {"start": "00:12.25", "end": "01:58.55", "type": "object-object",
   "objects": ["food container", "food"]},
  {"start": "00:21.55", "end": "01:58.55", "type": "object-object",
   "objects": ["food container lid", "food container"]},
  {"start": "00:36.85", "end": "01:58.55", "type": "object-object",
   "objects": ["food container", "fridge"]}
]
```


Listing 2: Contact primitives associated with the example action.


```
{
  "left_hand": [
    {"start_time": "00:24.3", "end_time": "00:27.65", "primitive_type": "move_free(left)
     "},
    {"start_time": "00:27.65", "end_time": "00:27.65",
     "primitive_type": "make_contact(left, food container)"},
    {"start_time": "00:27.65", "end_time": "00:27.65",
     "primitive_type": "make_contact(left, food container lid)"},
    {"start_time": "00:27.65", "end_time": "00:39.15",
     "primitive_type": "move_object(left, food container)"},
    {"start_time": "00:27.65", "end_time": "00:39.15",
     "primitive_type": "move_object(left, food container lid)"},
    {"start_time": "00:39.15", "end_time": "00:39.15",
     "primitive_type": "break_contact(left, food container)"},
    {"start_time": "00:39.15", "end_time": "00:39.15",
     "primitive_type": "break_contact(left, food container lid)"},
    {"start_time": "00:39.15", "end_time": "00:48.1", "primitive_type": "move_free(left)"}
  ],
  "right_hand": [
    {"start_time": "00:17.15", "end_time": "00:32.35", "primitive_type": "move_free(right)
     "},
    {"start_time": "00:32.35", "end_time": "00:32.35",
     "primitive_type": "make_contact(right, fridge door)"},
    {"start_time": "00:32.35", "end_time": "00:39.4",
     "primitive_type": "move_object(right, fridge door)"},
    {"start_time": "00:39.4", "end_time": "00:39.4",
     "primitive_type": "break_contact(right, fridge door)"},
    {"start_time": "00:39.4", "end_time": "00:59.25", "primitive_type": "move_free(right)
     "}
  ]
}
```

action: approach, contact manipulation, release, and withdrawal belong to one episode rather than being treated as four task actions. The video provides the context needed to distinguish an action's purpose and state change from undirected contact motion.

Candidates are first identified independently for each hand. Temporally overlapping candidates are then checked for their coordination requirements. An uncoordinated pair admits a feasible serial execution with the same task-relevant outcome. A coordinated pair requires a simultaneous physical dependency, such as support or stabilization. Coincident timing alone does not establish that dependency. A typed bimanual parent retains both nested hand-specific candidates and their own boundaries; the parent spans their union. Nested candidates are not duplicated as standalone top-level actions. An action is labeled information gathering when its primary purpose is to reduce uncertainty, rather than merely because it moves an object.

Listing 3 shows the resulting semantic parent and its two hand-specific components. The right hand opens the door while the left hand transfers the container. This pair is uncoordinated because opening the door and then inserting the container is also a feasible serial execution. The listing retains boundaries, summaries, and the coordination rationale; the primitive arrays already shown above and repeated descriptive prose are omitted for compactness.

See Listing 25 for high-level semantic-segmentation prompt.

See Listing 3 for semantic segmentation of the refrigerator-storage action.

Listing 3: Semantic segmentation of the refrigerator-storage action.


```
{
  "hands": "both", "bimanual_type": "uncoordinated", "start_time": "00:25.975",
  "end_time": "00:43.625", "action_type": "manipulation",
  "summary": "Open the fridge door and move the food container from the countertop into
     the fridge",
  "primary_object": "food container",
  "unimanual_actions": {
    "left_hand": {"hand": "left", "start_time": "00:25.975", "end_time": "00:43.625",
     "action_type": "manipulation",
     "summary": "Move the food container from the countertop into the fridge",
     "primary_object": "food container"},
    "right_hand": {"hand": "right", "start_time": "00:29.163", "end_time": "00:41.575",
     "action_type": "manipulation", "summary": "Open the fridge door",
     "primary_object": "fridge door"}
  },
  "bimanual_type_rationale": "The actions can be executed sequentially with the same
     outcome: the user could open the fridge door first with one hand, and then move the
     food container into the fridge afterward."
}
```


A.1.3 Generating the Abstract World Model

For each semantic action, the generator receives its target segment and surrounding semantic actions as context. Context helps identify entities, state dependencies, and task outcomes; it does not license absorbing a neighboring action into the target action. The abstract world model specifies a task-level name, minimal relevant preconditions, and net effects. Mutable properties are represented as predicates rather than embedded into new object identities.

A standalone action is represented as unimanual in the abstract world model. A coordinated bimanual action is represented jointly with its execution type. An uncoordinated parent additionally retains two unimanual components in a feasible serial order. The parent preconditions express the initial requirements of this composition, while its effects express the net transition. A condition established by the first component for the second is not automatically an initial precondition of the parent.

The generator also distinguishes changes to physical state from acquisition of information. An observed information-gathering occurrence can record both that a property became known and the value observed in that demonstration. Such an observed outcome is not a license to assume the same truth value before a new inspection at deployment. The refrigerator-storage example illustrates manipulation; information-gathering actions also appear in the complete abstract world models below.

After generation, a sequence-level audit checks consistency between each target action and its preconditions and effects, and between earlier effects and later requirements. The following listings provide the generation and audit interfaces and the abstract world model entries for the selected action. Its context includes the preceding container-closing action and the subsequent transfer of a condiment bottle into the refrigerator. These neighboring actions supply context, not extra operations to include in the target action. The generated parent requires a closed refrigerator and a container on the countertop; only its nested insertion action requires the door to be open, since opening establishes that condition within the composition.

See Listing 26 for abstract world model generation prompt.

See Listing 4 for sequence-level abstract world model audit prompt.

See Listing 5 for generated abstract world model entries for the example action and its serial components.

A.1.4 Predicate Normalization

Normalization standardizes predicate wording while preserving truth conditions, object identities, polarity, and epistemic status. Each predicate is assigned an ID, and the model returns one normalized sentence for that ID. The pipeline replaces these strings in the action structures without changing their order, temporal boundaries, or typed nesting. Explicit false and unknown values remain distinct. The preceding abstract world model entries are the input to this stage. For example, "The fridge door is closed" becomes "It is not that the fridge door is open." The output below retains the parent and both components.

See Listing 6 for predicate-normalization prompt.

See Listing 7 for example action after predicate normalization.

A.1.5 Predicate-Level Unification

Different demonstrations can express the same predicate with different wording. Predicate-level unification groups expressions that have equivalent truth conditions and selects an existing member as the canonical expression. Matching must preserve the state-bearing entity, semantic roles, polarity, and whether the statement concerns a physical fact or knowledge of that fact. Related causes or consequences are not interchangeable predicates merely because they frequently occur together.

The main grouping prompt receives a target and candidate groups. A canonical expression is selected for each group and propagated throughout the abstract world model. The normalized example above supplies this action's predicates. For this particular action, their wording already matches the canonical predicates, so its preconditions and effects remain unchanged in the output below. This step need not change every action to unify the vocabulary across videos.

See Listing 8 for predicate-equivalence grouping prompt.

See Listing 9 for example action after predicate-level unification.

A.1.6 Action-Level Unification of the Abstract World Model

The final refinement stage consolidates the abstract world model across videos by principal task operation and outcome. Names provide the principal action meaning, with preconditions and effects providing evidence about the intended transition. Identical predicate lists are not required, since independently generated descriptions can differ in incidental detail. Conversely, a compound action is not equivalent to one of its components simply because their names share an object or verb.

Equivalent actions are grouped and represented by an existing complete semantic triple. A demonstrated producer–consumer dependency constrains the corresponding actions to remain distinct. Execution metadata is retained separately: alternative manualities can belong to one semantic action, and occurrence-specific hand assignments remain attached to the demonstration. The unified catalog records available execution forms in `execution_variants`.

For the running example, this stage compares the refrigerator-storage action with actions from other videos, consolidates equivalent operations, and links each occurrence to the selected canonical action

Listing 4: Sequence-level abstract world model audit prompt.

```
You are an independent planning-domain auditor. Review every generated typed
action definition for one demonstrated video. The semantic segmentation is the
trusted interpretation of the demonstration; the definitions are proposals.

For every indexed proposal, verify:

1. Its action name denotes the same complete task outcome and preserves stable
   entity identities. Treat the semantic target's `summary` as the authoritative
   inventory of PRINCIPAL task subgoals. Detailed analysis, contact primitives,
   and neighboring segments may explain means or state but cannot authorize an
   additional independent task verb/outcome. Reject a definition that combines
   the target with a preceding, following, sibling, or incidental component
   action absent from that summary. For an uncoordinated wrapper, apply this
   exact-subgoal rule independently to the parent summary and to each nested
   hand's summary.
2. Preconditions are the planning-critical states required immediately before
   execution. Effects are the complete net state transition. Reject invented
   restrictions, contradictions, missing sources/destinations, or missing
   state changes that a later action consumes.
3. Standalone, coordinated, and uncoordinated schemas match the semantic target.
   A coordinated parent has no nested definitions. An uncoordinated wrapper has
   exactly two source-hand-traceable nested definitions in executable semantic
   order, and its parent equals their serial net transition.
4. Across the sequence, a later precondition must be established by an earlier
   effect or be a plausible unchanged initial fact. If an earlier action moves,
   opens, closes, attaches, detaches, activates, deactivates, cleans, fills, or
   empties the same entity, do not misclassify its resulting state as initial.
5. Producer and consumer predicates use the same state-bearing entity and a
   logically equivalent property. A relation such as one component not being
   attached does not automatically establish a separately asserted open state.
6. Predicates are atomic world states; hand predicates and mutable state inside
   entity names are forbidden. Apply only the explicit optional guidance below.
7. This audit is before predicate normalization. Do not reject clear lexical
   equivalents merely because their raw wording differs. Do not require
   unchanged effects or incidental visual details. Report only planning-relevant
   semantic errors that should change an action definition.

Contact-annotation-derived object vocabulary:
{object_vocabulary}

Optional explicit user-provided examples and conventions:
```json
{domain_examples}
```

Indexed semantic targets and proposed definitions:
```json
{sequence_records}
```

Return one review for every input index in order. `overall_verdict` is `pass`
only when every review passes; otherwise it is `needs_correction`. Return
exactly one final fenced JSON object:
```json
{{
  "audited_indices": [0],
  "reviews": [
    {{
      "index": 0,
      "verdict": "pass | needs_correction",
      "issues": [],
      "evidence": "concise planning-semantic justification"
    }}
  ],
  "overall_verdict": "pass | needs_correction"
}}
```
```

Listing 5: Generated abstract world model entries for the example action and its serial components.

```
{
  "bimanual_action_definition": {"action_name": "open fridge door and move food container
     from countertop into fridge",
   "manuality": "bimanual", "bimanual_type": "uncoordinated",
   "precondition": ["The fridge door is closed", "The food container is on the countertop
     "],
   "effect": ["The fridge door is open", "The food container is in the fridge", "The food
     container is not on the countertop"]},
  "sequential_unimanual_action_definitions": [
    {"source_hand": "right", "action_name": "open fridge door", "manuality": "unimanual",
     "precondition": ["The fridge door is closed"], "effect": ["The fridge door is open
     "]},
    {"source_hand": "left", "action_name": "move food container from countertop into
     fridge",
     "manuality": "unimanual",
     "precondition": ["The fridge door is open", "The food container is on the countertop
     "],
     "effect": ["The food container is in the fridge", "The food container is not on the
     countertop"]}
  ]
}
```

Listing 6: Predicate-normalization prompt.

```
You normalize predicate language without changing action semantics or typed
structure. The caller supplies stable predicate IDs and performs deterministic
replacement. Return exactly one normalized string for every input ID, in the
same order, with no missing or extra IDs.

Rules:
1. Preserve every entity name and truth condition exactly.
2. Use one concise atomic simple-present state sentence ending with a period.
3. Remove tense history, progressive aspect, modals, temporal qualifiers, and
   true passive voice while preserving meaning. Copular states are valid.
4. Mutable attributes are states, never object-identity adjectives.
5. Use `clean` as the canonical cleanliness concept.
6. Normalize negative-form concepts to an affirmative base, then express every
   negative fact exactly as `It is not that <affirmative clause>.` Examples:
   closed -> not open; off -> not on; unplugged -> not plugged in; outside ->
   not inside; unknown -> not known; nonempty -> not empty. The clause after
   `that` contains no direct negation or negative-form state word.
7. Device predicates describe current state: `The device is on.` or
   `It is not that the device is on.` Never use `turned on/off`.
8. Preserve epistemic meaning: `Whether P is unknown` -> `It is not that it is
   known whether P.`; `Whether P is known` -> `It is known whether P.`
9. Do not infer, add, delete, merge, reorder, or domain-canonicalize facts.
10. Never introduce hand-related predicates.

Review every record against all rules and self-correct before output. Return
exactly:

```json
{{
  "normalized_predicates": [
    {{"predicate_id": "<input ID>", "normalized_predicate": "<sentence>."}}
  ]
}}
```

Input:
```json
{input}
```

The fenced JSON block must be the final content in the response.
```

Listing 7: Example action after predicate normalization.

```
{
  "bimanual_action_definition": {"action_name": "open fridge door and move food container
     from countertop into fridge",
    "manuality": "bimanual", "bimanual_type": "uncoordinated",
    "precondition": ["It is not that the fridge door is open.", "The food container is on
     the countertop."],
    "effect": ["The fridge door is open.", "The food container is in the fridge.", "It is
     not that the food container is on the countertop."]},
  "sequential_unimanual_action_definitions": [
    {"source_hand": "right", "action_name": "open fridge door", "manuality": "unimanual",
     "precondition": ["It is not that the fridge door is open."],
     "effect": ["The fridge door is open."]},
    {"source_hand": "left", "action_name": "move food container from countertop into
     fridge",
     "manuality": "unimanual",
     "precondition": ["The fridge door is open.", "The food container is on the countertop
     ."],
     "effect": ["The food container is in the fridge.", "It is not that the food container
      is on the countertop."]}
  ]
}
```

Listing 8: Predicate-equivalence grouping prompt.

```
Group normalized planning predicates only when they have identical truth
conditions at the supplied planning abstraction. Return existing group IDs;
never rewrite or invent predicate text.

Preserve polarity, epistemic status, stable entity identity, property,
relation, location, source/destination, containment, support, availability,
quantity, and argument roles. Allow grammatical paraphrases and inverse
surface relations only when they are bidirectionally substitutable.

Also recognize a component-defined functional state: a whole artifact state
and the state of its uniquely named closure or control component are equivalent
when the component state is exactly what defines the artifact state at the
planning abstraction. For example, `The enclosure is open.` and `The enclosure
door is open.` may be equivalent, as may their matching negative forms. This is
not permission to ignore entity identity generally. A component's location,
motion, availability, cleanliness, or unrelated state is not automatically the
state of its parent artifact; `The cover is on the workbench.` is not equivalent
to `The enclosure is open.`. Likewise, a correlated component state that does
not bidirectionally define the whole artifact state must remain separate.

Related causes, consequences, correlations, or domain conventions are not
equivalent unless the relation is the component-defined functional-state case
above or is explicitly stated in optional user-provided domain examples. When
uncertain, do not match.

Review target semantics, compare every candidate bidirectionally, reject every
meaning change, and verify every returned ID exists.

Optional user-provided domain examples:
```json
{domain_examples}
```

Target predicate:
{target_predicate}

Candidate groups:
{candidate_groups_json}

Return exactly one final fenced JSON object:
```json
{{"matching_group_ids": ["<existing ID>"], "equivalence_summary": "<reason>"}}
```
```

Listing 9: Example action after predicate-level unification.

```
{
  "bimanual_action_definition": {"action_name": "open fridge door and move food container
     from countertop into fridge",
   "manuality": "bimanual", "bimanual_type": "uncoordinated",
   "precondition": ["It is not that the fridge door is open.", "The food container is on
     the countertop."],
   "effect": ["The fridge door is open.", "The food container is in the fridge.", "It is
     not that the food container is on the countertop."]},
  "sequential_unimanual_action_definitions": [
    {"source_hand": "right", "action_name": "open fridge door", "manuality": "unimanual",
     "precondition": ["It is not that the fridge door is open."],
     "effect": ["The fridge door is open."]},
    {"source_hand": "left", "action_name": "move food container from countertop into
     fridge",
     "manuality": "unimanual",
     "precondition": ["The fridge door is open.", "The food container is on the countertop
     ."],
     "effect": ["The food container is in the fridge.", "It is not that the food container
      is on the countertop."]}
  ]
}
```

in the abstract world model while preserving its execution metadata. We do not repeat the action's output here. The listing gives the main grouping prompt, and the three complete domain-level catalogs follow in Section A.2.

See Listing 10 for action-equivalence grouping prompt.

A.2 Complete Abstract World Model Catalogs

This section provides the complete language-based abstract world models for the three task domains. Predicate lists define the explicit state vocabulary, while action catalogs specify names, preconditions, effects, and execution constraints. These abstract world models are reused across planning instances, not plans for one particular language goal.

For a common serialization format, the Boil Water and Make Instant Noodles action lists include `execution_variants` with the single value `manuality: unimanual`. This is a metadata-only conversion of their original catalogs: names, preconditions, effects, ordering, and predicate strings are unchanged. Clean Up Kitchen retains its existing typed catalog.

A.2.1 Boil Water

See Listing 11 for complete Boil Water predicate vocabulary.

See Listing 27 for Boil Water abstract world model: complete action library, preconditions, effects, and execution metadata.

A.2.2 Make Instant Noodles

See Listing 12 for complete Make Instant Noodles predicate vocabulary.

See Listing 28 for Make Instant Noodles abstract world model: complete action library, preconditions, effects, and execution metadata.

A.2.3 Clean Up Kitchen

See Listing 13 for complete Clean Up Kitchen predicate vocabulary.

See Listing 29 for Clean Up Kitchen abstract world model: complete action library, preconditions, effects, and execution constraints.

Listing 10: Action-equivalence grouping prompt.

```
Group planning actions by shared task-action semantics. `action_name` semantics
are the primary criterion. Manuality, bimanual type, source hand, provenance,
and observed execution timing are deliberately outside the comparison and must
neither cause nor prevent grouping.

Treat two actions as the same action when their names denote the same main
operation and task outcome over the same stable entities, tools, semantic
roles, source, and destination. Exact wording does not matter: verb paraphrases
and different levels of motion detail may describe the same task action.

Preconditions and effects are secondary evidence, not equality gates. Their
wording, order, number, specificity, optional context, or incidental physical
details may differ because definitions were generated independently. Missing
or extra predicates alone must never block a match. Use predicates only to
clarify the action-name meaning or to detect a genuine conflict in the core
action identity, such as different objects, reversed source/destination,
opposite activation outcomes, different queried properties or observed
values, or a different principal task result.

