# UniDomain: Pretraining a Unified PDDL Domain from Real-World Demonstrations for Generalizable Robot Task Planning

Haoming Ye<sup>1,2</sup>\*, Yunxiao Xiao<sup>2,3</sup>\*, Cewu Lu<sup>1,2</sup>, Panpan Cai<sup>1,2</sup>†

<sup>1</sup>Shanghai Jiao Tong University 

<sup>2</sup>Shanghai Innovation Institute

<sup>3</sup>Beijing University of Posts and Telecommunications

\* Equal contribution

† Corresponding author: cai\_panpan@sjtu.edu.cn

#### **Abstract**

Robotic task planning in real-world environments requires reasoning over implicit constraints from language and vision. While LLMs and VLMs offer strong priors, they struggle with long-horizon structure and symbolic grounding. Existing methods that combine LLMs with symbolic planning often rely on handcrafted or narrow domains, limiting generalization. We propose UniDomain, a framework that pre-trains a PDDL domain from robot manipulation demonstrations and applies it to online robotic task planning. It extracts atomic domains from 12,393 manipulation videos to form a unified domain with 3,137 operators, 2,875 predicates, and 16,481 causal edges. Given a target class of tasks, it retrieves relevant atomics from the unified domain and systematically fuses them into high-quality meta-domains to support compositional generalization in planning. Experiments on diverse real-world tasks show that UniDomain solves complex, unseen tasks in a zero-shot manner, achieving up to 58% higher task success and 160% improvement in plan optimality over state-of-the-art LLM and LLM-PDDL baselines. <sup>1</sup>

#### 1 Introduction

Robotic task planning in real-world environments requires reasoning over complex constraints that are often implicitly specified in natural language instructions and grounded in visual observations. For instance, the task "partition the stack into even and odd numbers, sorted in ascending order" implicitly encodes long-term dependencies involving unstacking, sorting, and placement. Similarly, "make a cup of tea" entails a sequence of preparatory steps such as opening the cabinet, finding the tea cup, and boiling the water. These tasks demand structured reasoning over action preconditions, temporal dependencies, and physical affordances, in order to ensure safety (e.g., avoiding spills), prevent irreversible states, and minimize human intervention.

However, these problems remain fundamentally challenging: instructions are open-ended, scenes are unstructured, and constraints are implicit. Recent approaches leverage the commonsense priors of Large Language Models (LLMs) and Vision-Language Models (VLMs) [1, 2] to generalize across real-world tasks. Yet, despite their strengths in language and visual understanding, LLMs and VLMs often fail to model action preconditions and effects accurately, and struggle with generating coherent, long-horizon plans [3]. To improve reasoning, recent work [4, 5, 6] integrates LLMs with symbolic planning using the Planning Domain Definition Language (PDDL) [7]. A common pipeline translates language instructions and scene images into structured PDDL domain and problem files, then invokes a symbolic planner [8] to produce a plan. While LLMs can reliably generate PDDL problems when the domain is given [9, 10], they struggle to construct realistic domain files [11, 12] for planning tasks.

<sup>&</sup>lt;sup>1</sup>The code and demonstration video are available at: https://roboticsjtu.github.io/UniDomain/

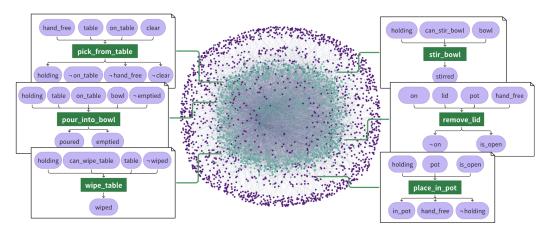


Figure 1: Visualization of our pre-trained unified domain, with 3,137 operator nodes (green) and 2,875 predicate nodes (purple).

Most models are only familiar with abstract domains [13, 14] such as BlocksWorld, Logistics, and Rovers, and lack grounding in real-world robot interactions.

To overcome this limitation, we propose leveraging large-scale demonstration datasets designed for training Vision-Language-Action (VLA) models [15, 16, 17] (e.g., DROID [18]). These datasets are grounded in real robot executions. Visual information in demonstrations captures the actual preconditions and effects of robot actions in diverse environments. Although each demonstration typically covers a single operation, the collection spans a broad spectrum of manipulation tasks, enabling *compositional generalization*. By learning atomic domains from **12,393** real-world demonstrations and merging them into a structured unified domain, we construct a connected symbolic knowledge graph (Figure 1)—containing **3,137** operators, **2,875** predicates, and **16,481** causal edges—to support long-term planning under complex constraints across diverse household tasks.

Particularly, we introduce UniDomain, a framework for *pre-training a unified, general-purpose PDDL domain* (referred to as the *unified domain*) from large-scale robot manipulation demonstrations, and applying it for online planning in unseen tasks. UniDomain comprises three stages: (1) *Domain Pretraining*. Given a demonstration dataset, each video is segmented into keyframes using an energy-based method. A VLM proposes an initial atomic domain from the keyframes, which is refined via closed-loop verification with an LLM to ensure syntactic correctness, solvability, and commonsense alignment. The resulting set of atomic domains constitutes a large-scale unified domain. (2) *Domain Fusion*. For any targeted task class, a relevant subset of atomic domains is retrieved from the large-scale unified domain and systematically fused into a high-quality *meta-domain*. The fusion merges functionally overlapping predicates and operators, yielding a compact yet expressive subgraph for generalizable task planning. (3) *Online Planning*. Given a specific task, UniDomain constructs a grounded PDDL problem with the high-quality meta-domain and solves it using a PDDL planner to generate an optimal plan.

We analogize these stages to the *pre-training*, *post-training*, and *inference* phases of foundation models [19, 20]. *Domain pretraining* builds a comprehensive unified domain encoding general manipulation knowledge. *Domain fusion* constructs a specialized meta-domain with enhanced symbolic connectivity. *Online planning* applies the meta-domain in a zero-shot fashion, to solve unseen tasks without additional demonstrations or feedback.

We evaluate UniDomain on four real-world task domains unseen during training, comprising 100 long-horizon tasks with complex constraints. Results show that UniDomain significantly outperforms popular LLM-only planners (e.g., Code-as-Policies [21], ReAct [22]) and state-of-the-art hybrid LLM-PDDL baselines (e.g., ISR-LLM [23], BoN-iVML [14]), achieving up to 58% higher task success and 160% better plan optimality than the strongest baselines. Ablation studies confirm that performance gains stem from data-driven domain learning, closed-loop verification, hierarchical fusion for meta-domain construction, and task-relevant grounding during online planning.

In summary, the main contributions of this work include: (1) The first framework to pre-train a unified PDDL domain for high-level robot task planning from large-scale, real-world demonstrations;

(2) A novel LLM-based domain fusion method for combining small, disconnected PDDL domains into a coherent and compact meta-domain thus supporting compositional generalization; and (3) A novel online task planner that applies the fused meta-domain to solve general, unseen tasks through VLM-grounded PDDL planning.

# 2 Background and Related Work

#### 2.1 PDDL Fundamentals

The Planning Domain Definition Language (PDDL) [7] formalizes classical planning problems as a tuple (D,P), where D is a domain and P is a problem instance. The domain is defined as  $D=(\mathcal{O},\mathcal{P})$ , consisting of a set of operators  $\mathcal{O}$  and predicates  $\mathcal{P}$ . Each predicate  $p\in\mathcal{P}$  is a Boolean-valued function over typed objects, representing properties or relations among entities. Each operator  $o\in\mathcal{O}$  is defined by its preconditions  $\operatorname{pre}(o)\subseteq\mathcal{P}\cup\neg\mathcal{P}$  and effects  $\operatorname{eff}(o)\subseteq\mathcal{P}\cup\neg\mathcal{P}$ , where  $\neg\mathcal{P}$  denotes the set of negated predicates. These preconditions and effects describe the logical requirements and state transitions induced by executing o. The problem instance is defined as  $P=(\mathcal{B},s_0,s_g)$ , where  $\mathcal{B}$  is the object set,  $s_0$  is the initial state given as a grounded conjunction of predicates, and  $s_g$  is a partially specified target state. A planner searches for an action sequence  $A=\langle a_1,a_2,\ldots,a_T\rangle$ , where each  $a_t$  is a grounded instance of an operator from  $\mathcal{O}$ , such that applying A to  $s_0$  results in a state satisfying  $s_g$ . Given (D,P), off-the-shelf symbolic planners such as Fast Downward [8] use heuristic search to compute a valid plan that transitions the world from  $s_0$  to a goal state. The symbolic structure of PDDL enables interpretable, verifiable, and constraint-aware planning in complex domains.

#### 2.2 LLM-based Task Planning

Large Language Models (LLMs) [1, 24] exhibit strong commonsense reasoning and structured generation, making them appealing for robotic task planning. Recent work translates free-form instructions into code-like plans (e.g., Code-as-Policies [21], ProgPrompt [25]), filters actions using affordance and cost (SayCan [26], SayCanPay [27]), or integrates feedback via closed-loop reasoning (ReAct [22], Inner Monologue [28], Reflexion [29]). Others incorporate search (LLM-MCTS [30]), symbolic grounding (Chain-of-Symbol [31]), or multimodal inputs (ViLa [32]). Yet, LLMs alone often fail to enforce implicit constraints and produce coherent long-horizon plans, motivating their integration with structured symbolic systems, such as PDDL.

# 2.3 Integration of PDDL and LLM Planning

Recent work has explored hybrid LLM-PDDL approaches to task planning. One common paradigm fixes the PDDL domain and uses LLMs to generate problems. For example, LLM+P [4], AutoGPT+P [9], and ViLaIn [10] translate natural language into PDDL problems conditioned on predefined domains, then solve them using classical planners. Some frameworks [5, 6] combine this with adaptive planning. However, their reliance on manually crafted domains requires significant effort from PDDL experts and restricts generalization across tasks and environments.

Several works attempted to construct PDDL domains directly from language instructions. ISR-LLM [23] generates a domain-problem pair from language. NL2Plan [13] uses five stages of LLM-based verification to iteratively construct a PDDL domain. BoN-iVML [14] uses Best-of-N sampling followed by iterative refinement to generate a domain. The domains generated by these methods have limited quality due to restricted information. Other work attempts to iteratively construct PDDL domains through external feedback. InterPreT [12] and LLM-DM [11] incorporate human-in-the-loop to iteratively refine operators. Ada [33] and LASP [34] use environment feedback to refine the domain. While effective in simulation, these methods are impractical for scalable deployment due to the cost, latency, and noise in external feedback. In contrast, our method uses internal closed-loop validation with synthetic test problems, which automates domain learning without human input.

There also exist a few works that learn PDDL domains from visual demonstration. BLADE [35] learns a domain from a robot manipulation trajectory with a given set of actions. pix2pred [36] extracts operators from visual demonstrations given a predefined set of predicates. Diehl et al. [37] and Huang et al. [38] automate the generation of a robotic planning domain from single or a few repeated demonstrations in simulation. These works typically aim to generate a narrow domain

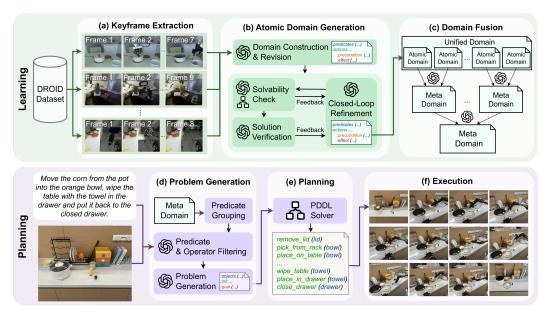


Figure 2: Overview of UniDomain. See detailed descriptions in Section 3.

tied to the demonstrated task, and require task-specific priors (action, predicate sets or in-domain demonstrations) as input. UniDomain, however, learns a unified, general-purpose PDDL domain from large-scale robotic manipulation datasets, to enable compositional generalization and support symbolic task planning in diverse manipulation tasks under complex constraints.

