Learning Relational Concepts in Neuro-Symbolic Reinforcement Learning

Erlernen von relationalen Konzepten in Neuro-Symbolischem Reinforcement Learning
Master thesis in the department of Computer Science by Henri Johannes Rößler
(Student ID: 2758240)
Date of submission: September 30, 2025

1. Review: Prof. Dr. Kristian Kersting
2. Review: M.Sc. Hikaru Shindo
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Computer Science Department

Artificial Intelligence and Machine Learning Lab

Abstract

Neuro-symbolic agents rely on logical rules to infer their actions, which often requires knowledge about how objects are related to each other. Understanding concepts such as *left of* or *nearby* is therefore essential for solving abstract tasks. In existing systems, such relations are typically defined by human experts, which limits extensibility since the meaning of a concept can vary across different environments. To address this limitation, we introduce a novel framework that grounds relational concepts through interaction with the environment. We further utilize large language models to provide additional weak supervision and to complement sparse reward signals. Empirical evaluations on the ATARI environments *Kangaroo* and *Seaquest* demonstrate that our agents match, and in some cases exceed, the performance of logic agents with hand-crafted relational concepts. Furthermore, our framework effectively mitigates concept misalignment in underdetermined environments.

Zusammenfassung

Neuro-symbolische Agenten verwenden logische Regeln, um ihre Handlungen abzuleiten, was häufig Wissen darüber erfordert, wie Objekte zueinander in Beziehung stehen. Das Verständnis von Konzepten wie *links von* oder *in der Nähe von* ist daher unerlässlich um abstrakte Aufgaben zu lösen. In bestehenden Systemen werden solche Beziehungen typischerweise von menschlichen Experten definiert, was die Erweiterbarkeit einschränkt, da die Bedeutung eines Konzepts je nach Umgebung variieren kann. Um diese Einschränkung zu überwinden, stellen wir ein neuartiges System vor, das relationale Konzepte durch Interaktion mit der Umgebung erlernt. Darüber hinaus nutzen wir große Sprachmodelle, um zusätzliche schwache Überwachung bereitzustellen und so seltene Belohnungssignale zu ergänzen. Empirische Auswertungen für die ATARI-Umgebungen *Kangaroo* und *Seaquest* zeigen, dass unsere Agenten die Leistung von logischen Agenten mit manuell entworfenen Relationen erreichen und in einigen Fällen übertreffen. Darüber hinaus mindert unser System effektiv Konzept-Fehlausrichtungen in unterbestimmten Umgebungen.

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit erkläre ich, Henri Johannes Rößler, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 30. September 2025

H. Rößler

H. Rößler

Contents

1. Introduction

Deep Reinforcement Learning (RL) has seen a lot of advances in the recent years which pushed forward the deployment of agents in highly critical areas like autonomous driving and robotics. Despite their successes in domain-specific tasks, they often lack essential planning and reasoning capabilities. This limits their ability to solve novel tasks and to adapt to environments they have never been exposed to before – which humans, on the other hand, excel at.

Studies in the fields of philosophy and cognitive science suggest that humans can generalize so well because they perceive the world in terms of *concepts* (Bruner et al., 1956; Rosch, 1973; Mao et al., 2025). A concept can be thought of as an abstract attribute or relation that a set of things can have in common (Archer, 1966). For example, objects might be described by their color, shape and how they are positioned relative to other objects.

Inspired by that, neuro-symbolic agents have emerged that operate on conceptual representations rather than raw sensory input, which is promising for a variety of reasons. Firstly, concepts capture relevant information about an environment at a higher level of abstraction. This facilitates learning policies that are less sensitive to noise and hence more robust to extraneous changes. Secondly, it allows agents to reason over objects in the environment by expressing their policy as logical rules. For instance, one such rule could enforce the agent to move right if it is left of a ladder, using concepts like ”left of” and ”is ladder”. This further makes the policy more transparent and interpretable because decisions can be justified in a logical and hence human-understandable way, as opposed to purely neural agents whose policies are opaque. Moreover, the agent can be better controlled and safeguarded because humans can induce bias or restrict its behavior by interacting with its logical rules.

All these potential advantages drive interest in building neuro-symbolic agents, but it remains an open research question how abstract concepts can be grounded to specific objects in the environment. As an example, recall the agent that is instructed to move right when being left of a ladder. In order to satisfy this rule, the agent must learn a

function that maps the concepts ”is ladder” and ”left of” to the respective objects in the scene, *i.e.*, to all ladders and all objects right of the agent.

This challenge has been addressed in supervised domains like Visual Question Answering (Mao et al., 2019) and robotic manipulation (Hsu et al., 2023; Mao et al., 2025). For RL tasks, however, concept grounding remains underexplored and is commonly circumvented by implementing suitable grounding functions manually (Jiang et al., 2019; Vouros, 2022; Shindo et al., 2024). While this can be feasible in simple environments, it becomes impractical in more complex ones, especially for concepts that represent relations between multiple objects.

We therefore argue that neuro-symbolic agents must ultimately be able to ground relational concepts by interacting with their environment. To that end, our work contributes as follows[1]:

(1) We present a framework for learning neuro-symbolic agents that incorporate logical reasoning over relational concepts, whose grounding functions are learned through experience and weak supervision provided by large language models.

(2) We identify and investigate key challenges of learning relational concepts without strong supervision.

(3) We extensively evaluate our framework on two ATARI environments: Kangaroo and Seaquest.

This thesis is structured as follows: First, we will provide necessary background on reinforcement learning in general and, more specifically, on concept grounding for neuro-symbolic systems in Chapter 2. Then, we give an overview of related work in Chapter 3. Next, we present our framework in Chapter 4 and show experimental results thereafter in Chapter 5. We end the thesis by discussing and analyzing some of the key insights in Chapter 6 and give a conclusion in Chapter 7.

[1]We provide the full code for this work at: `https://github.com/ml-research/blendrl-dev/tree/valuation_synthesis`

2. Foundations

In the subsequent sections, we provide the foundations for training neuro-symbolic agents. First, we introduce common RL algorithms and methods used to learn policies from experience in Section 2.1. We then investigate how relational concepts can be incorporated by the agent to apply logical reasoning in Section 2.2.

2.1. Deep Reinforcement Learning

In Deep Reinforcement Learning, the goal is to learn a policy that, given the current state of an agent, returns the next action that maximizes the overall reward of the agent. The problem is modeled as a Markov decision process (MDP), $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where in each timestep t, the agent is in state $s_t \in \mathcal{S}$, takes action $a_t \in \mathcal{A}$ and transitions to the next state s_{t+1} with probability $P(s_{t+1} \mid s_t, a_t)$ while receiving a reward $r_t = R(s_t, a_t, s_{t+1})$. The objective of the agent is to learn a stochastic policy $\pi_\theta(a_t \mid s_t)$ with parameters θ that maximizes the discounted cumulative reward (also called *return*)

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \gamma^t r_t\right] \tag{2.1}$$

where $\gamma \in [0, 1]$ is the discount factor and $T \in \mathbb{N}_{>0}$ is the length of the episode. We will briefly explore common methods in RL to find such a policy in the following.

2.1.1. Policy-Gradient Methods

Policy-gradient methods aim to learn the policy directly by maximizing the expected return $J(\theta)$ through gradient ascent. As the optimal mapping from states to actions is unknown, policy-gradient methods instead define estimates $\Psi_t \in \mathbb{R}$ of how good a policy

was in each step of the observed episode. The general form of the policy gradient then becomes

$$\nabla_\theta J(\theta) = \mathbb{E}_{(s,a)\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\Psi_t\right] \tag{2.2}$$

Therefore, policy-gradient methods increase the likelihood of actions a_t that are positively scored according to Ψ_t.

The first such method was *REINFORCE* (Williams, 1992), which samples sequences of states and actions according to the current policy and then updates the policy parameters θ based on the collected rewards. As it does not explore all possible paths in the MDP but rather samples from it, REINFORCE can be considered a Monte-Carlo method. For each timestep t, it computes the remaining return, J_t (also called *Monte-Carlo return*), and uses it as the estimate Ψ_t, *i.e.*,

$$\Psi_t = J_t = \sum_{i=0}^{T-t} \gamma^i r_{t+i} \tag{2.3}$$

In practice, this approach is rather slow and unstable because the return can greatly vary between different episodes, leading to noisy gradients and hence erratic policy updates.

2.1.2. Value-Based Methods

Value-based methods, on the other hand, attempt to find an optimal policy π that maximizes the action-value function $Q^\pi(s,a)$ of the MDP $\mathcal{M}$. The action-value function computes the *quality* of action a in state s, *i.e.*, the future return that is expected when taking action a in state s and following the policy onward:

$$Q^\pi(s,a) = \sum_{s'\in\mathcal{S}} P(s' \mid s,a)\left[R(s,a,s') + \gamma V^\pi(s')\right] \tag{2.4}$$

Here, V^π denotes the value function and represents the overall expected return of the policy when starting in state s:

$$V^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a \mid s) Q^\pi(s,a) \tag{2.5}$$

Intuitively, a policy π^* is optimal if, for any state s, there does not exist another policy with a higher Q-value:

$$Q^{\pi^*}(s,a) = \max_{\pi} Q^{\pi}(s, \pi(s)) \tag{2.6}$$

For ease of notation, we henceforth define $Q^* \equiv Q^{\pi^*}$. Vice versa, if the maximum Q-values are known, the optimal policy π^* can be implicitly defined as

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} Q^*(s,a) \tag{2.7}$$

It is possible to retrieve Q^* for a MDP with bounded rewards and finite states and actions through dynamic programming (DP), however, it requires the reward function R and the transition probabilities P to be known in advance, making DP impracticable in an RL environment. In 1989, Watkins thus introduced *Q-Learning*, an algorithm that converges to the optimal Q^*, starting from arbitrarily initialized values $Q_0(s,a)$. With each step that the agent transitions from state s to s' by performing action a, the corresponding Q-values get updated according to

$$Q_n(s,a) \leftarrow (1-\alpha_n) \cdot Q_{n-1}(s,a) + \alpha_n \cdot \left[R(s,a,s') + \gamma \max_{a'} Q_{n-1}(s',a') \right] \tag{2.8}$$

where $\alpha_n \in [0,1)$ is the learning rate.

Although this iterative update scheme guarantees that $Q_n(s,a) \to Q^*(s,a)$ as $n \to \infty$, Q-Learning is intractable for most environments, because complexity increases with $|\mathcal{S}|$ and $|\mathcal{A}|$ and convergence only holds if the agent observes infinitely many episodes for each starting state and action (Watkins et al., 1992).

Temporal Difference (TD) learning is another value-based method for learning the value function V^π of a fixed policy π (Sutton, 1988). We note that TD learning itself cannot be used to improve a policy, but we still present it here as its concepts are used in state-of-the-art RL algorithms that we will introduce later. TD learning estimates the future value $G_t^{(n)}$ of a state s_t based on the total discounted rewards received in the next $n \in \mathbb{N}_{>0}$ steps and bootstraps the value for all remaining steps of the episode using the current estimate for the last state s_{t+n}:

$$G_t^{(n)} = \left(\sum_{i=0}^{n-1} \gamma^i r_{t+i} \right) + \gamma^n V^\pi(s_{t+n}) \tag{2.9}$$

It then updates V^π according to

$$V^\pi(s_t) \leftarrow V^\pi(s_t) + \alpha\delta_t^{(n)} \tag{2.10}$$

where $\alpha \in (0, 1]$ is the learning rate and $\delta_t^{(n)}$ is the so-called *n-step TD residual*

$$\delta_t^{(n)} = G_t^{(n)} - V^\pi(s_t) \tag{2.11}$$

Note how the hyperparameter n can control the trade-off between bias and variance: If $n = 1$, only immediate rewards are considered, leading to low variance but high bias. Vice versa, if $n = T$, the update step $G_t^{(T)}$ becomes equivalent to the Monte-Carlo return in REINFORCE (Eq. 2.3), which has high variance but low bias.

2.1.3. Advantage Actor-Critic

Both, policy-gradient and value-based methods, have practical limitations that make them inapplicable in many RL environments. To overcome these limitations, most RL tasks today are learned using so-called *actor-critic* (AC) methods, which combine both approaches. They consist of two components: an actor that is equivalent to the parametric policy π_θ used in policy-gradient methods, and a critic that approximates the value (Eq. 2.5) or action-value function (Eq. 2.4), respectively.

The purpose of the critic is to guide the actor by providing feedback about the actions that the actor takes. For instance, assume the critic properly approximates Q^{π_θ} by a differentiable function $Q_\phi(s, a)$ with parameters ϕ. The predicted Q-values can then be used as the policy gradient $\Psi_t = Q_\phi(s_t, a_t)$ from Eq. (2.2).

Advantage actor-critic (A2C) methods further reduce the variance of Ψ_t by learning an approximation $\hat{A}$ of the following advantage function:

$$A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s) \tag{2.12}$$

It quantifies how much better an action a is compared to sampling a random action according to the distribution $\pi_\theta(\cdot \mid s)$. However, this increases complexity, because it requires learning two estimators V_ϕ and Q_ϕ for both, the value and action-value function. In practice, just a single value estimator V_ϕ is used to approximate the advantage function instead.

One example of this is the n-step TD residual $\delta_t^{(n)}$ (Eq. 2.11) which already provides an unbiased estimation of the advantage (Mnih et al., 2016) and therefore is a common choice in current RL algorithms. While the choice of n can balance bias and variance to a certain extent, a fixed value for n is not necessarily equally appropriate for all episodes. For instance, a smaller n might be a better choice for episodes in which rewards occur early or more frequently. Vice versa, if rewards are greatly delayed, increasing n is more favorable.

To better adapt to the dynamics of different episodes and still balance bias and variance, multiple such estimates can be combined. This is the core idea of *generalized advantage estimation* (GAE) (Schulman et al., 2018), which effectively blends the TD residuals of all subsequent timesteps using an exponential decay factor $\lambda \in [0, 1]$, and is formally defined as

$$\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} = \sum_{i=0}^{T} (\gamma\lambda)^i \delta_{t+i}^{(1)} \tag{2.13}$$

Note that this estimate decomposes into the 1-step TD residual for $\lambda = 0$ and the T-step TD residual for $\lambda = 1$.

By choosing $\Psi_t = \hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)}$, the actor can be learned more efficiently compared to standalone policy-gradient methods like REINFORCE due to the reduced variance and bias. Albeit this approach might not converge to a global optimum as guaranteed by value-based methods like Q-Learning, it is tractable and converges to a policy that is locally optimal, as proven in (Sutton et al., 1999).

The critic V_ϕ is learned alongside the actor by minimizing the mean squared error (also called *value function loss*) between its predictions and target values V^{targ} using gradient descent:

$$L^{\mathrm{VF}}(\phi) = \mathbb{E}_{s\sim\pi_\theta}\left[\left(V_\phi(s_t) - V^{\mathrm{targ}}(s_t)\right)^2\right] \tag{2.14}$$

While any estimate for V^{π_θ} such as the Monte-Carlo return can be used as target, a natural choice is to reuse the advantage estimates computed by GAE (Eq. 2.13) and define V^{targ} as

$$V^{\mathrm{targ}}(s_t) = V_\phi(s_t) + \hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} \tag{2.15}$$

2.1.4. Proximal Policy Optimization

We observe that, in A2C, the probability of an action a_t is pushed toward the direction of its estimated advantage $\hat{A}_t$, *i.e.*, $\pi_\theta(a_t \mid s_t)$ approaches 0 if $\hat{A}_t < 0$ and 1 if $\hat{A}_t > 0$. Especially for large $\hat{A}_t$, the policy might get updated too forcefully, which can cause a low action entropy and hence the actor to get stuck in a suboptimal deterministic policy. Another limitation is that the gradient estimate $\nabla_\theta J(\theta)$ must be computed using the current parameters θ. As a consequence, vanilla A2C is rather sample inefficient because after performing one update step on a batch of trajectories, a new batch of trajectories must be sampled under the updated policy.

To solve these limitations, *Trust Region Policy Optimization* (TRPO) (Schulman et al., 2015) implements two key improvements: First, TRPO limits the Kullback-Leibler (KL) divergence between the action distribution of the policy before and after updating its parameters, denoted $D_{\mathrm{KL}}\left(\pi_{\theta_{\text{old}}}(\cdot \mid s_t) \parallel \pi_\theta(\cdot \mid s_t)\right)$. This prevents the policy from diverging too much from the old one. Second, it reuses the sampled trajectories of the old policy and their advantage estimates to allow π_θ to be updated relative to $\pi_{\theta_{\text{old}}}$ in multiple optimization steps. More precisely, the proposed optimization problem is

$$\begin{aligned} \max_\theta \quad & \mathbb{E}_{(s,a)\sim\pi_{\theta_{\text{old}}}}\left[r_t(\theta)\hat{A}_t\right] \\ \text{s. t.} \quad & \mathbb{E}_{(s,a)\sim\pi_{\theta_{\text{old}}}}\left[D_{\mathrm{KL}}\left(\pi_{\theta_{\text{old}}}(\cdot \mid s_t) \parallel \pi_\theta(\cdot \mid s_t)\right)\right] \leq \delta \end{aligned} \tag{2.16}$$

where $\delta \in \mathbb{R}_{>0}$ is the maximum allowed KL divergence and $r_t(\theta)$ is the probability ratio between the current and old policy

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \tag{2.17}$$

For constrained optimization problems like this, more sophisticated methods such as the conjugate gradient algorithm need to be applied, making training significantly more complex compared to standard policy-gradient methods.