Distinguish principal task roles from incidental implementation details. A
location or motion that only explains how a component, tool, cover, support,
or other auxiliary object was handled must not split two actions whose names
otherwise express the same principal operation and task outcome. Conversely,
a different source or destination of the principal moved, disposed, or queried
object is a core identity difference and must remain separate.

Do not collapse a compound action into one of its component actions. An
additional verb phrase joined as another task step (for example, "unlock an
enclosure and transfer a package") is a distinct principal subgoal when it
changes an independent state, even if it enables the other step. A means phrase
such as "open an enclosure by moving its cover" may instead elaborate one
operation when the component motion physically realizes that same outcome.

Candidate groups list their observed action names and representative complete
semantic triples. Compare the target against every group and merge groups
whenever they express the same task action. Use representative triples to
decide whether component motion or a means phrase physically realizes the same
principal outcome, but never infer a new compound outcome. When uncertain,
prefer the interpretation supported by complete names and net task transitions
rather than predicate-list equality. Return existing group IDs only and never
rewrite fields.

Optional user-provided domain examples:
```json
{domain_examples}
```

Target:
{target_action_json}

Candidate groups:
{candidate_groups_json}

Return exactly one final fenced JSON object:
```json
{{"matching_group_ids": ["<existing ID>"], "equivalence_summary": "<reason>"}}
```
```

Listing 11: Complete Boil Water predicate vocabulary.

```
[
  "It is known whether the kettle contains water.",
  "It is known whether the microwave is empty.",
  "It is known whether the pot contains water.",
  "The bowl is in the microwave.",
  "The bowl is on the counter.",
  "The faucet is turned on.",
  "The induction cooker is turned on.",
  "The kettle contains water.",
  "The kettle is in the sink.",
  "The kettle is on the counter.",
  "The kettle is on the kettle base.",
  "The kettle is plugged in.",
  "The kettle is turned on.",
  "The kettle is under the faucet water stream.",
  "The kettle lid is on the counter.",
  "The kettle lid is on the kettle.",
  "The kettle lid is open.",
  "The microwave door is open.",
  "The microwave is empty.",
  "The microwave is on.",
  "The mug contains water.",
  "The mug is in the sink.",
  "The mug is inside the microwave.",
  "The mug is on the counter near the microwave.",
  "The mug is on the counter.",
  "The mug is on the drying rack.",
  "The pot contains water.",
  "The pot is in the sink.",
  "The pot is on the counter.",
  "The pot is on the induction cooker.",
  "The pot is on the shelf.",
  "The pot lid is on the counter.",
  "The pot lid is on the pot.",
  "The pot lid is open.",
  "The water bottle cap is on the counter.",
  "The water bottle cap is on the water bottle.",
  "The water bottle cap is open.",
  "The water bottle contains water.",
  "The water bottle is on the counter.",
  "The water bottle is on the shelf."
]
```

Listing 12: Complete Make Instant Noodles predicate vocabulary.

```
[
  "It is known whether the water is hot.",
  "The ceramic bowl contains water.",
  "The ceramic bowl is in the microwave.",
  "The ceramic bowl is on the countertop.",
  "The clip is on the instant noodle bag.",
  "The electric kettle contains water.",
  "The electric kettle is plugged in.",
  "The fried egg is in the ceramic bowl.",
  "The fried egg is in the instant noodle bowl.",
  "The fried egg is in the pot.",
  "The fried egg is on the ceramic plate.",
  "The fried egg is on the pan.",
  "The instant noodle bag is open.",
  "The instant noodle block is in the ceramic bowl.",
  "The instant noodle block is in the instant noodle bag.",
  "The instant noodle block is in the pot.",
  "The instant noodle bowl contains water.",
  "The instant noodle bowl is in the microwave.",
  "The instant noodle bowl is on the countertop.",
  "The instant noodle bowl lid is on the countertop.",
  "The instant noodle bowl lid is on the instant noodle bowl.",
  "The instant noodle bowl lid is open.",
  "The microwave door is open.",
  "The microwave is turned on.",
  "The pan lid is on the ceramic bowl.",
  "The pan lid is on the countertop.",
  "The pan lid is on the pan.",
  "The pan lid is open.",
  "The pot contains water.",
  "The pot is on the stove.",
  "The pot lid is on the countertop.",
  "The pot lid is on the pot.",
  "The pot lid is open.",
  "The sausage is in the ceramic bowl.",
  "The sausage is in the pot.",
  "The sausage is inside the instant noodle bowl.",
  "The sausage is on the ceramic plate.",
  "The sausage is on the pan.",
  "The spatula is available.",
  "The stove is turned on.",
  "The water is hot."
]
```

Listing 13: Complete Clean Up Kitchen predicate vocabulary.

```
[
  "The ceramic plate has soap suds.",
  "The ceramic plate is clean.",
  "The ceramic plate is in the dish rack.",
  "The ceramic plate is in the sink.",
  "The ceramic plate is on the countertop.",
  "The ceramic plate is on the dining table.",
  "The countertop is clean.",
  "The dish soap bottle contains dish soap.",
  "The faucet is on.",
  "The food container is in the fridge.",
  "The food container is on the countertop.",
  "The food container is on the dining table.",
  "The food container is open.",
  "The food is in the food container.",
  "The food is on the ceramic plate.",
  "The food is on the paper plate.",
  "The fridge door is open.",
  "The ketchup bottle is in the fridge.",
  "The ketchup bottle is on the countertop.",
  "The ketchup bottle is on the dining table.",
  "The paper plate is in the trash can.",
  "The paper plate is on the countertop.",
  "The paper plate is on the dining table.",
  "The sponge is available.",
  "The tissue is in the tissue box.",
  "The tissue is in the trash can.",
  "The tissue is on the countertop."
]
```

A.3 Learning to Plan with the Abstract World Model

The induced abstract world model serves two complementary roles: its action preconditions and effects provide supervision for reconstructing demonstration states, and the same explicit model conditions the planner during training and deployment. The planner learns to select actions and predict successor belief states conditioned on a language goal, the current belief state, and the supplied model. We describe this process in two stages: complete belief–action trajectory reconstruction (Appendix A.3.1), followed by planner SFT data generation (Appendix A.3.2). The second stage extracts subtrajectories, generates their language goals, and constructs one-hand and two-hand supervision for bimanual training.

A.3.1 Complete Belief–Action Trajectory Reconstruction

Reconstructing boundary beliefs. We use the observation-grounding and belief-update components of the online planning framework to reconstruct demonstration trajectories. The induced semantic segments supply the demonstrated action sequence and its temporal boundaries; the planner does not generate replacement actions during this process. At each action boundary, a predicate classifier evaluates predicates associated with relevant visible objects. Belief update combines this visual evidence with the preceding belief and executed action. The reconstruction then incorporates the preceding action's effects and the current demonstrated action's preconditions, in that order. Thus, for a state immediately before action a_k, the adjacent constraints come from the effects of a_{k-1} and the preconditions of a_k, not from the effects of a future action. The first boundary has no preceding task action; the terminal boundary has no subsequent task-action preconditions.

The reconstructed belief assigns predicates true, false, or unknown. Explicit negative statements express false values; statements that a fact is not yet known express unknown values rather than false ones. This procedure yields state supervision from observations and induced action descriptions without requiring manually annotated intermediate states. These states are reconstructed supervision, not independently annotated ground-truth states.

A complete worked example. For a compact reconstruction example, we use a short Clean Up Kitchen demonstration. Its five demonstrated actions produce six boundary states. The listing preserves every reconstructed predicate statement and every demonstrated action's name, preconditions, effects, and execution metadata from the abstract world model. A final `goal is achieved` record marks termination and is not an additional physical action. The example stores food in the refrigerator and disposes of the paper plate within one continuous demonstration. Unlike the preceding action-level induction example, this listing contains the complete reconstructed trajectory.

Concretely, the five actions are: open the food container and pour food into it from the paper plate; close the food container; open the fridge and move the container inside; close the fridge; and put the paper plate in the trash can. At b_0, the container is closed and the food is on the plate. To reconstruct b_1, the first action's effects establish that the container is open and contains the food, and that the food is no longer on the plate; the next action's precondition also requires the container to be open. The remaining facts, such as the fridge door being closed and the paper plate remaining on the dining table, persist from the preceding belief unless updated by the reconstruction evidence. At b_2, closing the container makes its open predicate false while the food remains inside. The example therefore illustrates both changed predicate values and information retained across action boundaries.

See Listing 30 for complete reconstructed state–action trajectory for a short Clean Up Kitchen demonstration.

A.3.2 Planner SFT Data Generation

Extracting belief–action subtrajectories. We form additional training examples by taking contiguous slices of a complete trajectory. A slice from boundary i to boundary j contains beliefs $b_i, \ldots, b_j$ and demonstrated actions $a_{i+1}, \ldots, a_j$, followed by the termination record at b_j. A language model describes the intended outcome of the actions within that slice; preceding actions outside the slice are not inputs to this goal-generation prompt. The initial belief, generated goal, and abstract world model form the planner input, while the interleaved actions and successor beliefs provide the supervised target. The shared domain-level abstract world model supplies the action semantics; capability- and

target-specific input views for bimanual training are described in the bimanual training construction below.

Here a_k takes the demonstration from boundary b_{k-1} to b_k, as in the main text. The exported JSON uses zero-based step indices: record i contains b_i and its next action a_{i+1}.

Goal generation. The following template takes the actions within a selected slice, including their preconditions and effects, and requests a concise description of their intended outcome. Braced fields denote runtime inputs.

See Listing 14 for goal-generation prompt.

Examples of extracted subtrajectories. The following two examples use boundaries 0–2 and 2–4 of the complete trajectory. The first transfers food into the container and closes it, with the generated goal *Have the food inside the closed food container rather than on the paper plate.* The second stores the container in the refrigerator and closes the door, with the generated goal *Have the food container in the closed fridge rather than on the dining table.* Their beliefs and actions are exact slices of Listing 30, with termination inserted at the selected endpoint. These listings preserve the demonstrated top-level form; the one-hand variants constructed below expand eligible bimanual actions and insert intermediate beliefs.

See Listing 15 for language goal for boundaries 0–2.

See Listing 16 for complete subtrajectory for boundaries 0–2.

See Listing 17 for language goal for boundaries 2–4.

See Listing 18 for complete subtrajectory for boundaries 2–4.

Listing 14: Goal-generation prompt.

```
### Objective
You are an intelligent assistant specialized in inferring the intended high-level goal
    of a robot task plan.

You will be given:
- A **task plan**: a chronological sequence of high-level actions. Each action is
    described in natural language and may include its name, preconditions, and effects.

Your task is to infer the **single most appropriate high-level language goal** that
    this task plan is intended to achieve.

### Core Principle
A task plan is typically generated to achieve a goal. Here, you must perform the
    reverse inference:
- given the task plan,
- infer the **intended task goal** that the plan is designed to accomplish.

The goal should capture **what the plan is trying to achieve**, rather than merely
    restating **how** the actions are executed.

### Requirements
1. Infer a goal that is **consistent with the entire task plan**.
2. The goal must be a **succinct, meaningful, and high-level language description** of
    the intended outcome.
3. The goal must be **achievable by executing the given task plan step by step**.
4. Prefer describing the **final intended accomplishment** of the plan, rather than
    listing intermediate states or action details.
5. Prefer the **most specific high-level goal that is clearly supported by the full
    plan**, but **do not invent intentions that are not grounded in the plan**.
6. Avoid goals that are:
   - overly low-level,
   - merely a paraphrase of the action sequence,
   - overly vague when a clearer high-level goal is supported.
7. Your inference should reflect a deep understanding of:
   - the semantic meaning of each action,
   - the dependencies between actions,
   - and the cumulative outcome of the full sequence.

### Output Instructions
First, carefully analyze the task plan step by step and reason about what overall goal
    the full sequence is intended to achieve.

Then, after completing your reasoning, output the final inferred high-level language
    goal in the following JSON format.

IMPORTANT:
- You should explicitly show your reasoning and analysis before the final JSON block.
- The final answer must appear as a fenced JSON code block.
- The JSON must strictly follow the format below.
- The final JSON block must contain only the inferred goal, with no extra fields.

'''json
{{"language_goal": "<your inferred high-level language goal>"}}
'''

### Task Input

#### Task Plan
{task_plan}

Now begin by analyzing the task plan step by step, infer the single most appropriate
    high-level goal it is intended to achieve, and then output the final result as a
    fenced JSON code block strictly following the required format.
```

Listing 15: Language goal for boundaries 0–2.

```
{"language_goal": "Have the food inside the closed food container rather than on the paper
    plate."}
```

Listing 16: Complete subtrajectory for boundaries 0–2.


```
[
  {
    "state": [
      "It is not that the food container is open.",
      "The food is on the paper plate.",
      "It is not that the fridge door is open.",
      "The food container is on the dining table.",
      "The paper plate is on the dining table.",
      "It is not that the paper plate is in the trash can.",
      "It is not that the food container is in the fridge.",
      "It is not that the food is in the food container."
    ],
    "action": {"action_name": "open food container and pour food from paper plate into
     food container",
     "manuality": "bimanual", "bimanual_type": "uncoordinated",
     "precondition": ["It is not that the food container is open.", "The food is on the
     paper plate."],
     "effect": ["The food container is open.", "The food is in the food container.", "It
     is not that the food is on the paper plate."]}
  },
  {
    "state": [
      "The food container is open.",
      "The food is in the food container.",
      "It is not that the food is on the paper plate.",
      "It is not that the fridge door is open.",
      "The food container is on the dining table.",
      "The paper plate is on the dining table.",
      "It is not that the paper plate is in the trash can.",
      "It is not that the food container is in the fridge."
    ],
    "action": {"action_name": "close food container with food container lid", "manuality":
      "unimanual",
     "precondition": ["The food container is open."],
     "effect": ["It is not that the food container is open."]}
  },
  {
    "state": [
      "The food is in the food container.",
      "It is not that the food is on the paper plate.",
      "It is not that the food container is open.",
      "It is not that the fridge door is open.",
      "The food container is on the dining table.",
      "The paper plate is on the dining table.",
      "It is not that the paper plate is in the trash can.",
      "It is not that the food container is in the fridge."
    ],
    "action": {"action_name": "goal is achieved", "precondition": [], "effect": []}
  }
]
```


Listing 17: Language goal for boundaries 2–4.


```
{"language_goal": "Have the food container in the closed fridge rather than on the dining
    table."}
```

Listing 18: Complete subtrajectory for boundaries 2–4.