#### 3 Method Overview

*UniDomain* (Figure 2) is a three-phase framework for vision-language-conditioned task planning with symbolic structure. In the first phase, *Domain Pretraining*, atomic PDDL domains are extracted from visual-language robot demonstrations using keyframe extraction (Figure 2a), VLM-based domain construction, and LLM-based closed-loop refinement (Figure 2b). These domains collectively form a unified domain capturing broad manipulation knowledge. In the second phase, *Domain Fusion* (Figure 2c), task-class-relevant atomic domains are retrieved and hierarchically merged into a compact meta-domain by aligning functionally-overlapping predicates and operators. In the final phase, *Online Planning*, a task instruction and scene image are used to construct a grounded PDDL problem (Figure 2d), which is solved with a classical planner (Figure 2e) using the fused meta-domain to generate executable plans (Figure 2f).

# 4 Domain Pretraining

This section pre-trains a general PDDL domain for robot manipulation tasks, which aggregates a large set of atomic domains learned from the DROID [18] dataset. Given each visual-language demonstration, UniDomain generates an atomic domain via two steps: unsupervised keyframe segmentation (Section 4.1) and LLM-guided domain construction with closed-loop verification (Section 4.2).

#### 4.1 Energy-Based Keyframe Extraction

Keyframe extraction is a well-studied problem, though many robotics approaches rely on task-specific signals such as human-object contact or end-effector trajectories [39, 40]. Other methods [41] extract keyframes based on frame similarity using embeddings from pretrained vision-language models like CLIP [42] or SigLIP [43], but these approaches incur high computational costs. We propose a simple, domain-agnostic method that identifies semantic transitions in videos by detecting changes in

grayscale intensity (Figure 2a). Let  $I_t \in \mathbb{R}^{W \times H}$  denote the grayscale version of the input frame t. We define the frame energy as

$$E(I_t) = \sum_{i=1}^{W} \sum_{j=1}^{H} I_t(i,j)^2.$$
 (1)

The energy sequence  $E(I_t)_{t=1:T}$  is computed across all frames, and keyframes are selected by identifying local extrema using a sliding window of size K: a frame t is chosen as a keyframe if

$$E_t = \max_{i \in [t-K, t+K]} E_i \quad \text{or} \quad E_t = \min_{i \in [t-K, t+K]} E_i.$$
 (2)

# 4.2 Closed-Loop Atomic Domain Generation

Given an ordered keyframe sequence  $\{k_1,\ldots,k_N\}$  and the associated task instruction T, we construct a symbolic PDDL domain through a multi-stage LLM-guided pipeline (Figure 2b). For each transition  $(k_i,k_{i+1})$ , a vision-language model (VLM) infers the operator name, identifies preconditions and effects, and expands the predicate set if necessary, yielding an initial grounded domain  $D_0$ . To improve consistency and generality, we pass  $D_0$  and T through a large language model (LLM) for holistic revision. The LLM enforces syntactic correctness, predicate reuse, and naming consistency, producing a revised domain  $D_r$ . We then apply two nested verification steps to refine the domain:

**Solvability Check.** To assess domain correctness, we prompt the LLM with  $(\mathcal{P}_r, T)$  to generate K test problems  $\{P_1, \ldots, P_K\}$  of increasing difficulty. Only the predicate set  $\mathcal{P}_r$  is used for problem generation, preventing the LLM from compensating for incorrect operators in  $D_r$ . Each pair  $(D_r, P_k)$  is evaluated by a PDDL planner. The solvability score is defined as

$$S(D_r) = \frac{1}{K} \sum_{k=1}^{K} \mathbb{I}\left[\text{PDDLSolver}(D_r, P_k) \text{ solves } P_k\right]. \tag{3}$$

If  $S(D_r) < \theta$  (default K=5,  $\theta=0.6$ ), the full, verbose feedback from the PDDL planner, including search process logs and any validation errors, is passed back to prompt the LLM for domain refinement, resulting in an updated atomic domain  $D_s$ .

**Solution Verification.** We then verify the solution to the solvable and most challenging test problem,  $A_K = \text{PDDLSolver}(D_s, P_K)$ , using another LLM to check whether the plan satisfies physical constraints and commonsense expectations. The LLM reads the action sequence and identifies steps that violate physical or operational commonsense. For example, it flags errors such as trying to pick an occluded object, stacking an object on itself, or applying an unsupported action to objects. If any violations are found, LLM feedback is used to prompt further domain refinement.

The two nested checks are repeated until both pass or a maximum of L=5 iterations is reached. If convergence fails, the learned atomic domain is discarded. The entire closed-loop process can be restarted to regenerate the domain.

#### 4.3 The Unified Domain

Using the domain learning pipeline described above, we process a total of **12,393** demonstrations from DROID, each yielding a corresponding atomic domain. While each atomic domain  $D_i$  captures task-specific knowledge grounded in a single demonstration, the complete set forms a comprehensive unified domain that spans the full task space present in the dataset. Each atomic domain can be interpreted as a minimal symbolic knowledge graph  $D=(\mathcal{V},\mathcal{E})$ , where the vertex set  $\mathcal{V}=\mathcal{P}\cup\mathcal{O}$  includes predicates  $p\in\mathcal{P}$  and operators  $o\in\mathcal{O}$ , and the edge set  $\mathcal{E}=\mathcal{E}_{\mathrm{pre}}\cup\mathcal{E}_{\mathrm{eff}}$  encodes preconditions  $(p\stackrel{\mathrm{pre}}{\longrightarrow} o)$  and effects  $(o\stackrel{\mathrm{eff}}{\longrightarrow} p)$ . Aggregating all atomic domains (by taking the union of predicate and operator sets and merging directly-overlapping nodes), the unified domain forms a large-scale symbolic knowledge graph for real-world robotic task planning (Figure 1), comprising **3,137** operator nodes grouped into **170** semantic categories, **2,875** predicate nodes, and **16,481** causal edges. This unified representation connects otherwise isolated behaviors such as pick\_from\_table, pour\_into\_bowl, stir\_bowl, remove\_lid, place\_in\_pot, and wipe\_table, enabling the planner to solve long-horizon tasks like "heat the milk using the pot and clean the table" through compositional generalization across atomic domains.

Despite its broad coverage, the unified domain is not directly suitable for online planning. Its large scale poses challenges for LLMs and VLMs, making task grounding less reliable and increasing computational overhead. Second, semantic inconsistencies across atomic domains—such as varying predicate names for equivalent concepts—break symbolic continuity and reduce planning effectiveness. Therefore, to ensure the quality of domains and enable effective planning, for a specific task class, we first retrieve a relevant subset of atomic domains based on the relevance of language instructions, and then fuse them into a compact, high-quality meta-domain.

#### 5 Domain Fusion

With a retrieved set of task-relevant atomic domains,  $\{D_i\}_{i=1:M}$ , this section constructs a metadomain,  $D = \bigcup_i D_i$ , an integrated graph with improved symbolic and causal connectivity, so that it better supports generalization across task variations. This is achieved by merging functionally-overlapping nodes, via hierarchical fusion along a binary tree (Figure 2c).

#### 5.1 Atomic Domain Retrieval

The set of task-relevant atomic domains can be retrieved either manually or automatically. For manual retrieval, atomic domains can be selected based on whether the associated language instructions show relevance to the target task class. For automatic retrieval, we prompt an LLM to infer the set of relevant actions based on the language description of the target task class. We then use sentence embedding similarity to find the top-K matching operators in the unified domain. Atomic domains containing these relevant operators are thus retrieved for fusion.

# 5.2 Binary Tree Fusion

We construct the meta-domain by recursively merging a set of atomic domains along a binary tree. At each level l, a parent node  $D_k^l$  is formed by fusing its two arbitrarily-paired child domains:  $D_k^l = f(D_{2k-1}^{l+1}, D_{2k}^{l+1})$ , where  $f(\cdot, \cdot)$  performs structured alignment of predicates and operators. This process proceeds bottom-up until a single, unified meta-domain resides at the root. Each node fusion is performed in two stages:

**Predicate Merging.** Let  $\mathcal{P}_1$  and  $\mathcal{P}_2$  be the predicate sets from two child domains. We compute predicate similarity using  $\phi(p_i,p_j)=\cos(\mathrm{E}(p_i),\mathrm{E}(p_j))$ , where  $\mathrm{E}(\cdot)$  denotes a text embedding from a pretrained language model [44]. Predicate pairs with  $\phi<\tau_p$  (with  $\tau_p=0.3$ ) are discarded. The remaining candidates are ranked by similarity and sequentially verified by an LLM for semantic equivalence. Equivalent predicates are merged.

**Operator Merging.** Following predicate alignment, we update all operators referencing merged predicates using LLM assistance. Operator similarity is computed via name embeddings:  $\phi_{\text{name}}(o_1, o_2) = \cos(\mathrm{E}(\mathrm{name}(o_1)), \mathrm{E}(\mathrm{name}(o_2)))$ . Pairs with  $\phi_{\text{name}} < \tau_o$  (with  $\tau_o = 0.3$ ) are filtered out. The remaining operator pairs are ranked and passed to the LLM along with their (name, pre, eff) tuples. Functionally equivalent ones are merged, inheriting the union of their preconditions and effects.

# 6 Task Planning with UniDomain

Given a new task specified by a language instruction T and a scene image I, we apply the meta-domain D for symbolic planning. The pipeline consists of two stages: (1) constructing a task-specific PDDL problem P, and (2) solving the pair (D,P) using a classical planner such as Fast Downward [8]. A central challenge is filtering task-relevant elements from D to reduce symbolic noise and improve solver efficiency.

**Predicate Grouping.** To help the LLM interpret the large predicate set  $\mathcal{P}$  in D, we pre-organize predicates into four semantic groups: object category descriptors, state or attribute indicators, spatial relations, and affordance-related predicates. This structured input improves the reliability of downstream problem construction.

**Predicate and Operator Filtering.** We first prompt a vision-language model with (D, I, T) to generate an initial problem:  $P_0 = \text{LLM}(D, I, T)$ . From  $P_0$ , we extract the predicate set  $\mathcal{P}_0$  used in

the initial and goal conditions, treating these as task-relevant predicates. Next, we extract from D the operators whose preconditions or effects involve

$$\mathcal{P}_0: \mathcal{O}_{\text{pre}} = \{ o \in \mathcal{O}: \exists p \in \mathcal{P}_0, \ p \xrightarrow{\text{pre}} o \}, \mathcal{O}_{\text{eff}} = \{ o \in \mathcal{O}: \exists p \in \mathcal{P}_0, \ o \xrightarrow{\text{eff}} p \},$$
(4)

and define the reduced operator set

$$\mathcal{O}' = \mathcal{O}_{\text{pre}} \cup \mathcal{O}_{\text{eff}}.\tag{5}$$

Using  $\mathcal{O}'$ , a compact domain  $D_{\text{new}} = (\mathcal{P}_0, \mathcal{O}')$  is constructed, and a refined problem is generated:  $P_{\text{new}} = \text{LLM}(D_{\text{new}}, I, T)$ .

