LEARNING NEURO-SYMBOLIC ABSTRACTIONS FOR TRANSFER IN REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning agents often require substantial interaction to adapt to a new environment, even when the new environment preserves the task semantics of a previously solved source environment. Reusing knowledge acquired in the source can reduce this cost, while doing so requires the agent to identify states that play the same decision-relevant role across environments. State abstractions provide a natural way to share action-value estimates and policies across such states, while abstractions induced by value signals from the source environment may encode layout-specific distinctions and therefore transfer poorly. We introduce Neuro-symbolic Abstraction and Transfer (NeuSAT), a single-source online transfer framework built on the observation that an abstraction effective for source learning is not necessarily suitable for transfer. NeuSAT first learns a visually routed abstraction tree and an abstract policy through interaction with the source environment. It then uses them as a teacher to rebuild the state partition with auto-generated semantic propositions and fits abstract action values on the resulting proposition-routed tree. The propositional tree and its action values are transferred to unseen target environments as a zero-shot policy prior. During the agent's interaction with the target environment, NeuSAT updates the transferred values and locally refines leaves that combine states with conflicting action preferences. Our evaluation results on three domains across held-out layouts and environment scales show that proposition-structured state abstractions can support sample-efficient transfer from a single source environment.

1 INTRODUCTION

Reinforcement learning (RL) agents often require extensive interaction with a fixed training environment to learn a good policy (Silver et al., 2016; Mnih et al., 2015). However, when a similar task appears in a new environment with a different layout or scale, learning from scratch discards useful source experience, while directly deploying the source policy may retain decisions specific to the source environment and lead to poor performance. This problem becomes more costly as target environments grow larger and require more interaction to explore. Effective transfer therefore requires identifying what knowledge can be reused and what must be adapted.

At their core, both source-domain sample efficiency and cross-environment transfer are state-abstraction problems: the former requires identifying which states can share value estimates and decisions within an environment (Li et al., 2006; Zhang et al., 2020; 2026), while the latter requires determining which of these groupings remain valid across environments. Many transfer methods learn such invariances from interaction across multiple source environments or from large offline datasets (Zhu et al., 2023; Watahiki et al., 2024; Verma et al., 2025). Although this environmental diversity can support robust representation learning, collecting it can be costly and may require anticipating the variations encountered at deployment. This reliance on broad environmental coverage also contrasts with how humans often generalize: humans can often solve a task in one setting, recognize the semantic relations underlying successful decisions, and reuse them in structurally different settings. This observation suggests that a successful source policy may already contain task-relevant information about which states can share decisions.

Our objective therefore is to learn, from a single, relatively small source environment, a compact decision structure that supports efficient source learning and can be reused and adapted across unseen

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

layouts and scales. The closest related approaches, CATRL (Dadvar et al., 2023) and PEARL (Nayyar et al., 2026), learn state abstractions online by starting with a single coarse abstraction and splitting it when its members exhibit inconsistent value signals. States in the same abstraction share abstract action values and a policy, improving learning efficiency within that environment. However, value-driven refinement can produce partitions tied to the source layout. In MiniGrid-DoorKey (Chevalier-Boisvert et al., 2023), moving walls, doors, or keys changes routes and Q-values, while carrying a key and facing a locked door still supports the same unlocking action. Thus, source-specific value distinctions may not transfer even when the underlying decision remains valid.

We therefore seek to learn transferable state distinctions through semantic propositions rather than directly reusing the source value-based partition. A propositional vocabulary specifies which semantic conditions can be expressed, without revealing which of them matter for the learned policy or how states should be grouped for decision making. Moreover, a semantic grouping that supports one action in the source may still combine states requiring different actions in a target environment. Transferable abstraction therefore requires both source-policy guidance to identify decision-relevant propositions and target feedback to revise the resulting grouping when it becomes insufficient.

We introduce Neuro-symbolic Abstraction and Transfer (NeuSAT), a single-source online transfer framework built around a separation between learning an effective source abstraction and constructing a transferable one. In Stage 1, NeuSAT learns an abstract policy and a visually routed abstraction tree by interacting with one source environment. In Stage 2, it freezes the learned tree and abstract action values as a teacher and grows a new semantic abstraction tree with propositional variables from scratch. The teacher identifies which state distinctions matter for its decisions, while automatically generated semantic propositions express those distinctions as rules that can be re-evaluated across layouts. The resulting propositional tree and its abstract action values are transferred to unseen target environments as a zero-shot policy prior. Rather than treating this prior as fixed, NeuSAT updates its values and locally refines leaves whose members require conflicting actions.

We evaluate NeuSAT on MiniGrid-DoorKey, Crafter (Hafner, 2021), and Sokoban, transferring from one small source layout to 15 held-out targets per domain with different layout and scale. Stage 1 learns successful policies with fewer interactions than PPO. After transfer, NeuSAT reaches 90% success in every evaluated run within 150k target steps. In Crafter and Sokoban, it requires 92% and 84% fewer target interactions than the fastest baseline, respectively. Direct transfer of the source visual tree remains unreliable even with continued target learning, supporting the need for semantic distillation. Ablations further show that value rebasing and local refinement work together to support fast and sustained adaptation.

2 Related works

State abstraction in RL State abstraction preserves decision-relevant structure across states. Bisimulation-based and predictive approaches learn representations that preserve reward and transition structure (Chen & Pan, 2022; Wang et al., 2025; Luo et al., 2025; Zhang et al., 2026). CATRL (Dadvar et al., 2023) and PEARL (Nayyar et al., 2026) refine abstractions online using value-learning signals but need explicit, factored state inputs. Stage 1 of NeuSAT extends this approach to visual observations and groups states by their value profiles across all actions, rather than a signal for one selected action. Beyond these source-learning differences, NeuSAT reconstructs the abstraction through semantic propositions for transfer and continued adaptation.

Interpretable policy learning in RL Prior work learns decision trees, symbolic expressions, and programmatic policies through distillation or direct optimization Acero & Li (2024); Marton et al. (2025); Vos & Verwer (2024); Costa et al. (2024); Dutta et al. (2024). Policy distillation transfers teacher knowledge to student models (Rusu et al., 2015), while VIPER extracts verifiable tree policies through imitation learning (Bastani et al., 2018). NeuSAT distills a visual abstraction into a proposition-routed tree for transfer, then updates its abstract values and locally refines its leaves when target states require different actions.

Transfer and online adaptation in RL Domain-invariant representations, cross-domain alignment, or reusable skills are commonly used for knowledge transfer (Shoeleh & Asadpour, 2020; Li et al., 2022; Verma et al., 2025). In-context methods use sequence models trained on learning

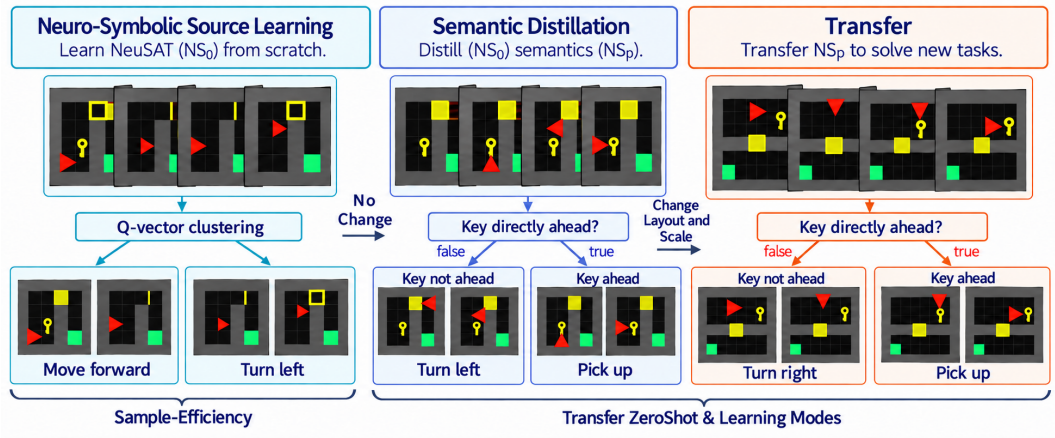

Figure 1: Overview of NeuSAT framework.

histories to support adaptation (Son et al., 2025). Continual RL methods such as CKA-RL reuse and combine task-specific parameter updates across a sequence of tasks (Hu et al., 2026). NeuSAT targets single-source online transfer by reusing an explicit semantic abstraction and its action values. Each target adapts this source knowledge independently, updating values and refining the abstraction without accumulating experience across targets.

3 Preliminaries and Problem Setting

We consider an episodic discounted MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma, \rho_0)$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and finite action spaces, P and R are the transition and reward functions, $\gamma \in [0, 1)$ is the discount factor, and ρ_0 is the initial-state distribution. The agent observes each state only through an RGB observation $o_t = \omega_{\mathcal{M}}(s_t)$ and acts according to $\pi(a \mid o_t)$. Its objective is to maximize $J_{\mathcal{M}}(\pi) = \mathbb{E}_{\rho_0,P,\pi}[\sum_{t=0}^{T-1} \gamma^t r_t]$. $Q(s, a)$ denotes the expected return from taking action a in state s under the current policy, and the state value is defined as $V(s) = \max_{a\in\mathcal{A}} Q(s, a)$.

For visual observations, a state abstraction is a mapping $\psi : \mathcal{O}_{\mathcal{M}} \to \mathcal{Z}$ that groups observations into a finite set of abstract states. Each abstract state z carries an action-value vector $Q_{abs}(z, \cdot)$, inducing the observation policy $\pi_{\psi,Q_{abs}}(o) \in \arg\max_{a\in\mathcal{A}} Q_{abs}(\psi(o), a)$. In our setting, ψ is learned online from the agent's interaction with the environment.

We study single-source online transfer: an agent learns in one source MDP $\mathcal{M}_s$ and adapts to unseen target MDPs that share its task objective, action space, and action semantics but may differ in layout and scale. No target observations or interactions are used during source learning or semantic distillation. Each target is initialized with a zero-shot policy $\pi_t^{(0)}$ from the same source solution and adapted independently; $\pi_t^{(b)}$ denotes its policy after b target interaction steps.