```
[
  {
    "state": [
      "The food is in the food container.",
      "It is not that the food is on the paper plate.",
      "It is not that the food container is open.",
      "It is not that the fridge door is open.",
      "The food container is on the dining table.",
      "The paper plate is on the dining table.",
      "It is not that the paper plate is in the trash can.",
      "It is not that the food container is in the fridge."
    ],
    "action": {"action_name": "open fridge door and move food container from dining table
     into fridge",
     "manuality": "bimanual", "bimanual_type": "uncoordinated",
     "precondition": ["It is not that the fridge door is open.", "The food container is on
      the dining table."],
     "effect": ["The fridge door is open.", "The food container is in the fridge.", "It is
      not that the food container is on the dining table."]}
  },
  {
    "state": [
      "The food is in the food container.",
      "It is not that the food is on the paper plate.",
      "It is not that the food container is open.",
      "The fridge door is open.",
      "The food container is in the fridge.",
      "It is not that the food container is on the dining table.",
      "The paper plate is on the dining table.",
      "It is not that the paper plate is in the trash can."
    ],
    "action": {"action_name": "close fridge door", "manuality": "unimanual",
     "precondition": ["The fridge door is open."],
     "effect": ["It is not that the fridge door is open."]}
  },
  {
    "state": [
      "The food is in the food container.",
      "It is not that the food is on the paper plate.",
      "It is not that the food container is open.",
      "The food container is in the fridge.",
      "It is not that the food container is on the dining table.",
      "It is not that the fridge door is open.",
      "The paper plate is on the dining table.",
      "It is not that the paper plate is in the trash can."
    ],
    "action": {"action_name": "goal is achieved", "precondition": [], "effect": []}
  }
]
```

Bimanual training: constructing one-hand and two-hand examples. After reconstructing complete belief–action trajectories, we construct capability-conditioned examples when extracting subtrajectories for planner supervised fine-tuning. This stage teaches two complementary operations: composing unimanual actions into an uncoordinated bimanual action when both hands are available, and serializing an uncoordinated bimanual action when only one hand is available. It does not require an additional video of the one-hand execution. The examples below are taken from the Clean Up Kitchen planner SFT dataset.

SHARED TASK, DIFFERENT HAND AVAILABILITY. For a selected subtrajectory, the two variants share the initial belief and the underlying language goal. We append either *You can use both hands.* or *You can only use one hand.* to that goal. The two-hand target preserves the demonstrated top-level actions and reconstructed boundary beliefs; it can contain both unimanual and bimanual actions. Two-hand availability does not require every action to use both hands. A one-hand counterpart is constructed only if the slice contains no coordinated bimanual action. Existing unimanual actions are retained, while each uncoordinated bimanual action is replaced by its ordered unimanual components. Both variants end at the original slice's final belief, followed by the `goal is achieved` termination record.

CONDITIONING ON THE ABSTRACT WORLD MODEL. The input language-based abstract world model is adapted to the example's capability and supervision target without changing the retained actions' preconditions and effects. For one-hand examples, its action library contains only unimanual execution variants. For two-hand examples, we omit the uncoordinated bimanual execution variant of each target compound action in the selected slice, while retaining its unimanual components. Other actions and their available execution variants remain. The compound action is still present in the supervised target: the planner must learn to compose the components rather than copy that compound action from its input. This target-specific omission is not a removal of all bimanual actions from the abstract world model.

SERIALIZING AN UNCOORDINATED BIMANUAL ACTION. Let a^{bi} denote an uncoordinated bimanual action with ordered unimanual components $a^{(1)}$ and $a^{(2)}$. We preserve its reconstructed pre-action and post-action beliefs, b_{pre} and b_{post}, and insert an intermediate belief b_{mid}:

$$\begin{aligned} \text{Two hands:} \quad & b_{\mathrm{pre}} \xrightarrow{a^{\mathrm{bi}}} b_{\mathrm{post}}, \\ \text{One hand:} \quad & b_{\mathrm{pre}} \xrightarrow{a^{(1)}} b_{\mathrm{mid}} \xrightarrow{a^{(2)}} b_{\mathrm{post}}. \end{aligned} \tag{A1}$$

The order follows the components' action dependencies, not a fixed left-hand or right-hand ordering. The hand used in the original demonstration is provenance; both components can be executed sequentially by the available hand. If a slice contains several uncoordinated bimanual actions, each is expanded in this way while the surrounding actions and boundary beliefs are preserved.

CONSTRUCTING THE INTERMEDIATE BELIEF. Crucially, b_{mid} is constructed from *three* inputs: the preceding belief b_{pre}, the first component's effects, and the second component's preconditions. Writing Pre and Eff for the corresponding statements in the abstract world model, the construction is

$$b_{\mathrm{mid}} = \mathrm{Revise}\Big(b_{\mathrm{pre}}; \mathrm{Eff}(a^{(1)}), \mathrm{Pre}(a^{(2)})\Big). \tag{A2}$$

Here Revise denotes the belief-editing rule used for reconstruction: start from b_{pre}, incorporate the first action's effects and the next action's preconditions, replace conflicting predicate values, and preserve facts not affected by these constraints. It is not merely the union of the effects and preconditions, which would discard unaffected information from the preceding belief. Unknown values remain unknown unless the supplied constraints resolve them; unknown and false are not interchangeable. The second action's effects are not applied at this boundary. The resulting b_{mid} is constructed belief supervision for the sequential execution, not an additional observed video frame. We check that the first action is applicable in b_{pre}, that its effects and the second action's preconditions hold in b_{mid}, and that the second action's effects hold in b_{post}.

EXAMPLE 1: OPENING THE FRIDGE AND STORING A CONTAINER. An actual pair shares the goal *Have the food container in the open fridge rather than on the dining table.* The two-hand target

contains *open fridge door and move food container from dining table into fridge*. The one-hand target instead contains (1) *open fridge door*, followed by (2) *move food container from dining table into fridge*. The first component requires that the door is not open and makes it open. The second requires the open door and the container on the dining table; it moves the container into the fridge and makes its dining-table predicate false. Table A1 shows all eight predicate values in this pair. At b_{mid}, only the door has changed: the container is still on the dining table and is not yet in the fridge. The facts about the food, container lid, and paper plate are carried forward from b_{pre}. Thus the inserted belief supervises what has changed after the first action *and what has not yet changed* before the second action.

Table A1: **Belief supervision for the fridge-storage example.** The two-hand target connects b_{pre} directly to b_{post} through one compound action; the one-hand target inserts b_{mid}. T and F denote true and false. All eight predicates of this example are shown; none has an unknown value at these boundaries.

Predicate	b_{pre}	b_{mid}	b_{post}
The fridge door is open.	F	T	T
The food container is on the dining table.	T	T	F
The food container is in the fridge.	F	F	T
The food is in the food container.	T	T	T
The food is on the paper plate.	F	F	F
The food container is open.	F	F	F
The paper plate is on the dining table.	T	T	T
The paper plate is in the trash can.	F	F	F

EXAMPLE 2: OPENING A CONTAINER AND TRANSFERRING FOOD. Another pair shares the goal *Have the food in the open food container rather than on the paper plate.* Its two-hand target is *open food container and pour food from paper plate into food container*. The one-hand target first opens the container, then pours the food into it. At the constructed b_{mid}, *The food container is open.* is true, *The food is on the paper plate.* remains true, and *The food is in the food container.* remains false. The food-location values change only after the pouring action. As in the first example, the two targets share their initial and final beliefs, but the one-hand target adds an explicit intermediate consequence for the planner to predict.

PRESERVING COORDINATED ACTIONS. The dataset also contains *wash ceramic plate with sponge*, a coordinated bimanual action in which one hand holds the plate while the other cleans it. An example with the goal *Clean the ceramic plate. You can use both hands.* retains this action as a single step. Its preconditions include a not-yet-clean plate in the sink, soap suds, the faucet being on, and an available sponge; its effects make the plate clean and remove the soap suds. We do not create a one-hand counterpart for a slice containing this action: separating holding from scrubbing would remove the required simultaneous support. This is a constraint on the represented action, not a claim that no alternative one-hand cleaning strategy could exist.

These paired examples teach the planner to compose or serialize actions and predict the corresponding belief transitions, subject to the abstract world model's coordination constraints.

A.4 PLANNER INFERENCE

DPK fine-tunes a Qwen3-VL-8B-Instruct planner on the reconstructed goal-conditioned state–action trajectories. At inference, the planner receives the current belief state, language goal, and explicit language-based abstract world model. It does not receive the image observation or an action history. Visual predicate classification and belief update operate outside the planner. The planner alternates action selection with successor-belief prediction.

Only the next selected action is executed. A new observation then updates the belief before replanning. Predicted successor states are planning outputs, not observations that directly overwrite the current belief. In particular, an information-gathering action can make a previously unknown condition observable, but its actual truth value must come from the ensuing observation.

Planner prompt. Listing 31 specifies the planner's input and output interface. The planner conditions on the abstract world model, including its action library, preconditions, and effects, together with the language goal and current belief to predict an interleaved action–belief trajectory.

See Listing 31 for planner inference prompt.

A.5 Baseline Implementation and Prompts

VLM-as-Planner. The proprietary baseline uses GPT-5.6 Sol with high reasoning, while the open-weight and fine-tuned baselines use Qwen3-VL-8B. All three use the same action-sequence planning interface: the current image observation, language goal, action names with execution metadata, and executed action history. They do not receive DPK's belief state or the action preconditions and effects. The fine-tuned baseline learns action-sequence targets from the demonstration data. The listed prompt is the complete extracted baseline template; doubled braces are literal JSON braces escaped for Python template formatting.

See Listing 32 for complete VLM-as-Planner baseline prompt.

LLM-as-Domain-Generator. This baseline uses GPT-5.6 Sol with high reasoning to translate the language-based abstract world model into a reusable lifted PDDL domain. Predicates take arguments and action schemas have parameters; the translation does not simply create one zero-argument PDDL operator per natural-language action. Additional translation stages construct the problem's object list, initial state, and goal literals, and serialize the PDDL problem. Fast Downward then searches the resulting symbolic model through the Unified Planning interface. Finally, a language model renders grounded symbolic actions as natural-language action names while preserving their count and order.

The following five listings retain the prompt bodies from the implementation. Numbered fields stand for runtime expressions, documented after each template; they are not extra instructions sent to the model. The goal-translation listing also documents the implementation's two input modes, an imperative language goal or an explicit final-state description. These are alternative interfaces, not inputs combined into one planning query.

See Listing 19 for lifted PDDL domain-translation prompt.

See Listing 20 for PDDL object and initial-state translation prompt.

See Listing 21 for PDDL goal-translation prompt and supported input modes.

See Listing 22 for PDDL problem serialization prompt.

See Listing 23 for symbolic-plan to natural-language rendering prompt.

A.6 Planner Training Details

Both DPK and Finetuned VLM-as-Planner use Qwen3-VL-8B-Instruct loaded in 4-bit precision, with a maximum sequence length of 16,384 tokens. We use Unsloth with distributed data parallel training on three NVIDIA L40S GPUs. Both models are trained for a fixed two epochs, taking approximately two hours.

We use LoRA with rank 16, scaling parameter 16, dropout 0.05, and no trainable bias terms. The configuration enables adaptation of attention and MLP modules in the language and vision components. DPK's planner training examples are text-only, whereas the baseline consumes image observations. Enabling an adapter family in the configuration does not imply that a text-only example provides visual supervision. Gradient checkpointing reduces activation memory.

Optimization uses 8-bit AdamW with learning rate 10^{-4}, weight decay 0.001, a cosine learning-rate schedule, and a warmup ratio of 0.03. The per-device batch size is one with eight gradient-accumulation steps, giving an effective batch size of 24 across three devices. We clip the gradient norm at 0.5. The training seed is 42 and the LoRA initialization seed is 3407.

The two planners differ in their conditioning and targets rather than in the base model: DPK receives a belief state, goal, and action preconditions/effects and learns interleaved action–belief trajectories;

Listing 19: Lifted PDDL domain-translation prompt.

```
You are implementing the GPT-only Full Classical Symbolic Planner Baseline for
a robot task-planning dataset.

Convert the complete natural-language action catalog into a textbook lifted
classical PDDL domain. The output must be a reusable lifted domain: action
schemas have parameters, predicates have arguments, and repeated grounded
actions are abstracted into parameterized schemas. Do not create a grounded
zero-parameter action for every natural-language action, and do not preserve a
one-to-one action mapping.

PDDL constraints:
- Domain name must be exactly `{INPUT_1}`.
- Use only classical STRIPS plus negative preconditions:
  `(:requirements :strips :negative-preconditions)`.
- Use snake_case names for predicates, action schemas, variables, and objects.
- Prefer untyped PDDL; if type/class constraints are needed, express them as
  ordinary predicates and include their facts in each problem init.
- Encode negative preconditions/effects with `(not (...))`, not separate
  negative predicates.
- Every natural-language action's preconditions and effects must be covered by
  one or more lifted action schemas.

Natural-language action catalog:
{INPUT_2}

Return only a fenced JSON object with these keys:
- mode: "lifted"
- domain_name
- domain_pddl
- object_policy: short description of how problem objects should be inferred
- predicate_signatures: list of objects with name, parameters, arity, description
- action_schemas: list of objects with name, parameters, preconditions, effects,
    description
- natural_action_coverage: list of objects with natural_action_name, covered_by_schemas,
     notes

Runtime fields (source expressions):
INPUT_1: domain_name
INPUT_2: _json_blob(natural_language_actions)
```

the baseline receives an image, goal, action names with execution metadata, and action history and learns action sequences without intermediate belief-state targets.

Listing 20: PDDL object and initial-state translation prompt.

```
Convert the natural-language belief state into a lifted PDDL problem object list
and initial state for this GPT-generated lifted domain.

Rules:
- Use only predicates from the generated domain artifact.
- Infer all problem objects needed for the task instance from the belief state,
  the domain's object policy, and the natural-language action catalog.
- Problem object names must not be identical to any domain predicate name,
  action schema name, or the domain name. These names are reserved:
  {INPUT_1}
- If a physical entity naturally has a reserved name, rename only the object by
  adding a suffix such as `_obj`, e.g. use `faucet_obj` with `(faucet faucet_obj)`.
  Use the renamed object consistently in every atom.
- Include positive static/classification facts in init_positive when the domain
  uses predicates such as `container`, `location`, `bottle`, or similar.
- Convert positive natural-language facts into positive PDDL atoms.
- Convert "It is not that X" facts into init_negative atoms; these are recorded
  for auditing and should not appear as negative facts in the PDDL :init block.
- Do not use the ground-truth action sequence or future states.

Generated domain artifact:
{INPUT_2}

Natural-language belief_state.json:
{INPUT_3}

Return only a fenced JSON object with these keys:
- problem_objects: list of object names
- init_positive: list of positive PDDL atoms, e.g. "(at kettle sink)"
- init_negative: list of atoms known false, e.g. "(contains_water kettle)"
- unmapped: list of natural-language belief facts that could not be represented
- rationale

Runtime fields (source expressions):
INPUT_1: reserved_problem_object_names(domain_artifact)
INPUT_2: _json_blob({k: v for k, v in domain_artifact.items() if k != 'raw_response'})
INPUT_3: _json_blob(natural_belief_state)
```

Listing 21: PDDL goal-translation prompt and supported input modes.

```
Convert the provided natural-language goal specification into lifted PDDL goal
literals for this GPT-generated lifted domain.

Rules:
- Use only predicates from the generated domain artifact.
- Use only the provided problem objects.
- Do not introduce new predicates or new objects.
- Return positive goal atoms in goal_positive.
- Return atoms that must be false in goal_negative; the problem generator will
  write these as `(not atom)` in the PDDL goal.
- Goal source: {INPUT_1}
{INPUT_2}
- Some final-state sentences can describe derived/closure facts. If a fluent is
  not directly achievable in the generated domain, express the goal with
  achievable supporting predicates instead of requiring the unreachable fluent.
- Do not use ground-truth action names or intermediate states.

Generated domain artifact:
{INPUT_3}

Problem objects:
{INPUT_4}

Initial positive atoms:
{INPUT_5}

Natural-language initial belief state:
{INPUT_6}

{INPUT_7}:
{INPUT_8}

Return only a fenced JSON object with keys goal_positive, goal_negative, rationale.

Runtime fields (source expressions):
INPUT_1: goal_source
INPUT_2: goal_description.strip()
INPUT_3: _json_blob({k: v for k, v in domain_artifact.items() if k != 'raw_response'})
INPUT_4: _json_blob(list(problem_objects))
INPUT_5: _json_blob(list(init_positive))
INPUT_6: _json_blob(natural_belief_state)
INPUT_7: goal_label
INPUT_8: _json_blob(natural_goal_specification)

Goal-description alternatives supported by the implementation:
{
  "language_goal": {
    "goal_description": "\nThe goal specification is an imperative natural-language
     task instruction from\ngoal_from_action_sequence.json. Convert the intended final
     condition of that\ninstruction into symbolic goal literals. Encode what should be
     true/false in the\nfinal state, not the action sequence required to get there.\n",
    "goal_label": "Natural-language imperative goal instruction"
  },
  "final_state": {
    "goal_description": "\nThe goal specification is a natural-language final state
     from the last\nstate-action pair in state_action_sequence.json. Encode the facts
     that should\nhold in that final state.\n",
    "goal_label": "Natural-language final goal state"
  }
}
```

Listing 22: PDDL problem serialization prompt.

```
Generate a valid PDDL problem file for this GPT-generated lifted PDDL domain.

Rules:
- Problem name must be exactly `{INPUT_1}`.
- Domain name must be exactly `{INPUT_2}`.
- Use exactly the provided problem_objects in the PDDL `:objects` section.
- The object names have already been chosen to avoid domain predicate/action
  name conflicts. Preserve them exactly.
- `:init` must contain init_positive atoms only. Do not write negative init facts;
  init_negative is provided only for audit and closed-world reasoning.
- `:goal` must be a conjunction of goal_positive atoms plus `(not atom)` for
  each goal_negative atom.
- Do not add, remove, rename, or rewrite init/goal atoms.

Generated domain PDDL:
```pddl
{INPUT_3}
```

problem_objects:
{INPUT_4}

init_positive:
{INPUT_5}

init_negative:
{INPUT_6}

goal_positive:
{INPUT_7}

goal_negative:
{INPUT_8}

Return only a fenced JSON object with keys:
- problem_pddl
- problem_objects
- init_positive
- goal_positive
- goal_negative

Runtime fields (source expressions):
INPUT_1: problem_name
INPUT_2: domain_artifact['domain_name']
INPUT_3: domain_artifact['domain_pddl']
INPUT_4: _json_blob(list(problem_objects))
INPUT_5: _json_blob(list(init_positive))
INPUT_6: _json_blob(list(init_negative))
INPUT_7: _json_blob(list(goal_positive))
INPUT_8: _json_blob(list(goal_negative))
```

Listing 23: Symbolic-plan to natural-language rendering prompt.

```
Render a symbolic lifted planner result into natural-language action names for
the DPK LLM evaluation grader.

Rules:
- Input symbolic actions are grounded action instances from the generated PDDL
  domain.
- Convert each symbolic action instance into one concise natural-language action
  description.
- Prefer wording from the natural-language action catalog when a symbolic step
  clearly corresponds to one of those actions.
- The sequence length should match the symbolic action sequence length.
- Do not add explanatory text to the action list.

Natural-language goal state summary:
{INPUT_1}

Generated domain action schema metadata:
{INPUT_2}

Natural-language action catalog:
{INPUT_3}

Symbolic action records:
{INPUT_4}

Return only a fenced JSON object with keys:
- natural_action_sequence: list of strings
- rationale