One of the most frequently used RL algorithms today, *Proximal Policy Optimization* (PPO) (Schulman et al., 2017), addresses this issue of TRPO. Rather than enforcing a hard constraint on the KL divergence, PPO effectively limits how much the probability ratio $r_t(\theta)$ can deviate from 1, which has a similar effect as in TRPO but can be implemented much more efficiently using standard gradient ascent methods. The objective of PPO is defined as

$$L^{\mathrm{CLIP}}(\theta) = \mathbb{E}_{(s,a)\sim\pi_{\theta_{\text{old}}}}\left[\min\left(r_t(\theta)\hat{A}_t, \operatorname{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right] \tag{2.18}$$

where $\epsilon \in (0, 1)$ is the *clipping coefficient*. Note that $L^{\text{CLIP}}(\theta)$ becomes constant for $r_t(\theta) > 1 + \epsilon$ if $\hat{A}_t > 0$ and $r_t(\theta) < 1 - \epsilon$ if $\hat{A}_t < 0$. In other words, once $r_t(\theta)$ is pushed out of the interval $[1 - \epsilon, 1 + \epsilon]$ in an attempt to keep maximizing the unclipped TRPO objective $r_t(\theta)\hat{A}_t$, PPO suppresses any policy updates that would make the probability ratio deviate even further.

We can now assemble the final objective that jointly learns the critic and actor by minimizing

$$L^{\text{PPO}}(\theta, \phi) = \mathbb{E}\left[c_{\text{VF}} \cdot L^{\text{VF}}(\phi) - L^{\text{CLIP}}(\theta) - c_{\text{AE}} \cdot H(\pi_\theta)\right] \tag{2.19}$$

where $c_{\text{VF}}, c_{\text{AE}} \in \mathbb{R}_{\geq 0}$ and

$$H(\pi_\theta) = \mathbb{E}_{s \sim \pi_\theta}\left[-\sum_{a \in \mathcal{A}} \pi_\theta(a \mid s_t) \log \pi_\theta(a \mid s_t)\right] \tag{2.20}$$

is an optional action entropy term that can foster exploration (Haarnoja et al., 2017).

2.2. Neuro-Symbolic Reinforcement Learning

While deep neural networks such as Convolutional Neural Networks (CNN) (LeCun et al., 2015) have been successfully applied in various RL tasks (Mnih et al., 2015), they lack general reasoning capabilities that are crucial for performing novel tasks in previously unseen environments. In recent years, this limitation has intensified research into neuro-symbolic agents that can understand their environment on an abstract level and then deduce their next actions based on logical rules. This requires strong visual perception capabilities of deep neural networks as well as symbolic reasoning. However, bridging the gap between both systems involves several challenges that we will detail in the following.

2.2.1. First-Order Logic

Neuro-symbolic agents use formal systems like *First-Order Logic* (FOL) to encode their world knowledge and actions in a logical and structured manner. A function-free language in FOL, $\mathcal{L} = (\mathcal{P}, \mathcal{D}, \mathcal{V})$, is comprised of a set of predicate symbols $\mathcal{P}$, constants $\mathcal{D}$ and variables $\mathcal{V}$. In an FOL language, a *term* is either a constant or a variable. An *atom* $\mathtt{p}(\mathtt{t_1}, \ldots, \mathtt{t_n})$ is the smallest unit in a logical statement, where $\mathtt{t_1}, \ldots, \mathtt{t_n}$ are terms and $\mathtt{p}$

is a predicate of arity $\alpha(\mathtt{p}) = n$. If an atom has only constant terms, it is called *ground atom* and has a *truth value* which is either true or false. A *substitution* $\sigma = \{\mathtt{X_1} \mapsto \mathtt{t_1}, \ldots, \mathtt{X_n} \mapsto \mathtt{t_n}\}$ maps variables $\mathtt{X_i}$ to other terms $\mathtt{t_i}$ and can be applied to an atom A to construct a new atom $A\sigma$ with its variables replaced accordingly. A *Horn clause* (or *rule*) is a finite disjunction of atoms, where all atoms except exactly one are negated, *e.g.*, $A \vee \neg B_1 \vee \cdots \vee \neg B_n$. This is equivalent to the logical statement $A \Leftarrow B_1 \wedge \cdots \wedge B_n$, or in words: if all atoms $B_1, \ldots, B_n$ are true, then A must hold as well. We write Horn clauses in the form A :- $B_1, \ldots, B_n$, where A is called the *head* and the set $\{B_1, \ldots, B_n\}$ is called the *body* of the clause. A *ground clause* (or *ground rule*) is a Horn clause with no variables. We use the term *extensional predicate* to refer to predicates that only occur in clause bodies, but never in a clause head.

We can now express an action of the neuro-symbolic agent as a rule, where the head atom is associated with an action (*e.g.*, `go_right`, `go_left`, *etc.*) and the body atoms represent all preconditions that must hold for that action to be executed. For example, the rule

$$\mathtt{go\_right}(\mathtt{player}) \text{ :- } \mathtt{left\_of}(\mathtt{player}, \mathtt{X}), \mathtt{is\_ladder}(\mathtt{X})$$

can be read as "the `player` must go right if there is any ladder `X` left of it", where `player` is a constant and `X` is a variable. For better comprehensibility, we will henceforth refer to a clause that explicitly models an action as *action rule* and call the atoms in its head and body *action atom* and *state atoms*, respectively. The set of all action rules forms the *logic program* of the agent.

A *knowledge base* consists of a set of true ground atoms (also called *facts*) and rules. The process of recursively applying the rules to the facts to infer new facts is known as *logic inference* or *symbolic reasoning*. It allows the neuro-symbolic agent to deduce which action to take next according to its logic program. More precisely, an action is selected if there exists a substitution σ under which a corresponding action atom can be inferred. Assume, for example, a knowledge base with facts

$$\{\mathtt{is\_ladder}(\mathtt{ladder_1}), \mathtt{is\_ladder}(\mathtt{ladder_2}), \mathtt{left\_of}(\mathtt{player}, \mathtt{ladder_2})\}$$

and the action rule from the previous example. The action atom $\mathtt{go\_right}(\mathtt{player})$ can be inferred from this knowledge base by applying the substitution $\sigma = \{\mathtt{X} \mapsto \mathtt{ladder_2}\}$ to all state atoms, hence the agent will go right.

Incorporating logic inference into agents is powerful because it enables them to adapt their actions appropriately to different facts. In order to apply symbolic reasoning, however, the agent must first translate raw sensory data to a faithful knowledge base. The main

challenge here is to train a neural system that can extract logical facts out of visual scenes without supervision, but rather from experience only. That is, the reward signal must be propagated back to the neural system to allow for an end-to-end training. With classical, boolean logic, this is not possible, because each rule is expressed as a chain of discontinuous (hence non-differentiable) logical connectives like AND ($\wedge$), OR ($\vee$), *etc.* In consequence, the logic policy becomes deterministic and discontinuous itself, which breaks the gradient flow. Therefore, neuro-symbolic agents use fuzzy logic instead, where these connectives are replaced by differentiable operations and ground atoms have continuous truth values in $[0, 1]$. We provide further details on that end in sections 2.2.3 and 2.2.4. Henceforward, we will refer to a fact as a ground atom in fuzzy logic with a real-valued probability and assume that all rules in the knowledge base are already given.

In the following, we investigate how to learn a model that can retrieve all facts from sub-symbolic, *e.g.*, pixel-based input, in more detail.

2.2.2. Object-Centric Representation

The first step is to represent the agent's environment as a discrete set of constants, as required by FOL. These constants typically represent disentangled objects with arbitrary attributes such as their class, position, color and shape. Let $f_\Phi^{\mathrm{OC}} : \mathbb{R}^N \to \mathbb{R}^{E \times D}$ with parameters Φ be the *perception function* that extracts E objects with D features each from an image $\mathbf{X} \in \mathbb{R}^N$ of size N. We then call $\mathbf{Z} = f_\Phi^{\mathrm{OC}}(\mathbf{X})$ the *object-centric (OC) representation* of the input image, and $\mathbf{z}_i \in \mathbb{R}^D$ the feature vector of object $i \in [E]$. Note that $\mathbf{z}_i$ can also be a latent embedding.

Decomposing an image into objects and their features can be done supervised or unsupervised. In a supervised setting, object detection models like *Faster R-CNN* (Ren et al., 2015), *Mask R-CNN* (He et al., 2017) or *YOLO* (Redmon et al., 2016) are commonly deployed. Alternatively, pretrained region proposal networks (as found in Faster R-CNN and Mask R-CNN) can be used to generate image patches of each object which are then fed to another feature extractor network (Yi et al., 2018; Mao et al., 2019). Object-centric learning methods like *IODINE* (Greff et al., 2019) or *Slot Attention* (Locatello et al., 2020), on the other hand, use an encoder-decoder approach to learn latent features for a fixed number of object slots in an unsupervised fashion.

2.2.3. Concept Grounding

Given all object constants and the predefined rule set, which implicitly defines all predicates and variables, the first-order language $\mathcal{L}$ is complete. The final step in connecting neural perception with symbolic reasoning is to determine the probabilities of all possible ground atoms under $\mathcal{L}$. Recall that a ground atom expresses some relation or *concept*, represented by a predicate, over a set of objects. In order to faithfully compute their truth values, the agent must therefore be able to understand the concept that each predicate embodies. This process is also referred to as *concept grounding* and remains a key challenge in the field of neuro-symbolic RL.

Let $v_\psi^{\mathtt{p}} : \mathbb{R}^{\alpha(\mathtt{p}) \times D} \to [0, 1]$ with parameters ψ be the *valuation function* of predicate $\mathtt{p}$ that maps object-centric features to the probability of the corresponding ground atom. While some frameworks simply use hand-crafted or LLM-generated valuation functions (Yi et al., 2018; Vouros, 2022; Shindo et al., 2024; Sun et al., 2025), any differentiable function can be learned instead. Viable choices include a simple multilayer perceptron (MLP), a CNN that processes image patches directly instead of object-centric features (Manhaeve et al., 2018) and even a Gaussian distribution (Koudouna et al., 2021). Another approach is to use expert knowledge to design tailored functions with typically few parameters. As an example, the closeness of two objects could be explicitly modeled as a linear transformation over their distance (Shindo et al., 2021).

Other neural architectures have been employed specifically in neuro-symbolic systems that do not rely on expert knowledge. *Logic Tensor Networks* (Serafini et al., 2016), for instance, apply symbolic reasoning on top of relations learned by a generalized version of Neural Tensor Networks (Socher et al., 2013). They capture relations of arbitrary arity using a combination of multiple linear and bilinear transformations. Another framework, the *Neuro-Symbolic Concept Learner* (Mao et al., 2019), associates each concept (*e.g.*, `Left`, `Right`, `Above`, *etc.*) with a vector embedding. A neural function is then learned to generate embeddings that align with those of the corresponding concepts given object-centric features. Finally, the truth values are computed based on the similarity between the generated embeddings and each individual concept embedding. Depending on the complexity of the predicates that need to be grounded, some neural architectures might be more suited than others.

2.2.4. Differentiable Reasoning

Regardless of which neural valuation function is chosen, the difficulty in learning their parameters is that we do not have access to supervised examples for each concept. Instead, they need to be learned indirectly from the reward signal and the symbolic policy $\pi_\theta(\cdot \mid s)$ they produce. The neuro-symbolic agent must therefore be able to calculate action probabilities based on its logic program and a given set of probabilistic facts in a differentiable manner.

Before we describe some viable methods, let us first introduce some auxiliary notations. We use the shorthand notation $v(G) = v_{\mathtt{p}}(\mathbf{z}_1, \ldots, \mathbf{z}_{\alpha(\mathtt{p})})$ for the probability of a ground atom $G = \mathtt{p}(\mathtt{d}_1, \ldots, \mathtt{d}_{\alpha(\mathtt{p})})$. Further, let $\mathcal{R}$ be the set of all rules, where $\mathrm{head}(R)$ is the head and $\mathrm{body}(R)$ is the body of a rule $R \in \mathcal{R}$. Finally, we denote $\mathcal{G}(R)$ the set of all ground rules that can be produced through variable substitution on R.

In classical logic, the head of some rule R is equal to

$$\mathrm{head}(R) = \bigvee_{R' \in \mathcal{G}(R)} \bigwedge_{G \in \mathrm{body}(R')} G \tag{2.21}$$

where each ground atom represents a logic value (true or false) and disjunction ($\vee$) and conjunction ($\wedge$) are logical connectives that again return a boolean value. In fuzzy logic with truth values in $[0, 1]$, those connectives are replaced by fuzzy operators.

A fuzzy conjunction is commonly implemented as a so-called *t-norm* function, T, which is commutative, associative, monotonic and has 1 as neutral element (Esteva et al., 2001). The complementary function for fuzzy disjunction, called *t-conorm* or *s-norm*, is defined as $S(x_1, x_2) = 1 - T(1 - x_1, 1 - x_2)$. This dual definition ensures that both, t-norm and its t-conorm, are consistent with fuzzy negation $v(\neg G) = 1 - v(G)$ under De Morgan's laws, where $\neg(A \wedge B) \equiv \neg A \vee \neg B$.

Many such t-norms and t-conorms exist, the most notable are:

Gödel:	$T(x_1, x_2) = \min(x_1, x_2)$	$S(x_1, x_2) = \max(x_1, x_2)$
Product:	$T(x_1, x_2) = x_1 \cdot x_2$	$S(x_1, x_2) = x_1 + x_2 - x_1 \cdot x_2$
Łukasiewicz:	$T(x_1, x_2) = \max(0, x_1 + x_2 - 1)$	$S(x_1, x_2) = \min(x_1 + x_2, 1)$

Note that these functions have different characteristics with regard to gradient flow and accuracy. The Gödel functions mimic boolean operators best, but only provide gradient signals for one atom. Similarly, the Łukasiewicz t-norm has zero gradients for either atom

if $x_1 + x_2 < 1$. Other frameworks explicitly use the Product t-norm and t-conorm (also called *probabilistic sum*) because they distribute gradients to both atoms (Evans et al., 2018). However, the probabilistic sum can be inaccurate compared to the more faithful maximum t-conorm by Gödel. For example, if $x_1 = x_2 = 0.5$, it greatly overestimates the maximum ($0.75 > 0.5$). The *Neuro-Symbolic Forward Reasoner* (NSFR) (Shindo et al., 2021) hence uses log-sum-exp (LSE), a smooth approximation of the maximum

$$\mathrm{LSE}(x_1, \ldots, x_n) = \frac{\gamma}{S} \log \left(\sum_{i=1}^{n} \exp(x_i/\gamma) \right) \tag{2.22}$$

where $\gamma \in \mathbb{R}_{>0}$ is a smoothing parameter and

$$S = \max \left(1.0, \gamma \log \left(\sum_{i=1}^{n} \exp(x_i/\gamma) \right) \right) \tag{2.23}$$

is a normalization term. Note that a better approximation of the maximum function with smaller γ comes at the cost of reducing gradient information for the non-maximum values.

Now, each rule head can be grounded according to Eq. (2.21), where ground atoms G are substituted by their valuations $v(G)$ and the logical connectives are replaced by any appropriate fuzzy operator. This is done in multiple steps until no more facts can be deduced or for a fixed number of steps (Evans et al., 2018; Shindo et al., 2021).

Further, each rule can be assigned a weight $w_R \in \mathbb{R}$. This can be particularly useful for logic programs that contain multiple action rules for the same action, as it allows the agent to learn the importance of each rule individually (Shindo et al., 2024).

Taken all together, the action probabilities can be computed by taking the softmax over the truth values of all action atoms. Let $\mathcal{R}(a) \subset \mathcal{R}$ be the set of action rules associated to action $a \in \mathcal{A}$. We further denote $v(\mathrm{head}(R))$ the fuzzy truth value of a rule head, or $v(R)$ for short. The final symbolic policy then becomes

$$\pi_\theta(a \mid s) = \frac{\sum_{R \in \mathcal{R}(a)} \exp\left(w_R \cdot v(R)\right)}{\sum_{a \in \mathcal{A}} \sum_{R \in \mathcal{R}(a)} \exp\left(w_R \cdot v(R)\right)} \tag{2.24}$$

3. Related Work

Building neuro-symbolic agents involves two fundamental challenges: (1) finding (action) rules which optimize the policy of the agent and (2) learning a model which retrieves all facts from sub-symbolic input. Many frameworks address the former challenge by applying *Differentiable Inductive Logic Programming* (δILP) (Evans et al., 2018), provided that the agent has access to ground-truth symbolic information (Jiang et al., 2019; Vouros, 2022; Shindo et al., 2024). Our work, however, examines the opposite case, in which we assume the rules (including the logic program) to be given and instead are interested in finding the corresponding valuation functions. While solutions to that have been studied in knowledge-based tasks (Serafini et al., 2016), Visual Question Answering (VQA) tasks (Mao et al., 2019; Mao et al., 2025) as well as robotic manipulation (Hsu et al., 2023; Mao et al., 2025), concept grounding is still underexplored in the field of RL (Acharya et al., 2023).

