Interpretable Reinforcement Learning via Meta-Policy Guidance

Raban Emunds[*1], **Jannis Blüml**[1,2], **Quentin Delfosse**[1,3], **and Kristian Kersting**[1,2,4,5]

[1]Department of Computer Science, Technical Universty Darmstadt, Germany
[2]Hessian Center for Artificial Intelligence (hessian.AI)
[3]National Research Center for Applied Cybersecurity (ATHENE), Germany
[4]Centre for Cognitive Science, Darmstadt
[5]German Research Center for Artificial Intelligence (DFKI)

Abstract

Deep reinforcement learning has demonstrated strong performance across a range of tasks, but its reliance on opaque neural policies hinders interpretability and alignment. Neurosymbolic approaches attempt to improve transparency by integrating symbolic reasoning, but when applied at the level of fine-grained actions, they often produce policies whose complexity obscures interpretability. We introduce LENS, an object-centric hierarchical reinforcement learning framework that combines neural low-level skill policies with symbolic high-level meta-policies to achieve both efficiency and interpretability. Our approach leverages the use of object-centric representations, which structure the environment in a way that enables large language models to generate meaningful skill definitions, reward functions, and meta-policy rules. We further extend the symbolic reasoning layer with a neuro-symbolic formulation of the meta-policies, improving expressiveness and generalization. All components are trained jointly using an off-policy algorithm that supports efficient and parallel learning of the sub-policies. LENS demonstrates strong performance while maintaining interpretability.

1 Introduction

Recent advancements in Deep Reinforcement Learning (DRL) have led to highly capable agents on a diverse set of tasks (Mnih et al., 2015; Schulman et al., 2017; Gallici et al., 2024); however, these policies are most often neural that operate as black boxes and exhibit difficult to interpret behaviors that may be misaligned (Rudin, 2019). Without interpretability, identifying misalignment or correcting undesirable behaviors, remains a significant challenge for practitioners (Zahavy et al., 2016; Zhang et al., 2018; Delfosse et al., 2024c).

Neurosymbolic approaches address this issue by combining neural networks for perceptual grounding with symbolic reasoning modules for decision-making (Delfosse et al., 2023a; Hazra & Raedt, 2023; Acharya et al., 2024). These architectures aim to enhance transparency by representing policies through symbolic structures that are more readily interpretable. Despite their promise, applying symbolic reasoning directly to low-level action spaces often results in policies of prohibitive complexity (*cf.* Figure 1). The combinatorial explosion of symbolic rules at fine-grained action levels undermines interpretability and scalability. Hierarchical Reinforcement Learning (HRL) offers an alternative by abstracting sequences of actions into higher-level skills or options (Sutton et al., 1999; Dietterich, 2000; Barto & Mahadevan, 2003). While these framework introduce a natural structure, the resulting policies often remain entangled and difficult to interpret.

[*]raban.emunds@cs.tu-darmstadt.de

18th European Workshop on Reinforcement Learning (EWRL 2025).

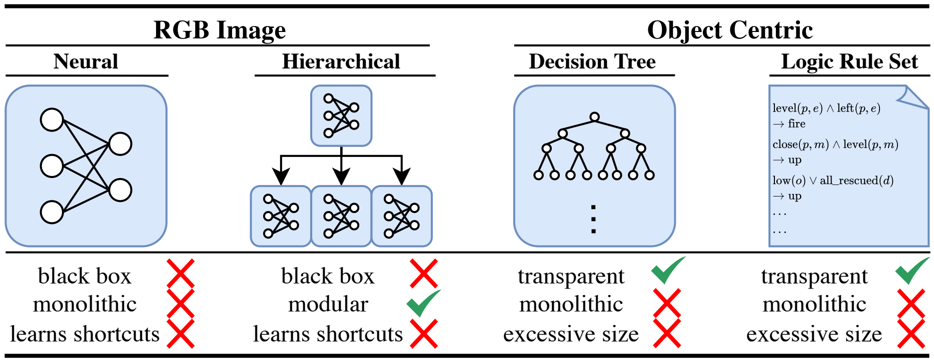

Figure 1: **Common approaches are not truly interpretable.** Conventional RL based on NNs fails to reveal the agent's current objective. Hierarchical RL can help by decomposing into sub-policies, however, there is no reasoning that explains why a specific sub-policy is preferred. Symbolic approaches leverage logical rule sets or decision trees on symbolic states for interpretable atomic actions, but quickly become excessively complex.

To address these limitations, we draw inspiration from the dual-process theory of cognition (Kahneman, 2011), which distinguishes between fast, intuitive (System 1) and slow, deliberative (System 2) reasoning. We emulate this cognitive structure by combining reactive, neural skills with symbolic, high-level policies to achieve both robust performance and interpretability. Our framework LENS, the **L**ogically **E**nhanced **N**euron **S**kills, preserves the effectiveness of neural policies at the low level while introducing a meta-policy layer composed of simple, interpretable rule functions that retain clarity even in complex environments (*cf.* Figure 2). We leverage the reasoning capabilities of Large Language Models (LLMs) to identify the necessary skills, define high-level meta-policy functions, and generate reward signals to guide the learning of low-level neural controllers. Both the low-level skills and the meta-policy rule weights are learned jointly in an off-policy manner. This design enables expressive and transparent high-level reasoning without compromising the efficiency of low-level execution.

Our primary contributions are as follows:

(i) We introduce LENS, a hierarchical and interpretable reinforcement learning framework that integrates high-level decision-making with low-level skill execution.

(ii) We extend PQN to hierarchical settings for efficient and scalable sub-policy learning.

(iii) We demonstrate that generated rewards and high-level meta-policy functions can be leveraged to guide training towards aligned policies.

(iv) We propose a neuro-symbolic extension to the meta-policy functions, enhancing their generalization, and empirically evaluate its effectiveness.

We proceed as follows. We start off by discussing background and related work in Section 2, followed by a detailed introduction of LENS in Section 3. Before concluding, we empirically evaluate our method in Section 4.