4 NeuSAT: Learning, Distilling and Transferring State Abstractions

Our goal is to use interaction with a single source environment to learn an effective policy and to extract a reusable decision structure for unseen target environments. NeuSAT proceeds in three stages (Figure 1). Stage 1 learns a Neuro-Symbolic abstraction tree (NSAT) and abstract policy through value-driven refinement. Stage 2 uses them as a teacher to reconstruct its decision-relevant distinctions with auto-generated semantic propositions. Stage 3 transfers the resulting propositional tree and its abstract action values as an initial target policy, then updates the values and locally adds value-driven visual refinements based on agent's interaction in the target environment.

4.1 NEURO-SYMBOLIC SOURCE LEARNING

Building on coarse-to-fine abstraction learning(Dadvar et al., 2023; Nayyar et al., 2026), NeuSAT starts with an NSAT consisting of a single abstract state and alternates between value learning and tree refinement. A heterogeneity score identifies *which* leaves of a NSAT require refinement. For each selected leaf, NeuSAT clusters all states belonging to it using their complete Q-value vectors and trains a visual router to implement the resulting partition. The learned tree and its abstract action values form the source teacher used in Stage 2.

Definition 1 *A* Neuro-Symbolic abstraction tree *(NSAT) is a rooted binary tree* $T = (N, E, n_0, \{h_n\}_{n \in N_{\text{int}}})$*, where* N *is a set of nodes,* E *is a set of edges,* $n_0 \in N$ *is the root and each internal node contains a visual router* $h_n : \mathcal{O}_s \to \{0, 1\}$ *that selects one of its two children. The leaves* $\mathcal{Z} = N \setminus N_{\text{int}}$ *are abstract states. Routing an observation from the root to a leaf defines the abstraction mapping* $\psi : \mathcal{O}_s \to \mathcal{Z}$*.*

Each leaf $z \in \mathcal{Z}_1$ carries an abstract action-value vector $Q_{\text{abs},1}(z, \cdot)$, inducing the source policy $\pi_1(o) \in \arg\max_{a \in \mathcal{A}} Q_{\text{abs},1}(\psi_1(o), a)$. The abstract values are learned with Watkins $Q(\lambda)$ (Watkins et al., 1989) and determine the agent's behavior.

As the RL agent interacts with the source environment using Watkins's $Q(\lambda)$, new data is collected, NSAT is refined and $Q_{\text{abs},1}$ is learned. After each refinement phase, NeuSAT updates $Q_{\text{abs},1}$ under the new tree during a fixed stabilization window. It then collects transitions $\mathcal{D} = \{(o_i, a_i, r_i, o_i')\}$ and computes the next refinement selection from this data. This cycle continues until the source interaction budget is exhausted.

Following Nayyar et al. (2026), NeuSAT scores each leaf–action pair by combining the dispersion of its abstract TD errors with the dispersion of the concrete-state values of its members. Let $\mathcal{H}(z, a)$ denote this score and define $\mathcal{H}(z) = \max_{a \in \mathcal{A}} \mathcal{H}(z, a)$. Leaves with the highest eligible scores are selected for refinement. Let z be a leaf selected for refinement. Prior refinement methods construct splits from a scalar signal associated with the action that triggers refinement (Dadvar et al., 2023; Nayyar et al., 2026). However, a child leaf may ultimately select a different greedy action. An action-specific split can therefore ignore value differences that determine the child's eventual policy and leave states with conflicting action preferences in the same abstraction. Instead, we split z based on the complete Q-value vectors of its member states.

A registry $\mathcal{R} : \mathcal{O}_s \to \mathcal{K}$ assigns each observation to a concrete key k. NeuSAT represents each key k by its complete action-value vector $\mathbf{q}(k) = Q_{\text{con}}(k, \cdot) = \big(Q_{\text{con}}(k, a)\big)_{a \in \mathcal{A}} \in \mathbb{R}^{|\mathcal{A}|}$. For a selected leaf z, let $\mathcal{K}_z = \{\mathcal{R}(o_i) : \psi_1(o_i) = z\}$ be its observed concrete keys. NeuSAT applies two-cluster Ward agglomerative clustering to $\{\mathbf{q}(k) : k \in \mathcal{K}_z\}$, producing a label $y(k) \in \{0, 1\}$ for each key.

A local visual router h_z is trained to predict these labels from the corresponding observations. If both clusters have sufficient support and the router reproduces the partition with the required accuracy, z is replaced by the router and two child leaves. Router training and child-value initialization are detailed in the appendix A.1

After Stage 1, NeuSAT returns $(T_1, Q_{\text{abs},1})$ as the teacher for Stage 2's semantic distillation.

4.2 SEMANTIC DISTILLATION

Stage 2 converts the source-specific NSAT T_1 and $Q_{\text{abs},1}$ into a transferable proposition-based abstraction by distillation. During learning, NeuSAT first freezes them as a teacher and collects source trajectories using its policy. Starting from a single leaf, it grows a new $NSAT_P$ T_P whose splits must be expressed as rules over auto-generated propositions. The teacher's action values determine which leaves need refinement and which propositional rules best separate their members. As the tree grows, NeuSAT fits its abstract action values on the teacher trajectories. The resulting proposition-routed tree can evaluate the same rules in unseen environments. An example of our proposition-routed tree is in appendix A.3.

Given example environment observations, we first prompt an LLM to generate a set of Boolean semantic propositions $P = \{p_1, \ldots, p_m\}$. We then prompt the LLM to generate a visual parser g that evaluates these propositions from an RGB observation o, where $o \in \mathcal{O}_{\mathcal{M}}$. The resulting propositional representation is $\mathbf{p}(o) = \big(p_1(g(o)), \ldots, p_m(g(o))\big) \in \{0, 1\}^m$.

The proposition $NSAT_P$ T_P has the same binary-tree form as T_1, but each internal node contains a Boolean rule over $\mathbf{p}(o)$ rather than a learned visual router. Its leaves $\mathcal{Z}_P$ are abstract states, and routing from the root to a leaf defines $\psi_P : \mathcal{O}_s \to \mathcal{Z}_P$. Proposition and parser generation are detailed in the appendix. Once generated, neither LLM calls nor simulator-state access is required to evaluate the tree.

For each registered source key $k = \mathcal{R}(o_k)$, the frozen teacher provides the action values $q_1(k, a) = Q_{\mathrm{abs},1}(\psi_1(o_k), a)$ and the greedy action $a_1(k) \in \arg\max_a q_1(k, a)$. NeuSAT collects a fixed trajectory dataset $\mathcal{D}_P$ using an ϵ_b-greedy version of the teacher policy.

Stage 2 also alternates between a learning phase and a refinement phase just as in Stage 1, except that the fixed NSAT T_1 and $Q_{\mathrm{abs},1}$ determines agent policy while $NSAT_P$ is learned. During learning, observations in $\mathcal{D}_P$ are routed through the current T_P, and $Q_{\mathrm{abs},P}$ is fitted by SARSA(λ) Rummery & Niranjan (1994) on the recorded teacher trajectories. Each update uses the next action recorded in the trajectory, including exploratory actions selected by the teacher's ϵ_b-greedy policy. After the tree is refined, the values of the new leaves are initialized from their members' teacher Q-vectors and fitted again on $\mathcal{D}_P$.

Let $\mathcal{K}_z$ be the distinct teacher-data keys assigned to leaf z. If all of these keys must share one action, the teacher-preferred shared action is $a_z^{\mathrm{T}} \in \arg\max_{a \in \mathcal{A}} \sum_{k \in \mathcal{K}_z} q_1(k, a)$. We measure the loss of enforcing this shared action using

$$H_P(z) = \frac{1}{c_1 |\mathcal{K}_z|} \sum_{k \in \mathcal{K}_z} \left[\max_a q_1(k, a) - q_1(k, a_z^{\mathrm{T}}) \right], \tag{1}$$

where $c_1 > 0$ is a global value scale. NeuSAT prioritizes leaves with high $H_P(z)$, where sharing one action incurs substantial loss under the teacher. If a common action is optimal for every member, $H_P(z) = 0$ even when their Q-values differ.

For a selected leaf z, let $\mathcal{K}_z^p \subseteq \mathcal{K}_z$ contain the keys covered by the propositional representation. Candidate rules are p_i, $p_i \wedge p_j$, or $p_i \vee p_j$, with sufficient support in both children. This limits the search to $O(m^2)$ rules per node for m propositions; more complex conditions are expressed across tree levels rather than through unrestricted CNF or DNF formulas.

To choose a rule, NeuSAT follows the greedy split-selection strategy of decision trees, maximizing the reduction in teacher action-preference dispersion. These vectors retain preferences over all actions, beyond the greedy-action label. Although a rule could instead be scored by its reduction in teacher regret, that reduction is zero whenever both children retain the parent's best shared action. Preference dispersion can distinguish such rules by how consistently the members within each child prefer the available actions, even before their best shared actions change.

Let $\mathbf{u}_1(k)$ denote the normalized action-preference vector derived from $q_1(k, \cdot)$, and let $D_1(K)$ be the mean Euclidean distance of these vectors from their mean over keys in K. A candidate rule r partitions the keys into $K_b^r = \{k \in \mathcal{K}_z^p : r(\mathbf{p}(o_k)) = b\}$ for $b \in \{0, 1\}$. Its gain is

$$G_1(r; \mathcal{K}_z^p) = D_1(\mathcal{K}_z^p) - \sum_{b \in \{0,1\}} \frac{|K_b^r|}{|\mathcal{K}_z^p|} D_1(K_b^r). \tag{2}$$

A larger G_1 means a greater reduction in average within-child action-preference dispersion. Thus, H_P prioritizes leaves where sharing an action is costly, while G_1 favors splits that group similar action preferences. NeuSAT selects the admissible rule with the largest gain. The rule replaces the selected leaf with an internal node and two children, whose values are initialized from their members' teacher Q-vectors. Refinement continues until the budget is exhausted or no admissible split remains. Normalization, support requirements, and value initialization are detailed in Appendix A.1.

Stage 2 returns $(T_P, Q_{\mathrm{abs},P})$ for transfer. Unlike the visual boundaries in T_1, the rules in T_P can be re-evaluated on observations from a new environment.