INSIGHT (Luo et al., 2024) is a neuro-symbolic RL framework in which the policy is modeled as a function over object coordinates. Learning spatial relations like this is akin to our approach, however, we do not map them directly to action probabilities but first compile them to predicates in a FOL language instead. INSIGHT does not incorporate explicit symbolic reasoning, yet its policy is made interpretable by parsing the learned functions into a textual explanation using GPT-4 (Achiam et al., 2023). Further, INSIGHT does not learn the policy directly from the reward signal but rather through knowledge distillation of a neural actor. Other frameworks have used neural guidance to learn symbolic policies in a similar way (Delfosse et al., 2023).

BlendRL (Shindo et al., 2024) also explores the idea of incorporating a large language model (LLM) like GPT-4, but not for explaining, rather for directly generating valuation functions. While recent LLMs are capable of rendering general spatial concepts such as "left of" or "close by", grounding those concepts to each individual environment is still challenging and can require manual refinement by human experts.

Our approach combines and extends on the core ideas of the above-mentioned frameworks: As in INSIGHT, we represent spatial relations by neural valuation functions instead of LLM-generated functions, as proposed in BlendRL. However, we still incorporate LLMs to support concept grounding rather than providing guidance through a neural actor.

4. Methodology

The core of our framework is BlendRL (Shindo et al., 2024), a neuro-symbolic RL framework that uses hand-crafted valuation functions to ground relational concepts. In Section 4.1, we elaborate why this is a fundamental limitation and detail in Section 4.2 how we modified BlendRL to overcome this limitation.

4.1. Problem Statement

Motivated by Kahneman's studies, according to which humans internalize two systems of thinking (Kahneman, 2011), BlendRL employs both a neural and logic policy. The *neural policy* is representative for *System 1*, which operates quickly and intuitively, whereas the *logic policy* resembles the foresightful and logically thinking *System 2*. For example, consider the ATARI environments Kangaroo and Seaquest depicted in Figure 4.1. One might associate System 1 with handling situations that require immediate reactions like punching an enemy or dodging a missile. Surviving in such situations can demand highly instinctive actions that are difficult to express in terms of logical rules. On the other hand, System 2 should take over when long-term planning is required to achieve a greater goal, such as reaching the child in Kangaroo or rescuing divers in Seaquest. A *blending module* finally controls whether the agent is more inclined toward the neural or logic policy.

BlendRL utilizes relational concepts in two components: the logic actor and the blending module. Both operate according to distinct logic programs, in which predicates represent abstract concepts such as spatial relations between multiple objects. In order for the logic actor to know, for example, in which particular direction it must move to approach a certain object of interest, it must know how it is positioned in relation to that object (*e.g.*, `left_of`, `right_of`, `above`, *etc.*). The blending module, on the other hand, must be able to switch to the neural policy if an enemy or missile is in critical proximity to the player.

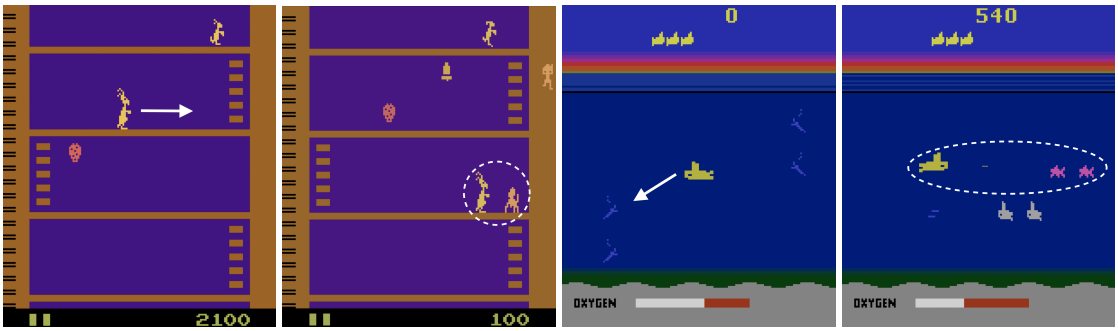

Figure 4.1.: **ATARI environments Kangaroo and Seaquest.** In Kangaroo (left), the player uses logic thinking to reach its joey while defending itself against monkeys along the way, which requires fast reactions. Likewise in Seaquest (right), the player navigates rationally through the environment to collect divers, but must quickly dodge or shoot approaching enemies.

In BlendRL, the valuation functions for these predicates are not learned, but rather hand-crafted by human experts. This is a major limitation of the current framework for multiple reasons:

(1) **The optimal valuation function is unknown or difficult to obtain**, *e.g.*, because it requires a deep understanding of the environment's physics and dimensions. Think of a predicate `close_by_missile` in Seaquest, which might be used by the blending module to evaluate if the player is going to be hit by a missile soon and should therefore favor the neural policy. To design a good valuation function, the interplay of several factors must be considered, such as the trajectory, velocity and size of the missile, as well as the defense mechanisms of the player.

(2) In consequence, **valuation functions must be adapted to the different characteristics of each environment individually.** A missile in Seaquest, for instance, has other dynamics than a missile (coconut thrown by a monkey) in Kangaroo. Although `close_by_missile` serves the same purpose in both environments, which is to dodge an object that can kill the player, their corresponding valuation functions must reflect these differences. BlendRL can thus not be extended to other environments without great effort.

(3) **Precisely implementing valuation functions by hand is time-consuming**, even if humans have a precise notion of the concepts. For example, it is clear that the player in Kangaroo must overlap with a ladder to be able to climb it. Without access to the

technical implementation details of the game, however, the exact region in which the player is recognized to be on the ladder is unknown and can only be retrieved by trial and error.

4.2. Framework

Our work aims to resolve these limitations by learning the valuation functions through experience instead. In the subsequent sections, we give an overview of our framework and explain how it extends BlendRL to achieve this.

4.2.1. Overview

The overall architecture of BlendRL consists of three components:

(1) A *neural policy* $\pi_\theta^{\text{neural}} : \mathbb{R}^{W\times H\times C} \to [0,1]^{|\mathcal{A}|}$ that maps raw images $\mathbf{X} \in \mathbb{R}^{W\times H\times C}$ to a probability distribution over all actions in $\mathcal{A}$.

(2) A *logic policy* $\pi_\phi^{\text{logic}} : \mathbb{R}^{E\times D} \to [0,1]^{|\mathcal{A}|}$ that uses a predefined logic program to infer its actions. It is implemented as an NSFR, that reasons over object-centric states $\mathbf{Z} \in \mathbb{R}^{E\times D}$ comprised of E objects with D features each, such as the object's position, orientation, value, *etc.*

(3) A *blending module* $B_\lambda : \mathbb{R}^{E\times D} \to [0,1]$ that combines the action distributions of both policies. It uses another NSFR with a separate logic program to decide when to use which policy based on the symbolic state $\mathbf{Z}$.

The final action distribution is computed as

$$\pi_{(\theta,\phi,\lambda)}(\mathbf{X},\mathbf{Z}) = \beta \cdot \pi_\theta^{\text{neural}}(\mathbf{X}) + (1-\beta)\cdot \pi_\phi^{\text{logic}}(\mathbf{Z}) \tag{4.1}$$

where $\beta = B_\lambda(\mathbf{Z}) \in [0,1]$ is the *blending weight*. All components of the hybrid agent are trained jointly using the PPO algorithm introduced in Section 2.1.4. In order to estimate the value function, sub-symbolic and symbolic states are processed separately by a *neural critic* $V_\mu^{\text{neural}} : \mathbb{R}^{W\times H\times C} \to \mathbb{R}$ and a *logic critic* $V_\omega^{\text{logic}} : \mathbb{R}^{E\times D} \to \mathbb{R}$. Finally, both estimates are blended into a single value function:

$$V_{(\mu,\omega,\lambda)}(\mathbf{X},\mathbf{Z}) = \beta \cdot V_\mu^{\text{neural}}(\mathbf{X}) + (1-\beta)\cdot V_\omega^{\text{logic}}(\mathbf{Z}) \tag{4.2}$$

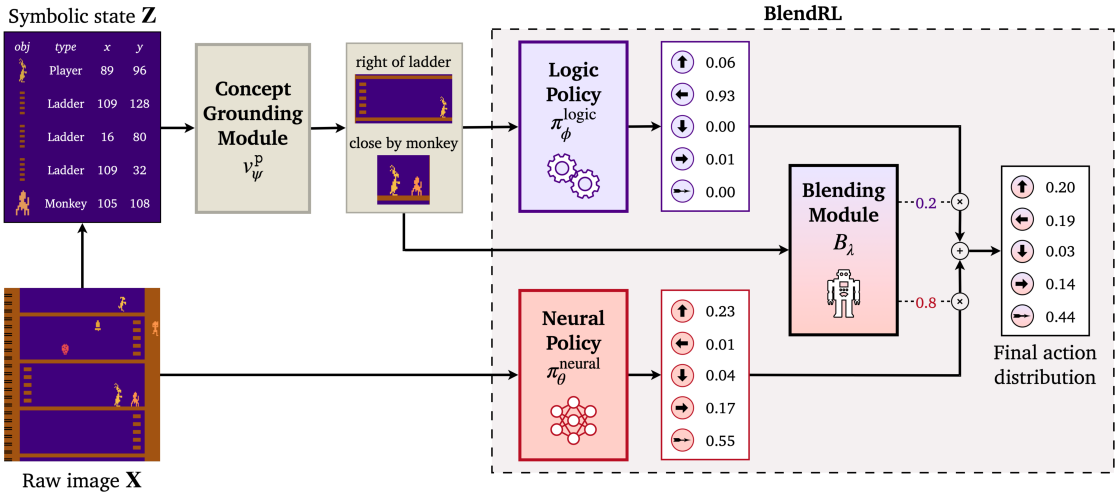

Figure 4.2.: **Overview of our framework.** A *concept grounding module* takes object-centric features $\mathbf{Z}$ extracted from an image $\mathbf{X}$ and computes object relations via differentiable valuation functions v_ψ^{p}. Those are then applied to a set of logical rules through forward reasoning to determine a *logic policy*. Likewise, a *blending module* utilizes the relational information to combine the logic policy with a *neural policy* that operates on the sub-symbolic state. All components can be trained jointly.

In addition to optimizing the PPO objective from Eq. (2.19), which includes loss terms for the value function, the policy ratio and the entropy of the action distribution, BlendRL adds another regularization term for the blending module:

$$H(B_\lambda) = -\beta \cdot \log \beta - (1-\beta) \cdot \log(1-\beta) \tag{4.3}$$

We call this the *blender entropy*, as it computes the entropy over the neural and logic blending weights β and $1-\beta$. It is imposed to ensure that the agent uses both policies and does not completely rely on a single policy. The final loss of BlendRL becomes $L^{\mathrm{BlendRL}}(\theta, \phi, \lambda, \mu, \omega) =$

$$\mathbb{E}\left[c_{\mathrm{VF}} \cdot L^{\mathrm{VF}}(\mu, \omega, \lambda) - L^{\mathrm{CLIP}}(\theta, \phi, \lambda) - c_{\mathrm{AE}} \cdot H(\pi_{(\theta,\phi,\lambda)}) - c_{\mathrm{BE}} \cdot H(B_\lambda)\right] \tag{4.4}$$

where we refer to $c_{\mathrm{BE}} \in \mathbb{R}_{\geq 0}$ as the *blender entropy coefficient*. All other terms and hyperparameters are analogous to Eq. (2.19), given the hybrid policy $\pi_{(\theta,\phi,\lambda)}$ and the hybrid value function $V_{(\mu,\omega,\lambda)}$.

As mentioned, a key limitation of BlendRL is that it uses hand-coded valuation functions to compute the truth values of all ground atoms. While some predicates are straight-forward to implement by hand (*e.g.*, counting objects or comparing a value against a threshold), grounding spatial relationships between multiple objects can be particularly challenging for the reasons outlined earlier. We hence propose to separate the valuation of spatial relations from the BlendRL framework and extend it by a *concept grounding module* which uses learnable functions instead. The resulting architecture of our framework is depicted in Figure 4.2.

This change alone may not be effective, as we analyze extensively in Section 6.1. In short, the spatial relations are ambiguous without stronger supervision, which can cause the agent to get stuck in a suboptimal local optimum because it relates the wrong objects to one another. To mitigate this effect, we introduce another component, the *concept aligner*, which utilizes an LLM to regularize the learned valuation functions.

We will now discuss both extensions in more detail.

4.2.2. The Concept Grounding Module

Recall that the logic actor and the blending module both operate logically and hence require faithful facts to reason on. To retrieve these facts, each extensional predicate must be grounded by appropriate valuation functions. Take, for instance, the logic programs of the ATARI games Kangaroo and Seaquest, shown in Figure 4.3. In order to decide if the player should go right in Kangaroo, the predicate `left_of_ladder` must be evaluated between the player and each ladder individually. The higher the truth value computed by its corresponding valuation function is for any ladder, the more likely it is that the agent goes right. For the blending program, this is analogous.

Notice how some of the extensional predicates query precise information about the environment, such as if an object of a specific type is present in the scene (`visible_diver`) or if the player has collected a certain amount of objects (`full_divers`). Given object-centric features, such information can be obtained exactly and with virtually no effort. We will therefore focus our attention on what we call *spatial predicates*, which represent spatial relations between two or more objects and can be much more difficult to model for the reasons provided in Section 4.1. For example, it might be plausible to activate `close_by_missile` when the distance between the player and a missile is below a certain threshold, but we may ask: *Which distance threshold is optimal? Is the distance equally important in both axes? And is distance even the best measure to capture that predicate?*

Kangaroo:

```
# Policy program
up_ladder()    :- on_ladder(player, Ladder)
left_ladder()  :- right_of_ladder(player, Ladder)
right_ladder() :- left_of_ladder(player, Ladder)

# Blending program
neural_agent() :- close_by_monkey(player, Monkey)
neural_agent() :- close_by_throwncoconut(player, ThrownCoconut)
logic_agent()  :- nothing_around(Z)
```

Seaquest:

```
# Policy program
up_rescue()    :- full_divers(Z) # all 6 divers collected
up_air()       :- oxygen_low(OxygenBar)
up_diver()     :- deeper_than_diver(player, Diver)
down_diver()   :- higher_than_diver(player, Diver)
left_diver()   :- right_of_diver(player, Diver)
right_diver()  :- left_of_diver(player, Diver)

# Blending program
neural_agent() :- close_by_enemy(player, Enemy)
neural_agent() :- close_by_missile(player, Missile)
logic_agent()  :- visible_diver(Diver)
logic_agent()  :- full_divers(Z)
logic_agent()  :- oxygen_low(OxygenBar)
```

Figure 4.3.: Logic programs employed by the logic actor (*policy program*) and blending module (*blending program*) in two ATARI environments. The constant `player` represents the only object the agent can control. The special variable `Z` entails the whole symbolic state $\mathbf{Z}$. Other uppercase variables are associated with objects that have the same type as the name of the variable, *e.g.*, `Ladder` can be any ladder. Spatial predicates that are learned by our framework are shown in blue and bold typeface.

These considerations can require a lot of manual engineering and trial and error to build good valuation functions. Although LLMs can technically be utilized as well, the generated functions are still not exact enough to solve the tasks reliably, as we show in Section 5.2.1. We hence argue that the valuation functions need to be learned by experience instead.

To do so, we replace the hand-coded valuation functions for spatial predicates in BlendRL with differentiable functions $f_\psi^{\mathtt{p}}$. Note that they can be jointly trained because both, the logic actor and the blending module, are implemented as an NSFR and hence support differentiable logic inference. Given an n-ary spatial predicate $\mathtt{p}$ with $n \geq 2$ and the positions $x_i \in [W], y_i \in [H]$ of each object, where $W, H \in \mathbb{N}$ are the width and height of the scene, we compute truth values as follows:

$$v_\psi^{\mathtt{p}}\left(\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \cdots, \begin{bmatrix} x_n \\ y_n \end{bmatrix}\right) = f_\psi^{\mathtt{p}}\left(\Big\|_{j=2}^{\alpha(\mathtt{p})} \begin{bmatrix} \frac{x_1 - x_j}{W} & \frac{y_1 - y_j}{H} \end{bmatrix}^T\right) \tag{4.5}$$

We want to emphasize three aspects of this function: (1) We assume that spatial relations are invariant to the position of the objects. That is, shifting the positions of each object by the same vector does not change their relation. We therefore use relative positions $x_1 - x_j, y_1 - y_j$ instead of absolute ones. (2) We normalize the x- and y-positions by dividing them by the width and height of the scene, respectively. Each offset vector then has values in $[-1, 1]^2$, which makes them invariant to the actual scene size. (3) Even though the logic programs listed in Figure 4.3 do not contain spatial predicates with an arity greater than 2, we still provide a generalized form that can model relations between any number of objects. In our case, all spatial predicates are binary, the valuation functions hence simplify to

$$v_\psi^{\mathtt{p}}\left(\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}\right) = f_\psi^{\mathtt{p}}\left(\Delta x, \Delta y\right) \tag{4.6}$$

where $\Delta x = \dfrac{x_1 - x_2}{W}$ and $\Delta y = \dfrac{y_1 - y_2}{H}$.