2 Background and Related Work

2.1 Background

Deep Reinforcement Learning. Reinforcement Learning (RL) is a framework for sequential decision making in which an agent learns to interact with an environment in order to maximize cumulative reward. The environment is typically modeled as a Markov Decision Process (MDP), defined by the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $P(s' \mid s, a)$ is the transition probability from state s to state s' under action a, $R(s, a)$ is the reward function and $\gamma \in [0, 1)$ is the discount factor. The goal of the agent is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected discounted return: $\mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$.

In *Q-learning*, a value-based RL algorithm, the agent seeks to learn the optimal action-value function $Q^*(s,a)$, which satisfies the Bellman optimality equation: $Q^*(s,a) = \mathbb{E}_{s'}\left[R(s,a) + \gamma \max_{a'} Q^*(s',a')\right]$. This function is updated iteratively via the Q-learning update rule: $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_t + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t)\right]$, where α is the learning rate.

With growing state space $\mathcal{S}$, it becomes infeasible to store and update a tabular Q-function. *Deep Q-Networks (DQN)* (Mnih et al., 2015) address this by using a deep neural network $Q_\theta(s,a)$, parameterized by θ, to approximate $Q(s,a)$. DQN introduces several key modifications to stabilize learning, including experience replay, where transitions (s_t,a_t,r_t,s_{t+1}) are stored in a replay buffer and sampled randomly to break correlations between consecutive updates and target networks, where a separate network $Q_{\theta^-}(s,a)$ is used to compute the target value and is updated periodically: $y_t = r_t + \gamma \max_{a'} Q_{\theta^-}(s_{t+1},a'), \quad \mathcal{L}(\theta) = (Q_\theta(s_t,a_t) - y_t)^2$.

More recently, Gallici et al. introduced *Parallelised Q-Networks (PQN)*, a simplified variant of DQN that eliminates the use of experience replay and target networks. Instead, PQN leverages a large number of parallel (ideally vectorized) environments and applies normalization techniques to mitigate training instabilities. This high degree of parallelization enables substantially faster training while maintaining performance comparable to state-of-the-art reinforcement learning algorithms.

Interpretable Reinforcement Learning. Interpretability in RL is a widely studied problem and can be addressed at various stages of the RL pipeline (Glanois et al., 2024). A prevalent strategy involves extracting symbolic representations from raw observations using computer vision techniques such as object recognition or instance segmentation. These representations can be derived using simple methods like template matching (Li et al., 2017; Iyer et al., 2018), or through learned approaches (Lin et al., 2020; Delfosse et al., 2023b).

In this work, we assume access to structured object-level information from the environment, allowing us to concentrate on enhancing the transparency of the decision-making process. Prior research leveraging symbolic state representations has explored interpretable policy learning via decision trees (Silva et al., 2019; Likmeta et al., 2020; Pace et al., 2022), logical formulas (Maes et al., 2012; Hein et al., 2018), (fuzzy) logic rules (Akrour et al., 2019; Jiang & Luo, 2019; Payani & Fekri, 2019; Delfosse et al., 2023a), and programmatic policies (Verma et al., 2019; Anderson et al., 2020). Most recently, Kohler et al. proposed distilling policies into Python programs to improve transparency and facilitate user intervention. In a similar vein, our approach aims to produce concise Python functions operating at the highest level of abstraction, prioritizing readability and ease of modification by end users.

Hierarchical Reinforcement Learning. Hierarchical reinforcement learning (HRL) introduces a structured approach to decision-making by decomposing complex tasks into a hierarchy of simpler sub-tasks, often modeled as temporally extended actions or "options" (Sutton et al., 1999). In its simplest form, HRL involves a high-level policy that selects among a set of lower-level sub-policies, each responsible for achieving a specific subgoal. This abstraction enables more efficient exploration and transfer, particularly in environments with long horizons or sparse rewards. These sub-policies — or options — can either be learned autonomously using intrinsic objectives or temporal abstractions (Bacon et al., 2017; Vezhnevets et al., 2017), or provided with manually designed reward functions to guide learning toward interpretable or task-relevant behaviors (Dietterich, 2000). This flexibility allows HRL frameworks to balance between automatic skill discovery and user-defined structure, making them well-suited for interpretable and modular policy design.

2.2 Related Work

Prior work on interpretable reinforcement learning also explored hierarchical decompositions, where a high-level interpretable policy selects among low-level, non-interpretable sub-policies.

Andreas et al. (2017) propose using annotated task sketches to define high-level structure and jointly learn reusable sub-policies. Leonetti et al. (2016) constrain agent behavior via symbolic planning, enabling more adaptable low-level RL. Yang et al. (2018) similarly integrates symbolic planning with RL, using experience to iteratively refine the symbolic planner. Jin et al. (2022) plan in symbolic state spaces using PDDL and execute predefined symbolic options. More recently, Ye et al. (2025) utilizes differentiable symbolic planning to compose neural subpolicies. In contrast, James et al. (2022) autonomously learn object-centric representations for transferable symbolic planning. Object-centric

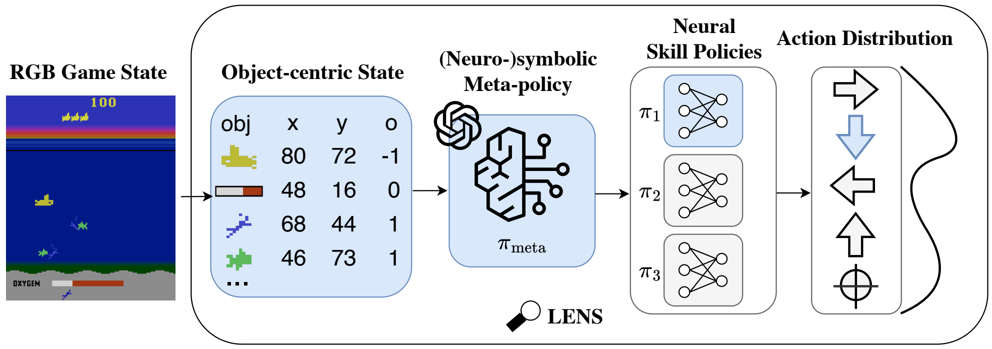

Figure 2: **LENS enables interpretable decision making via symbolic meta-policies**. The process begins with extracting an object-centric state from the image. Next, the meta-policy selects the most suitable neural skill policy for the current subgoal. Finally, the chosen expert policy determines the action to execute. The meta-policy can be generated by an LLM and consists of a set of rules, potentially weighted by a learned meta Q-function.

hierarchical RL is also effective in robotics, e.g., Sharma et al. (2020) uses RL to select and sequence object-axis controllers for manipulation tasks.