**PDDL Planning.** We then solve  $(D, P_{\text{new}})$  using a symbolic planner,  $A = \text{PDDLSolver}(D, P_{\text{new}})$ , yielding an action sequence A that satisfies  $s_g$  under the symbolic constraints and minimizes cost. This filtering process improves both planning accuracy and computational efficiency by reducing irrelevant symbolic clutter.

# 7 Experiments

We evaluate UniDomain on diverse real-world tasks. Results demonstrate that UniDomain achieves substantial improvements over the strongest baseline methods, obtaining up to 58% higher task success rate and 160% higher plan optimality. Specifically, UniDomain maintains consistently high success and optimality across diverse and previously unseen tasks, significantly reducing planning overhead compared to both LLM-only planners (e.g., Code-as-Policies [21] and ReAct [22]) and hybrid LLM-PDDL methods (e.g., ISR-LLM [23] and BoN-iVML [14]). Further ablation studies confirm that these performance gains result from learning a comprehensive domain via closed-loop verification and structured fusion and effectively filtering irrelevant predicates and operators during planning.

#### 7.1 Experimental Setup

**Tasks.** The evaluation tasks span 4 unseen task domains: *BlockWorld*, *Desktop*, *Kitchen*, and *Combination*. *BlockWorld* involves block sorting and stacking with ordering constraints; *Desktop* includes drawer use, wiping, folding, and document organization; *Kitchen* covers object transfers and food-tool manipulation; *Combination* mixes all domains to test cross-context generalization. There are 100 tasks in total. 40 atomic domains learned from DROID demonstrations are retrieved to construct a meta-domain for all evaluation tasks, which includes 78 predicates and 61 operators. See details of the evaluation tasks and the meta-domain in Appendix B and Appendix C, respectively.

**Evaluation Metrics.** We report success rate (SR), success-weighted relative path length (SPL), and optimality rate (OR), defined as the fraction of plans whose cost  $c_i$  falls within a threshold K of the optimal  $c_i^*$ . Specifically,

$$SPL = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}_{succ} \cdot \frac{c_i^*}{c_i}, \tag{6}$$

$$OR(K) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I} \left[ 0 < c_i \le c_i^* + K \right].$$
 (7)

We additionally report the thinking time (LLM wall-clock runtime) and number of LLM calls per task to assess efficiency and overhead.

**Baselines.** We compare UniDomain against two categories of methods. The first uses LLMs or VLMs as planners: Code-as-Policies [21] directly generates executable Python-style plans from language instructions; ReAct [22] improves robustness through closed-loop reasoning with feedback; VLM-CoT applies chain-of-thought prompting [31] in a zero-shot vision-language setting. The second category integrates LLMs with PDDL planning: ISR-LLM [23] translates instructions into PDDL specifications for building LLM planning and iteratively refines plans with validator feedback; VLM-PDDL grounds scene and language into symbolic specifications and plans with classical solvers; BoN-iVML [14] generates an initial PDDL domain via Best-of-N sampling, refines it with verbalized feedback, and then constructs the problem file for planning.

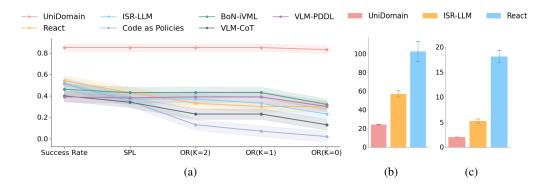


Figure 3: Comparison results of UniDomain and state-of-the-art methods on unseen evaluation tasks: (a) success rates  $\uparrow$ , success-weighted relative path lengths  $\uparrow$ , and optimality rates with thresholds  $(K=2,1,0)\uparrow$ ; (b) Thinking time (s)  $\downarrow$  of the top-performing methods; (c) number of LLM calls  $\downarrow$  of the top-performing methods. Average values are shown with standard errors.

**Evaluation Protocol.** We evaluate UniDomain as a high-level task planner, not as an integrated robot system. To focus evaluation on the performance of high-level symbolic planning, we followed a standard practice in the task planning literature [45, 46] and assumed a perfect low-level control policy (human teleoperation in our experiments for both UniDomain and baselines), so that the measured performance does not get confounded with potential imperfections in the low-level controller. We used a semi-automatic evaluation approach, wherein an LLM reads the task and the plan to provide an initial assessment, followed by final verification by human experts. Example results in Appendix E show that these judgments reflect well-defined objectives and commonsense constraints.

Despite the stand-alone evaluation, UniDomain is ready for seamless integration into a complete robotic system. Its high-level plan can be straightforwardly translated into natural language commands and input to any low-level language-conditioned skill policy, like modern Vision-Language-Action (VLA) models [15, 17], or modular approaches combining perception, motion planning, and affordance learning. See real-world demonstrations of such an integrated system built upon UniDomain on our project website: https://roboticsjtu.github.io/UniDomain/.

#### 7.2 Comparison Results

Figure 3a reports performance across three metrics: *Success Rate*, *SPL*, and *Optimality Rate* at increasing strictness levels (K=2, 1, 0), transitioning focus from task feasibility to plan optimality. All methods were evaluated using GPT-4.1 via API under a fixed temperature of 0.0.

Among LLM-only planners, *Code-as-Policies* achieves moderate success (51%) but degrades rapidly under stricter thresholds, highlighting its limited global reasoning capacity. *VLM-CoT* produces slightly more optimal plans due to stronger visual grounding, but struggles with task completion due to the absence of symbolic structure. *ReAct* yields the highest success among LLM-only methods by leveraging action feedback, yet its lack of explicit state tracking results in redundant or invalid steps—especially on long-horizon tasks.

For PDDL-integrated planners, *VLM-PDDL* performs comparably to *VLM-CoT*, but is hindered by fragile domain and problem generation—minor grounding or typing errors often lead to unsolvable plans. *ISR-LLM* achieves the highest task success rate among all baselines through iterative validator feedback, but its reliance on LLM-based planning results in sharp optimality drops (matching *VLM-CoT* at *K*=0). *BoN-iVML* improves over *VLM-PDDL* with verbalized refinement but still fails to construct reliable high-quality domains on the fly.

In contrast, UniDomain achieves strong and consistent performance across all metrics, attaining 85% success rate and 83% optimality at K=0. It also incurs the lowest LLM thinking time and fewest LLM calls among top-performing methods (Figures 3b and 3c). The meta-domain effectively supports compositional generalization. In 83% of tasks, our planner successfully produced not only feasible but also optimal plans through the composition of learned operators in unified domain. These results show that pre-training the unified domain, paired with post-training using domain fusion and

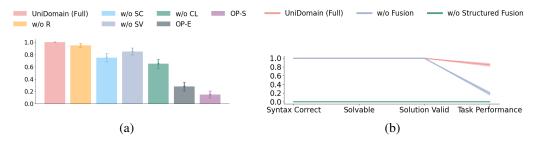


Figure 4: Results for ablation studies on domain generation: (a) ablation on the atomic domain learning method; (b) ablation on the domain fusion method. All values are success rates \( \gamma with standard errors.

test-time problem generation, enables symbolic planning to outperform both end-to-end LLM agents and the best existing hybrid planners in robustness, efficiency, and plan quality.

#### 7.3 Ablation Studies

We conduct ablation studies to understand the contributions of core components in UniDomain. Results show that removing the closed-loop verification significantly reduces atomic domain quality, causing failures in solvability and task logic. Hierarchical fusion is critical, as a naive union of atomic domains or direct LLM-based merging yields unusable domains due to semantic and structural inconsistencies. Additionally, predicate grouping and task-relevant filtering substantially boost planning performance, particularly in tasks requiring complex reasoning and compositional generalization.

**Ablations on Domain Learning.** In Figure 4a, we perform ablation studies on our atomic domain learning pipeline using 40 DROID demonstrations with paired instructions. A domain is considered successful if it passes both the solvability check, ensuring that test problems can be solved by a PDDL planner, and the plan verification step, which checks whether the resulting plans conform to real-world physical constraints and commonsense logic. We report the success rate as the primary metric.

To assess the contribution of each module, we compare against several ablated variants. Removing the LLM-based revision step (w/o R) results in domains with more syntax or logical inconsistencies, increasing the average number of required refinement iterations (from 0.49 to 1.36). Disabling the solvability check (w/o SC) leads to domains that are syntactically valid but often fail due to disconnected operators and incomplete predicates. Removing the solution verification stage (w/o SV) produces domains that are solvable but fail to capture essential task logic, resulting in mis-aligned plans. Eliminating all feedback mechanisms (w/o CL) reduces the process to single-pass LLM generation, which significantly degrades domain quality.

We also evaluate one-shot variants that use no refinement or validation, generating the domain from a single LLM query. Using our energy-based keyframe extraction (*OP-E*) achieves 28% single-pass success, while the similarity-based approach [41] (*OP-S*) drops to 15%. In addition to higher accuracy, our energy-based method is also substantially more efficient, reducing the processing time per demonstration from 47.8 seconds to just 0.6 seconds on average. <sup>2</sup>

**Ablations on Domain Fusion.** In Figure 4b, we perform ablation studies on our hierarchical domain fusion method using test problems generated from evaluation tasks (sampled from those in Section 7.2), assessing whether the fused domain supports correct and generalizable planning. We report four metrics: syntax validity (as measured by a PDDL syntax verifier), solvability (the rate of passing solvability tests), solution validity (the rate of passing plan verification), and task performance measured as the success rate of online planning.

Our full method, UniDomain, achieves perfect scores on syntax, solvability, and verification, and attains 85% task success. In contrast, using atomic domains without fusion (*w/o Domain Fusion*)—where the planner selects the closest atomic domain for each task, akin to retrieval-based

<sup>&</sup>lt;sup>2</sup>We performed similarity-based keyframe extraction using SigLIP-2 [47] on an NVIDIA A800 GPU (80GB VRAM), running in parallel across batches. The energy-based method was executed in single-threaded mode on an i7-14700HX CPU (32GB RAM).



Figure 5: Results for ablation study of the UniDomain planner. Each bar shows average task success rates  $\uparrow$  with standard errors.

methods—yields only 19% success, despite perfect syntax and solvability. This confirms the power of compositional generalization. Furthermore, replacing our structured fusion with a direct LLM-based merging strategy (*w/o Structured Fusion*) fails entirely: the merged domains contain structural errors, violate syntax rules, and are unusable by downstream planners.

**Ablations on the Planning Method.** In Figure 5, we assess the impact of predicate organization and domain filtering on planning performance by measuring task success rates across all evaluation tasks. In *w/o Grouping*, we remove the structural organization of predicates into semantic categories, providing the full flat list to the LLM during problem generation. In *w/o Filtering*, we disable pruning based on task relevance and directly use the full meta-domain to perform a single-pass problem generation.

Removing predicate grouping degrades performance significantly, particularly in the *Combination* domain, where complex task composition requires the LLM to interpret a rich and diverse predicate space. Without grouping, the flat structure overwhelms the LLM's capacity to localize task-relevant semantics. Disabling predicate and operator filtering also leads to sharp performance drops, especially in the *BlockWorld* domain. These tasks rely on long-horizon action dependencies, and a compact, task-focused domain allows more coherent grounding and reasoning by reducing irrelevant information.