4.3 Target Transfer and Online Refinement

Stage 3 transfers the propositional tree and adapts it through interaction with a target environment $\mathcal{M}_t$. NeuSAT initializes $T_t \leftarrow T_P$ and $Q_{\mathrm{abs},t}$ from $Q_{\mathrm{abs},P}$, while retaining the propositional vocabulary $\mathcal{P}$ and visual parser g from Stage 2. For any target observation o, including observations not

encountered in the source environment, the parser extracts its propositional vector $\mathbf{p}(o)$. Evaluating the internal rules of T_t on this vector routes o to a leaf z. The policy initially uses the abstract action values inherited by that leaf, providing a zero-shot policy prior.

Source values learned in a smaller environment may overestimate returns in a larger target. We therefore keep the source action preferences but lower their value level before target learning. For each leaf z, we initialize $Q_{\mathrm{abs},t}(z,a) = Q_{\mathrm{abs},P}(z,a) - (1-\kappa)\big(\bar{Q}(z) - V_{\min}\big)$, $\bar{Q}(z) = \max_b Q_{\mathrm{abs},P}(z,b)$, where $V_{\min} = -1/(1-\gamma)$ is the return from never reaching the goal and $\kappa = 0.1$. The same offset is applied to every action in a leaf, so its greedy action and zero-shot policy are unchanged. Lower initial values also reduce the chance that, during target learning, an untried action retains a higher source value than an action already updated with target experience. Setting $\kappa = 1$ transfers the values unchanged; we compare both settings in Section 5.3.

The transferred $NSAT_P$ is used as an initial abstraction rather than a fixed target policy. Starting from $(T_t, Q_{\mathrm{abs},t})$, target training repeats the learning–refinement cycle of Stage 1. During learning, target observations are routed through the current tree using propositional rules, and $Q_{\mathrm{abs},t}$ is updated from target interactions using Watkins's $Q(\lambda)$.

At each refinement round, NeuSAT follows the value-driven procedure from Stage 1. It selects heterogeneous leaves using target value signals, clusters the complete target Q-value vectors of their members, and trains a local visual router for each resulting partition. An accepted router replaces the selected leaf with two children, whose values are initialized and subsequently updated through target interaction. The transferred propositional rules remain unchanged; target refinement only grows new visual routers below leaves that require additional refinement.

Stage 3 returns $(T_t, Q_{\mathrm{abs},t})$. The resulting target policy is $\pi_t(o) \in \arg\max_{a\in\mathcal{A}} Q_{\mathrm{abs},t}(\psi_t(o), a)$.

Algorithm 1 NeuSAT

Input: Source $\mathcal{M}_s$, target $\mathcal{M}_t$
Stage 1: Source Abstraction Learning
Initialize single-leaf NSAT T_1
while source budget remains **do**
    Value update then collect $\mathcal{D}$
    Select leaves using heterogeneity $\mathcal{H}$
    Cluster their complete Q-vectors and Refine T_1 with the accepted routers
Stage 2: Semantic Distillation
Freeze $(T_1, Q_{\mathrm{abs},1})$ as the teacher
Generate propositions $\mathcal{P}$ and collect teacher data $\mathcal{D}_P$
Initialize a single-leaf propositional tree T_P
repeat
    Fit $Q_{\mathrm{abs},P}$ on $\mathcal{D}_P$
    Select leaves by teacher regret H_P
    Refine them using propositional rules maximizing G_1
until no split is accepted or the budget is reached
Stage 3: Target Transfer and Online Refinement
Initialize $T_t \leftarrow T_P$ and rebase $Q_{\mathrm{abs},P}$ to obtain $Q_{\mathrm{abs},t}$
while target budget remains **do**
    Update target values from interaction
    Refine heterogeneous leaves using target Q-vector clustering and visual routers
return $(T_t, Q_{\mathrm{abs},t})$

5 Empirical Results

Experimental setup. We evaluate NeuSAT on three visual RL domains: MiniGrid-DoorKey (Chevalier-Boisvert et al., 2023), Crafter (Hafner, 2021), and Sokoban, using RGB observations and 5% action noise. All tasks provide a $+1000$ reward only upon successful completion, with a -1 per-step cost and no intermediate rewards for task progress. For transfer, each domain uses one source layout (6×6, 8×8, and 7×7, respectively) and 15 larger held-out targets. All meth-

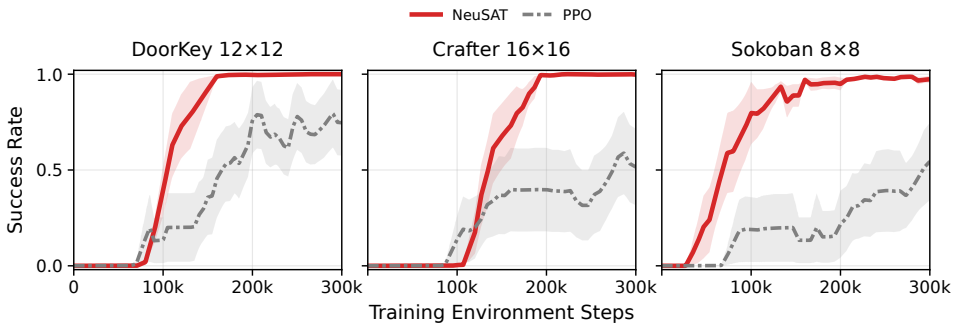

Figure 2: NeuSAT Stage 1 compared to PPO

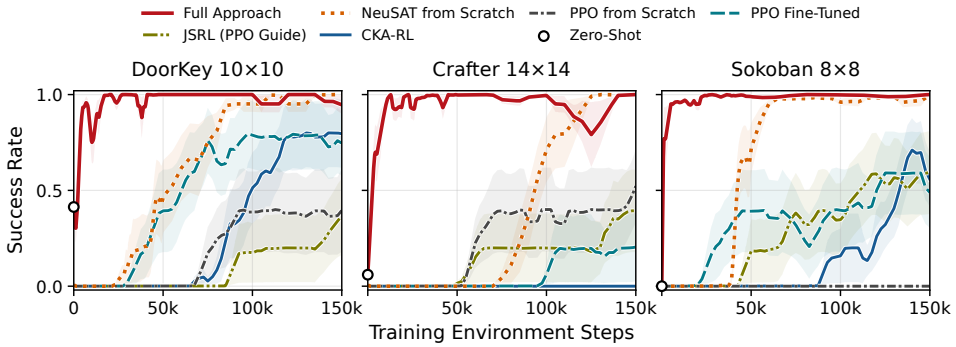

Figure 3: Target adaptation on one held-out layout per domain. Open circles on the Y-axis mark NeuSAT's zero-shot success before target learning; curves show success during online adaptation.

ods use five random seeds per environment. Each target run has a 150k-step budget and is evaluated using greedy policies. Metrics are defined in Table 1. Implementation details, hyperparameters, and layouts appear in Appendices A.1 and A.6.

Baselines. We compare with PPO trained from scratch (Schulman et al., 2017), PPO fine-tuning, JSRL (Uchendu et al., 2023), and CKA-RL (Hu et al., 2026). A source-trained PPO policy initializes fine-tuning and guides JSRL. We also run NeuSAT's Stage 1 directly in each target from scratch to isolate the benefit of source knowledge. The details of baseline implementation is in AppendixA.1.

5.1 Evaluation of Sample Efficiency in Source Learning

RL algorithms can quickly reach high success on small layouts, making differences in learning speed difficult to observe. We therefore evaluate Stage 1 on larger layouts: DoorKey 12×12, Crafter 16×16, and Sokoban 8×8. We compare with PPO trained from scratch using the same RGB observations, rewards, and fixed start, with five seeds each. Figure 2 shows that NeuSAT matches PPO's mean success at 300k steps after only 114k/129k/67k steps on each domain, requiring approximately $62\%/57\%/78\%$ fewer interactions. By 300k steps, every NeuSAT seed achieves success above 0.97, while PPO's mean success is $0.66/0.40/0.47$. These results indicates that NeuSAT abstraction leads to more sample efficient learning in Stage 1 as well.

5.2 Evaluation of Transfer Performance

Each target independently adapts the same source-trained propositional tree and action values. Figure 3 shows learning curves on one target per domain; Table 1 summarizes all 15 targets per domain.

Table 1 shows that NeuSAT achieves zero-shot success of $0.84/0.29/0.07$ on DoorKey/Crafter/Sokoban and reaches 90% success in every target run. In Crafter and Sokoban, it reaches the threshold in 8.5k and 8.5k steps, respectively, using 92% and 84% fewer interactions

Domain	Method	Zero-shot	Reached 0.9	$T_{0.9}$ (IQM, 95% CI)	AUC	Final
DoorKey	**NeuSAT (our approach)**	**0.84**	**100%**	**1.3k [0.8, 1.8]**	**0.98**	**1.00**
	NeuSAT stage 1 from scratch	0.00	99%	66.5k [62.6, 70.4]	0.57	0.98
	PPO from scratch	0.00	43%	136.6k [125.3, 146.6]	0.18	0.42
	PPO fine-tuned	0.00	83%	64.7k [50.3, 81.6]	0.51	0.85
	JSRL	0.00	30%	148.4k [144.4, 150.0]	0.08	0.32
	CKA-RL	0.00	33%	145.6k [136.2, 150.0]	0.16	0.34
Crafter	**NeuSAT (our approach)**	**0.29**	**100%**	**8.5k [7.2, 10.1]**	**0.95**	**1.00**
	NeuSAT stage 1 from scratch	0.00	88%	110.3k [106.4, 114.7]	0.29	0.94
	PPO from scratch	0.00	47%	130.6k [116.2, 144.1]	0.22	0.38
	PPO fine-tuned	0.00	27%	149.4k [143.8, 150.0]	0.11	0.24
	JSRL	0.00	13%	150k	0.04	0.13
	CKA-RL	0.00	17%	150k	0.07	0.15
Sokoban	**NeuSAT (our approach)**	**0.07**	**100%**	**8.5k [7.2, 10.3]**	**0.88**	**0.99**
	NeuSAT stage 1 from scratch	0.00	100%	51.8k [47.7, 56.0]	0.65	0.97
	PPO from scratch	0.00	60%	104.7k [93.4, 115.0]	0.35	0.52
	PPO fine-tuned	0.00	47%	136.2k [123.4, 146.2]	0.22	0.45
	JSRL	0.00	43%	129.7k [118.4, 140.3]	0.20	0.39
	CKA-RL	0.00	40%	130.9k [113.1, 145.9]	0.25	0.43

Table 1: Summary of transfer performance on 15 held-out targets per domain within 150k steps. Zero-shot: success before target interaction; Reached 0.9: percentage of runs achieving success ≥ 0.9 for three consecutive evaluations within 150k steps; $T_{0.9}$: the first evaluation step in a sequence of three consecutive evaluations with success ≥ 0.9, reported as the interquartile mean (IQM) with a 95% confidence interval; AUC: normalized area under the success curve over 150k steps; Final: success at 150k steps. Runs that do not satisfy this criterion within the budget are assigned $T_{0.9} = 150\text{k}$ and included in the IQM and CI.