Runtime fields (source expressions):
INPUT_1: language_goal
INPUT_2: _json_blob(domain_artifact.get('action_schemas', []))
INPUT_3: _json_blob(natural_language_actions)
INPUT_4: _json_blob(list(symbolic_action_records))
```

A.7 High-Level Action Sequences and Evaluation Splits

Table A2 summarizes the dataset and domain model statistics.

Figures A1–A3 show the high-level action sequences in our three task domains. Each sequence corresponds to multiple human egocentric videos. Orange marks sequences withheld for *New Action Sequence*, and blue marks sequences withheld for *New Strategy*; the two highlighted sets are disjoint. For each highlighted sequence, all corresponding videos are withheld for testing in the indicated evaluation setting. Black denotes the remaining sequences. Arrows indicate the order of high-level actions.

Table A2: **Dataset and domain model statistics.** Action counts per video refer to top-level actions.

	Boil Water	Make Instant Noodles	Clean Up Kitchen
Videos	90	64	32
Total duration (min)	66.52	106.67	79.53
Mean duration (s/video)	44.35	100.00	149.12
Contact annotation units (total)	966	1,285	698
Contact annotation units (mean/video)	10.73	20.08	21.81
High-level actions per video	3–10	5–10	5–15
Domain actions / predicates	42 / 40	33 / 41	27 / 29
Planning examples	2,737	2,601	3,055

Boil Water

22 sequences

New Action Sequence **New Strategy** Other sequences

1.	Pick up electric kettle → open kettle lid → turn on faucet → fill electric kettle with water → turn off faucet → place on stand → close kettle lid → turn on electric kettle
2.	Pick up electric kettle → open kettle lid → turn on faucet → fill electric kettle with water → turn off faucet → place on stand → close kettle lid → plug in electric kettle cable → turn on electric kettle
3.	Pick up electric kettle → turn on faucet → fill electric kettle with water → turn off faucet → place on stand → close kettle lid → turn on electric kettle
4.	Pick up electric kettle → turn on faucet → fill electric kettle with water → turn off faucet → place on stand → close kettle lid → plug in electric kettle cable → turn on electric kettle
5.	Open kettle lid → pick up water bottle → pour water into electric kettle → close kettle lid → plug in electric kettle cable → turn on electric kettle
6.	pick up water bottle → pour water into electric kettle → close kettle lid → turn on electric kettle
7.	Open kettle lid → check water → close kettle lid → turn on the electric kettle
8.	Open kettle lid → check water → close kettle lid → plug in electric kettle cable → turn on electric kettle
9.	Check water → close kettle lid → turn on electric kettle
10.	Check water → close kettle lid → plug in electric kettle cable → turn on electric kettle
11.	Pick up pot → open pot lid → turn on faucet → fill pot with water → close pot lid → turn on stove
12.	Pick up pot → turn on faucet → fill pot with water → close pot lid → turn on stove
13.	Open pot lid → pick up water bottle → pour water into pot → close pot lid → turn on stove
14.	Pick up water bottle → pour water into pot → close pot lid → turn on stove
15.	Open pot lid → close pot lid → turn on stove
16.	Close pot lid → turn on stove
17.	Pick up mug → turn on faucet → fill mug with water → open microwave door → place mug in microwave → turn on microwave
18.	Pick up mug → turn on faucet → fill mug with water → open microwave door → remove bowl from microwave → place mug in microwave → turn on microwave
19.	Pick up water bottle → pour water into mug → open microwave door → place mug in microwave → turn on microwave
20.	Pick up water bottle → pour water into mug → open microwave door → remove bowl from microwave → place mug in microwave → turn on microwave
21.	Pick up mug → open microwave door → place mug in microwave → turn on microwave
22.	Pick up mug → open microwave door → place mug in microwave → remove bowl from microwave → turn on microwave

Figure A1: **High-level action sequences in Boil Water.** The 22 sequences cover alternative procedures using a kettle, pot, or microwave. Orange marks sequences withheld for *New Action Sequence* (2, 9, 10, 14, 18, 22); blue marks sequences withheld for *New Strategy* (5, 6, 11, 12, 19, 20). Each colored sequence represents all of its corresponding videos.

Make Instant Noodles

16 sequences

New Action Sequence **New Strategy** Other sequences

1.	Open cup noodle lid → move sausage slices from cutting board to cup noodle → check kettle water temperature and finding hot → pour hot water from kettle into cup noodle → close cup noodle lid
2.	Open cup noodle lid → move sausage slices from cutting board to cup noodle → check kettle water temperature and finding NOT hot → turn on kettle → wait the water to be hot → pour hot water from kettle into cup noodle → close cup noodle lid
3.	Open cup noodle lid → move egg from pan into cup noodle using spatula → check kettle water temperature and finding hot → pour hot water from kettle into cup noodle → close cup noodle lid
4.	Open cup noodle lid → move egg from pan into cup noodle using spatula → check kettle water temperature and finding NOT hot → turn on kettle → wait the water to be hot → pour hot water from kettle into cup noodle → close cup noodle lid
5.	Open cup noodle lid → move sausage slices from cutting board to cup noodle → check kettle water temperature and finding NOT hot → pour cold water from kettle into cup noodle → open microwave door → move cup noodle into microwave → close microwave door → turn on microwave
6.	Open cup noodle lid → move egg from pan into cup noodle using spatula → check kettle water temperature and finding NOT hot → pour cold water from kettle into cup noodle → open microwave door → move cup noodle into microwave → close microwave door → turn on microwave
7.	Open noodle bag → move noodle block from noodle bag to bowl → move sausage slices from cutting board to bowl → check kettle water temperature and finding hot → pour hot water from kettle into bowl → cover bowl with tool
8.	Open noodle bag → move noodle block from noodle bag to bowl → move sausage slices from cutting board to bowl → check kettle water temperature and finding NOT hot → turn on kettle → wait the water to be hot → pour hot water from kettle into bowl → cover bowl with tool
9.	Open noodle bag → move noodle block from noodle bag to bowl → move egg from pan into bowl using spatula → check kettle water temperature and finding hot → pour hot water from kettle into bowl → cover bowl with tool
10.	Open noodle bag → move noodle block from noodle bag to bowl → move egg from pan into bowl using spatula → check kettle water temperature and finding NOT hot → turn on kettle → wait the water to be hot → pour hot water from kettle into bowl → cover bowl with tool
11.	Open noodle bag → move noodle block from noodle bag to bowl → move sausage slices from cutting board to bowl → check kettle water temperature and finding NOT hot → pour cold water from kettle into bowl → open microwave door → move bowl into microwave → close microwave door → turn on microwave
12.	Open noodle bag → move noodle block from noodle bag to bowl → move egg from pan into bowl using spatula → check kettle water temperature and finding NOT hot → pour cold water from kettle into bowl → open microwave door → move bowl into microwave → close microwave door → turn on microwave
13.	Check pot water temperature and finding hot → open noodle bag → move noodle block from noodle bag to pot → move sausage slices from cutting board to pot → close pot lid
14.	Check pot water temperature and finding hot → open noodle bag → move noodle block from noodle bag to pot → move egg from pan into pot using spatula → close pot lid
15.	Check pot water temperature and finding NOT hot → open noodle bag → move noodle block from noodle bag to pot → move sausage slices from cutting board to pot → move pot from counter to stove/induction cooker → close pot lid → turn on stove/induction cooker
16.	Check pot water temperature and finding NOT hot → open noodle bag → move noodle block from noodle bag to pot → move egg from pan into pot using spatula → move pot from counter to stove/induction cooker → close pot lid → turn on stove/induction cooker

Figure A2: **High-level action sequences in Make Instant Noodles.** The 16 sequences vary the container, ingredients, water temperature, and heating procedure. Orange marks sequences withheld for *New Action Sequence* (4, 9, 12, 15); blue marks sequences withheld for *New Strategy* (5, 6, 13, 14). Each colored sequence represents all of its corresponding videos.

Clean Up Kitchen

8 sequences

New Action Sequence **New Strategy** Other sequences

1.	Open food container and pour food from paper plate into container → close food container → open fridge door and place food container in fridge → close fridge door → discard paper plate in trash can.
2.	Open fridge door and place ketchup bottle in fridge → close fridge door → move ceramic plate into sink → apply dish soap to ceramic plate → turn on faucet → wash ceramic plate with sponge → turn off faucet → move ceramic plate to dish rack.
3.	Open food container and pour food from paper plate into container → close food container → open fridge door and place food container in fridge → place ketchup bottle in the already-open fridge → close fridge door → discard paper plate in trash can → take tissue and clean countertop with tissue → discard tissue in trash can.
4.	Open food container and pour food from ceramic plate into container → close food container → open fridge door and place food container in fridge → close fridge door → move ceramic plate into sink → apply dish soap to ceramic plate → turn on faucet → wash ceramic plate with sponge → turn off faucet → move ceramic plate to dish rack → take tissue and clean countertop with tissue → discard tissue in trash can.
5.	Open food container and pour food from ceramic plate into container → close food container → open fridge door and place food container in fridge → place ketchup bottle in the already-open fridge → close fridge door → move ceramic plate into sink → apply dish soap to ceramic plate → turn on faucet → wash ceramic plate with sponge → turn off faucet → move ceramic plate to dish rack.
6.	Open food container and pour food from paper plate into container → pour food from ceramic plate into the same container → close food container → open fridge door and place food container in fridge → close fridge door → discard paper plate in trash can → move ceramic plate into sink → apply dish soap to ceramic plate → turn on faucet → wash ceramic plate with sponge → turn off faucet → move ceramic plate to dish rack.
7.	Open fridge door and place ketchup bottle in fridge → close fridge door → discard paper plate in trash can → move ceramic plate into sink → apply dish soap to ceramic plate → turn on faucet → wash ceramic plate with sponge → turn off faucet → move ceramic plate to dish rack → take tissue and clean countertop with tissue → discard tissue in trash can.
8.	Open food container and pour food from paper plate into container → pour food from ceramic plate into the same container → close food container → open fridge door and place food container in fridge → place ketchup bottle in the already-open fridge → close fridge door → discard paper plate in trash can → move ceramic plate into sink → apply dish soap to ceramic plate → turn on faucet → wash ceramic plate with sponge → turn off faucet → move ceramic plate to dish rack → take tissue and clean countertop with tissue → discard tissue in trash can.

Figure A3: **High-level action sequences in Clean Up Kitchen.** The eight sequences combine food storage, waste disposal, dishwashing, and countertop cleaning. Orange marks sequences withheld for *New Action Sequence* (4, 6); blue marks sequences withheld for *New Strategy* (3, 5, 8). Each colored sequence represents all of its corresponding videos.

B Additional Qualitative Results

We examine twelve paired online human-assistance episodes, four from each task domain, to show how planning decisions affect physical execution. Each pair shares the language goal, initial task conditions, and reference action sequence; the physical layouts can differ between runs. The baseline is Finetuned VLM-as-Planner. We select episodes in which DPK completes the task without intervention and the baseline requires corrections, drawing from both *New Action Sequence* and *New Strategy*. These diagnostic examples illustrate specific strengths and failure modes; they do not estimate their frequency. The aggregate online evaluation in the main text provides that broader comparison.

Reading the figures. Each panel shows six selected observations and actions from each planner. The labels s_i retain the original step indices; arrows connect consecutive steps, and ellipses indicate omitted steps. DPK's predicate strips show selected pre-action belief values: T (true), F (false), and ? (unknown). A plus sign joins the components of a bimanual action. Red frames/actions mark interventions, with abbreviated human feedback below. "Intv. / shown" distinguishes the total interventions in the episode from those displayed. All DPK episodes have zero interventions. The observations are original recorded images, and the action, predicate, and feedback text is abbreviated for readability.

B.1 Boil Water: Establishing Preconditions Before Acting

Figure A4 shows that producing a plausible sequence of familiar actions is insufficient: the next action must match both the physical state and what the assistant knows about it. In panel (a), DPK checks the kettle, moves it to the sink, and turns on the faucet before filling. The baseline attempts to fill the kettle while it is still on the counter and the faucet is off, and later repeats an already completed placement on the heating base. In (b), DPK checks microwave availability and removes the occupying bowl; the baseline attempts insertion before establishing that the microwave is accessible and empty. Its earlier filling attempt also precedes turning on the faucet.

In (c), DPK inspects the kettle and removes the bottle cap before pouring, then prepares the kettle for heating. The baseline attempts pouring before inspection and uncapping, and later attempts heating before placing the kettle on its base and closing its lid. Panel (d) extends the comparison to boiling water in a pot. DPK checks whether the pot contains water, fills it at the sink, moves it to the stove, and closes the lid before reaching the terminal state. The baseline instead declares completion at s_6 while the pot lid is still on the counter; human intervention is needed before it closes the lid at s_7. This contrast exposes a failure to satisfy a remaining task requirement, not merely an inefficient action order. Across all four episodes, DPK establishes action prerequisites and completes the required procedure without corrective intervention. These examples illustrate how the planner's use of the language-based abstract world model and current beliefs supports both feasible action selection and completion of the task, rather than merely producing a plausible action sequence.

B.2 Make Instant Noodles: Using Observations to Choose Feasible Procedures

Figure A5 illustrates how information gathering changes the subsequent plan. In (a), DPK checks the water temperature, observes that it is not hot, and heats it before pouring. In (d), the corresponding check finds hot water, so DPK pours it directly instead. The baseline initiates heating without checking in these episodes, including unnecessary heating in (d). The contrast is not simply that DPK inserts more inspections: it uses their outcomes to decide which actions are needed.

DPK also adapts the procedure to available tools and ingredients. In (b), it transfers the fried egg directly from the pan when no spatula is available; the baseline selects the missing spatula and later pours without checking the water. In (c) and (d), DPK uses the available sausage, whereas the baseline proposes an unavailable fried egg. The baseline also leaves the noodle bag open when the task requires closing it. These cases show the practical value of selecting actions against the current belief state: DPK combines information gathering, feasible alternatives, and goal requirements into procedures that complete without human correction. The cases illustrate state-conditioned adaptation, rather than attributing the outcome to any single component in isolation.

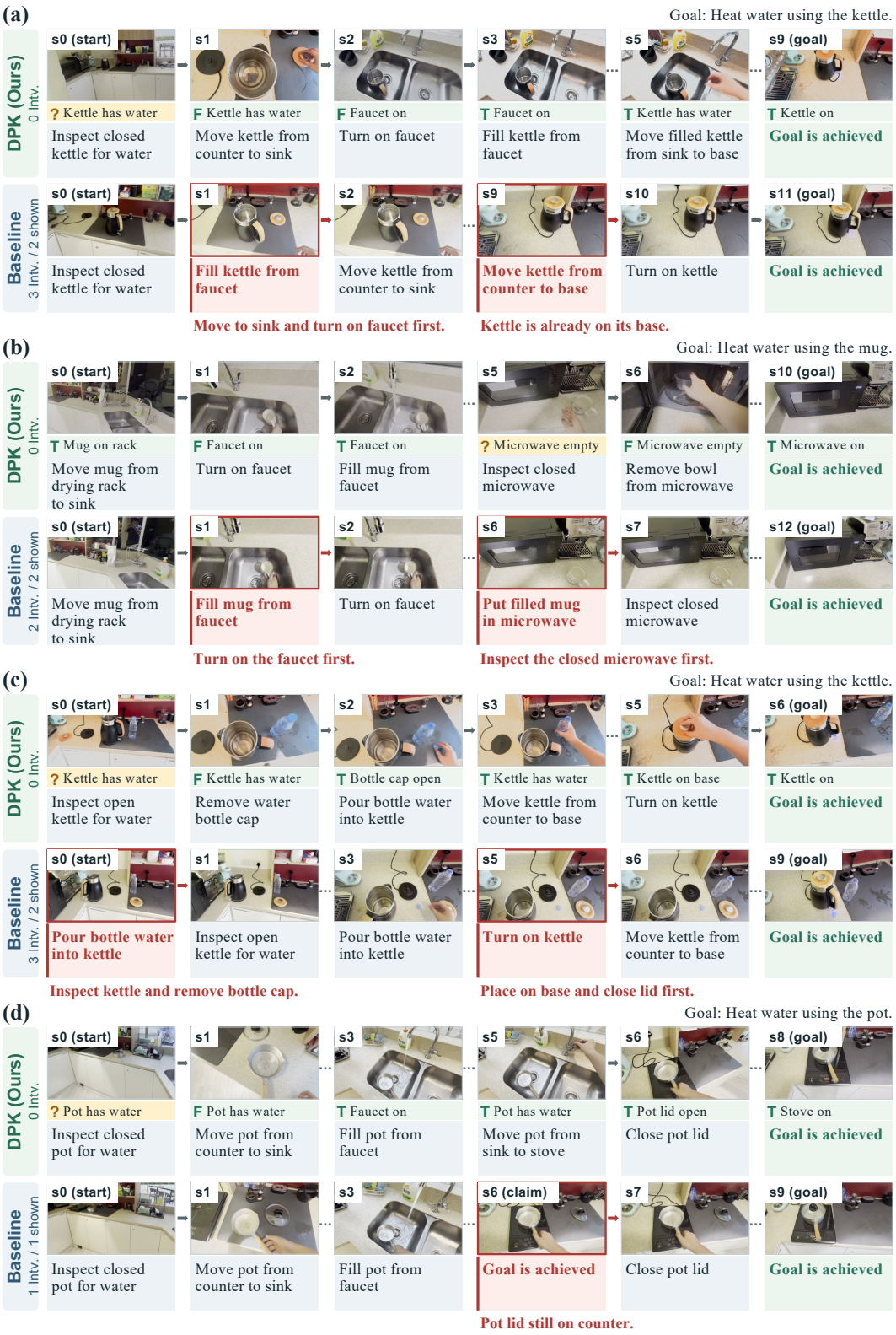


Figure A4: **Qualitative online human assistance in Boil Water.** (a,b) New Action Sequence; (c,d) New Strategy. DPK establishes action prerequisites and checks unknown conditions. All DPK episodes require zero interventions. Notation and selection protocol are defined in Appendix B.

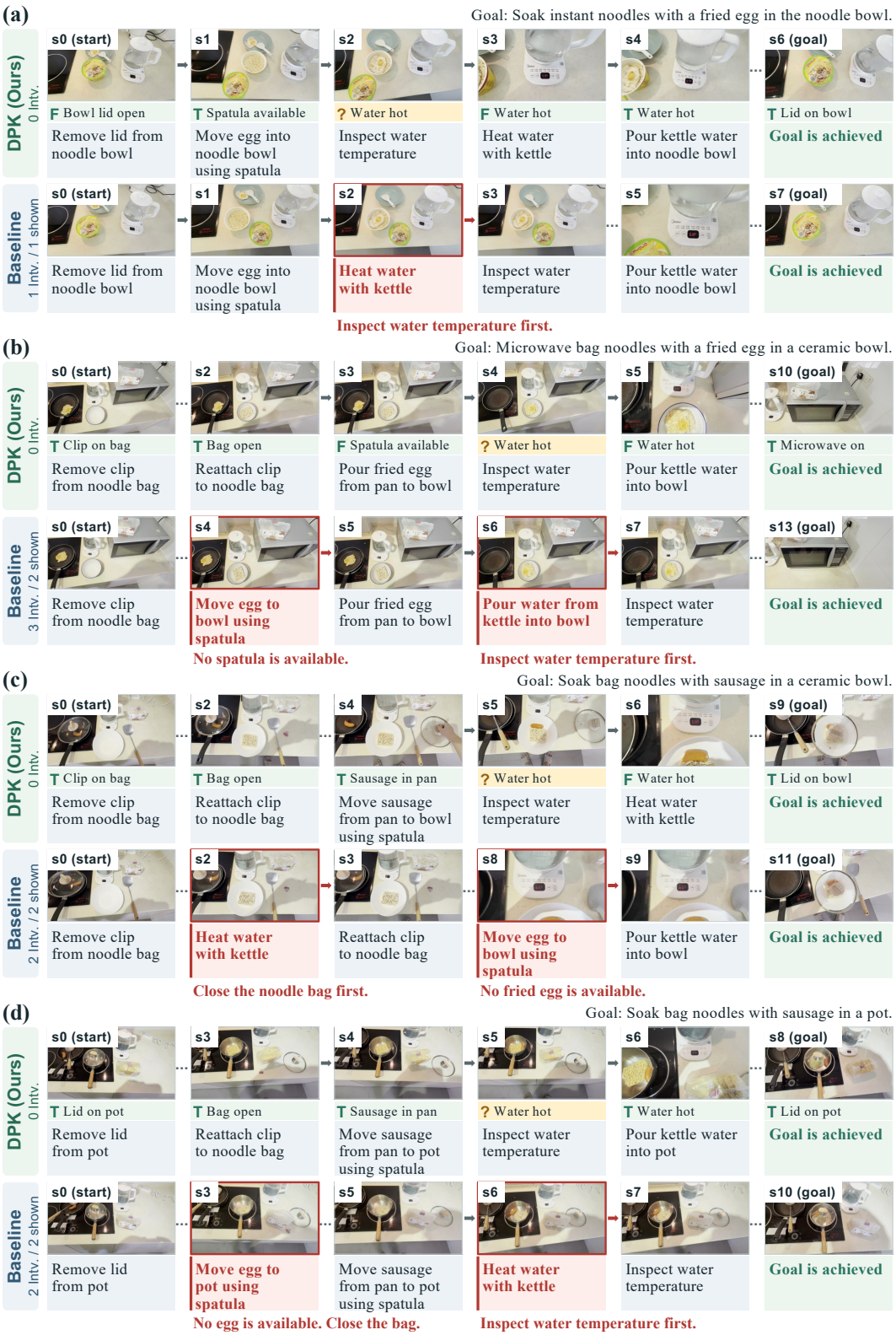


Figure A5: **Qualitative online human assistance in Make Instant Noodles.** (a,b) New Action Sequence; (c,d) New Strategy. DPK adapts preparation to water temperature and available ingredients/tools, requiring zero interventions. Notation follows Appendix B.

B.3 Clean Up Kitchen: Respecting Coordination and Tracking Completed Effects

Figure A6 shows why bimanual planning requires more than assigning one action to each hand. The combined action must respect access conditions and dependencies between its components. DPK selects valid combinations, such as opening a container while transferring food or opening the refrigerator while inserting an object. The baseline instead proposes pouring into a closed container while opening it in (b,c), closing the refrigerator while inserting an object in (c), and inserting an object before opening the refrigerator in (d). These four interventions expose specific coordination failures. Panel (a) illustrates a distinct constraint: a standalone opening action fails the goal's instruction to use both hands when feasible; the action is not physically impossible in isolation.

The same episodes also reveal the importance of retaining completed effects. The baseline repeats soap application in (a,b) and proposes discarding a paper plate that is already in the trash in (d). DPK's displayed beliefs track these conditions and support moving on to the remaining work. Together, the cases illustrate two complementary requirements of reusable planning knowledge: actions must compose into feasible coordinated behavior, and their consequences must inform later choices. DPK satisfies these requirements in all four selected episodes without intervention, linking the abstract world model's preconditions, effects, and execution constraints, and the planner's use of current beliefs to successful closed-loop assistance.

B.4 Failure Analysis

We observe two recurring failure modes in visual grounding. First, our default object detector, DINO-X (Ren et al., 2024), can miss task-relevant objects, such as a water bottle present in the scene. Predicates associated with a missed object cannot be evaluated by the predicate classifier, leaving them absent or outdated in the belief state. Second, predicate classification errors can assign incorrect values during belief update. Since both goal-conditioned action selection and state-transition prediction operate on the belief state, these errors can lead to infeasible actions and predictions inconsistent with the physical scene. These failures identify accurate visual grounding as a remaining requirement: tolerance of incomplete abstract world models does not eliminate the need for a reliable belief state.

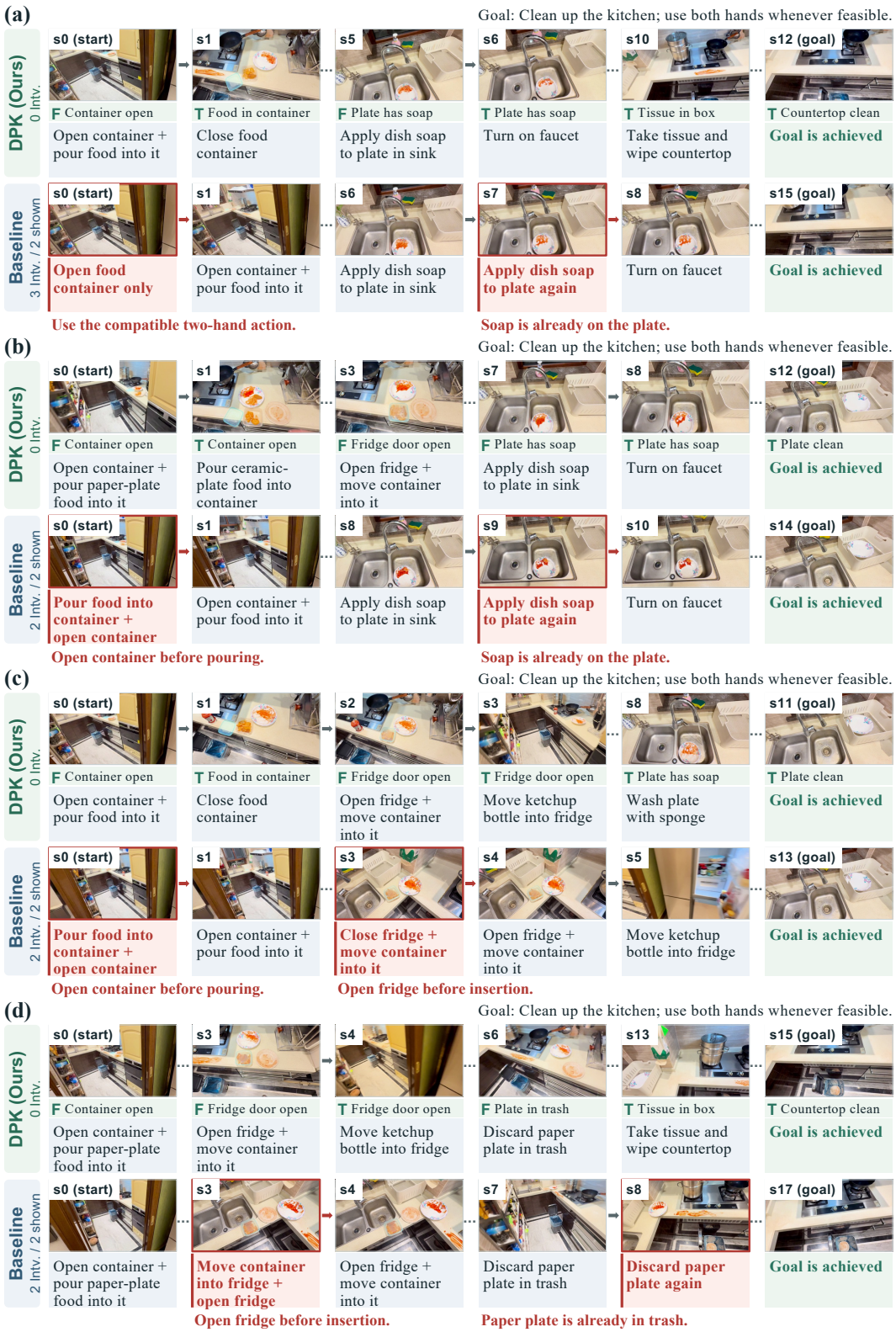


Figure A6: **Qualitative online human assistance in Clean Up Kitchen.** (a,b) New Action Sequence; (c,d) New Strategy. DPK respects coordination and completed effects without intervention. Four baseline bimanual-composition failures appear across (b–d). Notation follows Appendix B.

B.5 Temporal Segmentation without Contact Annotations

To illustrate the extraction errors discussed in the main text's contact-annotation ablation, Table A3 compares semantic segmentation of the same Clean Up Kitchen video with and without contact annotations. With annotations, the sequence contains five top-level semantic segments. Without annotations, two pairs of neighboring segments are merged, leaving three.

Table A3: **Over-grouping without contact annotations.** Paired segmentation outputs for the same Clean Up Kitchen video. Each numbered entry is one top-level semantic segment; timestamps (MM:SS) retain the recorded start and end boundaries. Action descriptions are shortened for readability.

With contact annotations	Without contact annotations
1. 00:00–00:14.45: Open the food container and pour food from the paper plate into it. **2.** 00:16.25–00:24.2: Close the food container.	**1.** 00:00–00:24.2: Transfer food from the paper plate into the food container and close it.
3. 00:25–00:38.175: Open the fridge door and move the food container from the dining table into the fridge. **4.** 00:38.175–00:45.625: Close the fridge door.	**2.** 00:25–00:45.625: Store the food container in the fridge and close the fridge door.
5. 00:45.625–00:59.7: Throw the paper plate from the dining table into the trash can.	**3.** 00:45.625–00:59.7: Throw the paper plate from the dining table into the trash can (unchanged).

Where over-grouping occurs. In the first merged segment, the left-hand component becomes "Open and close the food container," although its primitive sequence contains two separate contact episodes: the hand releases the lid after opening, then contacts it again to close the container. The second merge similarly combines opening and closing the fridge door in one right-hand component, despite a release and renewed contact. The plate-disposal segment remains unchanged. Thus, the issue is not bimanual grouping itself: the annotation-guided output already groups opening with pouring, and fridge opening with container placement. The error is absorbing a subsequent closing action into the preceding segment.

Implication for abstract world model induction. These merges remove separately represented closing actions and the intermediate boundaries at which their preconditions and effects would be associated with the demonstration. They illustrate how omitting contact annotations can alter the action granularity used to induce language-based abstract world models and reconstruct state–action supervision. This example makes the extraction noise concrete; the aggregate ablation in the main text evaluates the planner's ability to use the resulting imperfect abstract world models. The paired outputs here illustrate over-grouping, not oversegmentation or its frequency.

C EXTENDED LISTINGS

The following listings collect the longer prompts and complete catalogs, in the order in which they are referenced in Appendix A. Each listing links back to its corresponding description.

Return to the corresponding description.

Listing 24: Low-level contact-segmentation prompt.