The above approaches depend on expert knowledge of both the environment and symbolic reasoning, and are typically limited to simple domains. In contrast, we define the meta-policy as a rule-based code function generable by LLMs, reducing the need for manual expertise and evaluate on more complex environments. Furthermore, our neuro-symbolic extension to the meta-policy resolves rule conflicts.

3 Logically Enhanced Neuron Skills

To address the challenge of building interpretable yet performant reinforcement learning agents, we introduce LENS, a hierarchical framework that combines symbolic reasoning with neural skill execution. At a high level, LENS decomposes decision-making into two layers: a high-level meta-policy, defined as a set of human-readable rules, and a set of low-level neural skill policies optimized for specific sub-goals. Object-centric representations form the backbone of this architecture, enabling both structured reasoning and compatibility with LLMs for automated policy synthesis. Figure 2 describes the overall pipeline.

Each neural skill is associated with its own reward function and trained jointly with the overall pipeline, using shared off-policy data. For the meta-policy on the hand, we provide three approaches with various levels of interpretability and flexibility.

We first introduce hierarchical PQN (HPQN) in Section 3.1, which learns a neural meta-policy and serves as the foundation for subsequent variants, offering high flexibility but limited interpretability. In Section 3.2, we replace the learned meta-policy with a fixed, interpretable function, either manually defined or LLM-generated, sacrificing flexibility for clarity. Finally, Section 3.3 presents a hybrid approach that filters candidate skills via predefined rules and selects among them using Q-learning, yielding a soft meta-policy that balances adaptability and interpretability, especially in scenarios with overlapping skill applicability.

3.1 Hierarchical PQN

Inspired by MAXQ (Dietterich, 2000), we reformulate the environment as a collection of MDPs $\boldsymbol{\mathcal{M}} = \{\langle \mathcal{S}, \mathcal{A}, P, R_n, \gamma \rangle\}_{n=1}^{N}$, for N options, where each $M_n \in \boldsymbol{\mathcal{M}}$ corresponds to a distinct option policy $\pi_n \in \Pi$. Additionally, we define a meta-level MDP $\mathcal{M}_{\text{meta}} = \langle \mathcal{S}, \Pi, P, R, \gamma \rangle$, solved by a meta-policy π_{meta} that selects among the set of available options Π.

Unlike the MAXQ decomposition, in our setting, the option MDPs differ solely in their reward functions, while the meta-level MDP preserves the original environment reward R. Moreover, differing from the standard options framework (Sutton et al., 1999; Bacon et al., 2017), we invoke the meta-policy at every time step for more flexible top-down decisions, instead of delegating termination conditions to the option level.

To enable learning across sub-policies even when they are not actively selected, we adopt off-policy Q-learning. This allows skills to learn from trajectories generated by the active skill and meta-policy while optimizing for their respective reward structures. Exploration within each skill is conducted via ϵ-greedy action selection.

For its simplicity and scalability through vectorization, we build upon PQL with λ-returns (Gallici et al., 2024) as the foundation for our hierarchical training approach. However, instead of learning a single shared Q-network Q_ϕ as in PQL, we learn separate Q-functions Q_{ϕ_n} for each low-level skill, along with a meta-level Q-network $Q_{\phi_{\text{meta}}}$. Each Q-network is thus specialized for predicting returns under its corresponding reward signal.

Each update step starts by rolling out small trajectories $(s_0, ..., s_T)$ following the hierarchical policy by selecting the next active skill $\pi_n \in \Pi$ using the meta-policy Q-function $Q_{\phi_{\text{meta}}}$

$$\pi_n = \arg\max_{\pi_n \in \Pi} Q_{\phi_{\text{meta}}}(s_t, \pi_n) \tag{1}$$

and then sampling the next environment action a' from the active skill

$$a' = \arg\max_{a' \in \mathcal{A}} Q_{\phi_n}(s_t, a'). \tag{2}$$

This action is applied to the environment to retrieve the next state and obtain rewards. We employ ϵ-greedy for exploration during both the skill and the action selection.

Next to the environment reward $r_{\text{env},t}$, we require skill-specific reward functions $r_{n,t}$ that are based on the object-centric state s_t and may be automatically generated by LLMs (*cf.* Appendix D), which we use to compute the λ-returns recursively back in time (for details, we refer the reader to Gallici et al. (2024); Daley & Amato (2019)):

$$R^\lambda_{n,t} = r_{n,t} + \gamma \left[\lambda R^\lambda_{n,t+1} + (1-\lambda) \max_{a'} Q_{\phi_n}(s_{t+1}, a') \right], \tag{3}$$

and similarly, for the learned meta-policy using environment rewards:

$$R^\lambda_{\text{env},t} = r_{\text{env},t} + \gamma \left[\lambda R^\lambda_{\text{env},t+1} + (1-\lambda) \max_{\pi_n} Q_{\phi_{\text{meta}}}(s_{t+1}, \pi_n) \right], \tag{4}$$

or, if s_t is terminal $R^\lambda_{n,t} = r_{n,t}$ and $R^\lambda_{\text{env},t} = r_{\text{env},t}$. Finally, the learned Q-functions are updated towards the λ-returns. The full algorithm can be found in Appendix A.

3.2 Interpretable Meta-Policy Function

While the hierarchical structure introduced in previous sections enables identification of the currently active skill, it does not provide insight into the decision-making process that selected that skill. To address this gap, we propose interpretable meta-policy functions that facilitate understanding of the underlying selection rationale.