Domain	Method	Zero-shot	Reached 0.9	$T_{0.9}$ (IQM, 95% CI)	AUC	Final
DoorKey	NeuSAT	0.84	100%	1.3k [0.8, 1.8]	0.98	1.00
	Direct transfer (T_1)	0.00	32%	146.0k [139.1, 149.6]	0.16	0.35
Crafter	NeuSAT	0.29	100%	8.5k [7.2, 10.1]	0.95	1.00
	Direct transfer (T_1)	0.00	16%	150k	0.07	0.20
Sokoban	NeuSAT	0.07	100%	8.5k [7.2, 10.3]	0.88	0.99
	Direct transfer (T_1)	0.00	48%	131.3k [120.9, 139.5]	0.23	0.52

Table 2: Direct transfer versus semantic distillation on 15 held-out targets per domain. Column definitions follow Table 1.

than the fastest baseline. Its success AUC is the highest in all three domains ($0.98/0.95/0.88$), with final success of $1.00/1.00/0.99$. These results show that the transferred abstraction accelerates target learning even when its initial policy rarely succeeds, as in Sokoban. The full target environments learning curves are shown in Appendix A.5

5.3 Ablation Study

We test whether semantic distillation provides a better transfer prior than the source visual abstraction, and how value initialization and target refinement affect adaptation.

Necessity of Semantic Distillation A source-trained visual tree can solve the source task, but its partition may not remain useful when the layout changes. We test whether semantic distillation is needed by initializing the same Stage 3 learner with either the source solution $(T_1, Q_{\text{abs},1})$ or its distilled counterpart $(T_P, Q_{\text{abs},P})$. Both come from the same source teacher and are evaluated on the same 15 held-out targets per domain.

Direct visual transfer has zero-shot success of 0 in all three domains (Table 2). Even after target learning, only $32\%/16\%/48\%$ of its DoorKey/Crafter/Sokoban runs reach 90% success. With semantic distillation, every run reaches this threshold, and success AUC over the 150k-step 15 target

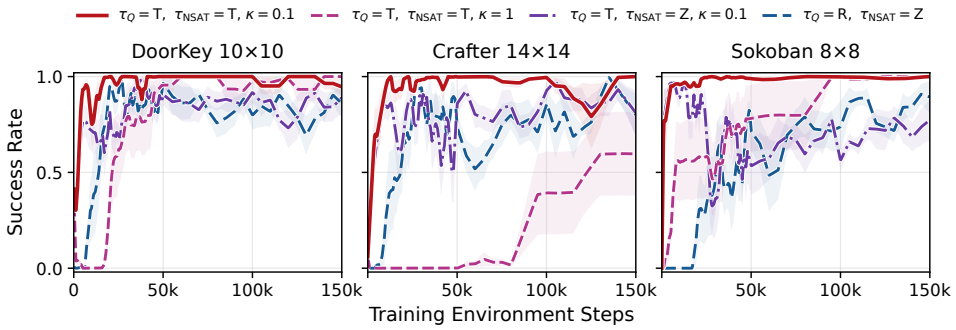

Figure 4: Effects of value initialization and target refinement

adaptation curves rises from $0.16/0.07/0.23$ to $0.98/0.95/0.88$. Continuing to learn from the source visual tree therefore does not reliably overcome its source-specific partition. The distilled propositional tree provides a more useful starting point for target adaptation. Per-target learning curves are shown in Appendix A.4.

Value initialization and target refinement. We test whether transferred values need rebasing and whether the transferred tree needs further refinement. We compare three value initializations: rebased ($\kappa = 0.1$), unchanged ($\kappa = 1$), and zero-initialized, with either a fixed or refined tree. All six variants start from the same T_P on one target per domain, with five seeds each.

We label configurations by $(\tau_Q, \tau_{\text{NSAT}})$, where τ_Q represents transfer of Q value, τ_{NSAT} represents transfer of NSAT, $\tau_Q, \tau_{\text{NSAT}} \in \{\textbf{(Z)}eroShot, \textbf{(R)}eset, \textbf{(T)}ransfer+Learn\}$. For example, (T, Z) updates transferred values while keeping T_P fixed. The full method uses (T, T) with $\kappa = 0.1$. Figure 4 shows four variants; the result for all six variants is in Appendix A.7.

Transferred values accelerate adaptation, but preserving their source scale can hinder learning. Raw and rebased values induce the same zero-shot policy, yet with refinement, rebasing raises success AUC from $0.80/0.21/0.80$ to $0.96/0.94/0.98$ on DoorKey/Crafter/Sokoban. The mean success reaches 0.9 after approximately 5k/9k/5k steps, compared with 45k/88k in DoorKey/Sokoban and no crossing within 150k steps in Crafter for raw values.

Refinement helps sustain this improvement. With rebased values and a fixed tree, $16\%/27\%/30\%$ of evaluations after a seed first reaches 90% success fall below 80%, versus $5\%/5\%/0\%$ with refinement. Together, rebasing and refinement achieve the highest AUC in all three tested targets, supporting both the adaptation of transferred values and the revision of transferred abstractions.

6 Conclusion

The main finding is that an abstraction optimized for source learning need not be the abstraction that should be transferred. NeuSAT first learns a Neuro-Symbolic abstraction tree and policy, then uses them as a teacher to build a proposition-routed tree. The tree and its action values provide an initial target prior; target interaction updates the values and adds local visual splits when a leaf groups states that require different decisions. Across 15 held-out targets per domain in MiniGrid-DoorKey, Crafter, and Sokoban, NeuSAT reaches 90% success in every run, with useful zero-shot behavior in DoorKey and Crafter. In Crafter and Sokoban, it requires 92% and 84% fewer target interactions than the fastest baseline, respectively. By comparison, transferring the source tree directly and using the same target learner reaches 90% success in only $32\%/16\%/48\%$ of DoorKey/Crafter/Sokoban runs. These results suggest that a source policy can guide the construction of reusable semantic abstractions when the transferred structure remains adaptable. Future work will examine how to expand the propositional vocabulary and revise internal rules as tasks change. Limitations are discussed in Appendix A.2.

AI USE STATEMENT

Large language models (LLMs) are a component of the proposed method: they generate the perception parser and the proposition vocabulary from images, as described in Section 4; the prompts are given in Appendices A.8, A.9. Separately, we used LLM-based assistants to polish and paraphrase the text and to help with software development. The authors reviewed all AI-assisted text and code and take full responsibility for the content of this paper.

ETHICS STATEMENT

This work studies reinforcement learning in simulated grid-world environments (DoorKey, Crafter and Sokoban). It involves no human subjects, no personal or sensitive data, and no deployment in the real world. We do not foresee direct negative societal impacts.

REPRODUCIBILITY STATEMENT

The full algorithm is given in Section 4, and implementation details including hyperparameters are in Appendix A.1. A single configuration is used for all domains. The LLM prompts are included, and we will release the code as an attachment with this submission.

REFERENCES

Fernando Acero and Zhibin Li. Distilling reinforcement learning policies for interpretable robot locomotion: Gradient boosting machines and symbolic regression. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6840–6847. IEEE, 2024.

Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. *Advances in neural information processing systems*, 31, 2018.

Jianda Chen and Sinno Jialin Pan. Learning generalizable representations for reinforcement learning via adaptive meta-learner of behavioral similarities. *arXiv preprint arXiv:2212.13088*, 2022.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and J K Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *Advances in neural information processing systems*, 36:73383–73394, 2023.

Vinicius G Costa, Jorge Perez-Aracil, Sancho Salcedo-Sanz, and Carlos E Pedreira. Evolving interpretable decision trees for reinforcement learning. *Artificial Intelligence*, 327:104057, 2024.

Mehdi Dadvar, Rashmeet Kaur Nayyar, and Siddharth Srivastava. Conditional abstraction trees for sample-efficient reinforcement learning. In *Uncertainty in Artificial Intelligence*, pp. 485–495. PMLR, 2023.

Rajdeep Dutta, Qincheng Wang, Ankur Singh, Dhruv Kumarjiguda, Li Xiaoli, and Senthilnath Jayavelu. Interpretable policy extraction with neuro-symbolic reinforcement learning. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 7570–7574. IEEE, 2024.

Danijar Hafner. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*, 2021.

Jinwu Hu, Zihao Lian, Zhiquan Wen, Chenghao Li, Guohao Chen, Xutao Wen, Bin Xiao, and Mingkui Tan. Continual knowledge adaptation for reinforcement learning. *Advances in Neural Information Processing Systems*, 38:36283–36313, 2026.

Dongfen Li, Lichao Meng, Jingjing Li, Ke Lu, and Yang Yang. Domain adaptive state representation alignment for reinforcement learning. *Information Sciences*, 609:1353–1368, 2022.

Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for mdps. *AI&M*, 1(2):3, 2006.

Ziyan Luo, Tianwei Ni, Pierre-Luc Bacon, Doina Precup, and Xujie Si. Understanding behavioral metric learning: A large-scale study on distracting reinforcement learning environments. *arXiv preprint arXiv:2506.00563*, 2025.