# **8 Conclusion and Limitations**

We present UniDomain, a framework that addresses the challenge of task planning under complex, implicit constraints from language and vision. UniDomain learns a reusable PDDL domain from large-scale visual demonstrations and applies it to zero-shot symbolic planning. By combining closed-loop domain learning, hierarchical fusion, and task-relevant filtering, UniDomain enables efficient and generalizable planning across diverse tasks. Experiments on 100 real-world task instances demonstrate that UniDomain substantially outperforms prior LLM-only and hybrid LLM-PDDL baselines, achieving higher success rates and plan optimality.

Despite its strong performance, UniDomain has a few limitations. First, automatically-retrieved atomic domains can be redundant, thus construction of a meta-domain can be time-consuming. Future work will focus on improving the accuracy and efficiency of the domain retrieval and fusion methods. Second, UniDomain operates under the PDDL 1.0 formalism, which lacks support for temporal constraints, numeric fluents, and cost-sensitive planning. Extending the framework to richer representations such as PDDL 2.1 [48] is an important direction. Finally, our experiments assume full observability, ignoring real-world challenges like occlusion and perceptual noise. Incorporating probabilistic planning frameworks such as PPDDL [49] or RDDL [50] is a promising path toward handling uncertainty.

# Acknowledgements

This work was supported in part by the National Key R&D Program of China (Grant No. 2024YFB4707600) and National Natural Science Foundation of China under grant No. 62303304.

We used generative AI to improve self-written texts to enhance readability. None of the presented methods and results (figures, equations, numbers, etc.) are generated by AI.

# References

- [1] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv* preprint arXiv:2410.21276, 2024.
- [2] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.
- [3] Cassie Huang and Li Zhang. On the limit of language models as planning formalizers. *arXiv* preprint arXiv:2412.09879, 2024.
- [4] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv* preprint arXiv:2304.11477, 2023.
- [5] Gautier Dagan, Frank Keller, and Alex Lascarides Keller. Dynamic planning with an llm. In *Proceedings of the Language Gamification Workshop 2024 at NeurIPS*, pages 1–14. Neural Information Processing Systems Foundation (NeurIPS), October 2024. Language Gamification Workshop 2024 at NeurIPS; Conference date: 14-12-2024 Through 14-12-2024.
- [6] Li Zhang, Peter Jansen, Tianyi Zhang, Peter Clark, Chris Callison-Burch, and Niket Tandon. PDDLEGO: Iterative planning in textual environments. In Danushka Bollegala and Vered Shwartz, editors, *Proceedings of the 13th Joint Conference on Lexical and Computational Semantics* (\*SEM 2024), pages 212–221, Mexico City, Mexico, June 2024. Association for Computational Linguistics.
- [7] Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins Sri, Anthony Barrett, Dave Christianson, et al. Pddll the planning domain definition language. *Technical Report, Tech. Rep.*, 1998.
- [8] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.
- [9] Timo Birr, Christoph Pohl, Abdelrahman Younes, and Tamim Asfour. Autogpt+ p: Affordance-based task planning with large language models. *arXiv preprint arXiv:2402.10778*, 2024.
- [10] Keisuke Shirai, Cristian C. Beltran-Hernandez, Masashi Hamaya, Atsushi Hashimoto, Shohei Tanaka, Kento Kawaharazuka, Kazutoshi Tanaka, Yoshitaka Ushiku, and Shinsuke Mori. Visionlanguage interpreter for robot task planning. In 2024 IEEE International Conference on Robotics and Automation (ICRA), pages 2051–2058, 2024.
- [11] Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.
- [12] Muzhi Han, Yifeng Zhu, Song-Chun Zhu, Ying Nian Wu, and Yuke Zhu. Interpret: Interactive predicate learning from language feedback for generalizable task planning. *arXiv* preprint *arXiv*:2405.19758, 2024.
- [13] Elliot Gestrin, Marco Kuhlmann, and Jendrik Seipp. Towards robust LLM-driven planning from minimal text descriptions. In ICAPS 2024 Workshop on Human-Aware Explainable Planning, 2024.
- [14] Zhouliang Yu, Yuhuan Yuan, Tim Z. Xiao, Fuxiang Frank Xia, Jie Fu, Ge Zhang, Ge lin, and Weiyang Liu. Generating symbolic world models via test-time scaling of large language models. *Transactions on Machine Learning Research*, 2025.
- [15] Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. RDT-1b: a diffusion foundation model for bimanual manipulation. In *The Thirteenth International Conference on Learning Representations*, 2025.

- [16] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Open-VLA: An open-source vision-language-action model. In 8th Annual Conference on Robot Learning, 2024.
- [17] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky.  $\pi_0$ : A vision-language-action flow model for general robot control, 2024.
- [18] Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024.
- [19] Ce Zhou, Qian Li, Chen Li, Jun Yu, Yixin Liu, Guangjing Wang, Kai Zhang, Cheng Ji, Qiben Yan, Lifang He, et al. A comprehensive survey on pretrained foundation models: A history from bert to chatgpt. *International Journal of Machine Learning and Cybernetics*, pages 1–65, 2024.
- [20] Guiyao Tie, Zeli Zhao, Dingjie Song, Fuyang Wei, Rong Zhou, Yurou Dai, Wen Yin, Zhejian Yang, Jiangyue Yan, Yao Su, et al. A survey on post-training of large language models. *arXiv* preprint arXiv:2503.06072, 2025.
- [21] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In 2023 IEEE International Conference on Robotics and Automation (ICRA), pages 9493–9500, 2023.
- [22] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.
- [23] Zhehua Zhou, Jiayang Song, Kunpeng Yao, Zhan Shu, and Lei Ma. Isr-llm: Iterative self-refined large language model for long-horizon sequential task planning. In 2024 IEEE International Conference on Robotics and Automation (ICRA), pages 2081–2088, 2024.
- [24] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.
- [25] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In 2023 IEEE International Conference on Robotics and Automation (ICRA), pages 11523–11530, 2023.
- [26] brian ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do as i can, not as i say: Grounding language in robotic affordances. In 6th Annual Conference on Robot Learning, 2022.
- [27] Rishi Hazra, Pedro Zuidberg Dos Martires, and Luc De Raedt. Saycanpay: Heuristic planning with large language models using learnable domain knowledge. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20123–20133, 2024.
- [28] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan James Richard Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Andrew Ichter.

- Innermonologue: Embodied reasoning through planning with language models. 2022. CoRL 2022 (to appear).
- [29] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652. Curran Associates, Inc., 2023.
- [30] Zirui Zhao, Wee Sun Lee, and David Hsu. Large language models as commonsense knowledge for large-scale task planning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 31967–31987. Curran Associates, Inc., 2023.
- [31] Hanxu Hu, Hongyuan Lu, Huajian Zhang, Yun-Ze Song, Wai Lam, and Yue Zhang. Chain-of-symbol prompting for spatial reasoning in large language models. In *First Conference on Language Modeling*, 2024.
- [32] 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. In *First Workshop on Vision-Language Models for Navigation and Manipulation at ICRA 2024*, 2024.
- [33] Lio 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.
- [34] Guanqi Chen, Lei Yang, Ruixing Jia, Zhe Hu, Yizhou Chen, Wei Zhang, Wenping Wang, and Jia Pan. Language-augmented symbolic planner for open-world task planning. *arXiv* preprint *arXiv*:2407.09792, 2024.
- [35] Weiyu Liu, Neil Nie, Ruohan Zhang, Jiayuan Mao, and Jiajun Wu. Learning compositional behaviors from demonstration and language. In 8th Annual Conference on Robot Learning, 2024.
- [36] Ashay Athalye, Nishanth Kumar, Tom Silver, Yichao Liang, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Predicate invention from pixels via pretrained vision-language models. *arXiv preprint arXiv:2501.00296*, 2024.
- [37] Maximilian Diehl, Chris Paxton, and Karinne Ramirez-Amaro. Automated generation of robotic planning domains from observations. In 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 6732–6738. IEEE, 2021.
- [38] Jinbang Huang, Allen Tao, Rozilyn Marco, Miroslav Bogdanovic, Jonathan Kelly, and Florian Shkurti. Automated planning domain inference for task and motion planning. In 2025 IEEE International Conference on Robotics and Automation (ICRA), pages 12534–12540. IEEE, 2025.
- [39] Zhenyu Jiang, Yuqi Xie, Kevin Lin, Zhenjia Xu, Weikang Wan, Ajay Mandlekar, Linxi Fan, and Yuke Zhu. Dexmimicgen: Automated data generation for bimanual dexterous manipulation via imitation learning. In *CoRL Workshop on Learning Robot Fine and Dexterous Manipulation: Perception and Control*, 2024.
- [40] Huayi Zhou, Ruixiang Wang, Yunxin Tai, Yueci Deng, Guiliang Liu, and Kui Jia. You only teach once: Learn one-shot bimanual robotic manipulation from video demonstrations. *arXiv* preprint arXiv:2501.14208, 2025.
- [41] Pingping Zhang, Jinlong Li, Kecheng Chen, Meng Wang, Long Xu, Haoliang Li, Nicu Sebe, Sam Kwong, and Shiqi Wang. When video coding meets multimodal large language models: A unified paradigm for video coding. *arXiv preprint arXiv:2408.08093*, 2024.
- [42] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.

- [43] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 11975–11986, 2023.
- [44] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding. *Advances in neural information processing systems*, 33:16857–16867, 2020.
- [45] Pierre Sermanet, Tianli Ding, Jeffrey Zhao, Fei Xia, Debidatta Dwibedi, Keerthana Gopalakrishnan, Christine Chan, Gabriel Dulac-Arnold, Sharath Maddineni, Nikhil J Joshi, et al. Robovqa: Multimodal long-horizon reasoning for robotics. In 2024 IEEE International Conference on Robotics and Automation (ICRA), pages 645–652. IEEE, 2024.
- [46] Wenqi Zhang, Mengna Wang, Gangao Liu, Xu Huixin, Yiwei Jiang, Yongliang Shen, Guiyang Hou, Zhe Zheng, Hang Zhang, Xin Li, et al. Embodied-reasoner: Synergizing visual search, reasoning, and action for embodied interactive tasks. *arXiv preprint arXiv:2503.21696*, 2025.
- [47] Michael Tschannen, Alexey Gritsenko, Xiao Wang, Muhammad Ferjad Naeem, Ibrahim Alabdulmohsin, Nikhil Parthasarathy, Talfan Evans, Lucas Beyer, Ye Xia, Basil Mustafa, et al. Siglip 2: Multilingual vision-language encoders with improved semantic understanding, localization, and dense features. arXiv preprint arXiv:2502.14786, 2025.
- [48] Maria Fox and Derek Long. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124, 2003.
- [49] Håkan LS Younes and Michael L Littman. Ppddl1. 0: An extension to pddl for expressing planning domains with probabilistic effects. *Techn. Rep. CMU-CS-04-162*, 2:99, 2004.
- [50] Scott Sanner et al. Relational dynamic influence diagram language (rddl): Language description. *Unpublished ms. Australian National University*, 32:27, 2010.
- [51] Michel Plancherel and Mittag Leffler. Contribution à l'étude de la représentation d'une fonction arbitraire par des intégrales définies. *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, 30(1):289–335, 1910.
- [52] Qingwen Bu, Jisong Cai, Li Chen, Xiuqi Cui, Yan Ding, Siyuan Feng, Shenyuan Gao, Xindong He, Xuan Hu, Xu Huang, et al. Agibot world colosseo: A large-scale manipulation platform for scalable and intelligent embodied systems. *arXiv preprint arXiv:2503.06669*, 2025.
- [53] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In 2011 IEEE international conference on robotics and automation, pages 1470–1477. IEEE, 2011.
- [54] Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning compositional models of robot skills for task and motion planning. *The International Journal of Robotics Research*, 40(6-7):866–894, 2021.
- [55] Meng Guo and Mathias Bürger. Geometric task networks: Learning efficient and explainable skill coordination for object manipulation. *IEEE Transactions on Robotics*, 38(3):1723–1734, 2021.