For $f_\psi^{\mathtt{p}}$, we use small MLPs in our experiments, but any other neural network such as the ones presented in Section 2.2.3 can be used as well.

4.2.3. The Concept Aligner

Learning those functions is challenging, particularly in symbolic RL environments. To understand why, we must consider how the loss gets propagated through the whole system

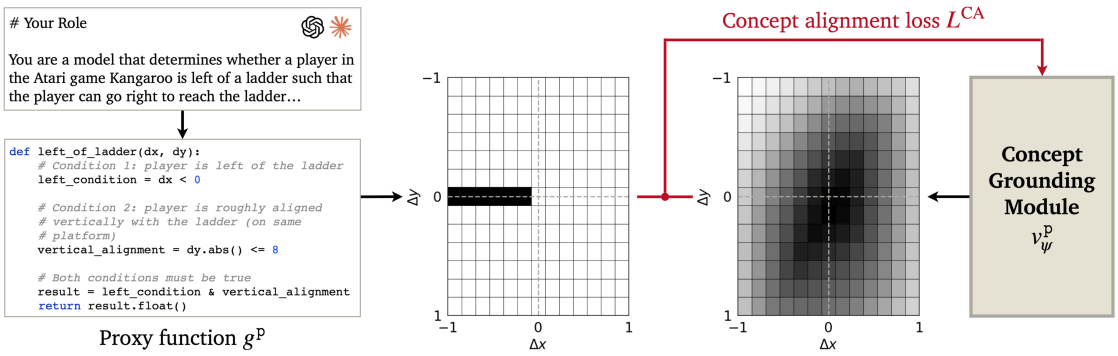

Figure 4.4.: **The Concept Aligner.** We instruct LLMs to generate proxy functions g^{p} for each extensional predicate. The proxy functions and the learned valuation functions v_ψ^{p} are then evaluated on a set of offset vectors $(\Delta x, \Delta y)$. Their outputs are compared against each other via a binary cross entropy loss L^{CA} that enforces the valuation functions to align with their proxy functions.

back to the valuation functions. First, the critic provides an advantage estimate to the symbolic policy that indicates how good or bad the selected action was: if the advantage was positive, the action probability shall increase. Now, recall that each action is represented as a rule in a logic program, and that its truth value is computed as a fuzzy disjunction over all possible substitutions of that rule. For example, if there are three ladders in the game Kangaroo, the action rule $\texttt{right\_ladder}() \texttt{ :- } \texttt{left\_of\_ladder}(\texttt{player}, \texttt{Ladder})$ has to be evaluated for all three ladders. This inevitably creates ambiguity, because the NSFR cannot disentangle for which of the ladders the valuation function should produce a higher activation in order to make the overall action more likely. In simpler terms, if the reward signal suggests that going right was a good decision, the symbolic policy can conclude that the player must have been left of any ladder, but it does not know of which ladder(s) specifically. Consequently, the valuation function might learn to output high values for the wrong pairs of objects. We refer to this problem as *concept misalignment* and will provide empirical results in Section 5.3 as well as an in-depth theoretical analysis on that end in Section 6.1.

To solve this problem, we propose to extend the current framework by another component: the *concept aligner*. We argue that in order to mitigate concept misalignment, we need to have weak supervision on the predicate-level in addition to the action-level reward signal. Conceptually, this requires access to a *proxy function* $g^{\mathrm{p}} : [-1, 1]^{2(\alpha(\mathrm{p})-1)} \to [0, 1]$ that approximates the optimal valuation function f_ψ^{p}.

Several approaches can be considered to achieve this, three of which we will briefly mention:

(1) **Human-in-the-loop (HIL)** is a method that integrates human feedback into the training (Stammer et al., 2022; Natarajan et al., 2025). The core idea of HIL is to let humans interact with the agent during training to provide weak supervision whenever needed, *e.g.*, when the concept of a predicate appears ambiguous. For this task, humans are able to provide highly accurate proxy functions, because they can naturally understand the semantics of each concept contained in the logic program thanks to their internal world model. That is, they can intervene with high confidence in cases where the predicted truth value for a ground atom deviates from their belief. On the other hand, this approach requires active participation by humans which is undesirable and often infeasible.

(2) **Primitive functions** are predefined spatial functions for basic concepts such as "left" or "close by" and considered primitive because they are not adapted to specific environments. For each predicate, the primitive which approximates it best can then be used as the proxy function in place of a human. Although this method is lightweight, the primitives might not be accurate enough to provide effective feedback.

(3) **LLM-generated functions** advance the idea of HIL by replacing humans with an LLM. With increasing zero-shot generalization capabilities of modern LLMs, it is possible to generate tailored and hence more accurate proxy functions compared to primitives by providing context about the environment and each predicate in the prompt. We acknowledge that this context must once be provided by a human expert, which requires more effort than selecting primitives, but completely eliminates human involvement in the training loop.

In our experiments, we used LLMs to generate proxy functions, as this option balances human overhead and accuracy best. For details on the technical implementation, see Appendix A. We will now describe how the concept aligner integrates such proxy functions into the framework.

First, we define a $K \times K$ grid with offset vectors $(\Delta x_i, \Delta y_i), i \in \{0, \dots, K^2 - 1\}$, uniformly spaced in $[-1, 1]^2$:

$$\begin{bmatrix} \Delta x_i \\ \Delta y_i \end{bmatrix} = \frac{2}{K+1} \cdot \begin{bmatrix} (i \bmod K) + 1 \\ \lfloor i/K \rfloor + 1 \end{bmatrix} - 1 \tag{4.7}$$

In each training step, we then query the outputs $v_i = f_\psi^{\mathtt{p}}(\Delta x_i, \Delta y_i)$ of the valuation functions and compare them to the outputs $\hat{v}_i = g^{\mathtt{p}}(\Delta x_i, \Delta y_i)$ of the proxy functions for these samples. We quantify the error between them by computing the mean binary cross entropy loss

$$L^{\mathrm{CA}}(\psi) = \frac{1}{|\mathcal{P}_e|} \sum_{\mathtt{p} \in \mathcal{P}_e} \frac{1}{K^2} \sum_{i=0}^{K^2-1} -\hat{v}_i \cdot \log v_i - (1 - \hat{v}_i) \cdot \log (1 - v_i) \tag{4.8}$$

over all extensional predicates $\mathcal{P}_e \subset \mathcal{P}$. See Figure 4.4 for a visualization of that process.

Finally, we extend the original BlendRL loss from Eq. (4.4) by the *concept alignment loss* $L^{\mathrm{CA}}(\psi)$. The total objective for our proposed framework becomes

$$L(\theta, \phi, \lambda, \mu, \omega, \psi) = L^{\mathrm{BlendRL}}(\theta, \phi, \lambda, \mu, \omega, \psi) + \left(1 - \gamma_{\mathrm{CA}} \cdot \frac{t}{T}\right) \cdot c_{\mathrm{CA}} \cdot L^{\mathrm{CA}}(\psi) \tag{4.9}$$

where we refer to $c_{\mathrm{CA}} \in \mathbb{R}_{\geq 0}$ as the *concept alignment coefficient*. Note the additional term $\left(1 - \gamma_{\mathrm{CA}} \cdot \frac{t}{T}\right)$, where $t \in \mathbb{N}$ is the current timestep out of T timesteps in total and $\gamma_{\mathrm{CA}} \in [0, 1]$ is a hyperparameter that controls how quickly the concept alignment coefficient attenuates. The rational behind the addition of that factor is to allow the influence of the concept alignment loss to gradually decrease over time. For example, if $\gamma_{\mathrm{CA}} = 1$, the concept alignment loss will completely vanish by the end of the training, whereas for $\gamma_{\mathrm{CA}} = 0$, the loss will not attenuate at all. This definition is motivated by the purpose of the concept aligner, which is to provide guidance to the valuation functions rather than strong supervision. That guidance is potentially more important at early steps where the signal might be more ambiguous. We provide ablation studies with different values for c_{CA} and γ_{CA} in Section 5.3.

5. Experiments

In this chapter, we present the results of our framework tested on two ATARI environments, Kangaroo and Seaquest. First, we will detail the overall experimental setup in Section 5.1. We then provide the empirical results in Section 5.2, followed by additional ablation studies in Section 5.3.

5.1. Experimental Setup

Our framework is comprised of multiple components, the specifications of which we will describe in the following. Further, we elaborate our optimization strategy, which includes a two-stage training pipeline that enables the spatial predicates of the logic actor and the blending module to be learned separately. We also explain the evaluation method that we use to assess the accuracy of the learned concepts.

5.1.1. Model

As described in Section 4.2, our framework consists of a neural and a logic policy, which both employ an actor and a critic. A separate blending module combines the action distributions of both policies. The spatial predicates used by the logic actor and the blending module are valuated using differentiable functions. We will now describe the architecture of all those components in more detail.

A CNN is used as a shared backbone for both, the neural critic V_{μ}^{neural} and neural actor $\pi_{\theta}^{\text{neural}}$, to extract high-level features from the sub-symbolic state. More specifically, the CNN processes the last 4 frames of the ATARI environment given as 84×84 grayscale images each and outputs a latent feature vector of size 512. The neural critic and actor then separately apply a linear transformation on top of those convolutional features to

compute a value estimate and logits for each of the 18 actions possible in an ATARI game. A detailed description of their model architecture is provided in Table 5.1.

Component	Layer	Input Size	Output Size	Nonlin.	Kernel Size	Stride
Shared layers	Conv2d	$4 \times 84 \times 84$	$32 \times 20 \times 20$	ReLU	8	4
	Conv2d	$32 \times 20 \times 20$	$64 \times 9 \times 9$	ReLU	4	2
	Conv2d	$64 \times 9 \times 9$	$64 \times 7 \times 7$	ReLU	3	1
	Flatten	$64 \times 7 \times 7$	3136			
	Linear	3136	512	ReLU		
Neural critic	Linear	512	1			
Neural actor	Linear	512	18			

Table 5.1.: **Model architecture of the neural critic and neural actor.**

The logic policy and the blending module, on the other hand, operate on the object-centric features. We assume that they are provided by an external component, such as one of those presented in Section 2.2.2. In our experiments, we extract object-centric features using *OCAtari* (Delfosse et al., 2023).

For the logic critic V_ω^{logic}, we use an MLP to process the features for all E objects, where the maximum number of objects is $E = 49$ in Kangaroo and $E = 42$ in Seaquest. For each object, we utilize 4 features: a boolean value indicating if the object is present in the scene, the x- and y-coordinates and either the orientation or the value of the object (depending on the object type). The layer configurations of the MLP are shown in Table 5.2.

Layer	Input Size	Output Size	Nonlin.
Linear	$4 \cdot E$	120	ReLU
Linear	120	60	ReLU
Linear	60	1	

Table 5.2.: **Model architecture of the logic critic.** $E = 49$ in Kangaroo and $E = 42$ in Seaquest.

Unlike the above-mentioned components, the logic actor π_ϕ^{logic} and the blending module B_λ are not implemented as neural networks, but as a NSFR instead (see Section 2.2.4).

Throughout all experiments, we used the fuzzy log-sum-exp operator (Eq. 2.22) to aggregate the truth values of action atoms with a smoothing parameter of $\gamma = 0.01$. To compute the truth values for spatial relations among two objects, we use a small MLP (see Table 5.3). It takes the positional offset vector $(x_1 - x_2, y_1 - y_2)$ between both objects as input and produces a probability between 0 and 1 as output.

Layer	Input Size	Output Size	Nonlin.
Linear	2	64	ReLU
Linear	64	32	ReLU
Linear	32	1	Sigmoid

Table 5.3.: **Model architecture of the valuation functions for spatial predicates.**

5.1.2. Optimization

We optimize the model parameters according to the PPO algorithm described in Section 2.1.4 with respect to the joint objective from Eq. (4.9). First, we sample 128 steps from the current policy for multiple environments in parallel. Then, we use GAE from Eq. (2.13) with a discount factor of $\gamma = 0.99$ and a decay factor of $\lambda = 0.95$ to compute advantage estimates. Based on these, the BlendRL objective is computed with coefficients $c_{\mathrm{VF}} = 0.5$ for the value function, $c_{\mathrm{AE}} = 0.01$ for the action entropy and $c_{\mathrm{BE}} = 0.01$ for the blender entropy loss. The clipping coefficient from Eq. (2.18) is set to $\epsilon = 0.1$.

As described in Section 4.2.3, the total objective includes an additional loss term for the concept aligner. We generate proxy functions using the LLMs *Claude 4 Sonnet* by Anthropic (Anthropic, 2025) and *ChatGPT 4o* by OpenAI (OpenAI, 2025). The concept alignment loss from Eq. (4.8) between the proxy functions and the trainable valuation functions is computed for a grid of 49×49 offset vectors (odd numbers to include the zero vector). We use an attenuation factor of $\gamma_{\mathrm{CA}} = 1$ to anneal the loss term by the end of the training. Furthermore, we train the model with different concept alignment coefficients $c_{\mathrm{CA}} \in \{0.03, 0.1, 0.3, 1.0\}$ and report the best result.

We employ the Adam optimizer with a linearly decaying learning rate of $2.5 \cdot 10^{-4}$ and gradient clipping at 0.5. The model parameters are updated in 10 epochs with a batch size of 32 environments. After that, the next steps are sampled from the updated policy.

5.1.3. Training Pipeline

Instead of training the model jointly, we train it in two stages: In the first stage, only the logic policy is learned, including the valuation functions for its spatial concepts. Thereafter, the logic policy is fixed and the neural policy is learned alongside the blending module in the second stage.

To understand why, recall that in BlendRL, both, the logic policy and the blending module, are effectively fixed (except for their clause weights) because their valuation functions are provided by a human expert. Therefore, the reward signal can be clearly distributed to one of the two agents and the neural policy can be learned virtually independently of the logic one. While this simplifies training, the amount of human inductive bias that must be provided is also one of the biggest limitations of BlendRL.

Our framework overcomes this restriction by making the logic policy and blending module learnable as well. However, training them jointly is significantly more complex than in the original BlendRL framework, because the neural and logic policy on one hand and the blending module on the other hand are highly interdependent. That is, if neither policy is provided in advance, the model is unable to distinguish if an action should have been performed by the neural or the logic actor from the reward signal alone, given that both have an overlapping action space. Another potential problem is that the action rules of the logic policy are typically designed to model long-term action sequences with sparse rewards. Therefore, the blending module might learn to completely ignore the logic program in order to maximize short-term rewards that can only be achieved by the neural agent.

All of the above-mentioned issues motivate our two-stage training pipeline, in which we learn both policies separately. To achieve this, we deactivate the neural policy in the first stage by fixing the blending weight at $\beta = 0$. We further simplify the ATARI environments by deactivating all enemies using *HackAtari* (Delfosse et al., 2024). This is necessary because the logic agent does not define rules for actions like punching, shooting, jumping or dodging. Defense and attack mechanisms like these are intended to be learned by the neural agent instead. This approach is similar to the curriculum learning proposed by (Mao et al., 2019), where simple concepts are learned first in less complex environments. Note that without any enemies, the episodes can become extremely long, so we terminate each episode after at most 3000 steps. Additionally, we only provide rewards when the player has achieved the high-level goal of the level, *i.e.*, reaching the child in Kangaroo or rescuing 6 divers in Seaquest. As these goals require long action sequences, we also increase the step size from 1 to 4. Finally, we fix the clause weights of the logic actor to

eliminate ancillary effects caused by prioritizing some actions more than others that are not directly visible from the spatial concepts themselves. We train the logic policy for a total of 10 million steps.

In the second stage, we freeze the model parameters for the logic agent and its valuation functions that have been learned in the first stage. All other components, *i.e.*, the neural agent and the blending module, are trained from scratch. Enemies are reactivated to the environments, the restriction on the maximum episode length is repealed and the step size is reset to 1. The reward function is designed such that the player receives a reward of 20 for reaching the goal and 1 for any other in-game points. This is to incentivize the agent to pursue high-level goals instead of frequent rewards. We train the second stage for a total of 60 million steps.

In both stages, we further modified the Kangaroo environment using HackAtari to disable falling coconuts, repeat the first level and start randomly on the middle or upper floor at the beginning of each episode.

5.1.4. Evaluation

In the following experiments, we empirically evaluate our framework on the ATARI environments Kangaroo and Seaquest for each training stage individually. After the first stage, we test the performance of the logic agent in modified environments without enemies. We do so to assess whether our framework is able to ground spatial concepts that align with the policy program of the logic agent. After the second stage, we then evaluate the hybrid agent in the original environments with enemies reactivated.

To quantify the performance of our framework, we report the average return per episode. Note that this measure alone does not disclose which specific actions have led to the returns, so we also report the number of goals that the agent has achieved. This is of particular interest in the second stage, in which the blending module must be able to appropriately switch between both policies, depending on the spatial concepts it has learned. The additional metric therefore allows us to better distinct if the returns are mainly due to the neural policy and short-term rewards (*e.g.*, defeating enemies), or the interplay of *both* policies, enabling the agent to eventually reach the goal of the level.