```
You perform low-level contact segmentation for an egocentric video. Manual
contact annotations are authoritative for which hand/object and object/object
contacts exist and for their temporal envelopes. The video is authoritative
for visible motion, primitive phase, and neutral description. Never override,
invent, rename, merge, or omit an annotated contact.

### Closed Object Vocabulary

Use object identities exactly as written below in every structured field and
descriptive field. Do not use synonyms, subtype guesses, state adjectives, or
names absent from this annotation-derived vocabulary.

```json
{object_vocabulary}
```

Core Rules

1. Produce independent `left_hand` and `right_hand` timelines. A video with
   only one active hand still contains the other hand as `move_free`.
2. For each hand, the union of every non-`move_free` primitive must exactly
   cover its manual hand-object annotations. A non-contact interval is
   `move_free`, including egocentric body/camera travel while that hand is
   outside the frame and annotation confirms no contact.
3. Use `make_contact(hand, object)` and `break_contact(hand, object)` as
   zero-duration boundary events; use `move_object(hand, object)` for the
   positive-duration interior of direct manipulation contact.
4. Use tool primitives only when the hand-held object and target-object
   contact are both supported by the annotations:
   `make_contact_tool(hand, tool, target)`,
   `move_object_tool(hand, tool, target)`, and
   `break_contact_tool(hand, tool, target)`.
5. Contact phases may share endpoints, but positive-duration primitives for
   one hand must not overlap. Endpoint equality is not overlap.
6. Preserve concurrent contacts independently. Do not drop one contact merely
   because the same hand has another annotated contact with the same envelope.
7. `object` and `target_object` must match primitive arguments exactly.
   `move_free` uses empty strings for both fields.
8. Use timestamps in `MM:SS` or `MM:SS.sss`. Do not infer contact before or
   after the annotation envelope.

Output Schema

```json
{{
  "left_hand": [
    {{
      "hand": "left",
      "start_time": "MM:SS",
      "end_time": "MM:SS",
      "primitive_type": "move_free(left) | make_contact(left, object) | move_object(
    left, object) | break_contact(left, object) | make_contact_tool(left, tool, target)
     | move_object_tool(left, tool, target) | break_contact_tool(left, tool, target)",
      "object": "exact vocabulary name or empty string",
      "target_object": "exact vocabulary name or empty string",
      "summary": "neutral concise contact description",
      "detailed_description": {{
        "action_what": "neutral physical description grounded in video and annotation"
      }}
    }}
  ],
  "right_hand": []
}}
```
```

```
### Your Step-by-Step Analysis Process

For every step, construct the result, review it against the step gate,
self-correct every failure, and proceed only after the gate passes.

Step 1. Audit the video, both manual-contact streams, and the closed vocabulary.
Gate: every annotation and both hands are accounted for with exact identities.

Step 2. Build the left-hand contact and hand-free timeline.
Gate: exact annotation coverage, chronological order, and no unsupported contact.

Step 3. Independently build the right-hand timeline.
Gate: the same checks pass without copying the other hand's structure.

Step 4. Select direct versus tool primitive phases from annotation and visible
motion. Gate: every tool-target primitive has both required contact evidences.

Step 5. Audit exact primitive arguments, object fields, timestamps, endpoint
rules, full hand-free complements, and complete coverage. Regenerate any
failing record, then emit exactly one final fenced JSON object as the last
content in the response.

### Optional User-Provided Domain Examples

These are explicit illustrations only. They are not an exhaustive catalog and
must never override the video or annotations.

'''json
{domain_examples}
'''

### Manual Hand-Object Contact Annotations
'''json
{hand_object_contact_annotation}
'''

### Manual Object-Object Contact Annotations
'''json
{object_object_contact_annotation}
'''
```

Return to the corresponding description.

Listing 25: High-level semantic-segmentation prompt.

```
You perform high-level typed-bimanual semantic action segmentation for an
egocentric video. First construct complete per-hand unimanual candidates, then
reason about cross-hand overlap, then replace qualifying overlapping pairs by
typed bimanual parents. Never create an untyped `hands: both` action.

### Authoritative Inputs

The low-level primitives and compiled contact episodes constrain hand-contact
membership. Manual annotations constrain contact and object relations. The
video constrains task outcome, direction, purpose, approach, and withdrawal.
Object identities must be copied exactly from the annotation-derived closed
vocabulary. Optional examples are illustrative only and cannot override these
inputs.

Compiled `relation_transition_candidates` are mechanically enumerated changes
in object-object relations near a contacted episode. They are hypotheses for
video verification, not asserted transfers. Reject a candidate when the shared
object is merely a support, access component, tool target, or coincident scene
relation rather than the item/material actually transferred by that hand.

### Complete Unimanual Candidate Definition

1. Construct left and right independently before comparing hands.
2. Exactly one compiled contact episode maps to exactly one unimanual semantic
   action. Do not split acquisition, use, and release into separate actions.
3. A candidate begins with a preparatory `move_free(hand)`, contains exactly
   one continuous positive-contact episode, and ends with cleanup
```

```
   `move_free(hand)`. No internal `move_free` is allowed between contact phases.
4. The nested candidate's timestamps are grounded independently in its own
   approach and withdrawal evidence. Never copy a sibling or future parent
   boundary. Equal boundaries are permitted only when independently supported.
5. Candidates for the same hand are chronological and never overlap. If their
   hand-free margins collide, choose a shared boundary inside the genuine
   hand-free interval while preserving both contact cores.
6. A one-hand video outputs only standalone actions for the active hand; never
   invent contact or a typed parent for the inactive hand.

### Semantic Outcome And Action Type

Name the concrete task outcome, not merely `holds`, `moves`, or `handles`, when
video or object-relation evidence establishes a state transition. For a
transfer, include the evidenced object, source, and destination. For a
direction-ambiguous mechanism, inspect state immediately before first contact
and immediately after release. Do not infer direction from the task context or
from undirected `move_object` alone.

If a continuous episode acquires an object or tool and then uses it before
release to create a visible task outcome, the summary must name that use or
outcome. An acquisition-only summary such as `pick up the tool` is invalid when
the same episode subsequently applies, dispenses, cuts, washes, fastens, opens,
or otherwise changes the task state. Acquisition-only wording is valid only
when acquisition itself is the complete evidenced outcome.

Use `information_gathering` only when the primary purpose is to reduce
uncertainty about a property or scene state. Use `manipulation` when the primary
purpose is to change the physical world. Opening or moving an object is not
automatically information gathering.

### Typed Bimanual Classification

Compute positive-duration intersection of the two candidates' task-active
contact cores. `end == start` is touching, not overlap. Free-only envelope
overlap is not a bimanual action.

For every qualifying left/right pair:

* `uncoordinated`: there exists a feasible serial order in which one hand can
  execute both complete actions and obtain the same relevant final state. A
  prerequisite ordering is compatible with this type. Determine the feasible
  semantic order; do not use observed onset order. Its rationale must include
  the explicit conclusion `can be executed sequentially with the same outcome`
  and then name the feasible order and why no simultaneous support is needed.
* `coordinated`: one action must remain physically active while the other runs,
  providing stabilization, support, alignment, counter-force, tension, load
  sharing, or another evidenced simultaneous constraint. Serial execution
  would fail, be unsafe, or change the relevant outcome. Its rationale must
  explicitly conclude that the components cannot be executed sequentially with
  the same outcome and identify the physical dependency.

Acting on the same object is not sufficient for `coordinated`; acting on
different objects is not sufficient for `uncoordinated`. Continued overlap or
convenience alone is not simultaneous dependency.

### Final Timeline Invariants

* Replace each qualifying overlapping pair by one typed parent. Nested actions
  remain inside that parent and are not duplicated as standalone entries.
* Parent `start_time` is the earlier nested start; parent `end_time` is the later
  nested end. Preserve each nested action's independently grounded times.
* Parent `action_type` is `information_gathering` if either nested action has
  that type; otherwise it is `manipulation`.
* Parent `contact_primitives.left_hand/right_hand` exactly copy the nested
  primitive sequences.
* An uncoordinated parent summary orders clauses by the feasible serial plan,
  even when observed onset order differs. A coordinated summary names the joint
  simultaneous outcome.
* Each coordinated nested summary names that hand's own physical contribution.
  Do not copy the joint parent summary into a supporting or stabilizing hand's
  nested action. A nested action may equal the parent wording only when that
  hand independently performs the entire stated outcome.
* The final top-level list contains only standalone unimanual actions and typed
  bimanual parents, sorted chronologically with zero overlap. Internal nested
```

```
  overlap inside a typed parent is legal and required.

### Exact Schemas

Standalone unimanual:
```json
{{
  "hands": "left",
  "start_time": "MM:SS",
  "end_time": "MM:SS",
  "action_type": "manipulation",
  "summary": "concise semantic outcome",
  "primary_object": "exact vocabulary object names",
  "contact_primitives": {{"left_hand": ["move_free(left)"]}},
  "detailed_analysis": {{
    "description": "neutral physical motion",
    "signal": "independent start and end evidence",
    "context": "purpose and sub-goal"
  }}
}}
```

Typed bimanual parent:
```json
{{
  "hands": "both",
  "bimanual_type": "uncoordinated",
  "start_time": "MM:SS",
  "end_time": "MM:SS",
  "action_type": "manipulation",
  "summary": "both outcomes in executable serial order or joint coordinated outcome",
  "primary_object": "main exact vocabulary object names used by both actions",
  "contact_primitives": {{
    "left_hand": ["complete left primitive sequence"],
    "right_hand": ["complete right primitive sequence"]
  }},
  "unimanual_actions": {{
    "left_hand": {{
      "hand": "left",
      "start_time": "MM:SS",
      "end_time": "MM:SS",
      "action_type": "manipulation",
      "summary": "complete left-hand outcome",
      "primary_object": "exact vocabulary object names",
      "contact_primitives": ["complete left primitive sequence"],
      "detailed_analysis": {{
        "description": "neutral physical motion",
        "signal": "independent left boundary evidence",
        "context": "purpose and sub-goal"
      }}
    }},
    "right_hand": {{
      "hand": "right",
      "start_time": "MM:SS",
      "end_time": "MM:SS",
      "action_type": "manipulation",
      "summary": "complete right-hand outcome",
      "primary_object": "exact vocabulary object names",
      "contact_primitives": ["complete right primitive sequence"],
      "detailed_analysis": {{
        "description": "neutral physical motion",
        "signal": "independent right boundary evidence",
        "context": "purpose and sub-goal"
      }}
    }}
  }},
  "detailed_analysis": {{
    "description": "neutral joint physical motion",
    "signal": "parent union boundary evidence",
    "context": "joint purpose and sub-goal"
  }},
  "bimanual_type_rationale": "explicit serialization or simultaneous-dependency test"
}}
```
```

```
### Generic Reference Cases

* If one hand opens a barrier that remains open and the other retrieves an
  object through it, classify as `uncoordinated` and phrase the parent in the
  feasible order `open barrier -> retrieve object`, regardless of which hand
  began approaching first.
* If one hand braces a workpiece while the other applies tool force, and the
  workpiece would move without the brace, classify as `coordinated`.
* If only one hand opens a cabinet primarily to determine whether a component
  is present, output one standalone `information_gathering` action.

### Your Step-by-Step Analysis Process

At every step: construct the complete intermediate result, review every rule
and evidence requirement for that step, identify failures, regenerate until all
checks pass, and only then proceed.

Step 1. Cross-check video, annotations, primitives, episodes, and vocabulary.
Step 1 gate: no missing evidence, renamed identity, or invented contact.

Step 2. Map every left episode to exactly one complete candidate and audit the
left timeline. Step 2 gate: count, primitive membership, boundaries, no internal
`move_free`, and same-hand zero overlap all pass.