# **NeurIPS Paper Checklist**

#### 1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

Answer: [Yes]

Justification: The abstract and introduction faithfully reflect the core contributions and scope of the paper. Specifically, the paper introduces UniDomain, a novel framework that learns a unified PDDL domain from large-scale real-world robotic manipulation demonstrations and applies it to enable zero-shot task planning. The three-stage pipeline—domain pretraining, hierarchical domain fusion, and online symbolic planning—is clearly outlined in both sections and substantiated by empirical results in later sections (e.g., Sections 4–7). The claims regarding improved task success, plan optimality, and generalization across unseen domains are directly supported by the experiments on 100 real-world tasks from four distinct domains. Moreover, the abstract explicitly states the key innovations: learning from demonstrations, structured domain fusion, and zero-shot planning—all of which are realized and evaluated throughout the paper.

#### Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the
  contributions made in the paper and important assumptions and limitations. A No or
  NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

#### 2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: The paper contains a dedicated discussion of limitations in Section 8 (Conclusion and Limitations). It explicitly identifies several practical and methodological constraints of the current work. These include: (1) The reliance on PDDL 1.0, which limits expressiveness in handling temporal constraints, numeric fluents, and cost-sensitive actions; (2) The assumption of full observability, which excludes challenges such as occlusion or sensory noise frequently encountered in real-world robotic planning.

#### Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.

- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

#### 3. Theory assumptions and proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: The paper does not present any formal theoretical results, theorems, or proofs. It is an empirical and methodological contribution focused on planning system design, evaluation, and application.

#### Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

#### 4. Experimental result reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper provides comprehensive methodological and experimental details sufficient for reproducibility. It clearly specifies the dataset used (DROID), the components of the UniDomain pipeline, domain fusion hyperparameters (Section 5.1), keyframe extraction algorithm (Section 4.1), evaluation metrics and baselines (Section 7.1–7.2), and the LLM configuration (GPT-4.1 with temperature 0.0).

#### Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.

- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
- (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
- (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
- (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

#### 5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The paper plans to release core code upon acceptance and provides access to test task data to support evaluation. Specifically, the learned unified PDDL domain mentioned in Section 4.3 will be released after acceptance. While full code and data are not yet released at submission time, the authors commit to sharing them with appropriate documentation to support reproducibility, and the released task data can already assist in verifying the results. The code is available at: https://roboticsjtu.github.io/UniDomain/

#### Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

#### 6. Experimental setting/details

Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: The paper clearly describes the task domains and splits (Section 7.1), evaluation metrics (Section 7.1), ablation setups (Section 7.3), and fusion-related hyperparameters (Section 5.2). The planning setup, including use of GPT-4.1 and temperature setting, is also explicitly stated.

#### Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental
  material.

# 7. Experiment statistical significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: The paper reports standard errors for all major quantitative results, including main comparisons (Figure 3), ablations (Figures 4 and 5), and task-level success rates (Figure 9 in the Appendix). These are described as averages with standard errors and visualize them in the figure as error bars or error bands

#### Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

#### 8. Experiments compute resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: The paper reports relative execution time (e.g., planning time and LLM calls per task in Figure 3b, 3c) In the keyframe extraction experiment (Section 7.3), the computing devices (NVIDIA A800 with 80GB VRAM) and memory are described.

#### Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.

• The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

#### 9. Code of ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

Answer: [Yes]

Justification: The research does not involve human subjects, crowdsourcing, or sensitive data. All datasets used are publicly available (e.g., DROID), and the methods comply with standard academic practice. No unethical content, misuse risk, or unconsented data collection is present.

# Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

### 10. Broader impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [No]

Justification: The paper focuses on foundational research in robotics planning and does not explicitly discuss societal impacts. Potential negative applications or risks (e.g., misuse, surveillance) are not addressed.

#### Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

# 11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: The work does not release high-risk models (e.g., LLMs or generative models). The planned release is limited to a symbolic PDDL domain file derived from public robot demonstrations (Section 4.3), which poses minimal risk of misuse.

#### Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with
  necessary safeguards to allow for controlled use of the model, for example by requiring
  that users adhere to usage guidelines or restrictions to access the model or implementing
  safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do
  not require this, but we encourage authors to take this into account and make a best
  faith effort.

#### 12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper explicitly cites the DROID dataset ([17]) and other models and techniques it builds upon, including GPT-4.1 and the Fast Downward planner. The referenced assets are standard research tools or public datasets, and there is no indication that their terms of use are violated.

#### Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

#### 13. New assets

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The paper introduces a unified PDDL domain constructed from over 12,000 robot demonstrations. This asset is described in detail in Section 4.3, including the number of actions, predicates, causal edges, and its structure as a symbolic knowledge graph. The authors also state that the domain file will be released upon acceptance.

#### Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.

- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

# 14. Crowdsourcing and research with human subjects

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: The paper does not involve crowdsourcing or human participants.

#### Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

# 15. Institutional review board (IRB) approvals or equivalent for research with human subjects

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: The research does not involve human subjects and thus does not require IRB approval or equivalent review.

#### Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

#### 16. Declaration of LLM usage

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: The usage of LLMs is central to the method and is thoroughly described throughout Sections 4–6. LLMs are used for domain generation, semantic fusion, task grounding, and planning. The specific model (GPT-4.1) is also mentioned explicitly in Section 7.2 along with its temperature setting.

#### Guidelines:

• The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.

• Please refer to our LLM policy (https://neurips.cc/Conferences/2025/LLM) for what should or should not be described.

# **Appendix**

#### A The Task Included in Overview

The task scene used in the overview is shown in Figure 6. The language instruction is "Move the corn from the pot into the orange bowl, wipe the table with the towel in the drawer and put it back to the closed drawer." The meta-domain used is fused from 40 atomic domains learned from DROID demonstrations, and includes key operators for accomplishing such tasks, for example:

```
(:action pick_from_rack
    :parameters (?r ?o ?rk)
    :precondition (and (on_rack ?o ?rk) (hand_free ?r) (rack ?rk))
    :effect(and(not(on_rack ?o ?rk))(not (hand_free?r))(holding ?r?o))
(:action place_on_table
    :parameters (?r ?o ?t)
    :precondition (and (holding ?r ?o) (table ?t))
    :effect (and(on_table ?o ?t)(hand_free ?r)(clear ?o)(not (holding
       ?r ?o))))
(:action pick_from_pot
    :parameters (?r ?o ?p)
    :precondition (and (in_pot ?o ?p) (is_open ?p) (hand_free ?r))
    :effect(and(holding ?r ?o)(not(in_pot ?o ?p))(not(hand_free ?r)))
(:action open_drawer
     :parameters (?r ?x)
     :precondition (and (hand_free ?r) (drawer ?x) (not (is_open ?x)))
     :effect (and (is_open ?x))
```

The online planner generates the following PDDL problem from the language and image input (showing only core predicates):

```
(:init
    (on_rack orange_bowl rack)
    (in_pot corn pot)
    (can_wipe_table towel)
    (in_drawer towel yellow_drawer)
    (hand_free robot)
    (on lid pot)
)
(:goal
    (and
          (in_bowl corn orange_bowl)
          (wiped table)
          (in_drawer towel yellow_drawer)
          (not (is_open yellow_drawer))
)
```

The PDDL solver produces the following optimal plan:

```
(remove_lid robot lid pot)
(pick_from_rack robot orange_bowl rack)
(place_on_table robot orange_bowl table)
(pick_from_pot robot corn pot)
(put_in_bowl robot corn orange_bowl table)
(open_drawer robot yellow_drawer)
(pick_from_drawer robot towel yellow_drawer)
(wipe_table robot towel table)
(place_in_drawer robot towel yellow_drawer)
(close_drawer robot yellow_drawer)
```

Visual results of the plan execution are shown in Figure 7.



Figure 6: The initial scene of the task in the overview (Figure 2).

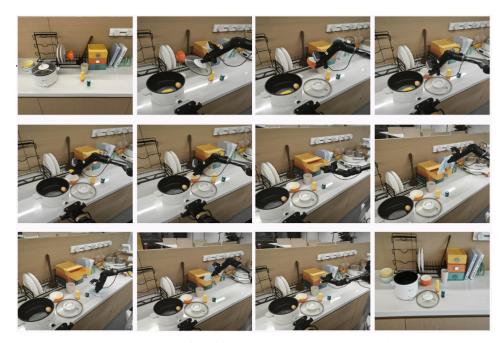


Figure 7: Execution of the high-level plan generated by UniDomain.

# **B** Evaluation Tasks

We provide representative examples of task images and corresponding language instructions for four real-world task domains—*BlockWorld*, *Desktop*, *Kitchen*, and *Combination*—each illustrated with two representative task instances used in our experimental evaluation.

# **B.1** Representative Tasks in *BlockWorld*



Figure 8: Arrange all blocks into two separate stacks on the table. The first stack should have blocks 1, 3, 5, and 7 in order from top to bottom. The second stack should have blocks 2, 4, 6, and 8 in order from top to bottom.

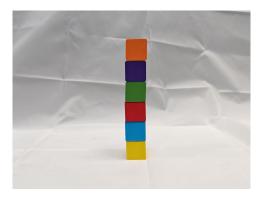


Figure 9: Arrange all blocks in a single stack (top to bottom): purple, orange, blue, green, yellow, red.

# **B.2** Representative Tasks in *Desktop*



Figure 10: There is a block in the green drawer. Please put it on the table, push it and put it in the yellow drawer.



Figure 11: There is a tissue in the yellow drawer. Put it on the table and put the white book on the pink book.

### **B.3** Representative Tasks in *Kitchen*



Figure 12: Put the jujube in the green bowl. And put the white plate on the rack.



Figure 13: Take the egg out of the pot and put it in the left bowl. Take the vegetable out of the pot and put it on the right bowl.

# **B.4** Representative Tasks in *Combination*



Figure 14: Put block 8 in the drawer, and arrange other blocks in a single stack on the table (from top to bottom): 1, 2, 3, 4, 5, 6, 7.



Figure 15: There are a spoon, a tissue, an orange block in the green drawer. Stir the bowl and put the spoon in the cup, put the orange block into the orange drawer, wipe the bowl and scrunch the tissue on the table.

# C Verb Distribution in the Unified Domain and the Meta Domain

UniDomain processes 12,393 real-world demonstrations to construct a unified domain consisting of 3,137 operators grouped into 170 verb types (Figure 16). For planning evaluation, we retrieve and fuse 40 household-related atomic domains from this unified set to form a compact meta-domain with 106 predicates (including negated predicates), 61 operators, and 332 causal edges (Figure 17). This compositional meta domain integrates a diverse set of kitchen-related verbs (Figure 18).