Although various interpretable policy representations exist, such as decision trees or FOL-based formulations, we opt for programmatic rule-based functions due to two primary considerations. First, rule-based policies offer accessibility to a broader audience: unlike FOL, simple conditional statements can typically be understood and modified even by individuals with limited programming experience. Second, LLMs are proficient in handling code-like structures, owing to extensive pretraining on programming corpora. This allows for direct use of LLMs in synthesizing rule-based meta-policies, enabling efficient generation of high-quality, interpretable policies with minimal manual intervention. Further details on how we employ LLMs to generate the meta-policy function are available in Appendix D.

Formally, instead of relying on a learned meta-level Q-function $Q_{\phi_{\text{meta}}}$, we define a meta-policy $\pi_{\text{meta}} : \mathcal{S} \to \Pi$ as a set of human-readable rules that directly maps the current object-centric state $s_t \in \mathcal{S}$ to the selected low-level policy $\pi_n \in \Pi$:

$$\pi_n = \pi_{\text{meta}}(s_t). \tag{5}$$

Algorithm 1 LENS Variants

Require: Skill policies $\{\pi_n\}_{n=1}^{N}$ with Q-functions $\{Q_{\phi_n}\}_{n=1}^{N}$, meta-policy $Q_{\phi_{\text{meta}}}$ (for learned/soft)

for each episode **do**
  for each environment step (in parallel) **do**
    Extract object-centric state s_t
    Select skill $\pi_{n,t}$ as follows:
     I) Learned: $\pi_{n,t} = \arg\max_{\pi_n} Q_{\phi_{\text{meta}}}(s_t, \pi_n)$
    II) Fixed: $\pi_{n,t} = \pi_{\text{meta}}(s_t)$
    III) Soft: $\pi_{n,t} = \arg\max_{\pi_n} (c_t \odot Q_{\phi_{\text{meta}}}(s_t, \pi_n))$
    Select action a_t using ϵ-greedy from $\pi_{n,t}$
    Execute a_t and observe transition $(\{r_{n,t}\}_{n=0}^{N}, r_{\text{env},t}, s_{t+1})$
  end for
  for each gradient step **do**
    Compute $R_{n,t}^{\lambda}$ and $R_{\text{env},t}^{\lambda}$ (if I or III)
    Update each Q_{ϕ_n} using skill-specific $R_{n,t}^{\lambda}$
    Update $Q_{\phi_{\text{meta}}}$ using environment $R_{\text{env},t}^{\lambda}$ (if I or III)
  end for
end for

By constraining π_{meta} to consist of simple rules (*cf.* Figure 4, left side) that are mutually exclusive, we enable transparent inspection of policy decisions. The active skill at any time can be traced back to the specific rule that evaluates to true given the current state.

Beyond interpretability, rule-based meta-policies can also mitigate common failure modes in hierarchical reinforcement learning, such as premature convergence to a single sub-policy. This inductive bias encourages broader exploration and more robust policy composition.

3.3 Soft Meta-Policy Function

Thus far, the meta-policy has been treated as a deterministic function that selects skills based on fixed rules derived from the extracted environment state. However, this approach may become inefficient in situations where multiple skills are simultaneously applicable.

Consider, for example, a Seaquest agent faced with the choice between shooting a nearby enemy or resurfacing to replenish oxygen. Both actions could be viable depending on the specific context: if oxygen levels are critically low, resurfacing may be prioritized; if an enemy is dangerously close, attacking may take precedence. Crafting hand-coded rules to handle all such trade-offs would quickly lead to complex, less interpretable policies.

Instead, we propose maintaining a set of high-level, interpretable conditions (e.g., "surface if oxygen is low", "fight if enemy is close"), represented as a binary condition vector $c_t \in \{0,1\}^N$, where each entry indicates whether a particular condition is active at time t. We formulate these rule-sets again as simple functions and leverage LLMs for generation. However, this time we allow multiple conditions to be active concurrently.

The skill selection is then modulated by these conditions using a binary mask applied to the meta-policy Q-values:

$$\pi_n = \arg\max_{\pi_n \in \Pi} (c_t \odot Q_{\phi_{\text{meta}}}(s_t, \pi_n)) \tag{6}$$

where $\odot$ denotes element-wise multiplication, $Q_{\phi_{\text{meta}}}(s_t, \cdot) \in \mathbb{R}^N$ is the vector of meta-policy Q-values for each interpretable condition, and Π is the set of available skills.

This way, we preserve interpretability at the symbolic level while enabling the agent to resolve ambiguous scenarios adaptively based on learned preferences. An overview of the different meta-policy variations is provided in Algorithm 1.

4 Experimental Evaluation

The goal of this work is to define an object-centric pipeline for agents that can solve complex environments while keeping the higher-level goals interpretable. We evaluate whether this goal was reached by answering the following research questions:

(Q1) Does LENS provide meaningful and learnable reward functions?
(Q2) Are LENS policies interpretable?
(Q3) How does LENS compare to other agents?
(Q4) Does LENS improve robustness to game variations?

4.1 Experimental Setup

Our experiments assume access to pre-extracted object representations and attributes. We therefore provide the agent with symbolic environment states directly, allowing us to remain agnostic to the specific object extraction pipeline.

We conduct experiments on a JAX-based reimplementation of the Atari Learning Environment (ALE) (Bellemare et al., 2013), specifically using the games *Kangaroo* and *Seaquest*, with object-centric representations similar to *OCatari* (Delfosse et al., 2024b). These games are selected because their gameplay can be naturally decomposed into low-level skills that can be combined to solve the overall task. In Kangaroo, the objective is to ascend through platforms while avoiding enemies and collecting berries. In Seaquest, the player navigates a submarine to rescue divers and return them to the surface, while avoiding hazards such as sharks and enemy submarines.

To evaluate the agent's generalization abilities, we test the agents that were trained on the original games in modified versions generated using HackAtari (Delfosse et al., 2024a). These modifications are designed to highlight the sensitivity of standard reinforcement learning agents to even minor changes in the environment, including simplifications. For example, removing enemies from Seaquest to reduce the task's complexity already leads to a substantial drop in the agent's performance. We show that the decomposition of tasks into semantically meaningful, skill-level components, coupled with a skill selection meta-policy, can enhance robustness to such distribution shifts.