Step 3. Independently repeat for the right hand. Step 3 gate: the same checks
pass without copying left-hand timestamps.

Step 4. Audit every candidate's concrete outcome, direction, source,
destination, primary intent, action type, and exact identities. Step 4 gate:
every claim is supported by video/annotation evidence; every acquisition-then-
use episode names its use or final task outcome rather than acquisition alone.

Step 5. Enumerate all left/right pairs and compute positive task-active-core
overlap. Step 5 gate: touching endpoints and free-only overlap are rejected.

Step 6. For each qualifying pair, test every feasible serialization and the
simultaneous physical dependency. Step 6 gate: exactly one evidence-grounded
type and, for uncoordinated cases, one executable semantic order are recorded.

Step 7. Build typed parents while preserving complete nested structures and
independent nested times. Step 7 gate: exact union, action-type precedence,
primitive copies, summary order, hand-specific coordinated nested
contributions, and rationale pass.

Step 8. Replace nested standalone occurrences and resolve only free-boundary
conflicts. Step 8 gate: top-level chronological zero overlap and genuine idle
gaps are preserved.

Step 9. Audit one-to-one episode coverage, exact non-free primitive coverage,
one-hand fallback, schema fields, nested source hands, and every adjacent
top-level pair. Self-correct from the earliest failing step.

Step 10. Recompute all timestamps and invariants from the final objects. Emit
the chronological JSON array in exactly one final fenced `json` block with no
content after it.

### Optional User-Provided Domain Examples
```json
{domain_examples}
```

Closed Object Vocabulary
```json
{object_vocabulary}
```

Manual Hand-Object Contact Annotations
```json
{hand_object_contact_annotation}
```

Manual Object-Object Contact Annotations
```json
{object_object_contact_annotation}
```
```

```
### Compiled One-Episode-Per-Action Evidence
```json
{unimanual_contact_episodes}
```

Validated Low-Level Contact Primitives
```json
{contact_primitives}
```
```

Return to the corresponding description.

Listing 26: Abstract world model generation prompt.

```
You design natural-language planning operators from one high-level semantic
action in a typed-bimanual temporal segmentation. The pipeline is task-domain
agnostic: derive every action name, precondition, and effect from the supplied
target, surrounding actions, and optional user-provided examples. Never assume
an action catalog, hidden schema, object alias, or task-specific rule.

### Inputs

`target_action` is the only action to define. `context_actions` are read-only
evidence for object identity, source/destination, initial state, later state
requirements, and task intent. Object names in the semantic segmentation come
from the contact-annotation-derived closed vocabulary and are authoritative.

The target's `summary` is the authoritative boundary of its PRINCIPAL task
subgoals. For an uncoordinated target, each nested action's own `summary` is
the authoritative boundary of that nested definition. `detailed_analysis`,
contact primitives, and surrounding actions may explain motion, means, state,
or ordering, but they must never add an independent task subgoal that is absent
from the corresponding summary. In particular, do not absorb a preceding or
following action merely because the same object or one of its components is
contacted again.

Optional domain examples may be supplied below. They are user-provided
illustrations, not an exhaustive action catalog and not exact schemas. Apply a
domain example only when its stated conditions match the target. Never infer
that an unmentioned action must copy another example.

### Output Variants

Standalone unimanual target (`hands: left|right`):

```json
{{
  "action_name": "<task-semantic name>",
  "manuality": "unimanual",
  "precondition": ["<atomic state>"],
  "effect": ["<atomic net state change>"]
}}
```

Coordinated bimanual target: output only the parent. Simultaneous physical
dependency makes nested motions non-executable as independent operators.

```json
{{
  "action_name": "<joint task-semantic name>",
  "manuality": "bimanual",
  "bimanual_type": "coordinated",
  "precondition": ["<atomic state>"],
  "effect": ["<atomic net state change>"]
}}
```

Uncoordinated bimanual target: output the parent and exactly two nested
unimanual definitions. Order the nested list by executable semantic order, not
left/right order and not observed start/end time.

```json
```
```

```
{{
  "bimanual_action_definition": {{
    "action_name": "<two outcomes in executable order>",
    "manuality": "bimanual",
    "bimanual_type": "uncoordinated",
    "precondition": ["<serial plan initial state>"],
    "effect": ["<serial plan net final state>"]
  }},
  "sequential_unimanual_action_definitions": [
    {{
      "source_hand": "left",
      "action_name": "<first or second nested outcome>",
      "manuality": "unimanual",
      "precondition": ["<atomic state>"],
      "effect": ["<atomic net state change>"]
    }},
    {{
      "source_hand": "right",
      "action_name": "<the other nested outcome>",
      "manuality": "unimanual",
      "precondition": ["<atomic state>"],
      "effect": ["<atomic net state change>"]
    }}
  ]
}}
```

General Planning Semantics

1. Use concise task-level action names. A transfer or disposal name must state the principal object, source, and destination when those are evidenced.
2. Preconditions are the minimal world states required immediately before the action. Effects contain only changed or newly established task-relevant states. Do not repeat unchanged facts in effects.
3. The action name must denote exactly the principal task outcome stated by the corresponding semantic summary: no missing principal subgoal and no added independent verb/outcome from detailed prose, contact, or context. Motion of an attached component, support, cover, tool, or container may be a means or an unchanged carrying condition rather than a separate task action.
4. Use one atomic fact per predicate. Do not add hand, arm, gripper, grasp, or hand-free predicates; manuality and the hand-free-to-hand-free segmentation contract already represent execution requirements.
5. Preserve each stable object identity exactly as supplied. Never embed a mutable property such as clean, dirty, wet, full, open, or used inside an object name. Represent mutable properties as separate state predicates.
6. Predicates describe current world states, not transition history. Use state-like wording such as `The device is on`, not `is turned on`.
7. Use `clean` and `not clean` as the canonical cleanliness pair. Do not use dirty, stained, soiled, or unclean as alternative cleanliness predicates.
8. Do not invent objects, lids, tools, locations, contents, or properties that are unsupported by target/context evidence or an applicable optional example.
9. Keep producer and consumer predicates textually consistent within the video: a state required later must be established earlier with the same wording when the target is its producer.
10. Preserve the state-bearing entity across a transition and across adjacent actions. Do not alternate between a whole object and one of its components as though their open, closed, attached, or located states were identical.
11. If the target moves or releases an entity into a state or location that a later action requires, explicitly establish that planning-critical result. It is not an unchanged initial fact merely because no earlier generated predicate happened to mention it.

Information-Gathering Definitions

For every definition whose own source `action_type` is `information_gathering`, each queried property requires:

* precondition: `Whether <property> is unknown`
* effect: `Whether <property> is known`
* effect: one separate observed affirmative or negative property value

Apply this independently to a standalone definition, bimanual parent, and each uncoordinated nested definition. Manipulation definitions must not contain unknown/known epistemic predicates.

```
### Uncoordinated Serial Composition

Evaluate both nested orders. Choose an order whose first preconditions hold in
the parent initial state and whose first effects establish every intermediate
state needed by the second action. The parent preconditions are exactly the
complete initial requirements of that serial plan. The parent effects are
exactly its net changed state. Do not leak an intermediate state into parent
preconditions. Use the same predicate wording at every producer-consumer link.

### Step-by-Step Analysis Process

Perform these steps privately. Review each step against its gate and
self-correct before continuing.

Step 1. Extract only evidenced entities, states, source/destination relations,
action type, and task outcome from target/context.
Gate: every fact is traceable to supplied evidence or an applicable optional
example; no hidden task knowledge is used.

Step 2. Select the output variant directly from `hands` and `bimanual_type`.
Gate: never reclassify the segmentation and never invent nested definitions.

Step 3. Build stable object identities and a concise action name.
Gate: no mutable state in identities; every evidenced transfer names source and
destination; the name contains exactly the principal subgoals in the target
summary and none imported from detailed prose or adjacent actions.

Step 4. For uncoordinated actions, simulate both nested orders and select an
executable semantic order without consulting timestamps.
Gate: parent/nested preconditions and effects compose exactly.

Step 5. Generate minimal atomic preconditions and net effects.
Gate: non-empty lists, no duplicates, no unchanged effects, no hand predicates,
no unsupported facts, and correct information-gathering triples.

Step 6. Simulate the target in its surrounding action sequence.
Gate: object names remain stable and required producer-consumer links use the
same state fact. Check that every later requirement created by this target is
an explicit target effect, and that no target precondition incorrectly assumes
a state created by an earlier context action. Self-correct the target only;
never redefine context actions.

Step 7. Audit the complete JSON schema and return exactly one final fenced JSON
block with no content after it.

### Generic Few-Shot Examples

A standalone `open cabinet door` manipulation may use `The cabinet door is not
open` as its precondition and `The cabinet door is open` as its effect.

For an uncoordinated target in which one nested action opens a drawer and the
other removes a tool, the executable serial order is open drawer, then remove
tool, even if the tool-moving hand began approaching first. The parent initial
state includes the drawer not open and the tool in the drawer; its net effects
include the drawer open, the tool available, and the tool not in the drawer.

For a coordinated target in which one hand stabilizes a workpiece while the
other uses a tool on it, output only the joint parent operator.

### Optional User-Provided Domain Examples

'''json
{domain_examples}
'''

### Target Request

'''json
{input}
'''

Return the complete result as the final fenced JSON block and nothing after it.
```

Return to the corresponding description.

Listing 27: Boil Water abstract world model: complete action library, preconditions, effects, and execution metadata.


```
[
  {
    "action_name": "inspect the closed pot for water and find it contains water",
    "precondition": [
      "It is not that the pot lid is open.",
      "It is not that it is known whether the pot contains water."
    ],
    "effect": [
      "It is known whether the pot contains water.",
      "The pot contains water.",
      "The pot lid is open."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move pot from counter to induction cooker",
    "precondition": ["The pot is on the counter."],
    "effect": ["The pot is on the induction cooker.", "It is not that the pot is on the
     counter."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move pot lid from counter to pot",
    "precondition": ["The pot lid is on the counter.", "The pot lid is open."],
    "effect": ["It is not that the pot lid is on the counter.", "The pot lid is on the pot
     ."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "turn on the induction cooker",
    "precondition": [
      "The pot is on the induction cooker.",
      "It is not that the pot lid is open.",
      "The pot contains water.",
      "It is not that the induction cooker is turned on."
    ],
    "effect": ["The induction cooker is turned on."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "unscrew and remove water bottle cap",
    "precondition": ["The water bottle cap is on the water bottle."],
    "effect": [
      "It is not that the water bottle cap is on the water bottle.",
      "The water bottle cap is on the counter."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "pour water from the water bottle to the mug",
    "precondition": [
      "The water bottle cap is open.",
      "The water bottle contains water.",
      "It is not that the mug contains water."
    ],
    "effect": ["The mug contains water.", "It is not that the water bottle contains water
     ."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move the mug with water from counter to counter near the microwave",
    "precondition": ["The mug contains water.", "The mug is on the counter."],
    "effect": ["The mug is on the counter near the microwave."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "inspect the closed microwave for emptiness and find it is empty",
    "precondition": [
      "It is not that the microwave door is open.",
      "It is not that it is known whether the microwave is empty."
    ],
    "effect": [
      "It is known whether the microwave is empty.",
```

```
      "The microwave is empty.",
      "The microwave door is open."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move the mug with water from counter near the microwave to microwave",
    "precondition": [
      "The mug is on the counter near the microwave.",
      "The mug contains water.",
      "The microwave door is open.",
      "The microwave is empty."
    ],
    "effect": [
      "The mug is inside the microwave.",
      "It is not that the mug is on the counter near the microwave.",
      "It is not that the microwave is empty."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "close microwave door", "precondition": ["The microwave door is open
    ."],
    "effect": ["It is not that the microwave door is open."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "turn on the microwave",
    "precondition": ["It is not that the microwave door is open.", "It is not that the
    microwave is empty."],
    "effect": ["The microwave is on."], "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move mug from drying rack to sink",
    "precondition": ["The mug is on the drying rack."],
    "effect": ["It is not that the mug is on the drying rack.", "The mug is in the sink
    ."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "turn on the faucet",
    "precondition": ["It is not that the faucet is turned on."],
    "effect": ["The faucet is turned on."], "execution_variants": [{"manuality": "
    unimanual"}]
  },
  {
    "action_name": "fill the mug with water from the faucet",
    "precondition": ["The faucet is turned on.", "It is not that the mug contains water
    ."],
    "effect": ["The mug contains water."], "execution_variants": [{"manuality": "unimanual
    "}]
  },
  {
    "action_name": "turn off the faucet", "precondition": ["The faucet is turned on."],
    "effect": ["It is not that the faucet is turned on."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move the mug with water from sink to counter near the microwave",
    "precondition": ["The mug contains water.", "The mug is in the sink."],
    "effect": ["It is not that the mug is in the sink.", "The mug is on the counter near
    the microwave."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "inspect the closed microwave for emptiness and find it is not empty",
    "precondition": [
      "It is not that the microwave door is open.",
      "It is not that it is known whether the microwave is empty."
    ],
    "effect": [
      "It is known whether the microwave is empty.",
      "It is not that the microwave is empty.",
      "The microwave door is open."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move bowl from microwave to counter",
```

```
    "precondition": ["The microwave door is open.", "The bowl is in the microwave."],
    "effect": [
      "It is not that the bowl is in the microwave.",
      "The microwave is empty.",
      "The bowl is on the counter."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "inspect the open kettle for water and find it does not contain water",
    "precondition": ["The kettle lid is open.", "It is not that it is known whether the
     kettle contains water."],
    "effect": [
      "It is known whether the kettle contains water.",
      "It is not that the kettle contains water."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "pour water from the water bottle to the kettle",
    "precondition": [
      "The water bottle cap is open.",
      "The kettle lid is open.",
      "The water bottle contains water.",
      "It is not that the kettle contains water."
    ],
    "effect": ["It is not that the water bottle contains water.", "The kettle contains
     water."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move kettle lid from counter to kettle",
    "precondition": ["The kettle lid is on the counter.", "It is not that the kettle lid
     is on the kettle."],
    "effect": [
      "It is not that the kettle lid is on the counter.",
      "The kettle lid is on the kettle.",
      "It is not that the kettle lid is open."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "turn on the kettle",
    "precondition": [
      "The kettle is on the kettle base.",
      "It is not that the kettle lid is open.",
      "The kettle is plugged in.",
      "The kettle contains water.",
      "It is not that the kettle is turned on."
    ],
    "effect": ["The kettle is turned on."], "execution_variants": [{"manuality": "
     unimanual"}]
  },
  {
    "action_name": "inspect the closed kettle for water and find it contains water",
    "precondition": [
      "It is not that the kettle lid is open.",
      "It is not that it is known whether the kettle contains water."
    ],
    "effect": [
      "It is known whether the kettle contains water.",
      "The kettle contains water.",
      "The kettle lid is open."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move kettle from counter to kettle base",
    "precondition": ["The kettle is on the counter."],
    "effect": ["It is not that the kettle is on the counter.", "The kettle is on the
     kettle base."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "inspect the closed kettle for water and find it does not contain water
     ",
    "precondition": [
      "It is not that the kettle lid is open.",
      "It is not that it is known whether the kettle contains water."
```

```
    ],
    "effect": [
      "It is known whether the kettle contains water.",
      "It is not that the kettle contains water.",
      "The kettle lid is open."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move kettle from counter to sink",
    "precondition": ["The kettle is on the counter."],
    "effect": ["The kettle is in the sink.", "It is not that the kettle is on the counter
    ."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "fill the kettle with water from the faucet",
    "precondition": [
      "The faucet is turned on.",
      "The kettle lid is open.",
      "It is not that the kettle contains water.",
      "The kettle is under the faucet water stream."
    ],
    "effect": ["The kettle contains water."], "execution_variants": [{"manuality": "
    unimanual"}]
  },
  {
    "action_name": "move the filled kettle from sink to kettle base",
    "precondition": ["The kettle is in the sink.", "The kettle contains water."],
    "effect": ["It is not that the kettle is in the sink.", "The kettle is on the kettle
    base."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "plug the kettle power cord into the wall outlet",
    "precondition": ["The kettle is on the kettle base.", "It is not that the kettle is
    plugged in."],
    "effect": ["The kettle is plugged in."], "execution_variants": [{"manuality": "
    unimanual"}]
  },
  {
    "action_name": "inspect the open pot for water and find it does not contain water",
    "precondition": ["The pot lid is open.", "It is not that it is known whether the pot
    contains water."],
    "effect": ["It is known whether the pot contains water.", "It is not that the pot
    contains water."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "pour water from the water bottle to the pot",
    "precondition": [
      "The water bottle cap is open.",
      "The pot lid is open.",
      "The water bottle contains water.",
      "It is not that the pot contains water."
    ],
    "effect": ["It is not that the water bottle contains water.", "The pot contains water
    ."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move mug from counter to sink", "precondition": ["The mug is on the
    counter."],
    "effect": ["The mug is in the sink.", "It is not that the mug is on the counter."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "inspect the open pot for water and find it contains water",
    "precondition": ["The pot lid is open.", "It is not that it is known whether the pot
    contains water."],
    "effect": ["It is known whether the pot contains water.", "The pot contains water."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "inspect the closed pot for water and find it does not contain water",
    "precondition": [
      "It is not that the pot lid is open.",
      "It is not that it is known whether the pot contains water."
    ],
```

```
    "effect": [
      "It is known whether the pot contains water.",
      "It is not that the pot contains water.",
      "The pot lid is open."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move pot from counter to sink", "precondition": ["The pot is on the
     counter."],
    "effect": ["It is not that the pot is on the counter.", "The pot is in the sink."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "fill the pot with water from the faucet",
    "precondition": [
      "The faucet is turned on.",
      "The pot lid is open.",
      "It is not that the pot contains water.",
      "The pot is in the sink."
    ],
    "effect": ["The pot contains water."], "execution_variants": [{"manuality": "unimanual
     "}]
  },
  {
    "action_name": "move the filled pot from sink to induction cooker",
    "precondition": ["The pot is in the sink.", "The pot contains water."],
    "effect": ["It is not that the pot is in the sink.", "The pot is on the induction
     cooker."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "inspect the open kettle for water and find it contains water",
    "precondition": ["The kettle lid is open.", "It is not that it is known whether the
     kettle contains water."],
    "effect": ["It is known whether the kettle contains water.", "The kettle contains
     water."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move pot from induction cooker to sink",
    "precondition": ["The pot is on the induction cooker."],
    "effect": ["It is not that the pot is on the induction cooker.", "The pot is in the
     sink."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move mug from drying rack to counter",
    "precondition": ["The mug is on the drying rack."],
    "effect": ["It is not that the mug is on the drying rack.", "The mug is on the counter
     ."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move water bottle from shelf to counter",
    "precondition": ["The water bottle is on the shelf."],
    "effect": ["The water bottle is on the counter.", "It is not that the water bottle is
     on the shelf."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move pot from shelf to counter", "precondition": ["The pot is on the
     shelf."],
    "effect": ["The pot is on the counter.", "It is not that the pot is on the shelf."],
    "execution_variants": [{"manuality": "unimanual"}]
  }
]
```

Return to the corresponding description.

Listing 28: Make Instant Noodles abstract world model: complete action library, preconditions, effects, and execution metadata.