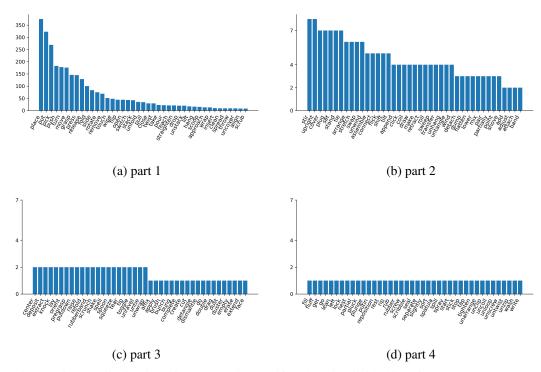


Figure 16: Verb distribution of our pre-trained unified domain, divided into four segments (a)–(d).

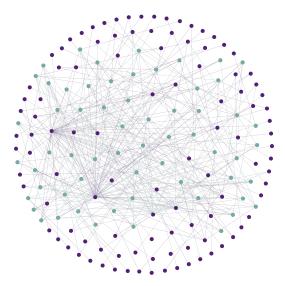


Figure 17: Visualization of meta domain used for planning across evaluation tasks, with 61 operator nodes (green) and 106 predicate nodes (purple).

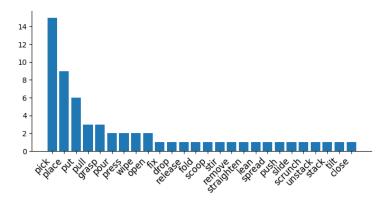


Figure 18: Verb distribution of meta domain used for planning in evaluation tasks.

# D Details and Justification on Energy-based Keyframe Extraction

**Hyperparameter.** To accommodate videos of varying lengths l, we employ an adaptive window size

$$K \text{, specified as follows: } K = \begin{cases} 10, l \leqslant 100 \\ 20, 100 < l \leqslant 150 \\ 30, 150 < l \leqslant 200 \\ 40, 200 < l \leqslant 500 \\ 40 + 10 \times \left( \lfloor \frac{l - 501}{200} \rfloor + 1 \right), l > 500 \end{cases}$$

To explain how and why our seemingly-simple keyframe extraction works, we offer both theoretical and empirical justification below.

**Theoretical Justification.** Our method of summing squared grayscale intensities is equivalent to measuring the total energy of the image treated as a 2D signal. By Parseval's theorem [51], this spatial-domain energy is proportional to the energy in the frequency domain. Semantic transitions in a video, such as an object being picked up or a drawer opening, cause significant changes in the image's structure and texture, which correspond to shifts in its frequency-space energy. Our method identifies keyframes by detecting the local extrema of this energy sequence, effectively capturing these of significant semantic change. Figure 19 shows an example set of keyframes extracted from a *DROID* demonstration via our energy-based method. The key-frames successfully captured semantic phase changes (from approach, grasp, lift, to place), echoing our theoretical justification.

**Empirical Justification.** We use the Agibot World [52] dataset, which provides human-annotated keyframes along with demonstrations, for empirical justification. We compare our automatically-extracted keyframes with human-annotated keyframes provided by Agibot World. Energy curves shown in Figure 20a and Figure 20b show that human-annotated keyframes consistently cluster around the local energy extrema we extracted, confirming the effectiveness of our method.

Connection to Primitive Actions. "Primitive actions", or "primitive skills", in a common terminology used in task planning (or task and motion planning) research [53, 54, 55]. A primitive skill is an atomic, specialized skill for manipulating one object or one object—relation tuple, such as "pick an object" or "open a container", inducing atomic, local effects, and acting as a building block for longer tasks. Our energy-based keyframe extraction detects local extrema in frame-energy curves, either a peak or a trough, that consistently occurs at primitive skill boundaries (for example, when a move ends and a pick begins). When spliting the demonstration at these extrema, each segment would span one primitive skill, resulting in the learned operator being atomic.

# E Evaluation Results and Failure Examples Per Task Class

Figure 21 reports comparison results in the four evaluation task classes separately, also evaluating three key metrics: Success Rate (SR), Success weighted by Path Length (SPL), and Optimality Rate



Figure 19: Energy-based keyframes on **DROID**. The automatically selected frames align with intuitive manipulation boundaries

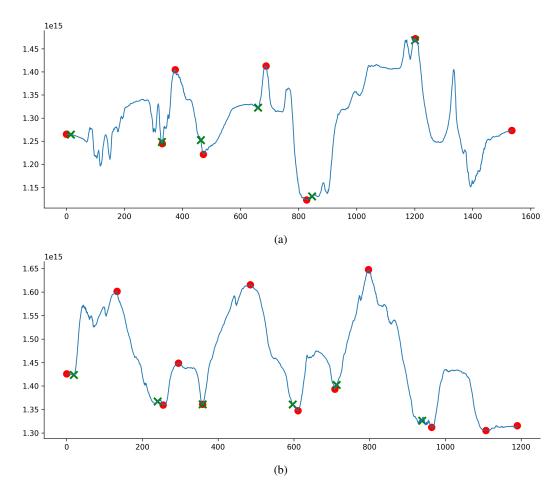


Figure 20: (a)(b) are two example of the energy curve in Agibot World datasets. Red circles: keyframes extracted by our energy-based method; green crosses: manual annotations in Agibot World. The horizontal axis represents the frame index, and the vertical axis represents the energy value.

(OR) at increasing strictness levels (K=2,1,0). Below, we provide a qualitative comparison between the best-performing and the worst-performing methods in the four task classes.

**BlockWorld.** In Figure 8, the worst-performing method (React) attempted to perform an infeasible action, commanding to pick block 3 without first unstacking block 4 on top of it. The best-performing method (UniDomain) correctly recognized this physical constraint.

Task.

Arrange all blocks into two separate stacks on the table. The first stack should have blocks 1, 3, 5, and 7 in order from top to bottom. The second stack should have blocks 2, 4, 6, and 8 in order from top to bottom.

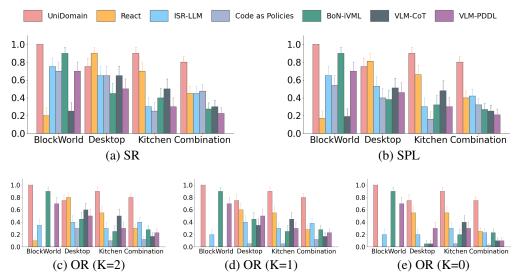


Figure 21: Separate comparison results of UniDomain and state-of-the-art baselines on the four classes of real-world tasks. Average values are shown with standard errors.

#### React plan (failed).

```
pick up block 1 from the top of the stack
put down block 1 on the table to start the first stack
pick up block 5 from the top of the stack
put down block 5 on top of block 1
pick up block 7 from the top of the tall stack
put down block 7 on top of block 5
pick up block 3 from the bottom of the second stack
pick up block 6 from the top of the second stack
put down block 6 on the table temporarily
pick up block 2 from the top of the second stack
put down block 2 on the table
pick up block 6 from the table
put down block 6 on the table
pick up block 2 from the table
put down block 2 on top of block 4
pick up block 6 from the table
pick up block 2 from the table
pick up block 3 from the bottom of the second stack
pick up block 7 from the top of the first stack
pick up block 7 from the top of the first stack
pick up block 6 from the table
pick up block 6 from the table
pick up block 6 from the table
```

# UniDomain PDDL problem (showing only core predicates).

```
(:init
    (clear block1)
    (on block1 block5)
    (on block5 block7)
    (on block7 block6)
    (on block6 block2)
    (on block2 block8)
    (on block8 block4)
    (on block4 block3)
    (on_table block3 table)
    (hand_free robot)
)
```

```
(:goal
  (and
      (clear block1)
      (on block1 block3)
      (on block3 block5)
      (on block5 block7)
      (on_table block7 table)
      (clear block2)
      (on block2 block4)
      (on block4 block6)
      (on block6 block8)
      (on_table block8 table)
)
```

UniDomain plan (succeeded).

```
(unstack robot block1 block5)
(place_on_table robot block1 table)
(unstack robot block5 block7)
(place_on_table robot block5 table)
(unstack robot block7 block6)
(place_on_table robot block7 table)
(pick_from_table robot block5 table)
(stack robot block5 block7)
(unstack robot block6 block2)
(stack robot block6 block5)
(unstack robot block2 block8)
(place_on_table robot block2 table)
(unstack robot block8 block4)
(place_on_table robot block8 table)
(unstack robot block6 block5)
(stack robot block6 block8)
(unstack robot block4 block3)
(stack robot block4 block6)
(pick_from_table robot block2 table)
(stack robot block2 block4)
(pick_from_table robot block3 table)
(stack robot block3 block5)
(pick_from_table robot block1 table)
(stack robot block1 block3)
```

**Desktop.** In Figure 10, the worst-performing method (BoN-iVML) commanded to open the wrong drawer, distinct from the one instructed, and omitted a critical "push" action. The best-performing method (React) correctly understood these key task elements.

Task.

```
There is a block in the green drawer. Please put it on the table, push it and put it in the yellow drawer.
```

BoN-iVML PDDL problem.

```
(:init
          (block_in_green_drawer block1)
          (not (drawer_open green_drawer))
          (not (drawer_open yellow_drawer))
          (robot_handempty)
)
(:goal
          (block_in_yellow_drawer block1)
)
```

BoN-iVML plan (failed).

```
(open_drawer yellow_drawer)
(pick_up_block_from_green_drawer block1 yellow_drawer)
(put_down_block_in_yellow_drawer block1 yellow_drawer)
```

React plan (succeeded).

```
open the green drawer
pick up the block from the green drawer
put down the block on the table
push the block on the table
open the yellow drawer
pick up the block from the table
put down the block in the yellow drawer
```

**Kitchen.** In Figure 12, the worst-performing method (Code as Policies) attempted to place an object into a bowl that is still leaning on a vertical drying rack. The highest-performing method (UniDomain) correctly captured this operational commonsense.

Task.

```
Put the jujube in the green bowl. And put the white plate on the rack.
```

Code as Policies plan (failed).

```
pick_up(jujube)
put_down(green bowl)
pick_up(white plate)
put_down(rack)
```

UniDomain PDDL problem (showing only core predicates).

```
(:init
    (on_rack bowl_pink rack1)
    (on_rack bowl_green rack1)
    (on_table bowl_white table1)
    (on_table plate_white table1)
    (in_bowl jujube1 bowl_white)
    (hand_free robot1)
)
(:goal
    (and
        (in_bowl jujube1 bowl_green)
        (on_rack plate_white rack1)
)
```

UniDomain plan (succeeded).

```
(pick_from_rack robot1 bowl_green rack1)
(place_on_table robot1 bowl_green table1)
(pick_from_bowl robot1 jujube1 bowl_white)
(put_in_bowl robot1 jujube1 bowl_green table1)
(pick_from_table robot1 plate_white table1)
(place_on_rack robot1 plate_white rack1)
```

**Combination.** In Figure 15, the worst-performing method (VLM-PDDL) suffered from syntaxerrors and incomplete operators in its PDDL problems and domains. The best-performing method (UniDomain) ensured the quality of the PDDL domain and the syntax correctness of PDDL problems.