Sascha Marton, Tim Grams, Florian Vogt, Stefan Lüdtke, Christian Bartelt, and Heiner Stuckenschmidt. Mitigating information loss in tree-based reinforcement learning via direct optimization. In *International Conference on Learning Representations*, volume 2025, pp. 37083–37115, 2025.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Rashmeet Kaur Nayyar, Naman Shah, and Siddharth Srivastava. Context-sensitive abstractions for reinforcement learning with parameterized actions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 24522–24531, 2026.

Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Farzaneh Shoeleh and Masoud Asadpour. Skill based transfer learning with domain adaptation for continuous reinforcement learning domains. *Applied Intelligence*, 50(2):502–518, 2020.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Jaehyeon Son, Soochan Lee, and Gunhee Kim. Distilling reinforcement learning algorithms for in-context model-based planning. In *International Conference on Learning Representations*, volume 2025, pp. 37293–37310, 2025.

Ikechukwu Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, Joséphine Simon, Matthew Bennice, Chuyuan Fu, Cong Ma, Jiantao Jiao, et al. Jump-start reinforcement learning. In *International Conference on Machine Learning*, pp. 34556–34583. PMLR, 2023.

Abhishek Verma, Balaraman Ravindran, et al. Generalized adaptive transfer network: Enhancing transfer learning in reinforcement learning across domains. *arXiv preprint arXiv:2507.03026*, 2025.

Daniël Vos and Sicco Verwer. Optimizing interpretable decision tree policies for reinforcement learning. *arXiv preprint arXiv:2408.11632*, 2024.

Rongrong Wang, Yuhu Cheng, and Xuesong Wang. Constrained visual representation learning with bisimulation metrics for safe reinforcement learning. *IEEE Transactions on Image Processing*, 34:379–393, 2025.

Hayato Watahiki, Ryo Iwase, Ryosuke Unno, and Yoshimasa Tsuruoka. Cross-domain policy transfer by representation alignment via multi-domain behavioral cloning. *arXiv preprint arXiv:2407.16912*, 2024.

CJCH Watkins et al. Learning from delayed rewards.(1989), 1989.

Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*, 2020.

Yivan Zhang, Ziyan Luo, and Manuel Baltieri. Compositional behavioral semantics for state abstraction in reinforcement learning. *arXiv preprint arXiv:2606.25357*, 2026.

Zhuangdi Zhu, Kaixiang Lin, Anil K Jain, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(11): 13344–13362, 2023.

A Appendix

A.1 Implementation Details

Test environment All experiments were run on a single workstation with an AMD Ryzen Threadripper PRO 7975WX (32 cores, 64 threads) and 64 GB of RAM, under Ubuntu 24.04.

All three domains use one configuration; hyperparameter values are listed in Table 3.

Observations The agent observes an RGB rendering of the whole grid. A concrete state is identified by a hash of the observation downsampled to 84×84. Visual routers operate on frozen ImageNet ResNet-18 features (512-d, averagepool) of the same 84×84 image. No simulator state or scene annotation is used to construct or execute the pixel parser. For each domain, parser generation uses only six unlabeled RGB frames from the source environment, the domain description and legend, and statistics computed directly from those image pixels.

Split score (Stage 1). The tree is refined every C_{ref} environment steps. In the second half of each period, the learner records the TD errors δ of Q_{abs} and the concrete states visited. The errors are computed against a copy of Q_{abs} frozen at the start of this window. Following PEARL (Nayyar et al., 2026), the heterogeneity of a leaf combines the dispersion of its TD errors with the dispersion of its concrete values:

$$H(z) = \Big[\beta \max_a \mathrm{SD}\left(\delta_{z,a}\right) + (1-\beta)\,\mathrm{SD}_{k\in z}\left(\hat{V}(k)\right)\Big], \qquad \hat{V}(k) = \max_a Q_{\text{con}}(k,a). \tag{3}$$

As in PEARL, β starts at 1: splits are first driven by inconsistent TD targets and later by differences in value. After each refinement that accepts a split, β decreases by $\Delta\beta$, down to $\beta_{\min}$. Leaves with $H(z) > \theta$ are split in decreasing order of H.

Router training and child-value initialization. A router learns to reproduce a proposed split from pixels. Its training set has one example per concrete state of the leaf: the input is the frozen ResNet-18 feature of the state's frame (the 84×84 image resized to 224×224, with ImageNet normalization), standardized by the mean and standard deviation over these examples, and the label is the state's Ward cluster. The router is an MLP with one hidden layer of 64 ReLU units, trained full-batch with Adam (learning rate 10^{-2}, weight decay 10^{-4}) on a class-balanced cross-entropy for at most 300 epochs, and training stops once every example is classified correctly. The split is accepted if the router's training accuracy is at least $\rho = 0.95$. After a visual split (Stages 1 and 3), both children copy the parent's row of Q_{abs}. After a propositional split (Stage 2), each child starts from the mean teacher Q-vector of the states routed to it; the values are then refitted on $\mathcal{D}_P$, and actions that do not occur in $\mathcal{D}_P$ keep this initial value.

Stage-2 normalization. Let $\mathcal{K}_P$ be the distinct keys in $\mathcal{D}_P$ and $V_1(k) = \max_a q_1(k,a)$ the teacher's value of key k. The value scale in Eq. (1) is the 5th–95th percentile range of these values over the distinct keys (unweighted),

$$c_1 = \max\Big(1,\ P_{95}\big\{V_1(k)\big\}_{k\in\mathcal{K}_P} - P_5\big\{V_1(k)\big\}_{k\in\mathcal{K}_P}\Big), \tag{4}$$

where the floor of 1 only guards against a degenerate range and is not active in our runs. A leaf is split only while $H_P(z) > \epsilon_{\text{rel}}\, H_P(\text{root})$ with $\epsilon_{\text{rel}} = 0.01$, so the threshold is relative to the root and does not depend on the reward scale. Here, $H_P(\text{root})$ is computed on the initial single-leaf tree and held fixed throughout distillation. The action-preference vector of a key is its teacher advantage vector scaled to unit length,

$$u_1(k) = \frac{A_1(k)}{\|A_1(k)\|_2}, \qquad A_1(k,a) = q_1(k,a) - \max_b q_1(k,b), \tag{5}$$

with $u_1(k) = \mathbf{0}$ when all actions tie. It retains how the teacher ranks and separates the actions but not the magnitude of the values.

Stage	Component	Value
1 & 3	value learning	Watkins Q(λ), γ=0.9, λ=0.9; α_{abs}=0.1, α_{con}=0.5
	exploration	ϵ-greedy, ϵ=0.1
	refinement	every 2.5k steps (half stabilize, half collect)
	leaf score H	β: $1 \to 0.1$ (-0.05 per refinement); threshold 0.005
	split	2-cluster Ward on Q-vectors; $\geq$4 visits per cluster
	visual router	MLP (512–64–2) on ResNet-18 features; train acc. ≥ 0.95
	tree size	$\leq$3000 leaves
2	teacher data	15k episodes, random starts, ϵ_b=0.15
	value fit	SARSA(λ), λ=0.9
	rules	p, $p \wedge q$, $p \vee q$
	tree size	$\leq$16 splits per round, $\leq$1000 leaves
all	evaluation	greedy, fixed start, 5% action noise; every 1k steps

Table 3: NeuSAT hyper-parameters, shared by all domains.

LLM configuration. The proposition vocabularies and the parsers were generated by `claude-opus-5-5`, queried on 2026-09-23 through the Anthropic Messages API. Each request is a single user turn that holds the prompt of Appendices A.8 and A.9 and its example frames, with no system prompt and no tools. We use adaptive thinking, the API's default sampling (temperature and top-p not set) and a limit of 64,000 output tokens, which no reply reached. Each of the six modules (one vocabulary and one parser per domain) was accepted in the first round, so each was produced by a single API call. They were taken from the first complete run with the final prompts and are used unchanged, with no manual edits.

Baselines

- **PPO** uses Stable-Baselines3 with a CNN policy. We evaluate both training from scratch and fine-tuning a source-trained policy. The same source-trained policy serves as JSRL's guide.
- **JSRL** (Uchendu et al., 2023) uses the curriculum variant with the source-trained PPO policy as its guide. The guide's zero-shot success is at most 0.02 on the targets. Guide interactions count toward the target budget.
- **CKA-RL** (Hu et al., 2026) uses the authors' implementation. In our single-source setting, the target encoder and actor are initialized from the source model, while the critic is initialized anew.

Hyperparameters Table 3 lists the single setting used in every domain and stage. Only the interaction budgets vary: Stage 1 trains for 200k/200k/500k steps (The separate Stage 1 comparison in Section 5.1 uses 300k steps per domain.), and Stage 2 uses $15{,}000$ teacher episodes, with a 150k-step budget per target.

A.2 Limitation

NeuSAT assumes shared proposition semantics across environments; changes in task semantics may require adapting the proposition vocabulary, which we leave to future work. Our current formulation also assumes discrete actions; extending it to continuous action spaces would require corresponding changes to action-value estimation and action selection.

A.3 EXAMPLE OF A PROPOSITION-ROUTED TREE

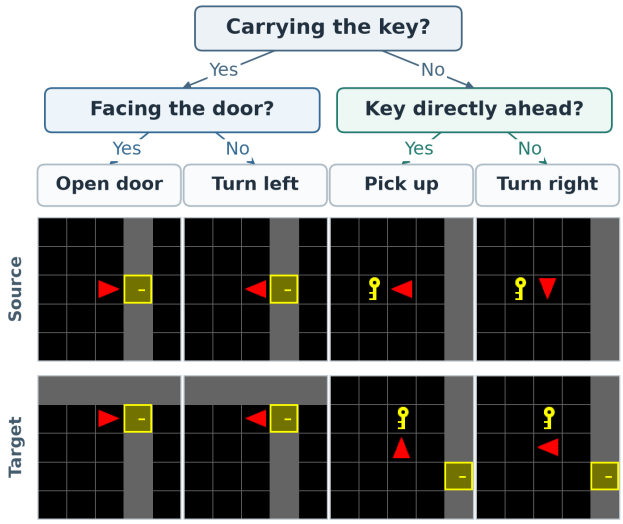

Figure 5: Illustration of proposition-based routing across DoorKey layouts. Within each vertical pair, the source and target observations have the same proposition vector and therefore reach the same leaf and select the same initial greedy action. The displayed branches organize verified decisions for illustration and do not reproduce the exact topology of the learned tree. In the door examples, the door is closed and locked, and "Open door" denotes the toggle action.