We train our models on 3 different seeds and report the results over 100 tested episodes. We compare our framework against a purely neural agent that has been trained with a fixed blending weight of $\beta = 1$, and also include results of a logic agent that directly uses

	Kangaroo		Seaquest	
Model	Return	Goals	Return	Goals
Neural	$1045_{\pm 577.8}$	$0.58_{\pm 0.35}$	$452.63_{\pm 92.88}$	$10.49_{\pm 2.21}$
ChatGPT	984	0.58	835	19.93
Claude	788	0.45	102.8	2.46
BlendRL	3724	2	863.1	20.52
Ours (ChatGPT)	$3540.33_{\pm 131.63}$	$2_{\pm 0}$	$874.13_{\pm 40.16}$	$20.76_{\pm 1.03}$
Ours (Claude)	$3625.67_{\pm 36.46}$	$2_{\pm 0}$	$981.3_{\pm 161.95}$	$23.56_{\pm 4.06}$

Table 5.4.: **Results of the logic agent.** Returns and number of achieved goals for Kangaroo ($c_{\text{CA}} = 0.3$) and Seaquest ($c_{\text{CA}} = 0.3$ for ChatGPT and $c_{\text{CA}} = 0.1$ for Claude). Average results per episode are reported over 3 seeds, with the standard deviation denoted in subscript. Enemies were disabled and the episodic length limited to 3000 steps.

the LLM-generated proxy functions as valuation functions. Furthermore, we report the results of the original BlendRL agent for reference.

5.2. Results

We now present the empirical results for both training stages individually.

5.2.1. Logic Policy Concepts (1st stage)

First, we investigate the logic agent and the spatial concepts it has learned after the first training stage. The results compared to the baselines and BlendRL are presented in Table 5.4. In general, our framework matches or even exceeds the performance of BlendRL. This is regardless of whether the proxy functions of Claude or ChatGPT have been utilized by the concept aligner. In the following, we will explore these results in more detail, first with respect to Kangaroo and then separately for Seaquest.

Figure 5.1 visualizes the valuation functions of the logic agent alongside their corresponding proxy functions for Kangaroo. The hand-crafted functions of BlendRL are also

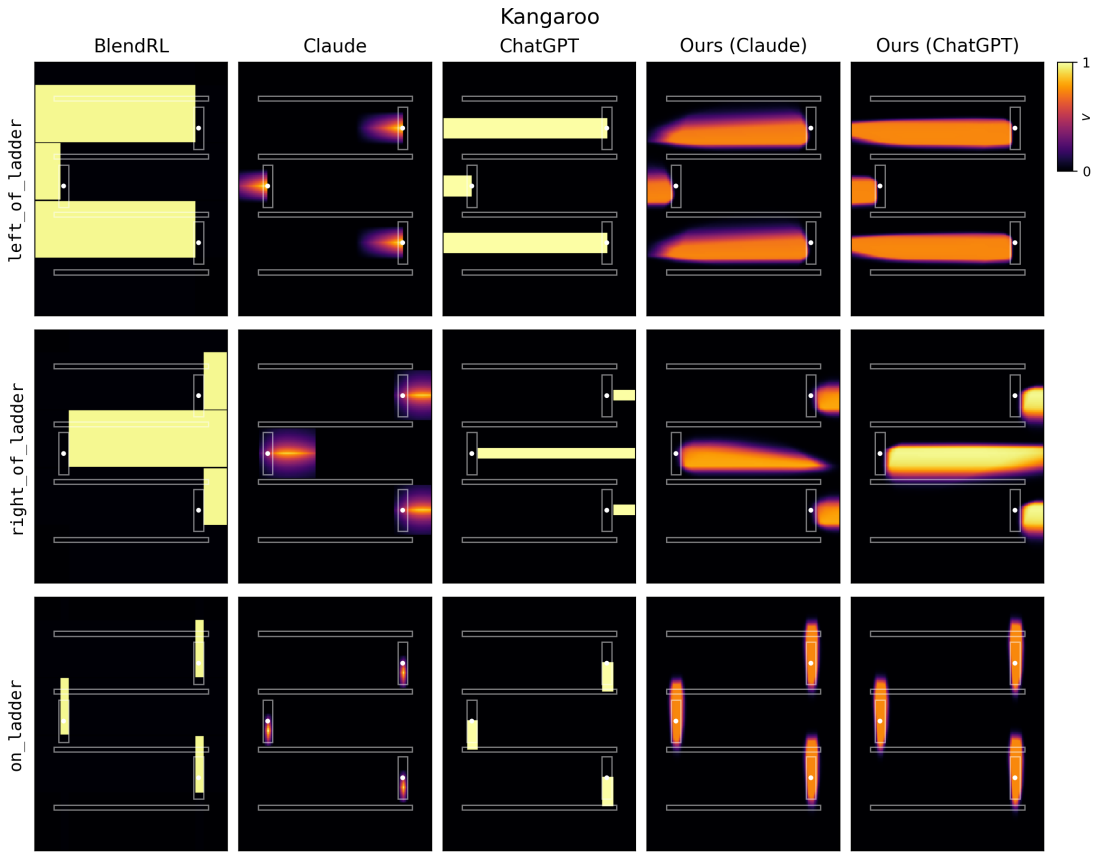

Figure 5.1.: **Spatial concepts learned by the logic agent (Kangaroo).** Heatmaps show the truth values computed by the valuation functions depending on the position of the player in the scene. Each white dot marks the position of a ladder, relative to which the truth values of the spatial predicates are computed. Results are shown (from left to right) for the hand-crafted valuation functions in BlendRL, the proxy functions generated by Claude and ChatGPT, as well as the valuation functions learned by our framework using either proxy functions for weak supervision. The contours of platforms and ladders are depicted as white boxes.

shown for reference. While both, Claude and ChatGPT, have managed to capture the spatial concepts in general, they are not accurately grounded to the environment. See, for instance, the predicates `left_of_ladder` and `right_of_ladder`: The area for which ChatGPT assigns high truth values relative to a ladder extends to the full width of each floor, but is too narrow vertically. Vice versa, Claude covers almost the full height of each floor but deactivates the predicates if the horizontal distance between the player and a ladder exceeds a certain threshold. This illustrates that the LLM-generated proxy functions on their own and without further adjustments are not suited to valuate the spatial predicates directly.

Our framework, on the other hand, uses the LLMs to guide the logic agent while still allowing it to adapt to the environment based on the reward signal. By doing so, our agent is able to overcome the inaccuracies of the proxy functions and successfully learn spatial concepts that align with the specific layout of Kangaroo. The resulting logic policy is on par with BlendRL and significantly outperforms either LLM. This clearly demonstrates that our agent does not simply overfit to the proxy functions, but can deviate from them to learn spatial concepts that are better adapted to the respective environment.

Figure 5.2 analogously shows the valuation and proxy functions for Seaquest. Note that BlendRL employs valuation functions for Seaquest that are much more generic compared to those of Kangaroo. This is plausible because the player can freely move across the whole scene in Seaquest, whereas in Kangaroo, the scene is separated into different floors between which the player can move. ChatGPT was able to account for these environmental differences and matched the hand-crafted functions of BlendRL almost perfectly. Given that the proxy functions of ChatGPT were very well grounded already, the valuation functions learned by our agent are virtually identical and hence have comparable performance. Claude generated similar proxy functions with a noticeable exception of `left_of_diver`, which factors in the distance of the player to the diver. However, the decay is so strong that even slightly distant divers are not covered anymore, which drastically limits the agent's ability to approach a diver to its right. This also leads to a significant drop in performance: Claude rescues divers about 8 times less often than ChatGPT and BlendRL.

Surprisingly, the logic agent that was guided by the proxy functions of Claude performed about 15% better than BlendRL. Looking at the final valuation functions, this may seem counterintuitive at first, but upon closer inspection, it becomes clear why their policy is indeed superior. For one, the agent learned that `left_of_diver` must cover a broader area in order to enclose more distant divers as well. The key differentiating factor, however, is, that the overall activations for `left_of_diver` and `higher_than_diver` are lower than their spatial counterparts `right_of_diver` and `deeper_than_diver`. This duality allows

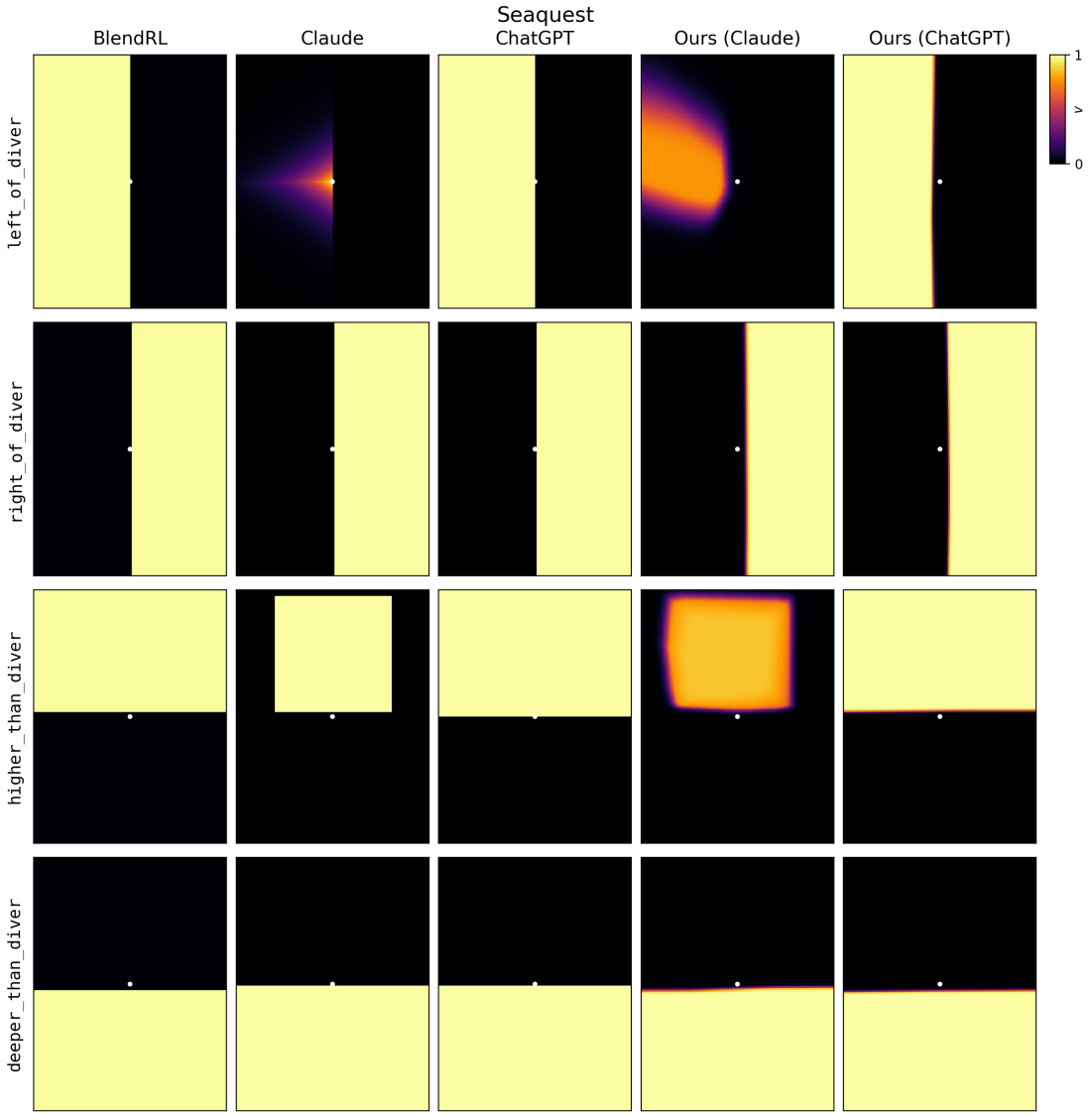

Figure 5.2.: **Spatial concepts learned by the logic agent (Seaquest).** Each white dot marks the position of a diver, relative to which the truth values of the spatial predicates are computed.

	Kangaroo		Seaquest	
Model	Return	Goals	Return	Goals
Neural	$2589.33_{\pm 81.9}$	$0.65_{\pm 0.07}$	$2215.6_{\pm 1209.69}$	$0_{\pm 0}$
BlendRL	5925	3.48	4706.4	0
Ours (ChatGPT)	$4993_{\pm 4619.32}$	$0.66_{\pm 0.21}$	$333.57_{\pm 53.90}$	$1.05_{\pm 0.66}$
Ours (Claude)	$8155_{\pm 4614.72}$	$0.79_{\pm 0.19}$	$867.57_{\pm 447.48}$	$0.87_{\pm 0.98}$

Table 5.5.: **Results of the hybrid agent.** Returns and number of achieved goals for Kangaroo and Seaquest (both $c_{\text{CA}} = 0.3$).

the agent to prioritize an action if two options are equally viable. That is, if the player is located in between two divers (horizontally or vertically), the agent will more likely pursue the diver on the left or above. This decisiveness is crucial and allows the agent to collect and hence also rescue the divers at a faster pace, which is reflected in the improved performance. With this insight, the proxy function for `left_of_diver` that Claude generated was actually close to optimal. If the truth values for either spatial predicate were dependent on the distance, the agent would be approaching the divers closest to the player instead of moving to one particular direction that is always preferred.

We acknowledge, however, that no such effect was observed for the agent that utilized the ChatGPT-generated proxy functions. It therefore is questionable if the outperforming policy was intentionally learned by our framework or just a coincidental effect of the concept aligner enforcing the low activations of the proxy function that Claude generated. Yet, the fact that spatial concepts other than those designed by a human expert perform objectively better reaffirms the argument of our framework: Grounding concepts by hand is difficult, even if they seem trivial, and should therefore be learned by the agent itself.

5.2.2. Blending Concepts (2nd stage)

The results for the hybrid agents learned in the second and final stage are presented in Table 5.5. We will again explore the results separately for Kangaroo and Seaquest.

For Kangaroo, we observe that our agents achieve similar or better performance with regard to the episodic return. However, the number of times the kangaroo child was reached is significantly lower compared to what the original BlendRL agent is able to achieve. The reason for this discrepancy is, that, instead of approaching the child, our agents use

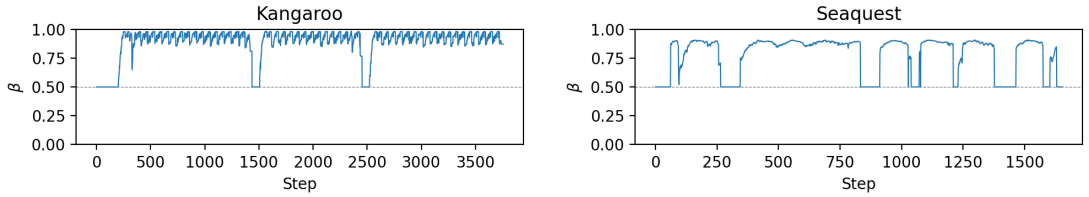

Figure 5.3.: **Neural blending weights in representative episodes of Kangaroo and Seaquest.** The hybrid agents mainly act according to the neural policy ($\beta > 0.5$).

their neural capabilities to get frequent rewards by repeatedly punching monkeys. Hence, the logic policy is largely ignored, as can be seen in Figure 5.3, which shows the neural blending weight β over the course of one episode. The blending module always favors the neural policy and as a result, the child is rarely reached. Also note that β sometimes drops to 0.5, which happens when there are no monkeys in the scene (*e.g.*, because the player has died and the level is reset). The fact that the logic policy is not activated even if monkeys are absent is because the blending module disables the logic agent almost completely by pushing the respective clause weight to near zero (see Figure 5.4).

This effect is also reflected in the spatial concepts that the blending module has learned, shown in Figure 5.5. Claude and ChatGPT suggest the agent to switch to the neural policy whenever a monkey is in close proximity to the player, which aligns with the valuation function employed by BlendRL. Note that, according to these functions, the player and the monkey must be on the same floor, which is reasonable given that the player cannot be attacked by a monkey on a different floor. Our framework, however, activates `close_by_monkey` even if a monkey is one floor above the player. The reason for this is that the agent learns to move to the very right, where it waits for the next monkey to climb down from the floor above. As monkeys keep constantly spawning from the top right, the neural policy stays activated, which allows the agent to get frequent rewards for punching the monkeys. This also causes `close_by_thrownconut` to be never evaluated by the agent because the monkeys get defeated before they can even throw a coconut at the player. Therefore, our framework just adopts the proxy functions provided by the LLMs. Ultimately, the valuation functions are fitted such that the blending module can predominantly use the neural policy.