We compare all three variations of LENS to the default, non-interpretable PQN (Gallici et al., 2024), and the actor-critic PPO (Schulman et al., 2017) as baselines. Additionally, we compare to HPQN, a hierarchical PQN variation that does not employ skill-specific rewards and has a purely neural meta-policy. All results (including baselines) are directly trained on object-centric inputs with 200M frames budget. We provide further implementation details in Appendix B.

Additional experiments that validate the applicability of LENS to other environments are available in Appendix C.

4.2 Results

LENS learns meaningful skills (Q1). We first evaluate whether the generated rewards enable efficient skill learning. Since each skill requires its own reward function, we leverage LLM's reasoning capabilities to generate them based on the game manual, skill definitions, and the object-centric state.

As shown in Figure 3, LENS successfully learns all target skills jointly using the generated rewards. Unlike PQN, which tends to overfit to a single skill that maximizes environment reward, LENS promotes balanced skill acquisition. This highlights a key advantage: LENS enables reward-aligned, generalizable behavior rather than environment reward exploitation.

LENS policies are interpretable and flexible (Q2).

A simple fixed meta-policy for the game Seaquest is visualized in Figure 4 (left side). Thanks to the abstraction of the raw observation to object-centric states and from raw actions to intelligible skills, the decision process is easily understandable to the user. For example, the agent selects the skill responsible for fighting enemies if and only if there is an enemy in close proximity to the player. We provide the meta-policies of all experiments in the appendix.

A flexible Seaquest agent trained with a soft meta-policy can be seen in action in Figure 5. The corresponding filtering rules are in Figure 4 (right side). At each point in time, we can comprehend

Figure 3: **LENS learns disentangled skills from off-policy data.** In Seaquest (top), the baseline methods mainly focus on shooting enemies, while the LENS approaches acquire the target skills more evenly. Similarly, in Kangaroo (bottom), the LENS approaches learn 'Move Up' and 'Collect Fruits' reliably, while the baselines focus mostly on 'Handle Threats'.

```python
def meta_policy(st):
    if enemy_close(st.enemies,
    st.player):
        fight_enemies()
    elif diver_available(state.divers):
        rescue_divers()
    elif oxygen_low(st.oxygen):
        surface()   # replenish_oxygen
    elif all_collected(st.divers):
        surface()   # deliver_divers
    rescue_divers()
```

```python
def meta_policy_rules(st):
    fight_enemies = False
    rescue_divers = True
    surface = False
    if enemy_close(st.enemies,
    st.player):
        fight_enemies = True
    if oxygen_low(st.oxygen):
        surface = True
    [fight_enemies, rescue_divers,
    surface]
```

Figure 4: **LENS policies and rules are clear and interpretable.** A fixed meta-policy (left) and similar filtering rules for the soft meta-policy (right) for Seaquest.

which skill was active and why it was possible to be chosen. Most of the time, only one of the skills can be chosen, because only one of the conditions is active. However, if multiple conditions resolve to true, we let the meta-policy decide intuitively with accordance to maximizing the environment reward.

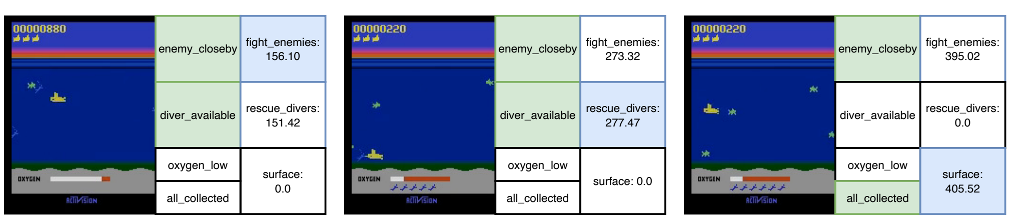

Figure 5: **LENS produces interpretable, but flexible high-level plans.** Left: the rules "enemy_closeby" and "diver_available" are true, the learned meta Q-function prefers to first fight the enemy. Middle: the same conditions are true, but in this case, the meta-policy prefers to rescue the diver. Right: There is an enemy close by and all divers are collected, in this case, returning to the surface is preferred.

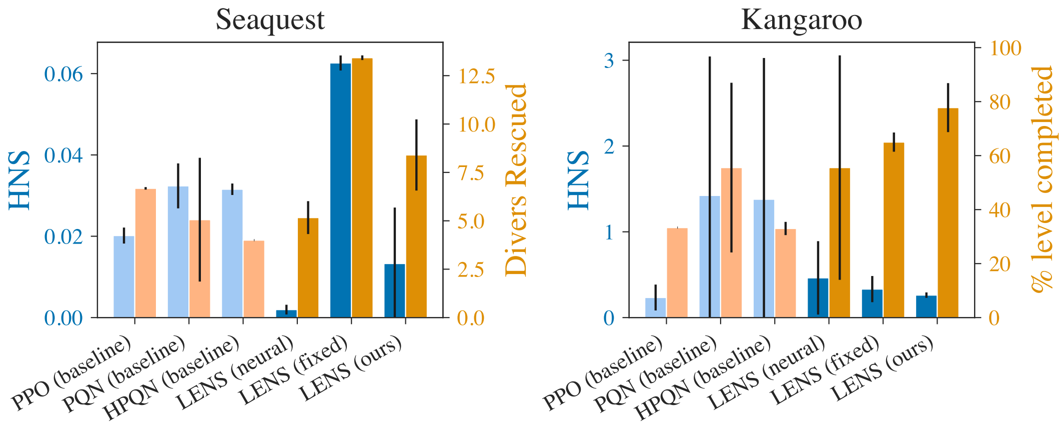

Figure 6: **LENS is better aligned.** Baselines score higher on HNS but are misaligned to the true game objectives, while LENS, particularly with symbolic and neuro-symbolic meta-policies, achieves better goal alignment.

LENS is better aligned than non-interpretable agents (Q3). We evaluate LENS against baselines using two metrics: Human Normalized Scores (HNS) (Badia et al., 2020) for environment reward performance, and Aligned Environment Goal Scores, which track progress toward goals defined in game manuals (e.g., divers rescued in Seaquest, level completion in Kangaroo). The latter aims to capture the overall alignment with the intended game objectives, which differ from maximizing the reward for these environments, where deep agents usually perform reward hacking (Shihab et al., 2025).