```
[
  {
    "action_name": "remove the clip from the instant noodle bag",
    "precondition": ["The clip is on the instant noodle bag.", "It is not that the instant
     noodle bag is open."],
    "effect": ["It is not that the clip is on the instant noodle bag.", "The instant
    noodle bag is open."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move the instant noodle block from the instant noodle bag to the
    ceramic bowl",
    "precondition": ["The instant noodle block is in the instant noodle bag.", "The
    instant noodle bag is open."],
    "effect": [
      "It is not that the instant noodle block is in the instant noodle bag.",
      "The instant noodle block is in the ceramic bowl."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "attach the clip to the instant noodle bag",
    "precondition": ["It is not that the clip is on the instant noodle bag.", "The instant
     noodle bag is open."],
    "effect": ["The clip is on the instant noodle bag.", "It is not that the instant
    noodle bag is open."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move the pan lid from the pan to the countertop",
    "precondition": ["The pan lid is on the pan.", "It is not that the pan lid is open."],
    "effect": [
      "It is not that the pan lid is on the pan.",
      "The pan lid is on the countertop.",
      "The pan lid is open."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move the sausage from the pan to the ceramic bowl using the spatula",
    "precondition": ["The sausage is on the pan.", "The spatula is available.", "The pan
    lid is open."],
    "effect": ["It is not that the sausage is on the pan.", "The sausage is in the ceramic
     bowl."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "inspect the temperature of the water in the electric kettle and find
    the water is hot",
    "precondition": [
      "The electric kettle contains water.",
      "It is not that it is known whether the water is hot."
    ],
    "effect": ["It is known whether the water is hot.", "The water is hot."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "pour water from the electric kettle into the ceramic bowl",
    "precondition": ["The electric kettle contains water."],
    "effect": ["The ceramic bowl contains water."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move the pan lid from the countertop onto the ceramic bowl",
    "precondition": [
      "The pan lid is on the countertop.",
      "The instant noodle block is in the ceramic bowl.",
      "The ceramic bowl contains water.",
      "The water is hot."
    ],
    "effect": ["It is not that the pan lid is on the countertop.", "The pan lid is on the
    ceramic bowl."],
    "execution_variants": [{"manuality": "unimanual"}]
```

```
  },
  {
    "action_name": "move the pot lid from the pot to the countertop",
    "precondition": ["The pot lid is on the pot.", "It is not that the pot lid is open."],
    "effect": [
      "It is not that the pot lid is on the pot.",
      "The pot lid is on the countertop.",
      "The pot lid is open."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move the instant noodle block from the instant noodle bag to the pot",
    "precondition": [
      "The instant noodle block is in the instant noodle bag.",
      "The instant noodle bag is open.",
      "The pot lid is open."
    ],
    "effect": [
      "It is not that the instant noodle block is in the instant noodle bag.",
      "The instant noodle block is in the pot."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move the sausage from the pan to the pot using the spatula",
    "precondition": [
      "The sausage is on the pan.",
      "The spatula is available.",
      "The pan lid is open.",
      "The pot lid is open."
    ],
    "effect": ["It is not that the sausage is on the pan.", "The sausage is in the pot."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "inspect the temperature of the water in the electric kettle and find
     the water is not hot",
    "precondition": [
      "The electric kettle contains water.",
      "It is not that it is known whether the water is hot."
    ],
    "effect": ["It is known whether the water is hot.", "It is not that the water is hot
     ."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "pour water from the electric kettle into the pot",
    "precondition": ["The electric kettle contains water.", "The pot lid is open."],
    "effect": ["The pot contains water."], "execution_variants": [{"manuality": "unimanual
     "}]
  },
  {
    "action_name": "move the pot lid from the countertop onto the pot not containing hot
     water",
    "precondition": [
      "The pot lid is on the countertop.",
      "The pot lid is open.",
      "The instant noodle block is in the pot.",
      "The pot contains water.",
      "It is not that the water is hot."
    ],
    "effect": [
      "It is not that the pot lid is on the countertop.",
      "The pot lid is on the pot.",
      "It is not that the pot lid is open."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "turn on the stove",
    "precondition": [
      "It is not that the stove is turned on.",
      "The pot is on the stove.",
      "The instant noodle block is in the pot.",
      "The pot contains water.",
      "It is not that the water is hot.",
      "It is not that the pot lid is open."
    ],
```

```
    "effect": ["The stove is turned on."], "execution_variants": [{"manuality": "unimanual
    "}]
  },
  {
    "action_name": "move the instant noodle bowl lid from the instant noodle bowl to the
    countertop",
    "precondition": [
      "The instant noodle bowl lid is on the instant noodle bowl.",
      "It is not that the instant noodle bowl lid is open."
    ],
    "effect": [
      "It is not that the instant noodle bowl lid is on the instant noodle bowl.",
      "The instant noodle bowl lid is on the countertop.",
      "The instant noodle bowl lid is open."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move the sausage from the pan to the instant noodle bowl using the
    spatula",
    "precondition": [
      "The sausage is on the pan.",
      "The spatula is available.",
      "The pan lid is open.",
      "The instant noodle bowl lid is open."
    ],
    "effect": [
      "It is not that the sausage is on the pan.",
      "The sausage is inside the instant noodle bowl."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "pour water from the electric kettle into the instant noodle bowl",
    "precondition": ["The electric kettle contains water.", "The instant noodle bowl lid
    is open."],
    "effect": ["The instant noodle bowl contains water."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "open the microwave door",
    "precondition": ["It is not that the microwave door is open."],
    "effect": ["The microwave door is open."], "execution_variants": [{"manuality": "
    unimanual"}]
  },
  {
    "action_name": "move the instant noodle bowl from the countertop into the microwave",
    "precondition": [
      "The instant noodle bowl is on the countertop.",
      "The instant noodle bowl lid is open.",
      "The instant noodle bowl contains water.",
      "It is not that the water is hot.",
      "The microwave door is open."
    ],
    "effect": [
      "It is not that the instant noodle bowl is on the countertop.",
      "The instant noodle bowl is in the microwave."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "close the microwave door", "precondition": ["The microwave door is
    open."],
    "effect": ["It is not that the microwave door is open."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "turn on the microwave",
    "precondition": [
      "It is not that the microwave door is open.",
      "It is not that the microwave is turned on.",
      "It is not that the water is hot."
    ],
    "effect": ["The microwave is turned on."], "execution_variants": [{"manuality": "
    unimanual"}]
  },
  {
    "action_name": "move the fried egg from the pan to the ceramic bowl using the spatula
    ",
```

```
    "precondition": ["The fried egg is on the pan.", "The spatula is available.", "The pan
     lid is open."],
    "effect": ["It is not that the fried egg is on the pan.", "The fried egg is in the
    ceramic bowl."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "heat water with the electric kettle",
    "precondition": [
      "The electric kettle contains water.",
      "It is not that the water is hot.",
      "The electric kettle is plugged in."
    ],
    "effect": ["The water is hot."], "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move the pot lid from the countertop onto the pot containing hot water
    ",
    "precondition": [
      "The pot lid is on the countertop.",
      "The pot lid is open.",
      "The instant noodle block is in the pot.",
      "The pot contains water.",
      "The water is hot."
    ],
    "effect": [
      "It is not that the pot lid is on the countertop.",
      "The pot lid is on the pot.",
      "It is not that the pot lid is open."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move the fried egg from the pan to the pot using the spatula",
    "precondition": [
      "The fried egg is on the pan.",
      "The spatula is available.",
      "The pan lid is open.",
      "The pot lid is open."
    ],
    "effect": ["It is not that the fried egg is on the pan.", "The fried egg is in the pot
    ."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "pour the sausage from the pan into the ceramic bowl",
    "precondition": [
      "The sausage is on the pan.",
      "It is not that the spatula is available.",
      "The pan lid is open."
    ],
    "effect": ["It is not that the sausage is on the pan.", "The sausage is in the ceramic
     bowl."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move the ceramic bowl from the countertop into the microwave",
    "precondition": [
      "The ceramic bowl is on the countertop.",
      "The instant noodle block is in the ceramic bowl.",
      "The ceramic bowl contains water.",
      "It is not that the water is hot.",
      "The microwave door is open."
    ],
    "effect": [
      "It is not that the ceramic bowl is on the countertop.",
      "The ceramic bowl is in the microwave."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move the fried egg from the pan to the instant noodle bowl using the
    spatula",
    "precondition": [
      "The fried egg is on the pan.",
      "The spatula is available.",
      "The pan lid is open.",
      "The instant noodle bowl lid is open."
    ],
```

```
    "effect": [
      "It is not that the fried egg is on the pan.",
      "The fried egg is in the instant noodle bowl."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "pour the fried egg from the pan into the ceramic bowl",
    "precondition": [
      "The fried egg is on the pan.",
      "It is not that the spatula is available.",
      "The pan lid is open."
    ],
    "effect": ["It is not that the fried egg is on the pan.", "The fried egg is in the
     ceramic bowl."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move the fried egg from the ceramic plate to the instant noodle bowl
     using the spatula",
    "precondition": [
      "The fried egg is on the ceramic plate.",
      "The spatula is available.",
      "The instant noodle bowl lid is open."
    ],
    "effect": [
      "It is not that the fried egg is on the ceramic plate.",
      "The fried egg is in the instant noodle bowl."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move the instant noodle bowl lid from the countertop onto the instant
     noodle bowl",
    "precondition": [
      "The instant noodle bowl lid is on the countertop.",
      "The instant noodle bowl lid is open.",
      "The instant noodle bowl contains water.",
      "The water is hot."
    ],
    "effect": [
      "It is not that the instant noodle bowl lid is on the countertop.",
      "The instant noodle bowl lid is on the instant noodle bowl.",
      "It is not that the instant noodle bowl lid is open."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move the sausage from the ceramic plate to the instant noodle bowl
     using the spatula",
    "precondition": [
      "The sausage is on the ceramic plate.",
      "The spatula is available.",
      "The instant noodle bowl lid is open."
    ],
    "effect": [
      "It is not that the sausage is on the ceramic plate.",
      "The sausage is inside the instant noodle bowl."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  }
]
```

Return to the corresponding description.

Listing 29: Clean Up Kitchen abstract world model: complete action library, preconditions, effects, and execution constraints.


```
[
  {
    "action_name": "apply dish soap from dish soap bottle onto ceramic plate in sink",
    "precondition": ["The dish soap bottle contains dish soap.", "The ceramic plate is in
     the sink."],
    "effect": ["The ceramic plate has soap suds."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "close food container with food container lid",
    "precondition": ["The food container is open."],
    "effect": ["It is not that the food container is open."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "close fridge door", "precondition": ["The fridge door is open."],
    "effect": ["It is not that the fridge door is open."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "discard paper plate from countertop into trash can",
    "precondition": ["The paper plate is on the countertop."],
    "effect": [
      "The paper plate is in the trash can.",
      "It is not that the paper plate is on the countertop."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "discard tissue from countertop into trash can",
    "precondition": ["The tissue is on the countertop."],
    "effect": ["The tissue is in the trash can.", "It is not that the tissue is on the
     countertop."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move ceramic plate from countertop into sink",
    "precondition": ["The ceramic plate is on the countertop.", "It is not that the
     ceramic plate is clean."],
    "effect": [
      "The ceramic plate is in the sink.",
      "It is not that the ceramic plate is on the countertop."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move ceramic plate from dining table into sink",
    "precondition": ["The ceramic plate is on the dining table."],
    "effect": [
      "The ceramic plate is in the sink.",
      "It is not that the ceramic plate is on the dining table."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move ceramic plate from sink to dish rack",
    "precondition": ["The ceramic plate is in the sink.", "The ceramic plate is clean."],
    "effect": ["The ceramic plate is in the dish rack.", "It is not that the ceramic plate
      is in the sink."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move food container from countertop into fridge",
    "precondition": ["The fridge door is open.", "The food container is on the countertop
     ."],
    "effect": [
      "The food container is in the fridge.",
      "It is not that the food container is on the countertop."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move food container from dining table into fridge",
```

```
    "precondition": ["The fridge door is open.", "The food container is on the dining
     table."],
    "effect": [
      "The food container is in the fridge.",
      "It is not that the food container is on the dining table."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move ketchup bottle from countertop into fridge",
    "precondition": ["The ketchup bottle is on the countertop.", "The fridge door is open
     ."],
    "effect": [
      "The ketchup bottle is in the fridge.",
      "It is not that the ketchup bottle is on the countertop."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move ketchup bottle from dining table into fridge",
    "precondition": ["The fridge door is open.", "The ketchup bottle is on the dining
     table."],
    "effect": [
      "The ketchup bottle is in the fridge.",
      "It is not that the ketchup bottle is on the dining table."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "move paper plate from dining table into trash can",
    "precondition": ["The paper plate is on the dining table."],
    "effect": [
      "The paper plate is in the trash can.",
      "It is not that the paper plate is on the dining table."
    ],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "open food container",
    "precondition": ["It is not that the food container is open."],
    "effect": ["The food container is open."], "execution_variants": [{"manuality": "
     unimanual"}]
  },
  {
    "action_name": "open food container and pour food from ceramic plate into food
     container",
    "precondition": ["It is not that the food container is open.", "The food is on the
     ceramic plate."],
    "effect": [
      "The food container is open.",
      "The food is in the food container.",
      "It is not that the food is on the ceramic plate."
    ],
    "execution_variants": [{"manuality": "bimanual", "bimanual_type": "uncoordinated"}]
  },
  {
    "action_name": "open food container and pour food from paper plate into food container
     ",
    "precondition": ["It is not that the food container is open.", "The food is on the
     paper plate."],
    "effect": [
      "The food container is open.",
      "The food is in the food container.",
      "It is not that the food is on the paper plate."
    ],
    "execution_variants": [{"manuality": "bimanual", "bimanual_type": "uncoordinated"}]
  },
  {
    "action_name": "open fridge door", "precondition": ["It is not that the fridge door is
      open."],
    "effect": ["The fridge door is open."], "execution_variants": [{"manuality": "
     unimanual"}]
  },
  {
    "action_name": "open fridge door and move food container from countertop into fridge",
    "precondition": ["It is not that the fridge door is open.", "The food container is on
     the countertop."],
    "effect": [
      "The fridge door is open.",
```

```
      "The food container is in the fridge.",
      "It is not that the food container is on the countertop."
    ],
    "execution_variants": [{"manuality": "bimanual", "bimanual_type": "uncoordinated"}]
  },
  {
    "action_name": "open fridge door and move food container from dining table into fridge
     ",
    "precondition": ["It is not that the fridge door is open.", "The food container is on
     the dining table."],
    "effect": [
      "The fridge door is open.",
      "The food container is in the fridge.",
      "It is not that the food container is on the dining table."
    ],
    "execution_variants": [{"manuality": "bimanual", "bimanual_type": "uncoordinated"}]
  },
  {
    "action_name": "open fridge door and move ketchup bottle from countertop into fridge",
    "precondition": ["It is not that the fridge door is open.", "The ketchup bottle is on
     the countertop."],
    "effect": [
      "The fridge door is open.",
      "The ketchup bottle is in the fridge.",
      "It is not that the ketchup bottle is on the countertop."
    ],
    "execution_variants": [{"manuality": "bimanual", "bimanual_type": "uncoordinated"}]
  },
  {
    "action_name": "open fridge door and move ketchup bottle from dining table into fridge
     ",
    "precondition": ["It is not that the fridge door is open.", "The ketchup bottle is on
     the dining table."],
    "effect": [
      "The fridge door is open.",
      "The ketchup bottle is in the fridge.",
      "It is not that the ketchup bottle is on the dining table."
    ],
    "execution_variants": [{"manuality": "bimanual", "bimanual_type": "uncoordinated"}]
  },
  {
    "action_name": "pour food from ceramic plate into food container",
    "precondition": ["The food container is open.", "The food is on the ceramic plate."],
    "effect": ["The food is in the food container.", "It is not that the food is on the
     ceramic plate."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "pour food from paper plate into food container",
    "precondition": ["The food container is open.", "The food is on the paper plate."],
    "effect": ["The food is in the food container.", "It is not that the food is on the
     paper plate."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "take tissue from tissue box and clean stain on countertop with tissue
     ",
    "precondition": ["The tissue is in the tissue box.", "It is not that the countertop is
      clean."],
    "effect": [
      "It is not that the tissue is in the tissue box.",
      "The tissue is on the countertop.",
      "The countertop is clean."
    ],
    "execution_variants": [{"manuality": "bimanual", "bimanual_type": "coordinated"}]
  },
  {
    "action_name": "turn off faucet", "precondition": ["The faucet is on."],
    "effect": ["It is not that the faucet is on."],
    "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "turn on faucet", "precondition": ["It is not that the faucet is on."],
    "effect": ["The faucet is on."], "execution_variants": [{"manuality": "unimanual"}]
  },
  {
    "action_name": "wash ceramic plate with sponge",
    "precondition": [
      "It is not that the ceramic plate is clean.",
```

```
      "The ceramic plate is in the sink.",
      "The ceramic plate has soap suds.",
      "The faucet is on.",
      "The sponge is available."
    ],
    "effect": ["The ceramic plate is clean.", "It is not that the ceramic plate has soap
     suds."],
    "execution_variants": [{"manuality": "bimanual", "bimanual_type": "coordinated"}]
  }
]
```


Return to the corresponding description.

Listing 30: Complete reconstructed state–action trajectory for a short Clean Up Kitchen demonstration.