A.4 $(T_1, Q_{\text{abs},1})$ VS $(T_P, Q_{\text{abs},P})$ FULL TRANSFER PLOTS

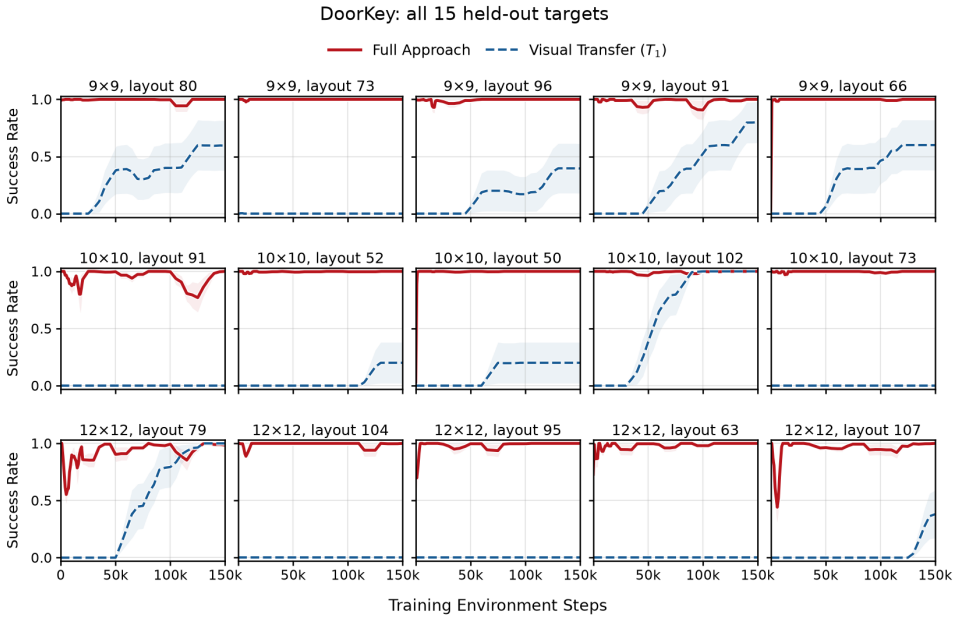

Figure 6: $(T_1, Q_{\text{abs},1})$ vs $(T_P, Q_{\text{abs},P})$ Transfer in Doorkey

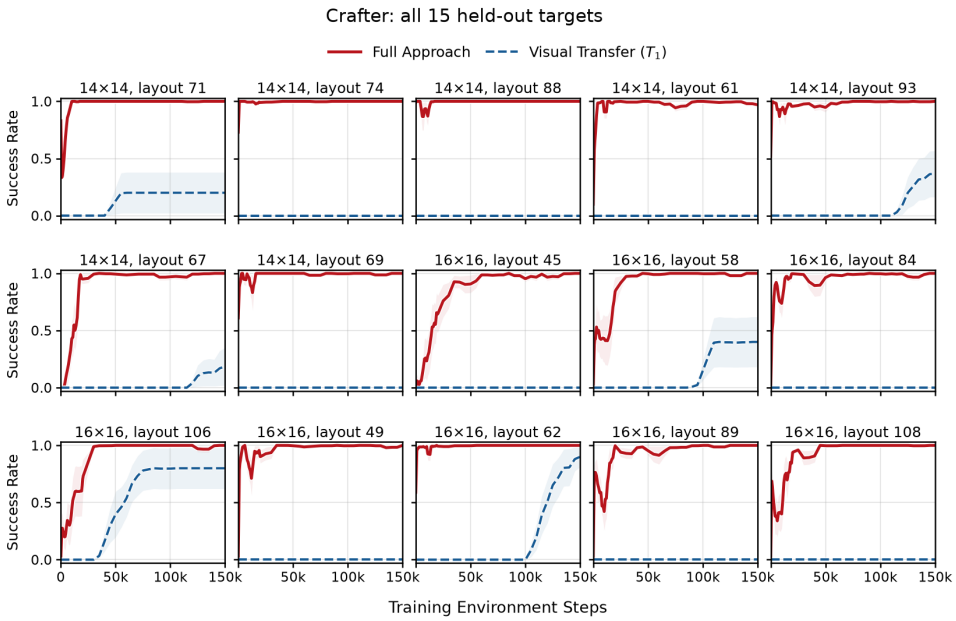

Figure 7: $(T_1, Q_{\text{abs},1})$ vs $(T_P, Q_{\text{abs},P})$ Transfer in Crafter

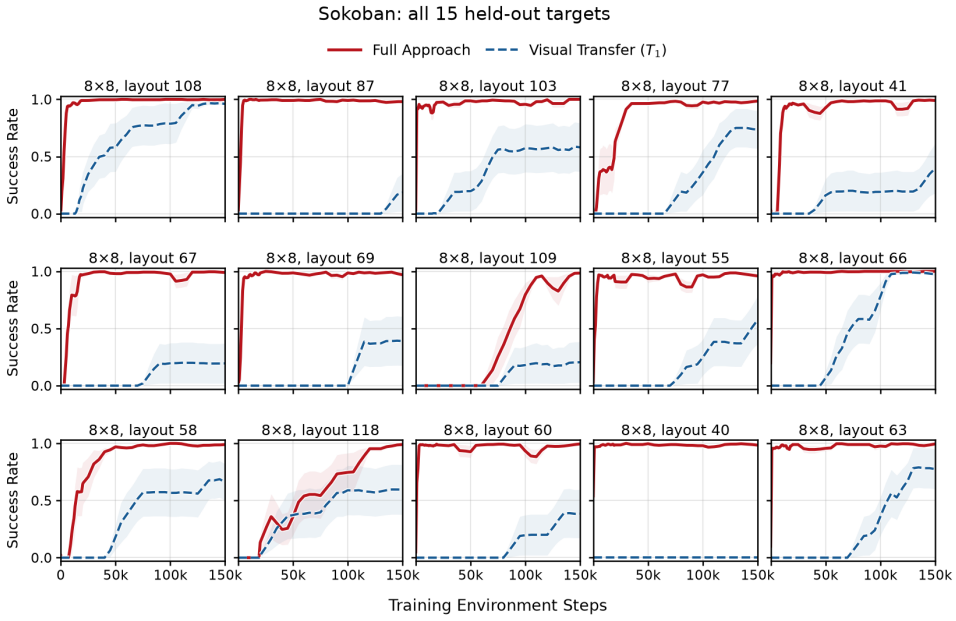

Figure 8: $(T_1, Q_{\text{abs},1})$ vs $(T_P, Q_{\text{abs},P})$ Transfer in Sokoban