As shown in Figure 6, baselines often achieve higher HNS, but underperform on the actual games' main goal, demonstrating their reward hacking tendency. In contrast, LENS with symbolic and neuro-symbolic meta-policies better aligns with environment goals. The purely neural variant lags behind, likely due to the absence of the positive inductive biases that the LLM incorporated in the rule-based policies.

These results suggest that LENS mitigates reward misalignment by incorporating domain priors into decision-making.

LENS is robust to game variations (Q4). Most RL algorithms struggle to adapt to even minor variations in the environment Delfosse et al. (2025). Surprisingly, their performances drops includes settings that simplify the game for humans, such as removing dealy threats like enemies and thier projectiles in the games Seaquest or Kangaroo.

We assess whether LENS can agents generalize to such simplified variants by evaluating them on the unseen simplifications (described above). We evaluate the agents trained on the standard versions of Kangaroo and Seaquest.

Figure 7 presents the results. As expected, all baselines suffer substantial performance degradation under simplifications. In contrast, the symbolic and neuro-symbolic LENS variants exhibit smaller drops in performance, and in the case of Seaquest, even show improvements, particularly with respect to the aligned goal metrics. We mainly attribute this to the meta-policy, which does not activate skills related to enemy handling when no enemy is present in the state.

These findings demonstrate that LENS enhances robustness to minor distribution shifts, such as game simplifications.

5 Conclusion

Practitioners are often faced with the choice between high-performing but opaque neural agents and interpretable symbolic systems that lack scalability. LENS addresses this trade-off by combining

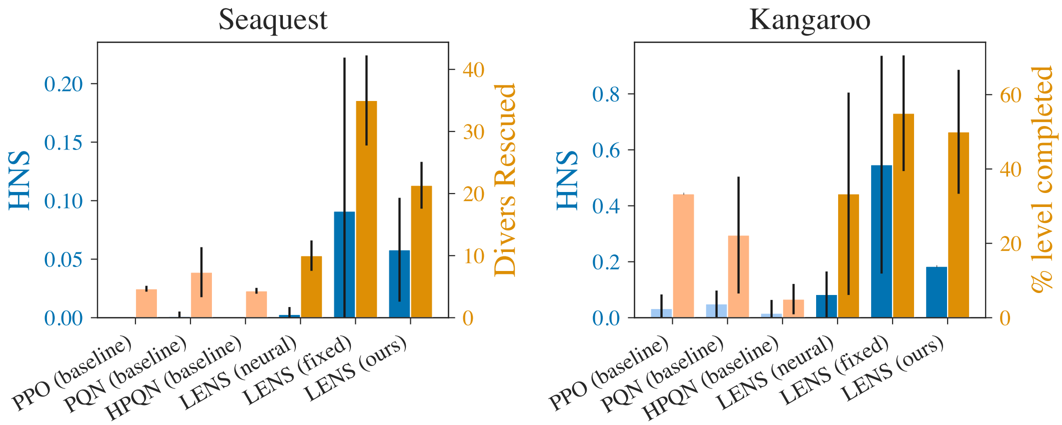

Figure 7: **LENS remains practical on simplified games.** Agents are trained on the original games and evaluated on unseen simplifications. While baseline performance drops significantly, symbolic and neuro-symbolic LENS exhibit improved robustness with smaller performance drops, and even gains in Seaquest, which we attribute to the meta-policy omitting unnecessary skills.

neural sub-policies with symbolic meta-policies in an object-centric hierarchical RL framework. This structure enables interpretable decision-making without sacrificing efficiency.

By leveraging LLMs for skill and reward design, and enhancing symbolic reasoning with a neuro-symbolic formulation, our method allows for flexible and transparent policy construction. The resulting agents demonstrate that interpretability and scalability need not be mutually exclusive.

Limitations and Future Work. While LENS enables transparent and modular policy learning, it still relies on accurate symbolic representations and skill definitions. In practice, we found LLMs to be effective at generating candidate skills, reward functions, and meta-policies based on the symbolic state definitions, but their outputs typically require human verification to ensure correctness and alignment with task intent.

Future directions include scaling to more complex, open-ended environments that demand broader skill discovery and task generalization. We also plan to explore tighter feedback loops between LLM-generated policies and environment-grounded corrections, moving toward more autonomous and interpretable agent design.

Broader Impact. This work contributes to efforts in making RL more interpretable and modular through symbolic structure and object-centric reasoning. By incorporating large language models to assist with skill and policy specification, LENS may reduce the expertise barrier for designing and understanding complex agent behaviors. However, as with any system involving language model outputs, care must be taken to validate the correctness and safety of generated rules and reward functions.

References

Kamal Acharya, Waleed Raza, Carlos M. J. M. Dourado Júnior, Alvaro Velasquez, and Houbing Herbert Song. Neurosymbolic Reinforcement Learning and Planning: A Survey. *IEEE Transactions on Artificial Intelligence*, 5(5):1939–1953, 2024.

Riad Akrour, Davide Tateo, and Jan Peters. Towards reinforcement learning of human readable policies. In *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases: The 1st Workshop on Deep Continuous-Discrete Machine Learning*, 2019.

Greg Anderson, Abhinav Verma, Işıl Dillig, and Swarat Chaudhuri. Neurosymbolic Reinforcement Learning with Formally Verified Exploration. In *Neural Information Processing Systems*, volume abs/2009.12612, 2020.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular Multitask Reinforcement Learning with Policy Sketches. In *International Conference on Machine Learning (ICML)*, pp. 166–175, 2017.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The Option-Critic Architecture. In *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1726–1734, 2017.

Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the Atari Human Benchmark. In *International Conference on Machine Learning (ICML)*, pp. 507–517, 2020.

Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13:341–379, 2003.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research (JAIR)*, 47:253–279, 2013.

Brett Daley and Christopher Amato. Reconciling λ-Returns with Experience Replay. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

Quentin Delfosse, Hikaru Shindo, Devendra Singh Dhami, and Kristian Kersting. Interpretable and Explainable Logical Policies via Neurally Guided Symbolic Abstraction. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2023a.