For Seaquest, equivalent observations can be made. Even though our agents are able to rescue all 6 divers occasionally (as opposed to BlendRL), the logic policy is still mainly

Kangaroo:

```
1.00 :: neural_agent() :- close_by_monkey(player, Monkey)
0.50 :: neural_agent() :- close_by_thrownconut(player, ThrownCoconut)
0.01 :: logic_agent()  :- nothing_around(Z)
```

Seaquest:

```
1.00 :: neural_agent() :- close_by_enemy(player, Enemy)
0.20 :: neural_agent() :- close_by_missile(player, Missile)
0.02 :: logic_agent()  :- visible_diver(Diver)
0.02 :: logic_agent()  :- full_divers(Z)
0.24 :: logic_agent()  :- oxygen_low(OxygenBar)
```

Figure 5.4.: **Clause weights of the blending programs.** The weights are listed next to their clause definitions. Logic policies are mainly suppressed in both ATARI environments.

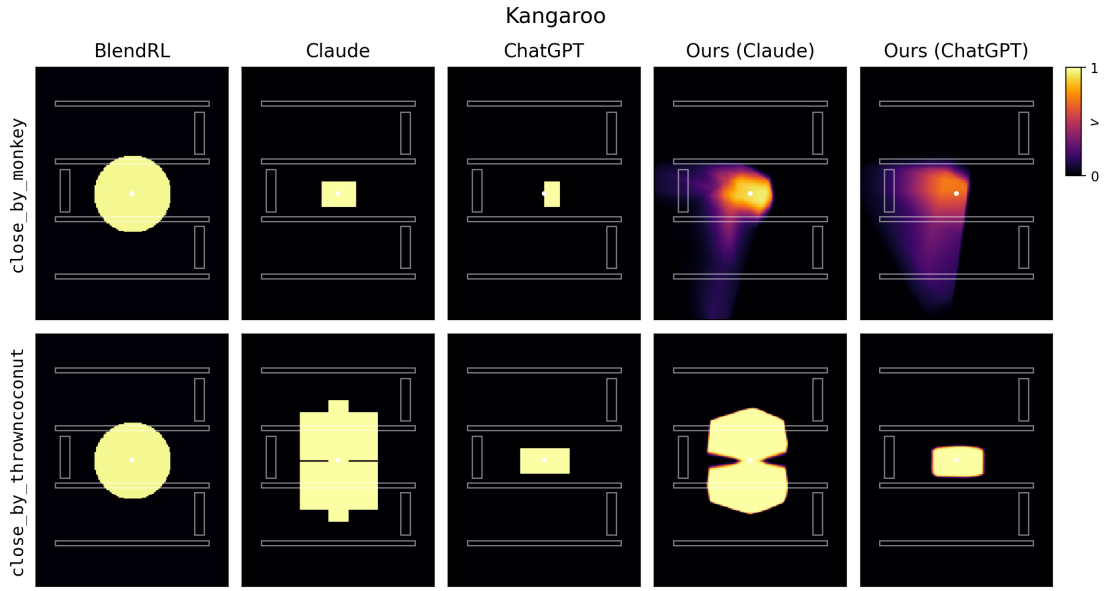

Figure 5.5.: **Spatial concepts learned by the blending module (Kangaroo).** Each white dot marks the position of a monkey or a thrown coconut, respectively.

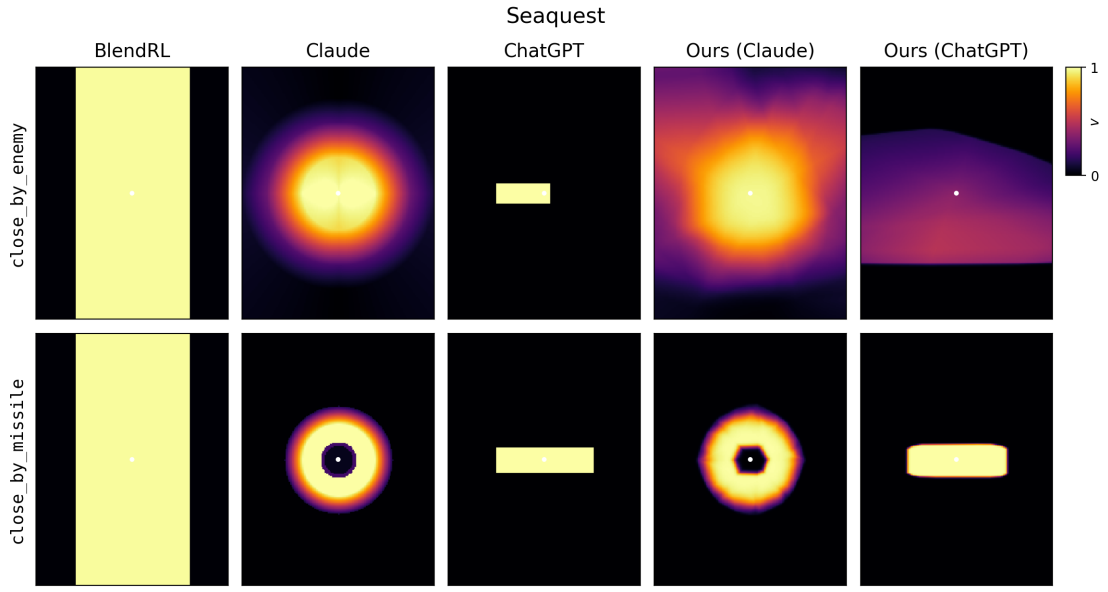

Figure 5.6.: **Spatial concepts learned by the blending module (Seaquest).** Each white dot marks the position of an enemy or missile, respectively.

neglected (see Figure 5.3). Hence, the attribution of the logic agent to these results can be considered rather insignificant. In fact, the logic policy is essentially ignored since the blending module decreased its corresponding clause weights (see Figure 5.4). Solely in the rare case that the player collected 6 divers, it is assigned a slightly higher importance. Like in Kangaroo, the blending module also does not learn spatial concepts that allow for a clear separation of both policies, as can be observed in Figure 5.6. Instead, the valuation functions for `close_by_enemy` cover almost the whole scene, causing high neural activation even if an enemy is far afield. We also observe that `close_by_missile` resembles the proxy functions of the LLMs. This is either because the agent eliminates enemies before they can shoot any missiles, or because the neural activation is mostly attributed to the presence of an enemy rather than a missile.

To summarize, in both ATARI environments, none of our agents were able to utilize the neural and logic policy in a synergistically way to achieve the desired goals.

5.3. Ablation Studies

In this section, we present ablation studies to gain further insights into our framework.

Learning Spatial Concepts without the Concept Aligner To highlight the importance of the concept aligner, we conduct experiments on the logic actor in which spatial concepts were learned without any weak supervision. For that, we disabled the concept alignment loss during the first training stage by setting $c_{\mathrm{CA}} = 0$ and kept all other hyperparameters unchanged.

The results for Kangaroo on 3 independent seeds are shown in Figure 5.7, alongside heat maps of the most likely actions. While none of the agents were able to faithfully ground the spatial concepts, the learned policies are not entirely ineffective. In fact, all agents succeed to reach the child when starting from the upper or, in one case, the middle floor. Due to the spurious valuation functions, however, they do not generalize globally.

To understand why this happens without weak supervision, recall that the agent only receives a reward for reaching the child, so the immediate actions that led to this event will be reinforced first (*i.e.*, going right to reach the ladder and then climbing up the ladder). As these actions are bound to their respective logical rules, the predicates `on_ladder` and `left_of_ladder` must get activated accordingly. The agent can achieve this by either increasing the valuations in relation to the ladder on the upper, middle or lower floor, or by increasing any combination thereof. As humans, we understand that the player can only be left of, right of or on a ladder that is on the same floor, but without some kind of supervision on the object-level, any other option is equally viable for the agent.

In consequence, we can observe that the agent attends to incorrect ladders in distinct ways. Compare, for instance, the global activations for `on_ladder` in all 3 seeds: While all agents have learned that the upper ladder must be climbed up in general, they failed to learn that the agent was on that exact ladder in particular. Instead, the valuation functions rather resemble the concepts ”on the ladder two floors above” or ”on the ladder one floor above on the opposing side”.

This phenomenon can also be observed for the other predicates and highlights a fundamental issue in concept grounding. It emphasizes the necessity for additional guidance and motivates the integration of our concept aligner.

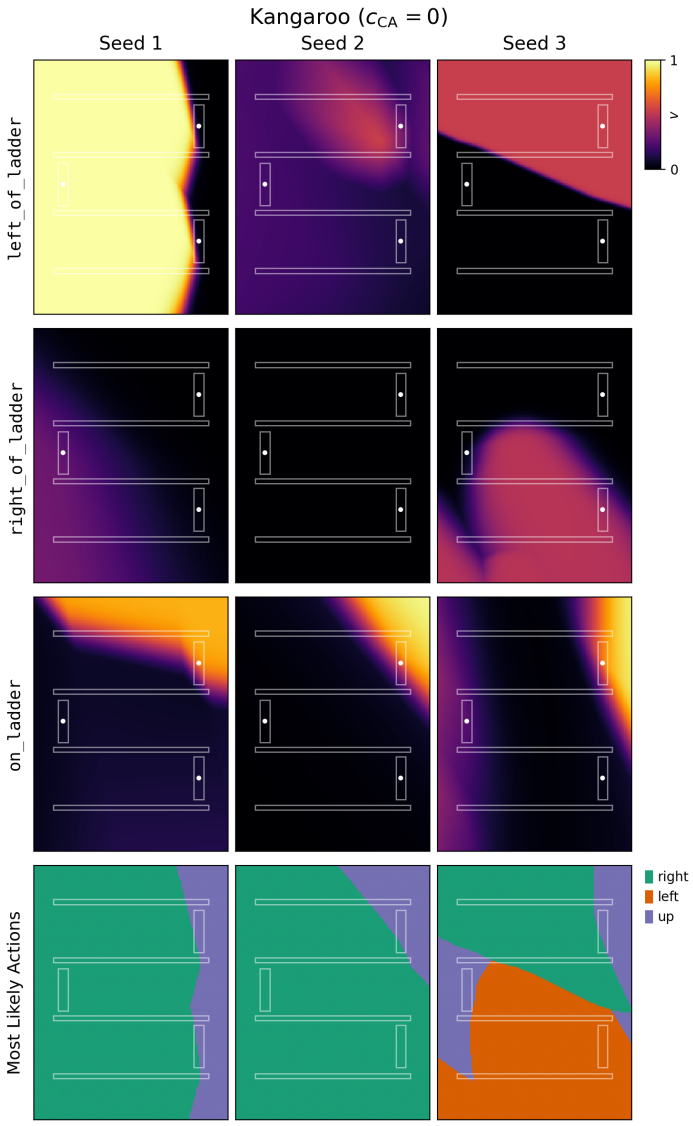

Figure 5.7.: **Spatial concepts learned by the logic agent with disabled concept aligner and most likely actions (Kangaroo).** Results are shown for 3 different seeds. The agent fails to align the concepts to the correct ladders.

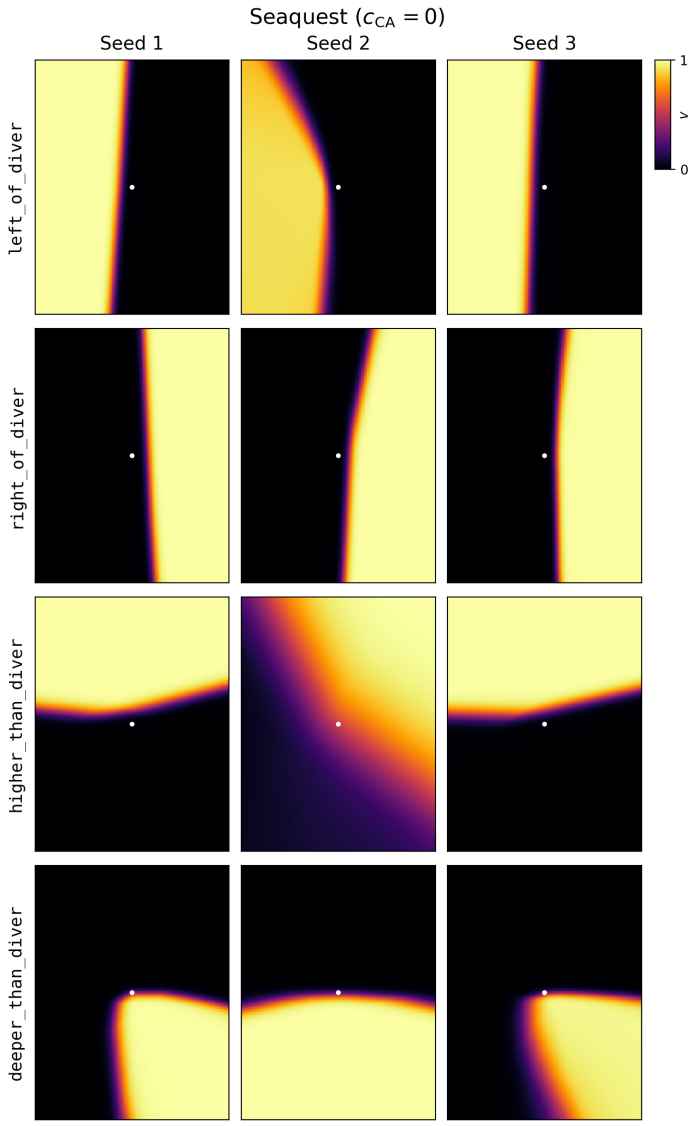

Figure 5.8.: **Spatial concepts learned by the logic agent with disabled concept aligner (Seaquest).** Results are shown for 3 different seeds. The agent is able to learn suitable concepts without requiring additional supervision.

Interestingly, the spatial concepts learned for Seaquest are mostly well-grounded, even without the weak supervision provided by the concept aligner (see Figure 5.8). This might be surprising because Seaquest is a more complex and dynamic environment with up to four divers that can be in the scene simultaneously, hence making concept misalignment potentially more likely. We provide a hypothesis for why the results for Seaquest contrast so strongly with those of Kangaroo in Section 6.1.

Impact of Different Concept Alignment Coefficients Our framework allows to control how strongly the valuation functions shall align with the proxy functions via the concept alignment coefficient c_{CA}. Lower values give the model more flexibility to adapt the concepts to the environment, but might not be effective enough to prevent concept misalignment. On the other hand, too high values for c_{CA} can cause the valuation functions to overfit on the suboptimal proxy functions, which in turn can lead to suboptimal policies.

To help mitigate this, we introduced a second hyperparameter, γ_{CA}, which influences how strongly the concept alignment loss L^{CA} attenuates over time. The rationale behind this factor is the following: We assume that the proxy functions are accurate enough to associate the concepts with the correct objects, yet too poorly adapted to the environment to render learnable valuation functions unnecessary. Therefore, enforcing L^{CA} is more important at the beginning when the agent has not yet figured out how to align the valuation functions correctly, but might be obstructive afterwards where we want it to freely adjust to the environment.

In Figure 5.9, we compare the spatial concepts learned for different values of c_{CA} without annealing the concept alignment loss ($\gamma_{\mathrm{CA}} = 0$). It demonstrates how crucial the choice of c_{CA} is to balance both, flexibility and alignment: On one hand, the agents clearly misalign the valuation functions for `on_ladder` or `left_of_ladder` when using $c_{\mathrm{CA}} \leq 0.1$. On the other hand, increasing the coefficient to $c_{\mathrm{CA}} = 1.0$ causes overfitting (compare with the Claude-generated proxy functions in Figure 5.1). Setting the coefficient to $c_{\mathrm{CA}} = 0.3$ has shown to provide the best tradeoff.

When we run the same experiments but allow the loss term to linearly decrease over time by setting $\gamma_{\mathrm{CA}} = 1$, we can make two interesting observations (see Figure 5.10): (1) The valuation functions are generally more precise and less shallow, which highlights the improved adaptability. (2) The agent does not overfit for $c_{\mathrm{CA}} = 1.0$, which indicates that annealing L^{CA} makes training less sensitive to the choice of c_{CA} and hence eases hyperparameter search.

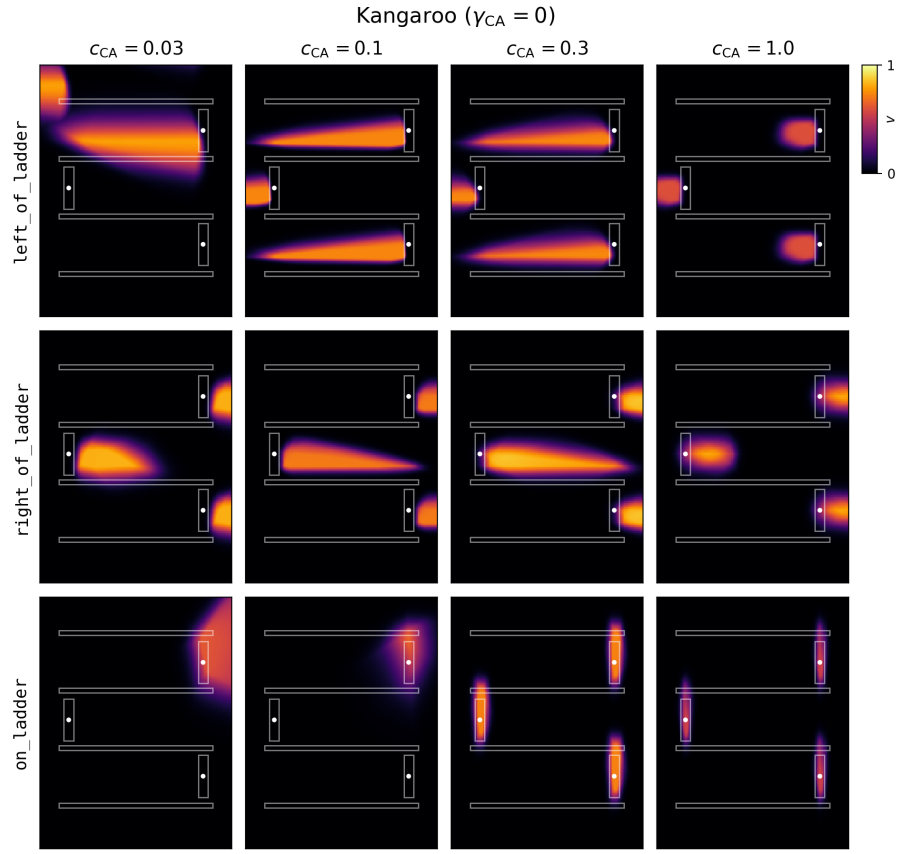

Figure 5.9.: **Spatial concepts learned by the logic agent without annealing the concept alignment loss (Kangaroo).** Results are shown for the agent that used the proxy functions generated by Claude with respect to different values for c_{CA}.