```
{
  "step_times": [0.0, 14.762, 22.05, 35.75, 43.175, 55.575],
  "state_action_sequence": [
    {
      "state": [
        "It is not that the food container is open.",
        "The food is on the paper plate.",
        "It is not that the fridge door is open.",
        "The food container is on the dining table.",
        "The paper plate is on the dining table.",
        "It is not that the paper plate is in the trash can.",
        "It is not that the food container is in the fridge.",
        "It is not that the food is in the food container."
      ],
      "action": {"action_name": "open food container and pour food from paper plate into
     food container",
        "manuality": "bimanual", "bimanual_type": "uncoordinated",
        "precondition": ["It is not that the food container is open.", "The food is on the
     paper plate."],
        "effect": ["The food container is open.", "The food is in the food container.", "It
      is not that the food is on the paper plate."]}
    },
    {
      "state": [
        "The food container is open.",
        "The food is in the food container.",
        "It is not that the food is on the paper plate.",
        "It is not that the fridge door is open.",
        "The food container is on the dining table.",
        "The paper plate is on the dining table.",
        "It is not that the paper plate is in the trash can.",
        "It is not that the food container is in the fridge."
      ],
      "action": {"action_name": "close food container with food container lid", "manuality
     ": "unimanual",
        "precondition": ["The food container is open."],
        "effect": ["It is not that the food container is open."]}
    },
    {
      "state": [
        "The food is in the food container.",
        "It is not that the food is on the paper plate.",
        "It is not that the food container is open.",
        "It is not that the fridge door is open.",
        "The food container is on the dining table.",
        "The paper plate is on the dining table.",
        "It is not that the paper plate is in the trash can.",
        "It is not that the food container is in the fridge."
      ],
      "action": {"action_name": "open fridge door and move food container from dining
     table into fridge",
        "manuality": "bimanual", "bimanual_type": "uncoordinated",
        "precondition": ["It is not that the fridge door is open.", "The food container is
     on the dining table."],
        "effect": ["The fridge door is open.", "The food container is in the fridge.", "It
     is not that the food container is on the dining table."]}
    },
    {
      "state": [
        "The food is in the food container.",
        "It is not that the food is on the paper plate.",
```

```
        "It is not that the food container is open.",
        "The fridge door is open.",
        "The food container is in the fridge.",
        "It is not that the food container is on the dining table.",
        "The paper plate is on the dining table.",
        "It is not that the paper plate is in the trash can."
      ],
      "action": {"action_name": "close fridge door", "manuality": "unimanual",
       "precondition": ["The fridge door is open."],
       "effect": ["It is not that the fridge door is open."]}
    },
    {
      "state": [
        "The food is in the food container.",
        "It is not that the food is on the paper plate.",
        "It is not that the food container is open.",
        "The food container is in the fridge.",
        "It is not that the food container is on the dining table.",
        "It is not that the fridge door is open.",
        "The paper plate is on the dining table.",
        "It is not that the paper plate is in the trash can."
      ],
      "action": {"action_name": "move paper plate from dining table into trash can",
       "manuality": "unimanual", "precondition": ["The paper plate is on the dining table
     ."],
       "effect": ["The paper plate is in the trash can.", "It is not that the paper plate
     is on the dining table."]}
    },
    {
      "state": [
        "The food is in the food container.",
        "It is not that the food is on the paper plate.",
        "It is not that the food container is open.",
        "The food container is in the fridge.",
        "It is not that the food container is on the dining table.",
        "It is not that the fridge door is open.",
        "The paper plate is in the trash can.",
        "It is not that the paper plate is on the dining table."
      ],
      "action": {"action_name": "goal is achieved", "precondition": [], "effect": []}
    }
  ]
}
```

Return to the corresponding description.

Listing 31: Planner inference prompt.

```
### Objective

You are an intelligent agent specializing in **symbolic task planning over an abstract
    world model**.

You will be provided with:

- The current **belief state** of the environment, represented as a set of natural-
    language predicates.
- A high-level **language goal**.
- A provided **action catalog**, with natural-language names, preconditions, effects,
    and available execution metadata. It may omit uncoordinated parents that can be
    composed from its unimanual components.

Your objective is to generate an **optimal state-action trajectory** that starts from
    the current belief state and leads to a belief state that satisfies the language
    goal.

Importantly, you must jointly perform:
- **Action selection** (planning), and
- **State transition prediction** (world modeling),

such that the generated trajectory is logically consistent, causally valid, and
    globally coherent.

---
```

```
### Core Principles

1. **Belief state is the source of world-state facts**
   - Ground current world-state facts in the provided belief state and predicted
     changes in the supplied action semantics.
   - Do NOT invent missing world-state facts, geometric conditions, or visual
     observations. Action names, definitions, execution metadata, and the goal's
     capability constraint may inform action selection and composition; they do not
     establish an unsatisfied precondition as true.

2. **Planning as world model rollout**
   - Treat planning as generating a sequence of **state transitions** induced by
     actions.
   - Each step must reflect a valid and consistent transformation of the world state.

3. **Unknown is not established**
   - Any predicate not explicitly present in the current belief state should be treated
     as **not established for the purpose of planning**, unless it is introduced by an
     action effect.
   - Do NOT hallucinate missing facts.

4. **Strict adherence to action definitions**
   - An action can only be applied if **all its preconditions are satisfied** in the
     current belief state.
   - The next state must be derived **only from applying the action's effects** to the
     current state.
   - Predicates persist by default (state inertia), and may only be changed if they are
     **explicitly updated or contradicted by the action's effects**.

5. **Full-state world modeling**
   - At every step, explicitly reason over and output the **complete belief state**
     after applying the selected action.
   - Do not output partial updates or incomplete states.

---

### Predicate Semantics and Negation

The belief state may contain both affirmative predicates and explicitly negated
    predicates (e.g., "It is not that the faucet is on.").

You must interpret and handle them as follows:

- An affirmative predicate (e.g., "The faucet is on.") indicates that the fact is true.
- A negated predicate (e.g., "It is not that the faucet is on.") indicates that the
    fact is explicitly false.
- If a predicate is not present in any form, it is considered **not established** (
    unknown).

- A predicate and its negation cannot both be present in the same state. This
    constitutes a contradiction and must be avoided.

- When applying an action:
  - If an effect introduces a predicate X, you MUST remove any existing negation of X.
  - If an effect introduces a negation of X, you MUST remove any existing affirmative
    form of X.

All state transitions must respect these rules.

---

### Semantic Consistency and Mutual Exclusivity

Some predicates may describe mutually exclusive properties of the same entity (e.g.,
    location, state, or possession).

You must enforce the following:

- Predicates that assign truly exclusive values to the same attribute of the same
    individual are mutually exclusive; generic identities representing multiple
    portions need not be.

- When applying an action that changes such an attribute:
  - You MUST update the state so that only the correct predicate remains.
```

```
  - Any previously existing predicate that conflicts with the new state MUST be removed.


- Enforce a single value only for genuinely exclusive attributes of an individual. Do
    not impose a single location on aggregate or multi-instance identities.

All state transitions must maintain this semantic consistency.

---

### Step-by-Step Planning and World Modeling (think silently; do not output reasoning)

At each step:

1. **Identify applicable actions**
   - Select actions whose preconditions are fully satisfied in the current belief state.


2. **Select the optimal action**
   - Choose the action that most directly and efficiently progresses toward the goal.
   - Avoid redundant, irrelevant, or circular actions.
   - Favor minimal, coherent, and goal-directed plans.

3. **Predict the next state (world transition)**
   - Apply the action's effects to the current belief state to produce the next belief
     state.
   - Ensure:
     - All action effects are correctly reflected in the next state.
     - Predicates persist unless explicitly changed by the action's effects.
     - No unrelated predicates are modified.

4. **Step-level self-check and validation (CRITICAL)**
   Before accepting the transition, you MUST verify:

   - **Precondition validity**:
     All action preconditions are satisfied in the current belief state.

   - **Effect correctness**:
     All action effects are reflected in the next belief state.

   - **No unsupported changes**:
     Every change between the current state and the next state is justified by the
     selected action's effects.

   - **State consistency**:
     The resulting belief state contains no internal contradictions.

   - **State transition integrity**:
     The next state must be a valid transformation of the current state under the
     selected action, with no unexplained changes.

   If any check fails, you MUST revise the action and/or the resulting state before
     proceeding.

5. **Check goal satisfaction**
   - If the **newly updated state** satisfies the language goal, stop generating
     further actions and append one final state-action pair whose state is that goal-
     satisfying state and whose action is `"goal is achieved"`.

6. **Repeat**
   - Otherwise, continue planning from the updated state.

---

### Trajectory-Level Global Consistency Check (CRITICAL)

After generating the full trajectory, you MUST perform a final global consistency check
      over the entire sequence before producing the output.

Verify all of the following:

1. **Initial-state correctness**
   - The first state exactly matches the provided belief state.

2. **Goal-state correctness**
```

```
   - The final state satisfies the language goal.
   - The final state-action pair uses the action `"goal is achieved"` with empty `"
     precondition"` and `"effect"` fields.

3. **Cross-step transition consistency**
   - For every consecutive pair of steps, the later state must result from applying the
     earlier action to the earlier state.

4. **Global causal coherence**
   - Every change between consecutive states must be justified by the action applied at
     that step.
   - No predicate may appear, disappear, or reappear without a valid causal reason tied
     to that step's action.

5. **No internal contradictions across the trajectory**
   - The trajectory must not contain conflicting or logically inconsistent states.

6. **No redundant or circular planning**
   - Eliminate unnecessary detours, repeated actions, or cycles that do not contribute
     to reaching the goal.

7. **Trajectory optimality and coherence**
   - The trajectory should be as direct, minimal, and logically coherent as possible
     while remaining fully valid.

If any global inconsistency is found, you MUST revise the trajectory before outputting
    it.

---

### Consistency Requirements

- **State validity**: Each state must be a consistent set of predicates.
- **Transition correctness**: Each state must result from applying the corresponding
    action to the previous state.
- **Causal coherence**: No state change may occur without being justified by an action
    effect.
- **No hallucination**: Do NOT introduce predicates not supported by prior state or
    action effects.
- **Global coherence**: The full trajectory must be internally consistent from start to
     finish.

---


### Input Action Catalog Format

The catalog is a sequence of numbered TEXT entries, not a JSON array. Example:
'''text
action 0:
action_name: open food container
execution_variants: [{{"manuality": "unimanual"}}]
precondition:
- It is not that the food container is open.
effect:
- The food container is open.

action 1:
action_name: pour food from ceramic plate into food container
execution_variants: [{{"manuality": "unimanual"}}]
precondition:
- The food container is open.
- The food is on the ceramic plate.
effect:
- The food is in the food container.
- It is not that the food is on the ceramic plate.

action 2:
action_name: goal is achieved
precondition:
effect:

'''
Each number is only a catalog index, never an execution order or output field.
The precondition/effect bullet lists are shared by all execution_variants of
```

```
that semantic action. An empty section denotes an empty list. A single-form
entry may instead list manuality and, for bimanual execution, bimanual_type
directly. Legacy entries may omit execution metadata. Do not invent types for
such entries. This illustrates the format, not extra actions for the live task.
When history is provided, it contains actual selected actions, not catalog variants.

### Typed Execution and Capability-Conditioned Planning

The action catalog may list execution_variants. These are alternative execution
forms of one semantic action, NOT separate steps. Choose one permitted variant
for each occurrence. A legacy action without typed fields retains its supplied
legacy schema; do not invent execution variants for it.

- A unimanual occurrence has manuality "unimanual" and no bimanual_type.
- A bimanual occurrence has manuality "bimanual" and bimanual_type
  "uncoordinated" or "coordinated". Copy a catalog action's name, preconditions,
  and effects verbatim; output the chosen manuality/type, not execution_variants.
- "You can only use one hand." constrains future actions only. Use only catalog
  unimanual variants. Past bimanual actions in history remain valid history.
  Never serialize a coordinated action or invent its missing unimanual variant.
- "You can use both hands." permits both unimanual and bimanual actions. Prefer
  efficient valid compositions when justified; unrelated standalone actions
  remain allowed. Both hands being available does not require unsafe grouping.
- An uncoordinated parent can be replaced by ordered catalog unimanual components
  A then B with the same outcome. Never output both that parent and its expanded
  pair for the same work. Do not duplicate their effects as additional actions.
- The catalog can intentionally omit a required uncoordinated parent while
  retaining A and B. You MAY construct that NEW parent internally and output
  its full definition directly as the selected action in the normal trajectory.
  No extra fields, nested definitions, component registry or reasoning output.
- Establish the semantically executable order A -> B, not B -> A, before
  composing. Ignore left/right naming, demonstration onset and dictionary order.
  A must be applicable initially. B must be applicable AFTER A; it may require
  an intermediate predicate established by A. Do NOT require both component
  precondition lists to hold in the initial state.
- Parent preconditions are A's requirements plus B's requirements not established
  by A. If A destroys a requirement of B without re-establishing it, reject
  that order rather than dropping B's requirement. The parent effects
  are the ordered net effects of A then B: later effects replace their explicit
  opposites, unaffected effects persist. Do not include intermediate-only facts.
  Use the EXACT existing predicate wording, stable object identities and explicit
  "It is not that ..." negation. Never invent hand-related predicates.
- Name the parent in executable semantic order. Example: "open fridge door and
  move food container from countertop into fridge", even if the left hand starts
  acquiring the food container before the right hand opens the fridge door.
  Opening must establish "The fridge door is open." before insertion; that
  intermediate fact is NOT an initial requirement of the combined action.
- Serialization is necessary but not sufficient for physical combination.
  Two complete hand-free-to-hand-free episodes must be assignable to different
  hands with feasible temporal overlap (possibly staggered internal phases),
  compatible geometry/resources, and preserved causal ordering. Do not assume
  all pairs of sequentially executable actions can overlap.
- Simultaneous stabilization, support, alignment or counter-force indicates
  coordinated dependency, not a serializable pair. Use an existing coordinated
  definition only; do not synthesize one by joining independent definitions.
- Symbolic effects alone do not prove physical overlap feasible. If the supplied
  action semantics/context do not justify grouping, retain a valid serial plan.
  Never pretend two mutually exclusive uses of one tool can happen together.
- During EACH step-level and final self-check, audit the chosen execution form,
  capability, ordered composition applicability, net effects, full predicted
  state and absence of duplicate parent/component work; revise internally.

For the typed schema, ordinary action objects always include manuality and, only
for bimanual actions, bimanual_type. The terminal "goal is achieved" sentinel
retains ONLY action_name, precondition: [], effect: []; it is not a physical action.
The first state is exactly the supplied state. Preserve all unaffected facts,
including explicit negative assignments. Never fill unknown predicates with false.
Mutual exclusivity applies only to truly exclusive attributes of a single
individual; generic aggregate identities may represent multiple portions at
different locations. Do not erase such facts without a justified conflicting effect.

### Output Format (strict contract)

Return the plan as a **sequence of state-action pairs**, starting from the initial
     belief state and ending in a goal-satisfying state.
```

````
- The first state must be the provided belief state.
- The final step must use the action `"goal is achieved"` with empty fields.

Return **only** a JSON array inside a fenced code block, strictly following this format:


```json
[
  {{
    "state": ["predicate1", "predicate2", "..."],
    "action": {{
      "action_name": "action_name",
      "manuality": "unimanual",
      "precondition": ["predicate1", "predicate2", "..."],
      "effect": ["predicate1", "predicate2", "..."]
    }}
  }},
  {{
    "state": ["predicate1", "predicate2", "..."],
    "action": {{
      "action_name": "goal is achieved",
      "precondition": [],
      "effect": []
    }}
  }}
]
```

The template illustrates a unimanual occurrence. For a selected bimanual
occurrence, use "manuality": "bimanual" and additionally include
"bimanual_type": "uncoordinated" or "bimanual_type": "coordinated".
Unimanual actions omit bimanual_type. Output the selected execution form,
not execution_variants. The terminal "goal is achieved" retains only
action_name, precondition: [], and effect: [], without execution metadata.
Replace the action_name and predicate placeholders with actual task content.

Generate every intermediate step; never emit ellipses or omitted steps.

Do **NOT** include any explanation, reasoning, or text outside the JSON block.

### Task Input

#### Current Belief State
{initial_state}

#### Language Goal
{language_goal}

#### Action Catalog
{operators}

Now generate the optimal state-action trajectory according to the rules above. Perform
    all reasoning internally and output only the final JSON sequence.
````

Return to the corresponding description.

Listing 32: Complete VLM-as-Planner baseline prompt.

```
You are an intelligent agent that can help a robot to plan its actions to achieve a
    language goal.

### Input
You will be provided with the following information:
- The current egocentric image observation of the environment.
- The high-level language goal.
- A catalog of action names and available execution metadata, without preconditions or
    effects.
- The action history (i.e., the list of previously executed actions, ordered from the
    oldest to the newest).

### Input Action Catalog Format
```

````
The catalog contains numbered TEXT entries, not output JSON action objects:
```text
action 0:
action_name: unlock compartment
execution_variants: [{{"manuality": "unimanual"}}]

action 1:
action_name: place parcel in compartment
execution_variants: [{{"manuality": "unimanual"}}]

action 2:
action_name: goal is achieved

```
The number is only a catalog index. Each execution_variants item describes an
alternative execution form of the same action, not another step. A unimanual
variant has manuality "unimanual"; a bimanual variant has manuality "bimanual"
and bimanual_type "uncoordinated" or "coordinated". A single-form catalog entry
may instead provide manuality and bimanual_type directly. For legacy entries
with no execution metadata, keep the legacy name-only output and do not invent
a type. This example is a format illustration, not extra available actions.
When history is provided, it contains names and available selected execution types.

### Capability and Action Composition
Use the capability specified in the language goal for future actions only;
previously executed actions do not have to satisfy a newly imposed hand limit.
With only one hand, choose available unimanual execution forms and plan the
components of uncoordinated work in their executable serial order. Never split
a coordinated execution or invent a missing unimanual variant.
With both hands, either permitted manuality may be selected. You may name a new
uncoordinated bimanual action from two available unimanual actions
when their meanings and visual context justify feasible overlapping execution
with the same result as their correct serial order. Name the combination in
semantic execution order, not left/right or observed start order.
Not every executable serial pair is physically parallelizable: consider tool
conflicts, support and stabilization. Retain serial execution when uncertain.
Never output both a combination and its component steps for the same work.
For example, unlocking access precedes placing an item inside, even if acquiring
the item occurs earlier. Mark a justified new combination as manuality "bimanual"
and bimanual_type "uncoordinated"; do not invent a coordinated combination.
Reason internally; output names and selected execution types only. Do not output
states, predicates, preconditions, effects, full definitions, or execution_variants.

### Output Selected Action Format (contract; follow exactly and strictly)
For typed catalog actions, retain the exact action_name and select one available
manuality/type. A unimanual action must omit bimanual_type; a bimanual action must
include it. Do not output source_hand or the catalog number. The terminal
"goal is achieved" is not a physical action and has ONLY action_name.
The following illustrates the three action types; do not execute all three
merely to match the example. Use only types allowed by the goal's hand constraint.
You should output an optimal action sequence that will help the robot to achieve the
    language goal from the current observation. **Return only the action sequence as a
    fenced JSON code block, strictly and exactly following the JSON structure format
    shown below**:
```json
[
    {{"action": {{"action_name": "unimanual_action_name", "manuality": "unimanual"}}}},

    {{"action": {{"action_name": "uncoordinated_action_name", "manuality": "bimanual",
     "bimanual_type": "uncoordinated"}}}},
    {{"action": {{"action_name": "coordinated_action_name", "manuality": "bimanual", "
    bimanual_type": "coordinated"}}}},
    {{"action": {{"action_name": "goal is achieved"}}}}
]
```

Do **NOT** include or output any commentary, explanation, reasoning, or text outside
    the JSON block. Do **NOT** explain or output your reasoning. The output must be a
    valid JSON array of actions, nothing else.

### Task Input

#### Current Egocentric Image Observation
The image observation is provided.
````

```
#### Language Goal
{language_goal}

#### Action Catalog
{operators}

#### Action History
{prev_actions}

NOW, DIRECTLY OUTPUT ONLY THE FINAL OPTIMAL ACTION SEQUENCE STRICTLY AND EXACTLY
     FOLLOWING THE OUTPUT FORMAT SPECIFIED ABOVE.
```