#### Task.

There are a spoon, a tissue, an orange block in the green drawer. Stir the bowl and put the spoon in the cup, put the orange block into the orange drawer, wipe the bowl and scrunch the tissue on the table.

#### VLM-PDDL PDDL domain (part).

### UniDomain PDDL domain (part).

# UniDomain PDDL problem (showing only core predicates).

```
(:init
    (in_drawer orange_block1 drawer_green)
    (in_drawer spoon1 drawer_green)
    (in_drawer tissue1 drawer_green)
    (on_table bowl1 table1)
    (on_table cup1 table1)
    (unfolded tissue1)
    (can_wipe_table tissue1)
    (can_wipe_bowl tissue1)
    (can_stir_bowl spoon1)
    (hand_free robot1)
(:goal
  (and
    (stirred bowl1)
    (wiped bowl1)
    (scrunched tissue1)
    (in_cup spoon1 cup1)
    (in_drawer orange_block1 drawer_orange)
    (on_table tissue1 table1)
  )
)
```

UniDomain plan (succeeded).

```
(open_drawer robot1 drawer_green)
(pick_from_drawer robot1 spoon1 drawer_green)
(stir_bowl robot1 spoon1 bowl1)
(place_in_cup robot1 spoon1 cup1)
(pick_from_drawer robot1 tissue1 drawer_green)
(wipe_bowl robot1 tissue1 bowl1)
(scrunch_on_table robot1 tissue1 table1)
(open_drawer robot1 drawer_orange)
(pick_from_drawer robot1 orange_block1 drawer_green)
(place_in_drawer robot1 orange_block1 drawer_orange)
```

# F UniDomain Main Failure Modes

In this section, we present the main failure modes of UniDomain.

**Domain learning failures** arise from: (1) Syntax errors, e.g., ill-formed parentheses, account for 48% of failures; (2) Missing operators—there are critical actions omitted in 39% of failed domains; (3) Logical conflicts—contradictory preconditions and effects exist within an operator, constituting 35% of failures; (4) Invalid tests—the generated test problems contain invalid or unreachable goal states in 26% of failures.

Crucially, our closed-loop refinement always successfully detects and corrects all the above issues; hence, the success rate of the closed-loop pipeline is 100%.

Online planning failures are mostly caused by perception and grounding issues: (1) Goal misinterpretation—incorrect interpretation of task goals (e.g., reversed stacking order), occurred in 13.3% of failures; (2) Visual grounding errors—mislocalized or misclassified objects, leading to incorrect initial states, occurred in 53.4% of failures; (3) Insufficient domain coverage—the retrieved meta-domain lacks necessary operators or predicates to support the test task, which occurred in 33.3% of failures.

# G Prompt Design of the UniDomain Planner

To facilitate symbolic planning with the meta-domain, UniDomain employs a prompt to guide two crucial modules in the online planning: predicate/operator filtering and problem generation. This prompt is used twice, first to identify task-relevant predicates and operators by analyzing the initial problem proposal, and second to refine the final problem file based on the filtered symbolic space.

```
Now you need to help a robot to generate a PDDL problem file based on
the given PDDL domain.
The given image shows the initial scene.
The robot's hand is free initially (even though it is not shown in the
image) and PDDL objects must include a robot.
The task is under table-top environment and PDDL objects must include
at least a table.
Instructions: {instructions}
given PDDL domain: {domain}
Your output should include four parts:
(1) reasoning: analyze the image and output the reasons.
(2) objects: you need to locate objects related to the task from the
iamge.
(3) init: you need to describe PDDL init from the image based on given
PDDL predicates.
(4) goal: you need to gnerate PDDL goal from human instructions.
When generating PDDL init, you should use type predicate, state
predicates, spatial or position relationship predicates and affordance
predicates in order.
Your output should be in JSON format like below:
    "reasoning:": "your analysis",
    "objects": ["apple_1", "apple_2", "bowl"],
    "init": ["(on_table bowl)"],
    "goal": "(and (in apple_1, bowl) (in apple_2, bowl))"
}}
```

# H Setting and Prompts for Baselines

#### H.1 Baseline Setting

In our evaluation, all LLM-based methods were adapted to use VLMs, to accept scene image as input and support visual planning. For ReAct, we provide online visual feedback and action executability (success/failure only) to perform closed-loop planning. The maximum closed-loop action steps (including both execution and thinking steps) are constrained to twice the optimal plan cost. For BoN-iVML, we make two modifications in our implementation: (1) employing multi-LLM voting for best-of-N selection, and (2) adding a PDDL problem file generation module, which uses VLM to ensure visual grounding.

# H.2 Prompt for VLM-CoT.

```
Now you need to help a single-armed robot to plan to finish the given
task.
The given image shows the initial scene.
The available action APIs for the robot are:
(1) pick up: pick up some object.
(2) put down: put down the object in the robot's hand on/in some
object.
(3) open: open some object such as drawer, door, etc.
(4) close: close some object such as drawer, door, etc.
(5) fold: fold some object such as towel, tissue, etc.
(6) wipe: wipe some object using the object in the robot's hand.
(7) scrunch: scrunch some object such as tissue, etc
(8) stir: stir some object using the object in the robot's hand.
(9) push: push some object such as block, etc.
(10) slide: slide some object such as block, etc.
(11) press: press some object such as button, etc.
(12) turn on: turn on some object such as light, tap, etc.
(13) turn off: turn off some object such as light, tap, etc
(14) pull: pull some object such as rod, handle, etc.
(15) pour: pour some object using the object in the robot's hand.
(16) lean: lean some object against some other object, such as lean a
board against a wall.
When you make decisions, you should consider constraints based on your
 common sense.
For example, the robot cannot pick up another object when it is
already holding one because it is single-armed.
Instructions: {instructions}
Your output should include two parts:
(1) reasoning: analyze the image and output the reasons.
(2) plan_sequence: output the sequence of actions in natural language
to complete the task.
Your output should be in JSON format like below (Do NOT output
comments):
    "reasoning:": "your analysis",
    "plan_sequence": ["pick up the green block from table", "place the
     green block on the red block"]
}}
```

#### H.3 Prompt for Code as policies.

```
You are an expert at writing modular Python functions for a single
-armed robot to complete a task using a fixed set of atomic APIs
and the image.
The given image shows the initial scene. Observe all objects and
their spatial relationships, then write code based on this initial
state to complete the task.
The available action APIs for the robot are:
(1) pick_up(obj): pick up some object.
(2) put_down(target): put down the object in the robot's hand on/
in some object.
(3) open(obj): open some object such as drawer, door, etc.
(4) close(obj): close some object such as drawer, door, etc.
(5) fold(obj): fold some object such as towel, tissue, etc.
(6) wipe(obj): wipe some object using the object in the robot's
(7) scrunch(obj): scrunch some object such as tissue, etc
(8) stir(obj): stir some object using the object in the robot's
hand.
(9) push(obj): push some object such as block, etc.
(10) slide(obj): slide some object such as block, etc.
(11) press(obj): press some object such as button, etc.
(12) turn on(obj): turn on some object such as light, tap, etc.
(13) turn off(obj): turn off some object such as light, tap, etc
(14) pull(obj): pull some object such as rod, handle, etc.
(15) pour(obj): pour some object using the object in the robot's
hand.
(16) lean(obj1, obj2): lean some object against some other object,
 such as lean a board against a wall.
Instructions: {instructions}
When you make decisions, you should consider constraints based on
your common sense. For example, the robot cannot pick up another
object when it is already holding one because it is single-armed.
Structure the code with reusable high-level functions that
encapsulate meaningful sub-tasks using atomic action APIs.
This code will be directly executed, so it must be syntactically
correct Python. No markdown formatting like python or text outside
the code.
Example: Open the drawer, take out a towel and a tissue, fold both
, use the tissue to wipe the table, and place the folded towel
neatly on a shelf
import numpy as np
from robot_utils import pick_up, put_down, open, close, fold, wipe
def retrieve_items_from_drawer(drawer, towel, tissue):
    open(drawer)
    pick_up(towel)
    put_down("table")
    pick_up(tissue)
   put_down("table")
def prepare_items(towel, tissue):
    pick_up(towel)
    fold(towel)
    put_down("table")
    pick_up(towel)
   fold(tissue)
    put_down("table")
def wipe_table(tissue, table):
    pick_up(tissue)
    wipe(table)
    put_down(tissue)
def store_towel_on_shelf(towel, shelf):
```

```
pick_up(towel)
   put_down(shelf)

# Main execution
retrieve_items_from_drawer("drawer", "towel", "tissue")
prepare_items("towel", "tissue")
wipe_table("tissue", "table")
store_towel_on_shelf("towel", "shelf")
```

#### **H.4** Prompt for ReAct.

```
Now you need to help a single-armed robot to plan to finish the given
task. The given image shows the initial scene. The available action
APIs for the robot are:
(1) pick up: pick up some object.
(2) put down: put down the object in the robot's hand on/in some
object.
(3) open: open some object such as drawer, door, etc.
(4) close: close some object such as drawer, door, etc.
(5) fold: fold some object such as towel, tissue, etc.
(6) wipe: wipe some object using the object in the robot's hand.
(7) scrunch: scrunch some object such as tissue, etc.
(8) stir: stir some object using the object in the robot's hand.
(9) push: push some object such as block, etc. (10) slide: slide some
object such as block, etc.
(11) press: press some object such as button, etc.
(12) turn on: turn on some object such as light, tap, etc.
(13) turn off: turn off some object such as light, tap, etc
(14) pull: pull some object such as rod, handle, etc.
(15) pour: pour some object using the object in the robot's hand.(16) lean: lean some object against some other object, such as lean a
board against a wall.
When you make decisions, you should consider constraints based on your
common sense. For example, the robot cannot pick up another object
when it is already holding one because it is single-armed.
You MUST use only the exact API formats listed above for your actions,
and output the sequence of actions in natural language to complete
the tasks (e.g., pick up the green block from table). If you need to
think through a problem first, start your response with 'think:'
followed by your reasoning, and you'll receive 'OK.' Then provide your
action in the proper format.,
react_example_1:
Instruction: Put the blue cup into the open drawer.
Initial scene: [Image showing a table with a blue cup, a closed drawer
, and other objects]
> think: I need to open the drawer first, then pick up the blue cup,
and finally put it in the drawer. The robot has only one arm, so I
need to follow this sequence carefully.
> open the drawer
Image: [Image showing the drawer now open]
Feedback: Action successfully executed. pick up the blue cup from the
Image: [Image showing the robot arm holding the blue cup]
Feedback: Action successfully executed.
> put down the blue cup into the drawer
Image: [Image showing the blue cup placed inside the open drawer]
Feedback: Action successfully executed.,
react_example_0:
Instruction: Fold the towel and place it on the shelf.
Initial scene: [Image showing a bathroom with an unfolded towel on a
rack and an empty shelf]
> pick up the towel from the rack
Image: [Image showing the robot arm holding the unfolded towel]
Feedback: Action successfully executed.
fold the towel
Image: [Image showing the robot arm holding a now folded towel]
Feedback: Action successfully executed.
> put down the towel on the shelf
Image: [Image showing the folded towel placed on the shelf]
Feedback: Action successfully executed.
```