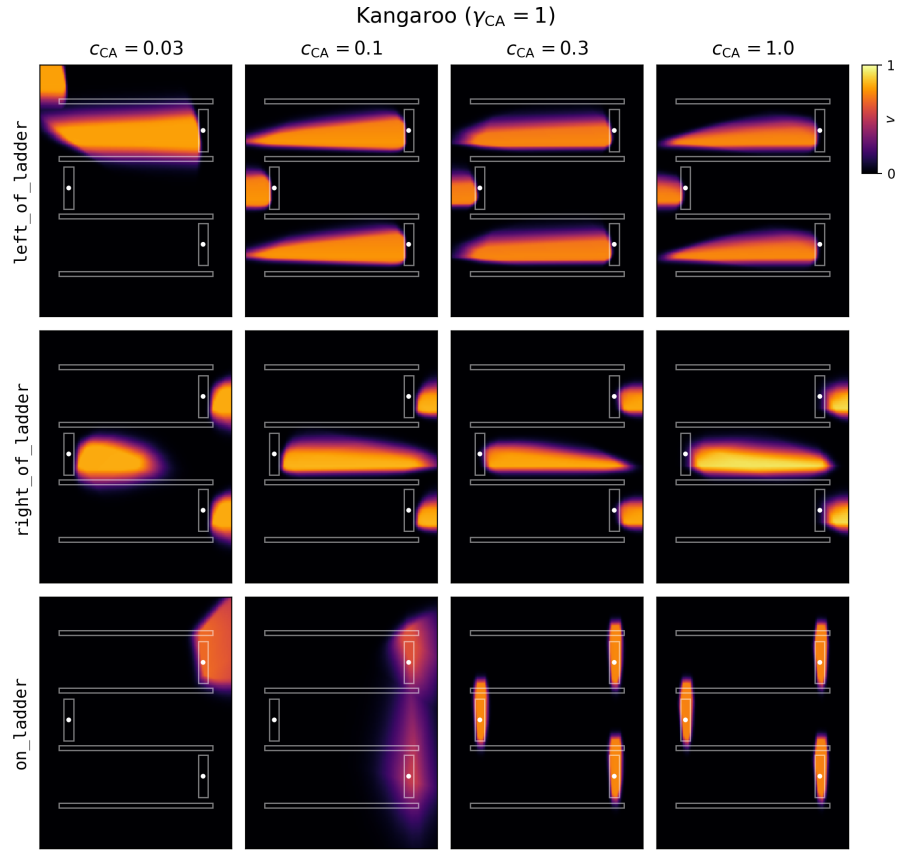

Figure 5.10.: **Spatial concepts learned by the logic agent when annealing the concept alignment loss (Kangaroo).** Results are shown for the agent that used the proxy functions generated by Claude with respect to different values for c_{CA}.

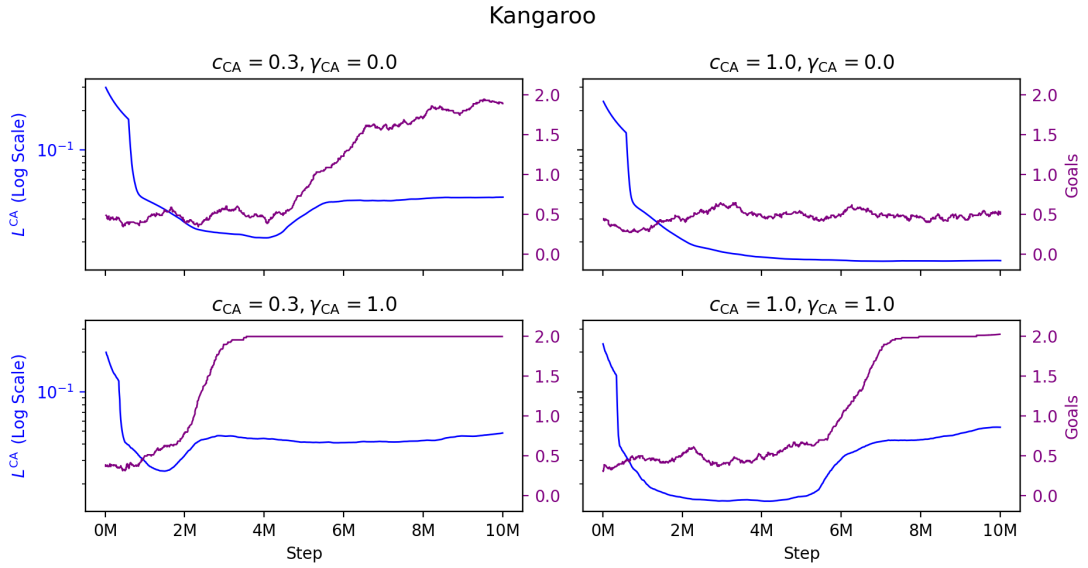

Figure 5.11.: **Comparison of the concept alignment loss L^{CA} and the performance during training (Kangaroo).** Results are compared between agents trained with different concept alignment coefficients ($c_{\mathrm{CA}} \in \{0.3, 1.0\}$) and attenuation factors ($\gamma_{\mathrm{CA}} \in \{0, 1\}$). The concept alignment loss is computed with respect to the proxy functions generated by Claude. Performance improves when concepts diverge from the proxy functions. Agents converge to the optimal policy earlier if the concept alignment loss is attenuated.

Another perspective on that end is provided in Figure 5.11, which compares the performance between the agents learned with $\gamma_{\mathrm{CA}} = 0$ and $\gamma_{\mathrm{CA}} = 1$ during training. First, we observe that an increase in performance is always incident to an increase in L^{CA}. This demonstrates that an agent must be able to diverge from the proxy functions in order to improve its policy. Second, note that for $c_{\mathrm{CA}} = 0.3$, both agents eventually learned a suitable policy, but the one that attenuated the concept alignment loss did converge much quicker.

Overall, these results suggest that a good hyperparameter configuration for the concept aligner is to use higher values for c_{CA} while decreasing it over time by setting the attenuation factor γ_{CA} to 1.

6. Analysis & Discussion

The empirical results give interesting insights into the challenges of learning concepts in neuro-symbolic RL systems that we want to discuss further. We first provide a theoretical analysis on the issue of concept misalignment and elaborate in more detail under which circumstances this problem can occur. We then point out limitations of our framework and suggest future directions of research to overcome these limitations and to improve the performance of our framework.

6.1. Concept Misalignment

The integration of a concept aligner to our framework was motivated by the inability of neuro-symbolic agents to link concepts to their corresponding objects in specific environments. If, for example, an agent correctly learns that it has been on a ladder in a certain state, identifying which of the three ladders the player climbed up in Kangaroo can be difficult from the available signal alone. Therefore, it might associate the concept `on_ladder` to an unrelated ladder, which we refer to as *concept misalignment*. Examples of this phenomenon can be viewed in Figure 6.1.

To understand why and when this problem can occur, let us recall how these concepts are learned using Kangaroo as an example. Every action a is represented by an action rule R that has one body atom of form $\texttt{p}(\texttt{player}, \texttt{Ladder})$. Here, `p` is a spatial predicate (*e.g.*, `on_ladder`) with a parametric valuation function $v_\psi^{\texttt{p}}$, and `Ladder` is a variable representing the set of all 3 ladders. Let $\mathbf{z}_{\texttt{player}}$ and $\mathbf{z}_{\texttt{ladder}}$ further be the object-centric features of the player and a ladder in the current state s. Then,

$$\pi_\theta(a \mid s) \sim v(R) = \bigvee_{\texttt{ladder} \in \texttt{Ladder}} v_\psi^{\texttt{p}}(\texttt{ladder}) \tag{6.1}$$

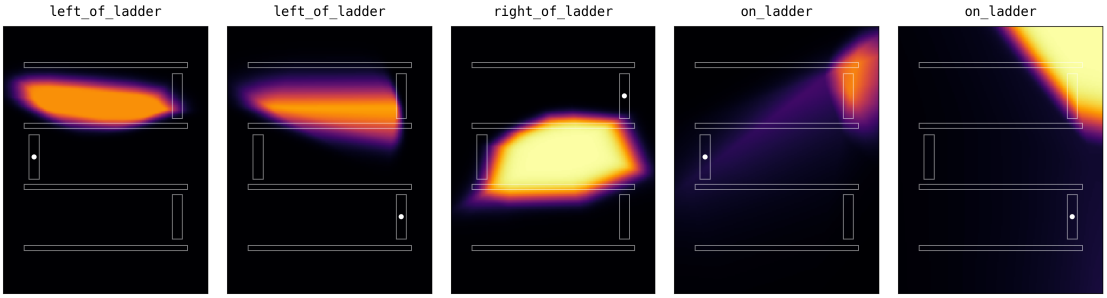

Figure 6.1.: **Examples of misaligned spatial concepts (Kangaroo).** Truth values for each predicate are shown relative to a single ladder object, marked as white dot.

where $v_\psi^{\mathtt{p}}(\mathtt{ladder})$ is shorthand for $v_\psi^{\mathtt{p}}(\mathbf{z}_{\mathtt{player}}, \mathbf{z}_{\mathtt{ladder}})$ and $\vee$ denotes a fuzzy operator for disjunction rather than the boolean logic connective.

Accordingly, the outputs of the valuation functions will be pushed in the direction of

$$\frac{\partial L}{\partial v_\psi^{\mathtt{p}}(\mathtt{ladder})} = \frac{\partial L}{\partial v(R)} \frac{\partial v(R)}{\partial v_\psi^{\mathtt{p}}(\mathtt{ladder})} \tag{6.2}$$

with L being the objective to be optimized. It enforces an increase in $v(R)$ if the action in the current state shall be reinforced, and, conversely, a decrease of $v(R)$ to make the action less likely. This change in $v(R)$ is then distributed to the individual ground atoms, *i.e.*, the outputs $v_\psi^{\mathtt{p}}(\mathtt{ladder})$ for each ladder. However, if the signal is underdetermined, it is unclear how to distribute the changes exactly.

To illustrate this, let us represent a state s in terms of the offset vectors between the player and each ladder. For instance, if the player is on the third ladder with zero offset, we can represent the state as

$$s = \left(\begin{bmatrix} 0 \\ 100 \end{bmatrix}, \begin{bmatrix} 100 \\ 50 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right)$$

Furthermore, assume that $v(\mathtt{up\_ladder})$ shall be increased for that state. Without more information, it is impossible for the agent to determine on which of the three ladders the player was. Hence, increasing $v_\psi^{\mathtt{on\_ladder}}(\mathtt{ladder})$ relative to either ladder leads to equally good policy updates for that particular state. Also, regardless of which fuzzy operator is used for disjunction, the gradient signal will mostly, if not solely, be distributed to the atom with the highest truth value, which amplifies the misalignment even further once the concept is associated with the wrong atom.

The agent can only make a consistent and globally optimal decision if it knows how to behave correctly in similar states. As an example, let us assume that the agent is certain that the player was *not* on any ladder in the following states:

$$s_2 = \left(\begin{bmatrix} -100 \\ 150 \end{bmatrix}, \begin{bmatrix} 0 \\ 100 \end{bmatrix}, \begin{bmatrix} -100 \\ 50 \end{bmatrix} \right) \qquad s_3 = \left(\begin{bmatrix} 100 \\ 50 \end{bmatrix}, \begin{bmatrix} 200 \\ 0 \end{bmatrix}, \begin{bmatrix} 100 \\ -50 \end{bmatrix} \right)$$

Now, the environment is not underdetermined anymore. This is because the agent can deduce from state s_2 and s_3 that the player is not on any ladder if their offset vector is $[0\ 100]^T$ or $[100\ 50]^T$. Therefore, when the player was in state s, it must have been on the third ladder.

Note that this is just a simplified theoretical example. In practice, especially for static environments such as Kangaroo, the state space that the agent can possibly explore might be too small, and even if the agent can observe arbitrary states, the reward signal might be too noisy to clearly identify the correct objects.

Dynamic environments like Seaquest, on the other hand, are potentially more robust against these issues compared to Kangaroo for two reasons: (1) The agent can freely move within the scene, so different arrangements of the player and the divers can be explored naturally. (2) Seaquest is about collecting divers, so the number of divers present in the scene will vary throughout an episode, with fewer divers inevitably causing less confusion.

In conclusion, the purpose of the concept aligner is to provide additional signal in underdetermined environments that can help overcome concept misalignment.

6.2. Limitations

Our framework comes with some limitations that we will address in the following. First, we specifically designed it to model spatial relations, assuming that the concept can be grounded solely based on the object positions. A relation such as `same_color`$(\cdot, \cdot)$ would require other attributes to reason on and is therefore not explicitly covered by our framework. Incorporating more object attributes might be relevant even for the spatial predicates that we employed. If, for example, ladders in Kangaroo had different heights, the valuation functions should take this into consideration accordingly.

In principle, the valuation functions can be extended to include additional features or to model relations between more than two objects, like `in_between`$(\cdot, \cdot, \cdot)$. However,

the LLMs must generate compatible proxy functions, and with an increasing number of features or objects, they might get less accurate.

Moreover, recall that the concept alignment loss is computed over a grid of two-dimensional offset vectors. If more features are needed to valuate a concept, the number of feature combinations to be sampled increases exponentially. While this can be mitigated to some extend by decreasing the sample resolution of continuous variables (like coordinates), it might negatively impact the effectiveness of the concept aligner.

Another limitation of utilizing proxy functions is that the object-centric features must be encoded in a format that the LLMs can understand. Latent encodings such as produced by unsupervised feature extractor models like Slot Attention or IODINE can hence not be used with our framework.

Also recall that the logic policy must be learned in simplified environments without enemies or, more abstractly, without obstacles that could only be overcome by the intervention of a neural agent. Modifying environments in this way is feasible in simulations such as ATARI games, but can be much more difficult or even impossible to achieve in the real world.

Finally, we acknowledge that the logic programs, including the extensional predicates to be learned, must be provided in advance. It remains an open research question how concepts can be grounded while simultaneously using frameworks such as δILP to construct suitable rules.

6.3. Future improvements

In its current state, the extensibility of our framework is limited with regard to the types of concepts it can learn. To solve more complex environments, other object features might need to be processed in addition to the positions, which does not scale well because it would require the concept alignment loss to be computed over significantly more samples. One way to remedy this problem is to adaptively reduce the number of samples, or to redesign the concept aligner altogether such that it does not require sampling at all. For instance, instead of comparing the outputs of the proxy and valuation functions for fixed states of the environment, the actual states observed by the agent could be used instead. In general, improving the scalability of our framework is an important direction for further research.

Another limitation is that proxy functions can only be generated by LLMs if the object features can be interpreted. To support latent feature representations like those learned by Slot Attention, the weak supervision must be provided on the sub-symbolic state, which is impossible to achieve with unimodal LLMs. Using multimodal Vision Language Models like *CLIP* (Radford et al., 2021) instead can make the framework less dependent on explicit object-centric information, but their effectiveness needs to be studied.

One of the key advantages of neuro-symbolic agents over purely neural ones is the fact that their policy can be inspected and even modified by a human. This does not hold for the learned concepts themselves if they are computed by an opaque function such as an MLP. Substituting them by more interpretable architectures such as *Deep Differentiable Logic Gate Networks* (Petersen et al., 2022) can hence make our framework more transparent. Adopting program synthesis (Wüst et al., 2024) for valuation functions could be another promising contribution in this regard.

Lastly, we have seen that our framework is incapable of learning appropriate concepts for the blending module, with the logic policy being ignored almost completely. This is potentially related to the increased frequency of rewards that only the neural agent can achieve (*e.g.*, by punching enemies), so reward shaping might be an option to put more emphasis on reaching the actual goals. To incentivize the blending module to only use the neural policy when necessary, another approach can be to explicitly reward the agent when using the logic policy. Overall, preventing the hybrid agent from unlearning the logic policy would be a significant contribution.