Quentin Delfosse, Wolfgang Stammer, Thomas Rothenbächer, Dwarak Vittal, and Kristian Kersting. *Boosting Object Representation Learning via Motion and Object Continuity*. Springer Nature Switzerland, 2023b.

Quentin Delfosse, Jannis Blüml, Bjarne Gregori, and Kristian Kersting. Hackatari: Atari learning environments for robust and continual reinforcement learning. *arXiv preprint arXiv:2406.03997*, 2024a.

Quentin Delfosse, Jannis Blüml, Bjarne Gregori, Sebastian Sztwiertnia, and Kristian Kersting. Ocatari: Object-Centric Atari 2600 Reinforcement Learning Environments. *Reinforcement Learning Conference (RLC)*, 1:400–449, 2024b.

Quentin Delfosse, Sebastian Sztwiertnia, Mark Rothermel, Wolfgang Stammer, and Kristian Kersting. Interpretable Concept Bottlenecks to Align Reinforcement Learning Agents. In *Conference on Neural Information Processing Systems (NeurIPS)*, volume abs/2401.05821, 2024c.

Quentin Delfosse, Jannis Blüml, Fabian Tatai, Théo Vincent, Bjarne Gregori, Elisabeth Dillies, Jan Peters, Constantin Rothkopf, and Kristian Kersting. Deep reinforcement learning agents are not even close to human intelligence. *arXiv preprint*, 2025.

Thomas G. Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research (JAIR)*, 13:227–303, 2000.

Matteo Gallici, Mattie Fellows, Benjamin Ellis, B. Pou, Ivan Masmitja, J. Foerster, and Mario Martin. Simplifying Deep Temporal Difference Learning. *arXiv.org*, abs/2407.04811, 2024.

Claire Glanois, Paul Weng, Matthieu Zimmer, Dong Li, Tianpei Yang, Jianye Hao, and Wulong Liu. A survey on interpretable reinforcement learning. *Machine Learning*, 113(8):5847–5890, 2024.

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

Rishi Hazra and Luc De Raedt. Deep Explainable Relational Reinforcement Learning: A Neuro-Symbolic Approach. In *European Conference on Machine Learning and Knowledge Discovery in Databases (PKDD)*, pp. 213–229, 2023.

Daniel Hein, Steffen Udluft, and Thomas A. Runkler. Interpretable policies for reinforcement learning by genetic programming. *Engineering Applications of Artificial Intelligence*, 76:158–169, 2018.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022. URL `http://jmlr.org/papers/v23/21-1342.html`.

Rahul Iyer, Yuezhang Li, Huao Li, Michael Lewis, Ramitha Sundar, and Katia Sycara. Transparency and Explanation in Deep Reinforcement Learning Neural Networks. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 144–150. ACM, 2018.

Steven James, Benjamin Rosman, and George Konidaris. Autonomous Learning of Object-Centric Abstractions for High-Level Planning. In *International Conference on Learning Representations (ICLR)*, 2022.

Zhengyao Jiang and Shan Luo. Neural Logic Reinforcement Learning. In *International Conference on Machine Learning (ICML)*, pp. 3110–3119, 2019.

Mu Jin, Zhihao Ma, Kebing Jin, Hankz Hankui Zhuo, Chen Chen, and Chao Yu. Creativity of AI: Automatic Symbolic Option Discovery for Facilitating Deep Reinforcement Learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 7042–7050, 2022.

Daniel Kahneman. *Thinking, fast and slow*. macmillan, 2011.

Hector Kohler, Quentin Delfosse, R. Akrour, Kristian Kersting, and Philippe Preux. Interpretable and Editable Programmatic Tree Policies for Reinforcement Learning. *arXiv.org*, abs/2405.14956, 2024.

Matteo Leonetti, Luca Iocchi, and Peter Stone. A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. *Artificial Intelligence*, 241:103–130, 2016.

Yuezhang Li, Katia P. Sycara, and Rahul Iyer. Object-sensitive Deep Reinforcement Learning. In *Global Conference on Artificial Intelligence (GCAI)*, pp. 20–35, 2017.

Amarildo Likmeta, Alberto Maria Metelli, Andrea Tirinzoni, Riccardo Giol, Marcello Restelli, and Danilo Romano. Combining reinforcement learning with rule-based controllers for transparent and general decision-making in autonomous driving. *Robotics and Autonomous Systems*, 131: 103568, 2020.

Zhixuan Lin, Yi-Fu Wu, Skand Vishwanath Peri, Weihao Sun, Gautam Singh, Fei Deng, Jindong Jiang, and Sungjin Ahn. Space: Unsupervised Object-Oriented Scene Representation via Spatial Attention and Decomposition. In *International Conference on Learning Representations (ICLR)*, 2020.

Chris Lu, Jakub Kuba, Alistair Letcher, Luke Metz, Christian Schroeder de Witt, and Jakob Foerster. Discovered policy optimisation. *Advances in Neural Information Processing Systems*, 35: 16455–16468, 2022.

Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew J. Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents (extended abstract). In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 2018.

Francis Maes, Raphael Fonteneau, Louis Wehenkel, and Damien Ernst. Policy Search in a Space of Simple Closed-form Formulas: Towards Interpretability of Reinforcement Learning. In *International Conference on Discovery Science (DS)*, pp. 37–51, 2012.

Michael Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Jackson, Samuel Coward, and Jakob Foerster. Craftax: A Lightning-Fast Benchmark for Open-Ended Reinforcement Learning. In *International Conference on Machine Learning*, volume abs/2402.16801, 2024.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Alizée Pace, Alex J. Chan, and Mihaela van der Schaar. Poetree: Interpretable Policy Learning with Adaptive Decision Trees. In *International Conference on Learning Representations (ICLR)*, 2022.

Ali Payani and F. Fekri. Inductive Logic Programming via Differentiable Deep Neural Logic Networks. *arXiv.org*, abs/1906.03523, 2019.

Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence*, 1(5):206–215, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv.org*, abs/1707.06347, 2017.