#### H.5 Prompt for VLM-PDDL.

```
Now a single-armed robot is going to finish the given task.
The given image shows the initial scene.
The available action for the robot are:
(1) pick up: pick up some object.
(2) put down: put down the object in the robot's hand on/in some
object.
(3) open: open some object such as drawer, door, etc.
(4) close: close some object such as drawer, door, etc.
(5) fold: fold some object such as towel, tissue, etc.
(6) wipe: wipe some object using the object in the robot's hand.
(7) scrunch: scrunch some object such as tissue, etc
(8) stir: stir some object using the object in the robot's hand.
(9) push: push some object such as block, etc.
(10) slide: slide some object such as block, etc.
(11) press: press some object such as button, etc.
(12) turn on: turn on some object such as light, tap, etc.
(13) turn off: turn off some object such as light, tap, etc
(14) pull: pull some object such as rod, handle, etc.
(15) pour: pour some object using the object in the robot's hand.
(16) lean: lean some object against some other object, such as lean a
board against a wall.
The robot must consider constraints based on common sense when making
decisions.
For example, the robot cannot pick up another object when it is
already holding one because it is single-armed.
Now you need to help the robot to generate both PDDL domain file and
PDDL problem file to finish the task, according to the given image
scene and instructions.
Instructions: {instructions}
Your output should include three parts:
(1) reasoning: analyze the image and output the reasons.
(2) domain: output the PDDL domain file.
(3) problem: output the PDDL problem file.
Your output should be in JSON format like below (Do NOT output
comments):
{{
    "reasoning:": "your analysis",
    "domain": "your PDDL domain file",
"problem": "your PDDL problem file"
}}
```

#### H.6 Prompts for ISR-LLM.

#### Translator.

```
You are a helpful, pattern-following assistant that translates given
task description into Planning Domain Definition Language (PDDL)
domain and problem files.
Now you need to help a single-armed robot to plan to finish the given
task.
The given image will show the initial scene.
The available action APIs for the robot are:
(1) pick up: pick up some object.
(2) put down: put down the object in the robot's hand on/in some
object.
(3) open: open some object such as drawer, door, etc.
(4) close: close some object such as drawer, door, etc.
(5) fold: fold some object such as towel, tissue, etc.
(6) wipe: wipe some object using the object in the robot's hand.
(7) scrunch: scrunch some object such as tissue, etc
(8) stir: stir some object using the object in the robot's hand.
(9) push: push some object such as block, etc.
(10) slide: slide some object such as block, etc.
(11) press: press some object such as button, etc.
(12) turn on: turn on some object such as light, tap, etc.
(13) turn off: turn off some object such as light, tap, etc
(14) pull: pull some object such as rod, handle, etc.
(15) pour: pour some object using the object in the robot's hand.
(16) lean: lean some object against some other object, such as lean a
board against a wall.
When you make decisions, you should consider constraints based on your
common sense.
For example, the robot cannot pick up another object when it is
already holding one because it is single-armed.
Below are some examples of PDDL files for the blocksworld problem.
{examples}
Now instruction is {instruction}.
```

#### Planner.

```
You are a confident and pattern-following assistant that determines action sequences to complete a given task, which is described by Planning Domain Definition Language (PDDL) domain and problem files.

Below are some examples: {examples}

Now the planning problem is {planning_problem}.
```

#### Validator.

```
You are a helpful, pattern-following assistant that examines the correctness of each action during a task planning process. You work like a state machine.

Below are some examples: {examples}

Question: initial state: {pddl_init_state}
Goal state: {pddl_goal_state}
Examined action sequence: {action_description}
```

#### H.7 Prompts for BoN-iVML.

#### initialization.

```
Now a single-armed robot is going to finish the given task.
The given image shows the initial scene.
Instructions: {instructions}
The available action for the robot are:
(1) pick up: pick up some object.
(2) put down: put down the object in the robot's hand on/in some
object.
(3) open: open some object such as drawer, door, etc.
(4) close: close some object such as drawer, door, etc.
(5) fold: fold some object such as towel, tissue, etc.
(6) wipe: wipe some object using the object in the robot's hand.
(7) scrunch: scrunch some object such as tissue, etc
(8) stir: stir some object using the object in the robot's hand.
(9) push: push some object such as block, etc.
(10) slide: slide some object such as block, etc.
(11) press: press some object such as button, etc.
(12) turn on: turn on some object such as light, tap, etc.
(13) turn off: turn off some object such as light, tap, etc
(14) pull: pull some object such as rod, handle, etc.(15) pour: pour some object using the object in the robot's hand.
(16) lean: lean some object against some other object, such as lean a
board against a wall.
The robot must consider constraints based on common sense when making
decisions.
For example, the robot cannot pick up another object when it is
already holding one because it is single-armed.
Now you are given the natural language instruction. Your task is to
generate PDDL domain code. according to the given image scene and
instructions.
This includes defining predicates and actions based on the information
provided.
Note that individual conditions in preconditions and effects should be
listed separately. For example, "object1 is washed and heated" should
be considered as two separate conditions: "object1 is washed" and "
object1 is heated".
Also, in PDDL, two predicates cannot have the same name even if they
have different parameters.
Each predicate in PDDL must have a unique name, and its parameters
must be explicitly defined in the predicate definition.
It is recommended to define predicate names in an intuitive and
readable wav.
Remember: Ignore the information that you think is not helpful for the
planning task.
You are only responsible for domain generation. Before you generate
the concrete domain code, you should first generate a natural
Language thought about the meaning of each variable, and the step-by-
step explanation of the domain code. Even if I didn't provide the
exact name of the predicates and actions, you should generate them
based on the information provided in the natural language description.
Note that you ONLY need to use PDDL 1.0!
Your output should be in JSON format like below:
```

```
"reasoning": "Analysis the image and output your reasoning, like:
    predicate_1: the name of predicate_1, explanation of predictate_1,
        ..., predicate_n: the name of predicate_n, explanation of
    predictate_n, action_1: the name of action_1, explanation of
    action, ..., action_n: the name of actio_n, explanation of
    action_n.",
    "domain": "The concrete pddl code for domain.pddl, in PDDL format
    ."
}}
Now it's your time to generate the solution, you have to follow the
format I provided above.
```

#### Best-of-N.

```
Now a single-armed robot is going to finish the given task.
The given image shows the initial scene.
Instructions: {instructions}
The available action for the robot are:
(1) pick up: pick up some object.
(2) put down: put down the object in the robot's hand on/in some
object.
(3) open: open some object such as drawer, door, etc.
(4) close: close some object such as drawer, door, etc.
(5) fold: fold some object such as towel, tissue, etc.
(6) wipe: wipe some object using the object in the robot's hand.
(7) scrunch: scrunch some object such as tissue, etc
(8) stir: stir some object using the object in the robot's hand.
(9) push: push some object such as block, etc.
(10) slide: slide some object such as block, etc.
(11) press: press some object such as button, etc.
(12) turn on: turn on some object such as light, tap, etc.
(13) turn off: turn off some object such as light, tap, etc
(14) pull: pull some object such as rod, handle, etc.
(15) pour: pour some object using the object in the robot's hand.
(16) lean: lean some object against some other object, such as lean a
board against a wall.
The robot must consider constraints based on common sense when making
decisions.
For example, the robot cannot pick up another object when it is
already holding one because it is single-armed.
Now a sequence of initial PDDL domains is provided, you must choose
the best one that is the most relevant to the task according to the
given image.
PDDL domains: {domains}
Your output should be in JSON format like below:
    "reasoning": "Analysis the image and output your reasoning.",
    "domain_index": "1"
}}
```

# $f_{\mathrm{opt}}\left(\cdot\right)$

```
Now a single-armed robot is going to finish the given task.
The given image shows the initial scene.
Instructions: {instructions}
The available action for the robot are:
(1) pick up: pick up some object.
(2) put down: put down the object in the robot's hand on/in some
object.
(3) open: open some object such as drawer, door, etc.
(4) close: close some object such as drawer, door, etc.
(5) fold: fold some object such as towel, tissue, etc.
(6) wipe: wipe some object using the object in the robot's hand.(7) scrunch: scrunch some object such as tissue, etc
(8) stir: stir some object using the object in the robot's hand.
(9) push: push some object such as block, etc.
(10) slide: slide some object such as block, etc.
(11) press: press some object such as button, etc.
(12) turn on: turn on some object such as light, tap, etc.
(13) turn off: turn off some object such as light, tap, etc
(14) pull: pull some object such as rod, handle, etc.
(15) pour: pour some object using the object in the robot's hand.
(16) lean: lean some object against some other object, such as lean a
board against a wall.
The robot must consider constraints based on common sense when making
decisions.
For example, the robot cannot pick up another object when it is
already holding one because it is single-armed.
Now a PDDL domain about the task based on the image is given with
intermediate thoughts explaining each predicate and action
Your task is to generate critical feedback on the PDDL domain code
based on the task and image scene.
You should evaluate the grammar and logic of the PDDL domain codes,
and the logic error in the intermediate thoughts.
natural language chain of thoughts: {thought}
Generated PDDL domain: {domain}
Note that you ONLY need to use PDDL 1.0!
Your output should be in JSON format like below:
    "reasoning": "Analysis the image and output your reasoning.",
    "feedback": "Your final feedback."
}}
```

## $f_{\text{update}}\left(\cdot\right)$

```
Now a single-armed robot is going to finish the given task.
The given image shows the initial scene.
Instructions: {instructions}
The available action for the robot are:
(1) pick up: pick up some object.
(2) put down: put down the object in the robot's hand on/in some
object.
(3) open: open some object such as drawer, door, etc.
(4) close: close some object such as drawer, door, etc.
(5) fold: fold some object such as towel, tissue, etc.
(6) wipe: wipe some object using the object in the robot's hand.(7) scrunch: scrunch some object such as tissue, etc
(8) stir: stir some object using the object in the robot's hand.
(9) push: push some object such as block, etc.
(10) slide: slide some object such as block, etc.
(11) press: press some object such as button, etc.
(12) turn on: turn on some object such as light, tap, etc.
(13) turn off: turn off some object such as light, tap, etc
(14) pull: pull some object such as rod, handle, etc.
(15) pour: pour some object using the object in the robot's hand.
(16) lean: lean some object against some other object, such as lean a
board against a wall.
The robot must consider constraints based on common sense when making
decisions.
For example, the robot cannot pick up another object when it is
already holding one because it is single-armed.
You will be provided a PDDL domain code about the task based on the
image and critical feedback on the PDDL domain code based on the task
and image.
Your task is to generate a new PDDL domain code that is more
consistent with the task and update the chain of thoughts.
Natural language chain of thoughts at the previous turn: {thought}
Generated PDDL domain at the previous turn: {domain}
The error of the PDDL domain {feedback}
Note that you ONLY need to use PDDL 1.0!
Your output should be in JSON format like below:
    "reasoning": "Analysis the image and output your reasoning.",
    "thought": "your updated thought",
    "domain": "your updated PDDL domain code in PDDL format"
}}
```

#### **Problem Formulation.**

```
Now a single-armed robot is going to finish the given task.
The robot's hand is free initially even though it is not shown in the image.
The given image shows the initial scene.

Now you need to help the robot to generate the PDDL problem file to finish the task, according to the given image scene and instructions.
Instructions: {instructions}
Given PDDL domain: {domain}

Note that you ONLY need to use PDDL 1.0!

Your output should be in JSON format like below (Do NOT output comments):
{{
    "reasoning:": "analysis the image and output your analysis",
    "problem": "your PDDL problem file in PDDL format"
}}
```