7. Conclusion

In the course of this thesis, we developed a framework based on BlendRL for learning neuro-symbolic agents which can ground relational concepts by interacting with their environment. In contrast to existing systems that rely on manually crafted valuation functions such as BlendRL, our method learns them directly from reward signals and hence reduces the need for human intervention.

Our analysis revealed that learning concepts purely from experience is difficult due to concept misalignment that occurs in underdetermined RL environments. To address this issue, we employ LLMs to provide additional weak supervision, which proved highly effective in our experiments. As a result, our method successfully learned logic policies for the ATARI environments Kangaroo and Seaquest that reached or even surpassed the performance of BlendRL agents.

Nonetheless, our framework was not able to learn a blending policy that effectively combined the logic and neural agents in a synergistic way. To the contrary, the logic policy was largely circumvented, which highlights the need for further research to understand and resolve this issue. It also remains an open question to which extent our approach can be extended to more complex concepts and environments. Moreover, we rely on object-centric representations and predefined logical rules, which may not always be available.

While these challenges are important directions for future work, our findings demonstrate that concept grounding in RL is generally feasible. This underlines the contribution of our framework toward a long-term vision, in which neuro-symbolic agents can autonomously solve abstract tasks without requiring any human bias in advance.

Use of AI tools Parts of the text were translated from German to English using DeepL.

Acknowledgments I would like to thank my supervisor, Hikaru Shindo, for his great support and academic guidance throughout the thesis. I especially enjoyed the constant exchange of novel ideas and interesting insights that went beyond the scope of this work. Furthermore, I want to thank all the members of the AIML lab who contributed to the success of this work with their professional feedback during critical phases, namely Quentin Delfosse and Wolfgang Stammer. I would also like to thank Vivian Taylor for her technical support.

Finally, I thank my parents and my friends for their invaluable emotional support, not only during this work, but throughout my entire studies.

Bibliography

Acharya, Kamal, Waleed Raza, Carlos Dourado, Alvaro Velasquez, and Houbing Herbert Song (2023). "Neurosymbolic Reinforcement Learning and Planning: A Survey". In: *IEEE Transactions on Artificial Intelligence* 5.5, pp. 1939–1953.

Achiam, Josh, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. (2023). "GPT-4 Technical Report". In: *arXiv preprint arXiv:2303.08774*.

Anthropic (2025). *Claude 4 Sonnet*. `https://claude.ai/`. [Large language model].

Archer, E. James (1966). "The Psychological Nature of Concepts". In: *Analyses of Concept Learning*. Elsevier, pp. 37–49.

Bruner, Jerome S., Jacqueline J. Goodnow, and George A. Austin (1956). *A Study of Thinking*. John Wiley and Sons.

Delfosse, Quentin, Jannis Blüml, Bjarne Gregori, and Kristian Kersting (2024). "HackAtari: Atari Learning Environments for Robust and Continual Reinforcement Learning". In: *arXiv preprint arXiv:2406.03997*.

Delfosse, Quentin, Jannis Blüml, Bjarne Gregori, Sebastian Sztwiertnia, and Kristian Kersting (2023a). "OCAtari: Object-Centric Atari 2600 Reinforcement Learning Environments". In: *arXiv preprint arXiv:2306.08649*.

Delfosse, Quentin, Hikaru Shindo, Devendra Dhami, and Kristian Kersting (2023b). "Interpretable and Explainable Logical Policies via Neurally Guided Symbolic Abstraction". In: *Advances in Neural Information Processing Systems* 36, pp. 50838–50858.

Esteva, Francesc and Lluıs Godo (2001). "Monoidal t-norm based logic: towards a logic for left-continuous t-norms". In: *Fuzzy sets and systems* 124.3, pp. 271–288.

Evans, Richard and Edward Grefenstette (2018). "Learning Explanatory Rules from Noisy Data". In: *Journal of Artificial Intelligence Research* 61, pp. 1–64.

Greff, Klaus, Raphaël Lopez Kaufman, Rishabh Kabra, Nick Watters, Christopher Burgess, Daniel Zoran, Loic Matthey, Matthew Botvinick, and Alexander Lerchner (2019). "Multi-Object Representation Learning with Iterative Variational Inference". In: *International conference on machine learning*. PMLR, pp. 2424–2433.

Haarnoja, Tuomas, Haoran Tang, Pieter Abbeel, and Sergey Levine (2017). "Reinforcement Learning with Deep Energy-Based Policies". In: *International conference on machine learning*. PMLR, pp. 1352–1361.

He, Kaiming, Georgia Gkioxari, Piotr Dollár, and Ross Girshick (2017). "Mask R-CNN". In: *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969.

Hsu, Joy, Jiayuan Mao, Josh Tenenbaum, and Jiajun Wu (2023). "What's left? concept grounding with logic-enhanced foundation models". In: *Advances in Neural Information Processing Systems* 36, pp. 38798–38814.

Jiang, Zhengyao and Shan Luo (2019). "Neural Logic Reinforcement Learning". In: *International conference on machine learning*. PMLR, pp. 3110–3119.

Kahneman, Daniel (2011). *Thinking, Fast and Slow*. New York: Farrar, Straus and Giroux.

Koudouna, Daniel and Kasim Terzić (2021). "Few-shot Linguistic Grounding of Visual Attributes and Relations using Gaussian Kernels". In: *Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications-(Volume 5)*. SCITEPRESS-Science and Technology Publications.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep Learning". In: *nature* 521.7553, pp. 436–444.

Locatello, Francesco, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf (2020). "Object-Centric Learning with Slot Attention". In: *Advances in neural information processing systems* 33, pp. 11525–11538.

Luo, Lirui, Guoxi Zhang, Hongming Xu, Yaodong Yang, Cong Fang, and Qing Li (2024). "End-to-End Neuro-Symbolic Reinforcement Learning with Textual Explanations". In: *arXiv preprint arXiv:2403.12451*.

Manhaeve, Robin, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt (2018). "DeepProbLog: Neural Probabilistic Logic Programming". In: *Advances in neural information processing systems* 31.

Mao, Jiayuan, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu (2019). "The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences from Natural Supervision". In: *arXiv preprint arXiv:1904.12584*.

Mao, Jiayuan, Joshua B. Tenenbaum, and Jiajun Wu (2025). "Neuro-Symbolic Concepts". In: *arXiv preprint arXiv:2505.06191*.

Mnih, Volodymyr, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu (2016). "Asynchronous Methods for Deep Reinforcement Learning". In: *International conference on machine learning*. PmLR, pp. 1928–1937.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. (2015). "Human-level control through deep reinforcement learning". In: *nature* 518.7540, pp. 529–533.

Natarajan, Sriraam, Saurabh Mathur, Sahil Sidheekh, Wolfgang Stammer, and Kristian Kersting (2025). "Human-in-the-loop or AI-in-the-loop? Automate or Collaborate?" In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 27, pp. 28594–28600.

OpenAI (2025). *ChatGPT-4o*. `https://openai.com/research/`. [Large language model].

Pallets (2025). *Jinja Documentation (3.1.6)*. `https://jinja.palletsprojects.com`. Accessed August 20, 2025.

Petersen, Felix, Christian Borgelt, Hilde Kuehne, and Oliver Deussen (2022). "Deep Differentiable Logic Gate Networks". In: *Advances in Neural Information Processing Systems* 35, pp. 2006–2018.

Radford, Alec, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. (2021). "Learning Transferable Visual Models from Natural Language Supervision". In: *International conference on machine learning*. PmLR, pp. 8748–8763.

Redmon, Joseph, Santosh Divvala, Ross Girshick, and Ali Farhadi (2016). "You Only Look Once: Unified, Real-Time Object Detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788.

Ren, Shaoqing, Kaiming He, Ross Girshick, and Jian Sun (2015). “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *Advances in neural information processing systems* 28.

Rosch, Eleanor H (1973). “Natural Categories”. In: *Cognitive psychology* 4.3, pp. 328–350.

Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz (2015). “Trust Region Policy Optimization”. In: *International conference on machine learning*. PMLR, pp. 1889–1897.

Schulman, John, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel (2018). “High-Dimensional Continuous Control Using Generalized Advantage Estimation”. In: *arXiv preprint arXiv:1506.02438*.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). “Proximal Policy Optimization Algorithms”. In: *arXiv preprint arXiv:1707.06347*.

Serafini, Luciano and Artur d’Avila Garcez (2016). “Logic Tensor Networks: Deep Learning and Logical Reasoning from Data and Knowledge”. In: *arXiv preprint arXiv:1606.04422*.

Shindo, Hikaru, Quentin Delfosse, Devendra Singh Dhami, and Kristian Kersting (2024). “BlendRL: A Framework for Merging Symbolic and Neural Policy Learning”. In: *arXiv preprint arXiv:2410.11689*.

Shindo, Hikaru, Devendra Singh Dhami, and Kristian Kersting (2021). “Neuro-Symbolic Forward Reasoning”. In: *arXiv preprint arXiv:2110.09383*.

Socher, Richard, Danqi Chen, Christopher D. Manning, and Andrew Ng (2013). “Reasoning with Neural Tensor Networks for Knowledge Base Completion”. In: *Advances in neural information processing systems* 26.

Stammer, Wolfgang, Marius Memmel, Patrick Schramowski, and Kristian Kersting (2022). “Interactive Disentanglement: Learning Concepts by Interacting with their Prototype Representations”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10317–10328.

Sun, Fan-Yun, Weiyu Liu, Siyi Gu, Dylan Lim, Goutam Bhat, Federico Tombari, Manling Li, Nick Haber, and Jiajun Wu (2025). “LayoutVLM: Differentiable Optimization of 3D Layout via Vision-Language Models”. In: *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 29469–29478.

Sutton, Richard S. (1988). “Learning to Predict by the Methods of Temporal Differences”. In: *Machine learning* 3.1, pp. 9–44.

Sutton, Richard S., David McAllester, Satinder Singh, and Yishay Mansour (1999). “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. In: *Advances in neural information processing systems* 12.

Vouros, George A. (2022). “Explainable Deep Reinforcement Learning: State of the Art and Challenges”. In: *ACM Computing Surveys* 55.5, pp. 1–39.

Watkins, Christopher John Cornish Hellaby (1989). “Learning from Delayed Rewards”. In.

Watkins, Christopher John Cornish Hellaby and Peter Dayan (1992). “Q-Learning”. In: *Machine Learning* 8.3, pp. 279–292. DOI: 10.1007/BF00992698. URL: https://doi.org/10.1007/BF00992698.

Williams, Ronald J. (1992). “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning”. In: *Machine learning* 8.3, pp. 229–256.

Wüst, Antonia, Wolfgang Stammer, Quentin Delfosse, Devendra Singh Dhami, and Kristian Kersting (2024). “Pix2code: Learning to Compose Neural Visual Concepts as Programs”. In: *arXiv preprint arXiv:2402.08280*.

Yi, Kexin, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Josh Tenenbaum (2018). “Neural-Symbolic VQA: Disentangling Reasoning from Vision and Language Understanding”. In: *Advances in neural information processing systems* 31.

A. LLM-Generated Proxy Functions

In the following, we provide technical details on how we prompt LLMs to generate proxy functions.

A.1. Prompt Generation

We use the templating engine Jinja (Pallets, 2025) to generate individual prompts for each extensional predicate from a template. It contains the overall instructions with placeholders for information about the environment and the predicates. The template configurations used in Kangaroo and Seaquest are provided in Appendix A.2. Using templates is advantageous for two reasons: (1) They make iterative refinements possible that can be applied to all prompts simultaneously and (2) they can be reused and therefore extended to other environments by replacing the template configuration.

In all of our experiments, we used the following prompt template:

```
# Your Role

You are a model that determines whether a player in the Atari game {{ env.name }} is
    {{ predicate.description }} a {{ predicate.object_types | join(' or ') }}{% if
    predicate.purpose %} such that the player can {{ predicate.purpose }}{% endif %}.
Specifically, given the positional offset between the player and a {{ predicate.
    object_types | join(' or ') }}, you must construct a Python function that
    calculates whether the player is {{ predicate.description }} the {{ predicate.
    object_types | join(' or ') }}{% if predicate.purpose %} and in a position to {{
    predicate.purpose }}{% endif %}, based on the game's mechanics and scene layout.

# Game Context

* The scene is {{ scene.width }} points wide and {{ scene.height }} points tall ({{
    scene.width }}x{{ scene.height }}).
* Origin (0,0) is at the top-left of the screen.
* All object coordinates refer to their center.
```

```
**Object Dimensions**

* Player: {{ scene.player.width }}x{{ scene.player.height }}
{% for object_group in scene.object_groups -%}
* {{ object_group.type | capitalize }}: {{ object_group.width }}x{{ object_group.
    height }}
{% endfor %}

**Game Rules and Mechanics:**

{% for rule in env.rules_and_mechanics -%}
* {{ rule }}
{% endfor %}

# Your Task

* You will create a function called `{{ predicate.name }}(x: torch.Tensor) -> torch.
    Tensor`. It has one PyTorch tensor `x` of size `[batch_size, 2]` as input and
    produces a PyTorch tensor of size `[batch_size]` as output.
* Each two-dimensional vector in the batch `x` is the offset between the positions of
     the player and a {{ predicate.object_types | join(' or ') }}. For instance, if
    the position of the player is `[100, 100]` and the position of the {{ predicate.
    object_types | join(' or ') }} is `[50, 80]`, the offset vector will be `[100 -
    50, 100 - 80] = [50, 20]`.
* For each sample in the batch, you must compute a probability between 0 and 1. This
    probability must indicate how certain you are that the player is {{ predicate.
    description }} the {{ predicate.object_types | join(' or ') }}{% if predicate.
    purpose %} and able to {{ predicate.purpose }}{% endif %}. You may output only 0
    or 1 as well.
* Think about the typical layout of objects in the Atari game {{ env.name }} and
    consider its game rules and mechanics for the function you create.

# Output Format

* End your answer with a Python function of the following form:

```
def {{ predicate.name }}(x: torch.Tensor) -> torch.Tensor:
    # x is a tensor of size [batch_size, 2]
    # function goes here and returns a tensor of size [batch_size]
```
```

A.2. Template Configurations

Each environment is represented by a template configuration that contains all necessary information to generate the prompts. These information include:

- **Environment:** Name of the ATARI game as well as the rules and game mechanics that provide relevant background for the proxy functions.
- **Scene:** Dimensions of the scene, the player and relevant object types in the scene.
- **Predicates:** Name, abstract description, triggered action and purpose of the predicate, as well as the types of related objects (except the player).

The template configuration for Kangaroo is given as:

```
env:
  name: Kangaroo
  rules_and_mechanics:
    - "The player can move left/right and jump only when on a platform."
    - "The player climbs up ladders to reach the platform above."
scene:
  width: 160
  height: 210
  player:
    width: 8
    height: 24
  object_groups:
    - type: ladder
      width: 8
      height: 35
    - type: platform
      width: 128
      height: 4
    - type: monkey
      width: 6
      height: 15
    - type: thrown coconut
      width: 2
      height: 3
predicates:
  - name: "left_of_ladder"
    object_types:
    - ladder
    description: "left of"
    action: go right
    purpose: go right to reach the ladder
  - name: "right_of_ladder"
    object_types:
    - ladder
    description: "right of"
    action: go left
    purpose: go left to reach the ladder
  - name: "on_ladder"
    object_types:
    - ladder
    description: "on"
    action: go up
```

```
    purpose: climb up the ladder
  - name: "close_by_monkey"
    object_types:
    - monkey
    description: "close by"
    action: throw punches
    purpose: throw punches at the monkey
  - name: "close_by_thrownccoconut"
    object_types:
    - thrown coconut
    description: "close by"
    action: dodge
    purpose: dodge the thrown coconut
```

Similarly, the template configuration for Seaquest is:

```
env:
  name: Seaquest
  rules_and_mechanics:
    - "The player can move left/right/up/down to collect divers."
    - "The player loses a life if hit by a missile, shark or submarine."
scene:
  width: 160
  height: 210
  player:
    width: 16
    height: 11
  object_groups:
    - type: diver
      width: 8
      height: 11
    - type: shark
      width: 8
      height: 7
    - type: submarine
      width: 8
      height: 11
    - type: missile
      width: 6
      height: 4
predicates:
  - name: "left_of_diver"
    object_types:
    - diver
    description: "left of"
    action: go right
    purpose: go right to get closer to the diver
  - name: "right_of_diver"
    object_types:
    - diver
    description: "right of"
    action: go left
    purpose: go left to get closer to the diver
```

```
  - name: "higher_than_diver"
    object_types:
    - diver
    description: "above"
    action: go down
    purpose: go down to get closer to the diver
  - name: "deeper_than_diver"
    object_types:
    - diver
    description: "below"
    action: go up
    purpose: go up to get closer to the diver
  - name: "close_by_enemy"
    object_types:
    - shark
    - submarine
    description: "close by"
    action: shoot
    purpose: shoot to kill the enemy
  - name: "close_by_missile"
    object_types:
    - missile
    description: "close by"
    action: dodge
    purpose: dodge the missile
```

The generated Python code for the proxy functions can be found on the GitHub repository.