A.5 3 DOMAIN, 15 HELD-OUT ENVIRONMENT TRANSFER PLOTS

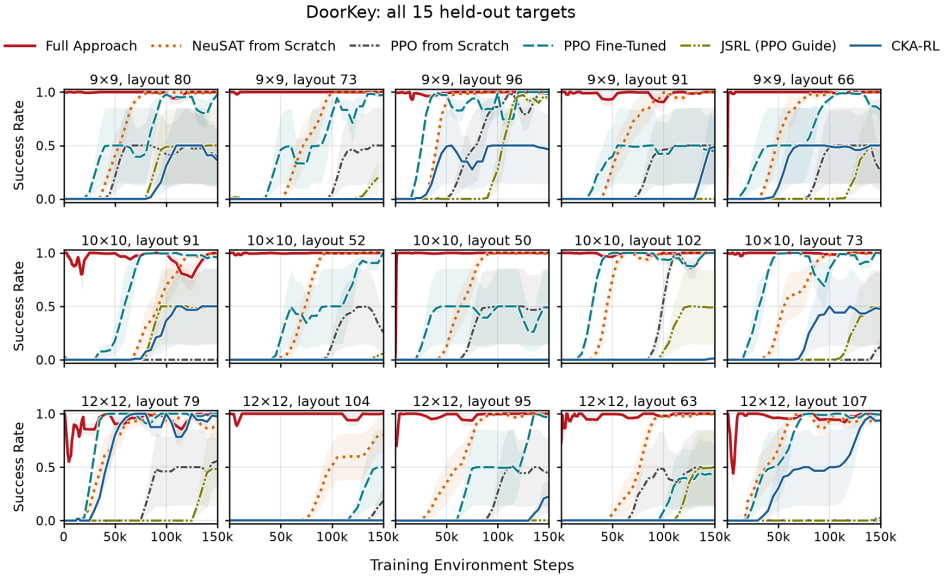

Figure 9: Target learning curves on all 15 held-out Minigrid-DoorKey layouts

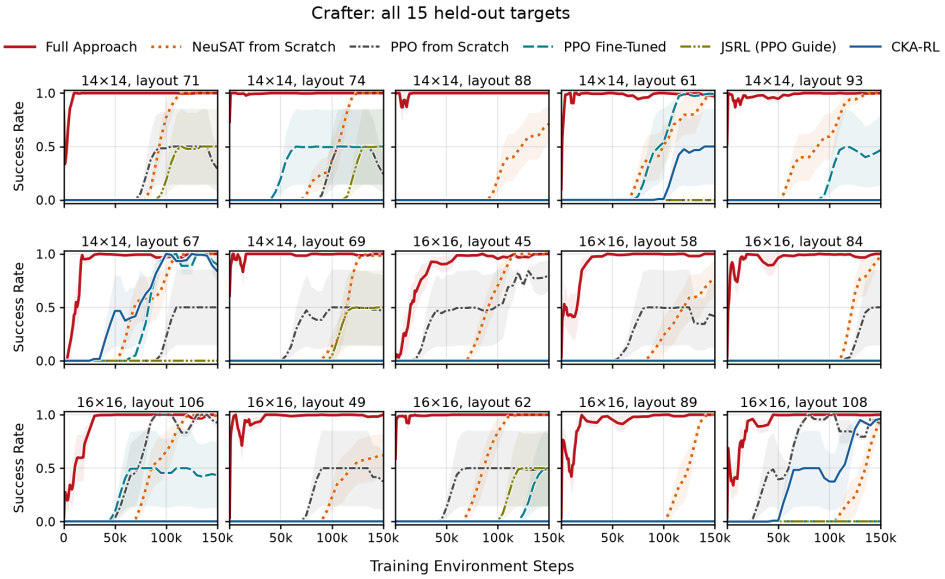

Figure 10: Target learning curves on all 15 held-out Crafter layouts

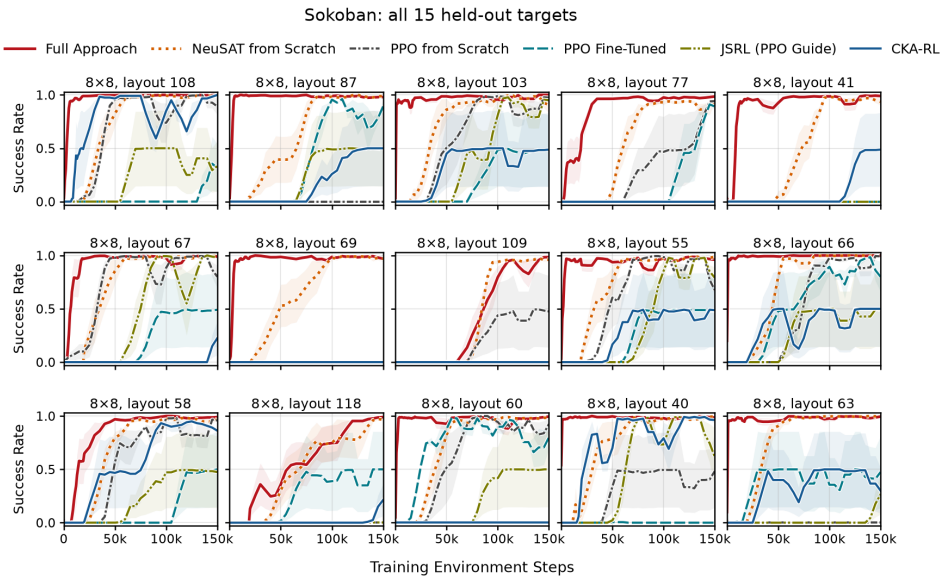

Figure 11: Target learning curves on all 15 held-out Sokoban layouts

A.6 Source layout and 15 held-out environments for each domain



Figure 12: Source layouts, shown at the fixed start and drawn to scale.

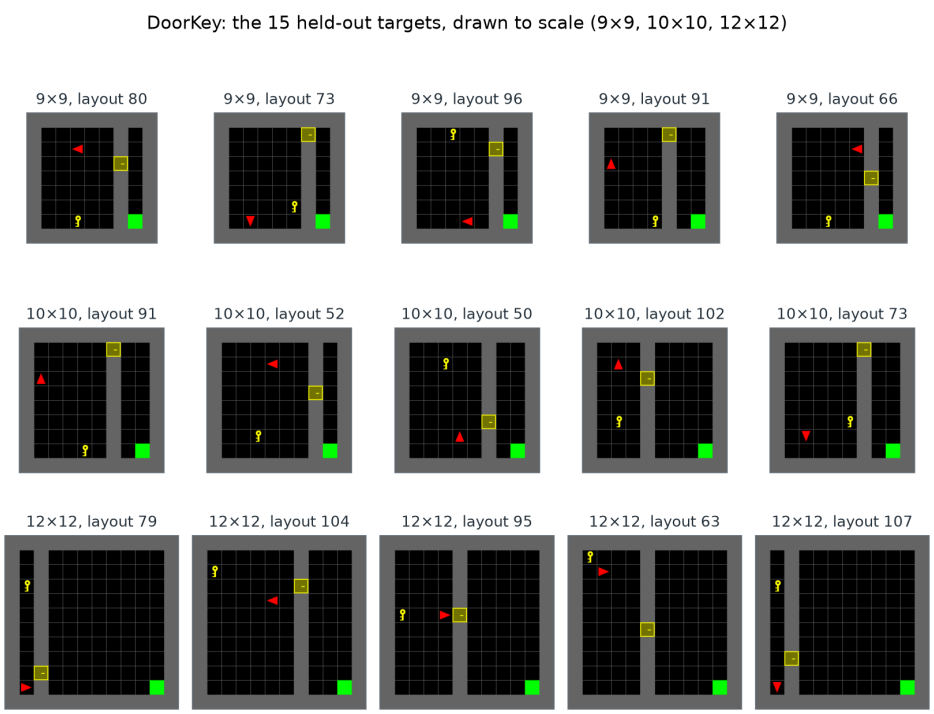

Figure 13: 15 held-out target layouts in Doorkey.

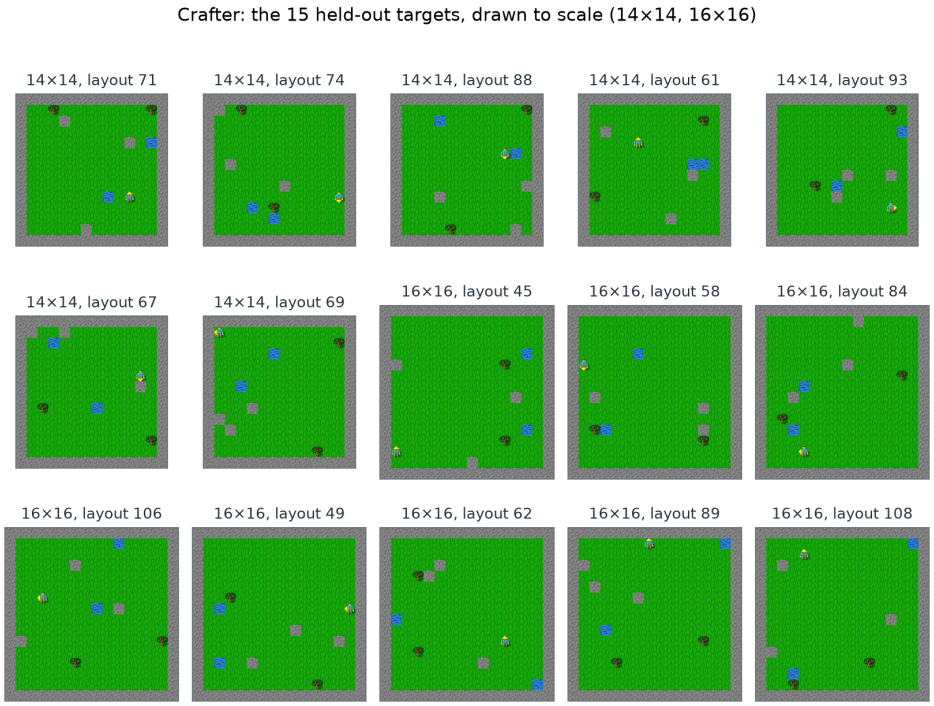

Figure 14: 15 held-out target layouts in Crafter.

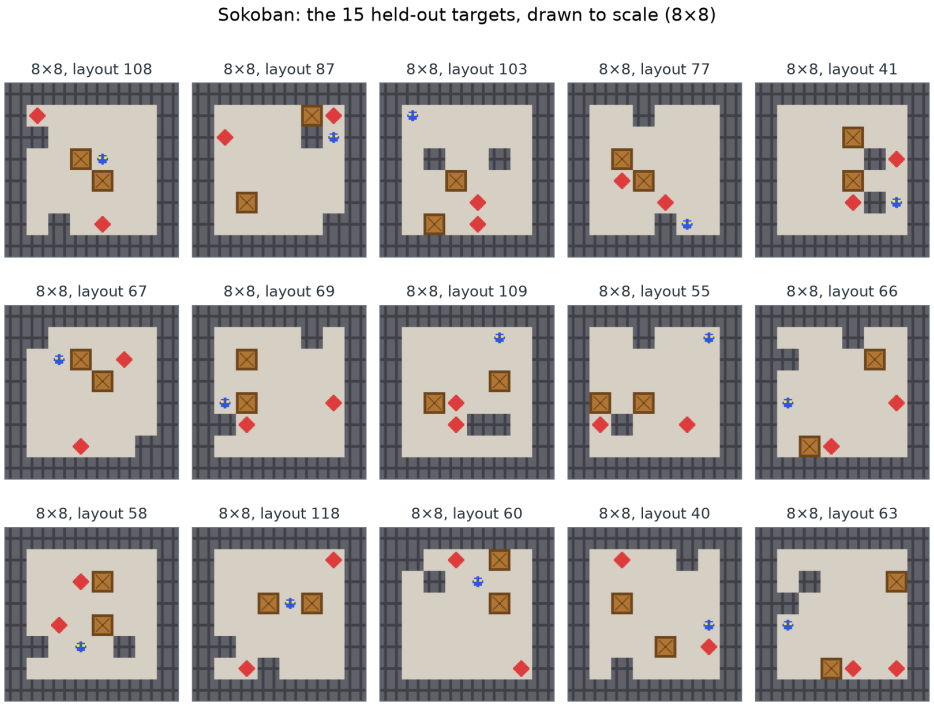

Figure 15: 15 held-out target layouts in Sokoban.

A.7 Full Ablation Study Plot

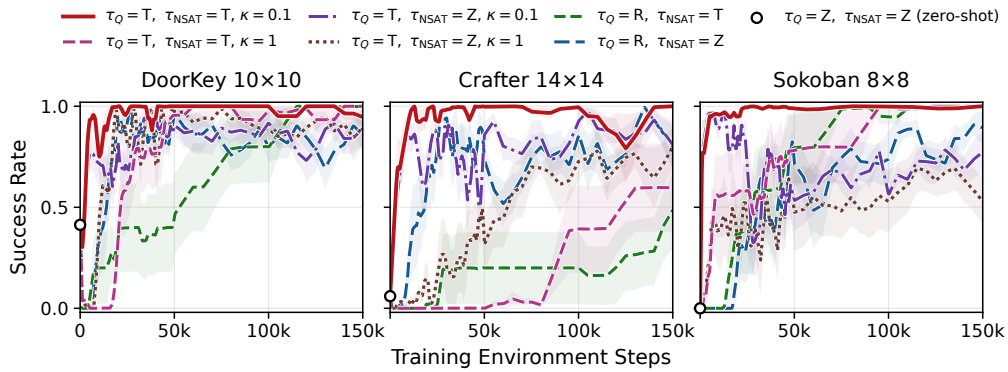

Figure 16: Effects of value initialization and target refinement

A.8 Proposition-vocabulary prompt

Both prompts are single templates shared by all domains; Six random example frames from single source environment for each domain, are attached as images before the text.

```
You are designing a PROPOSITION VOCABULARY for an interpretable reinforcement-learning
agent in the grid-world task described below. Output ONLY a single Python module.

Your goal is to identify candidate semantic propositions for this domain using examples from
↪  this single source environment only.

The agent's policy will be a decision tree whose every internal node tests one of
your propositions. Each proposition must describe a recognizable semantic fact,
and the vocabulary should provide candidate semantic distinctions that remain
meaningful across layouts of different sizes and arrangements.
```

```
WHAT A PROPOSITION READS: THE SCENE
  Every proposition is a function scene -> bool of one dict, the same for every frame:

    scene["map"]        list of strings, one per grid row (y = 0 at the top), one
                        character per cell (x = 0 at the left), saying what occupies
                        the cell now:
<<LEGEND>>
                        The agent is not drawn in the map: its own cell shows what it
                        stands on (or in).
    scene["agent_pos"]  (x, y), the agent's cell
    scene["agent_dir"]  <<AGENT_DIR_TEXT>>

DIRECTION FRAME OF THIS DOMAIN
  <<FRAME_TEXT>>
  Directional propositions must use exactly these four name suffixes, so that the
  pairing is machine-checkable:
      <<SUFFIXES>>

EXAMPLE FRAMES OF SOURCE ENV (attached above this text).They show what the agent sees; the
↪  vocabulary must work on layouts nobody has seen yet.

REQUIREMENTS

1. RELATIONAL AND SCALE-FREE. No absolute coordinates, no words for a place on the map
   ("top_half", "near_border"; direction suffixes of relations are fine), no fixed
   cell-count thresholds. Every proposition is stated relative to the agent and to
   concrete objects, and must mean the same thing on a layout of another size.
   Surrounding the map with extra rings of '<<WALL>>' cells changes no relation between the
   agent and any object, so it must not change the value of any proposition;

2. DIRECTIONS COME IN COMPLETE SETS. Where you name a direction, name all four with
   the suffixes above. A left-right mirror of the map is an exact isomorphism of this
   task once the left and right actions are swapped, so the vocabulary must satisfy
   vec(mirror(s)) = P . vec(s), where P swaps the <<LEFT>> / <<RIGHT>> member of every pair
   and fixes everything else. Never rely on the order of a sorted list of objects or on
   ↪  tie-breaking by
   coordinates -- quantify over the objects (any / all / count) instead.

3. FACTS, NOT DECISIONS. Do not write a proposition that names the correct next action,
   the stage of the task, or the object the agent should handle next. Object
   states and geometry are facts and are fine; "should_get_X" is not. The words
   "target", "subgoal", "objective", "current_aim", "goal_object", "next_goal",
   "desired" (or any synonym for the-thing-I-am-heading-for) MUST NOT appear in any
   proposition name (names are checked mechanically).

4. NOTHING TO POINT AT. When an object is gone, unreachable, or the task is finished,
   decide explicitly what your propositions return there and be consistent -- a bearing
   toward something that does not exist or cannot be reached is False, never an
   exception. A proposition must never raise and must never modify the scene.

5. EVERY PROPOSITION MUST VARY. Avoid propositions that are constant across all
   provided examples from the source environment, except when needed as a member
   of a complete directional set.

6. NO SILENT STATES. For each of these object words -- <<OBJECT_WORDS>> (each names a
   kind of map cell in the legend) -- at least one proposition whose name contains the
   word must be True in EVERY state from which the agent can walk to that object or to a
   cell beside it, including the states right beside it, where a bearing may already be
   False and the agent still has to act on it.

7. SMALL AND COMPLETE. Aim for 20-50 propositions. Every one must earn its place,
   and together they should provide sufficient candidate distinctions for a compact
   decision tree.

OUTPUT FORMAT

A standalone module (standard library only, optionally `from collections import deque`
and `functools`) that defines, at the end:

    PROPOSITIONS: list[tuple[str, Callable[[dict], bool]]] = [ ... ]
    PROPOSITION_NAMES = [n for n, _ in PROPOSITIONS]
    def prop_vec(scene) -> list[int]:
        return [int(bool(fn(scene))) for _, fn in PROPOSITIONS]

Proposition names are lowercase snake_case and unique. Group the propositions with short
comments saying what each group is FOR. No prose outside the code block, no
`if __name__ == "__main__"`, no I/O, no randomness.

Return exactly one ```python fenced code block.
```

A.9 PIXEL-PARSER PROMPT

For each domain, the parser is generated from exactly six RGB frames sampled from the single source environment. These frames are provided without scene annotations, object locations, agent states, or any other state-level ground-truth information. In addition to the images, the prompt contains the domain rules and map legend, together with `STATS`, which is computed mechanically from the pixels of the same six frames. No simulator state, additional source data, target observations, or held-out environments are available during parser generation.

`STATS` contains image-level statistics computed directly from the six RGB frames, including exact colour information and spatial colour-occupancy patterns used to help distinguish rendered objects and the agent. It contains no hand-written description of the rendering and is computed without scene labels or simulator information. The six frames are sampled from the same source environment and attached as images. The template assumes a grid-aligned rendering with square tiles drawn from the top-left corner.

```
You are writing deterministic computer-vision code. Output ONLY a single Python module.

TASK
Write `parse_image(image) -> dict` that reads one rendered frame of the grid-world task
described below and returns its SCENE, a dict with exactly these three keys:

  "map"        list of strings, one per grid row (y = 0 at the top), one character per
               cell (x = 0 at the left), saying what occupies the cell now:
<<LEGEND>>
               The agent is not a map character: its own cell must show what the agent
               stands on (or in).
  "agent_pos"  (x, y) integer cell coordinates of the agent.
  "agent_dir"  <<AGENT_DIR_TEXT>>

THE FRAME
  `image` is a uint8 numpy array of shape (H, W, 3) (also accept a PIL image or a
  float array by converting at the top). The whole map is drawn from the top-left
  corner at <<TILE>> pixels per cell: cell (x, y) occupies rows
  [y*<<TILE>>, (y+1)*<<TILE>>) and columns [x*<<TILE>>, (x+1)*<<TILE>>).
  The number of rows and columns MUST be derived from the image shape
  (H // <<TILE>>, W // <<TILE>>). The parser will be run unchanged on layouts
  of different sizes and arrangements. Never hard-code the grid size or any coordinate.

AVAILABLE INFORMATION
  You are given exactly six RGB example frames sampled from the single source
  environment. The images contain NO scene annotations, object locations,
  agent-state labels, or other state-level ground truth.

  You are also given the task rules, the map legend above, and image statistics
  computed mechanically from the pixels of these same six frames.

  These six images and the information explicitly included in this prompt are
  the ONLY information available for constructing the parser. You may NOT query
  or access the simulator, simulator state, hidden environment state, additional
  labeled or unlabeled frames, external datasets, or any held-out target environment.

IMAGE STATISTICS DERIVED FROM THE SIX SOURCE FRAMES
  (computed mechanically from RGB pixels of the six images only; no scene labels,
  simulator state, or frame/scene annotations are used)
<<STATS>>

EXAMPLE FRAMES FROM THE SINGLE SOURCE ENVIRONMENT
  (the <<N_IMAGES>> images attached above this text are sampled from the same
  source environment and are provided without state annotations)
<<SCALE_NOTE>>

REQUIREMENTS
1. The generated parser must be deterministic pure Python + numpy (PIL optional).
   It must contain NO learned components, NO randomness, NO file or network I/O,
   NO simulator access, and NO hard-coded positions or grid sizes.
2. Infer how rendered objects and agent states correspond to the map legend using
   only the supplied RGB examples, task rules, legend, and image statistics.
3. Classify cells from their image pixels using deterministic rules. Do not use
   information that is unavailable in the input image at runtime.
4. Find the agent from visual evidence in the frame and recover agent_pos,
   agent_dir using only information visible in the rendered image.
5. Return plain Python ints / str / tuples / lists / dicts (no numpy scalars).
6. It must be fast: well under 50 ms per frame.
7. At runtime, `parse_image` must use only the input RGB image. It must not query,
   call, or otherwise access the simulator, environment state, external data,
   or any hidden information.
```

```
Before returning the module, verify that the code can be applied consistently to
all six provided example images and that its output satisfies the required SCENE
format. No ground-truth SCENE is provided for these examples.

OUTPUT FORMAT
Return exactly one ```python fenced code block containing the complete standalone
module: imports, any helpers, and `parse_image`. No prose outside the block. No
`if __name__ == "__main__"` block.
```

A.10 PIXEL-PARSER GENERALIZATION WITHOUT SCENE SUPERVISION

We additionally evaluate whether the pixel parser can work without any state-level annotations or simulator supervision. For each domain, we independently generate three parsers using only six RGB frames randomly sampled from the single source environment. The frames are provided without scene annotations, agent-state labels, or object locations. No simulator state or target observation is available during parser generation.

For evaluation only, we sample 20 frames from each of the 15 held-out target layouts, giving 300 test frames per domain. Ground-truth scene information is used only after parser generation to score the predictions. We report *Scene Exact*, which requires the entire parsed scene, including the map, agent position, and direction to be correct, and *Proposition Exact*, which requires the complete proposition vector derived from the parsed scene to match that derived from the ground-truth scene.

Across three independent parser generations per domain, all nine generated parsers achieve 100% Scene Exact and 100% Proposition Exact on their corresponding 300 held-out frames. Each parser is evaluated in its initial generated form; no parser is manually edited or selected based on target performance.

Domain	Generations	Frames / generation	Scene Exact (%)	Proposition Exact (%)
DoorKey	3	300	100.0	100.0
Crafter	3	300	100.0	100.0
Sokoban	3	300	100.0	100.0

Table 4: Pixel-parser accuracy on held-out target layouts. Parsers are generated from six unlabeled source RGB frames without state annotations or simulator access. Ground truth is used only for evaluation.