Mohit Sharma, Jacky Liang, Jialiang Zhao, Alex LaGrassa, and Oliver Kroemer. Learning to Compose Hierarchical Object-Centric Controllers for Robotic Manipulation. In *Conference on Robot Learning (CoRL)*, pp. 822–844, 2020.

Ibne Farabi Shihab, Sanjeda Akter, and Anuj Sharma. Detecting and mitigating reward hacking in reinforcement learning systems: A comprehensive empirical study. *arXiv preprint*, 2025.

Andrew Silva, Taylor W. Killian, I. D. Rodriguez, Sung-Hyun Son, and M. Gombolay. Optimization Methods for Interpretable Differentiable Decision Trees in Reinforcement Learning. *arXiv: Learning*, 2019.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

Abhinav Verma, Hoang Minh Le, Yisong Yue, and Swarat Chaudhuri. Imitation-Projected Programmatic Reinforcement Learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, pp. 15726–15737, 2019.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal Networks for Hierarchical Reinforcement Learning. In *International Conference on Machine Learning (ICML)*, pp. 3540–3549, 2017.

Fangkai Yang, Daoming Lyu, Bo Liu, and Steven Gustafson. Peorl: Integrating Symbolic Planning and Hierarchical Reinforcement Learning for Robust Decision-Making. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 4860–4866, 2018.

Zihan Ye, O. Arenz, and Kristian Kersting. Learning from Less: Guiding Deep Reinforcement Learning with Differentiable Symbolic Planning. *arXiv*, 2025.

Tom Zahavy, Nir Ben-Zrihem, and Shie Mannor. Graying the black box: Understanding DQNs. In *International Conference on Machine Learning (ICML)*, pp. 1899–1908, 2016.

Chiyuan Zhang, O. Vinyals, R. Munos, and Samy Bengio. A Study on Overfitting in Deep Reinforcement Learning. *arXiv.org*, abs/1804.06893, 2018.

A Hierarchical PQN Algorithm

The complete algorithm for hierarchical PQN with a neural meta-policy is provided in Algorithm 2.

Algorithm 2 Hierarchical PQN

Require: Update period U, number of parallel environments E, number of skills N, exploration probability ϵ

Ensure: Learned Q-network parameters $\{\phi_n\}_{n=1}^{N}$, ϕ_{meta}

Initialize Q-network parameters $\{\phi_n\}_{n=1}^{N}$, ϕ_{meta}
Sample initial states $s_0^e \sim P_0$ for $e \in \{0, \ldots, E-1\}$
$t \leftarrow 0$
for each episode **do**
  for all $e \in \{0, \ldots, E-1\}$ **in parallel do**
    Sample skill $\pi_t^e \sim \pi_{\text{meta}}$
    With probability ϵ: $a_t^e \sim$ Unif, else $a_t^e \sim \pi_t^e$
    Sample rewards $r_t^e \sim P_R(s_t^e, a_t^e)$, skill rewards $r_t^{e,n} \sim P_{R,n}(s_t^e, a_t^e)$ for all n
    Sample next state $s_{t+1}^e \sim P_S(s_t^e, a_t^e)$
    $t \leftarrow t+1$
  end for
  if $t \bmod U = 0$ **then**
    Compute meta λ-returns $R_{\lambda,t-1}^e$ to $R_{\lambda,t-U}^e$ for all e
    Compute skill λ-returns $R_{\lambda,t-1}^{e,n}$ to $R_{\lambda,t-U}^{e,n}$ for all e, n
    for number of epochs **do**
      for number of minibatches **do**
        Sample minibatch B of size $b \leq EU$ from $\{(t-U, 0), \ldots, (t-1, E-1)\}$
        Update meta:

$$\phi_{\text{meta}} \leftarrow \phi_{\text{meta}} + \frac{\alpha_t}{2b} \nabla_{\phi_{\text{meta}}} \sum_{(j,\tau) \in B} \left(R_{\lambda,\tau}^{j} - Q_{\phi_{\text{meta}}}(s_\tau^j) \right)^2$$

        Update skills:

$$\phi_n \leftarrow \phi_n + \frac{\alpha_t}{2b} \nabla_{\phi_n} \sum_{(j,\tau) \in B} \left(R_{\lambda,\tau}^{j,n} - Q_{\phi_n}(s_\tau^{j,n}) \right)^2, \quad \forall n \in \{1, \ldots, N\}$$

      end for
    end for
  end if
end for

B Implementation Details

Setup. We base our implementations on CleanRL (Huang et al., 2022) and PureJaxRL (Lu et al., 2022), adapting them to object-centric inputs by replacing convolutional encoders with lightweight MLPs for feature extraction. Hyperparameters are listed in Table 1 and remain largely consistent with the original implementations, except for an increased number of parallel environments enabled by the efficiency of JAX-based code.

The exploration parameter ϵ was selected via a brief hyperparameter sweep in the range $[1, 0.001]$, using final test return as the selection criterion.

Each experiment is run with three random seeds (0, 1, 2) to ensure reproducibility. Reported plots include the corresponding standard deviation.

All experiments were conducted on a single NVIDIA Tesla V100-SXM3-32GB-H GPU on an NVIDIA DGX Server (Version 5.1.0) with CUDA 12.4.

Table 1: Hyperparameters

Parameter	PQN(-based)	PPO
Total Timesteps	5×10^7	5×10^7
Num Environments	1024	128
Num Steps per Update	128	128
Learning Rate	1.0×10^{-4}	2.5×10^{-4}
Max Grad Norm	10	0.5
Discount Factor (γ)	0.99	0.99
GAE Lambda (λ)	0.65	0.95
Num Epochs	5	2
Num Minibatches	128	4
Hidden Size	64	–
Num Layers	3	–
Normalization	Layer Norm	–
Clip ϵ	–	0.2
Entropy Coef	–	0.01
Value Function Coef	–	0.5
Anneal LR	False	True
ϵ-Start/End/Decay	1.0 / 0.1 / 0.3	–
Meta ϵ-Start/End/Decay	0.1 / 0.001 / 0.1	–

B.1 Evaluation Metrics

We empirically evaluate agent performance using three metrics: (1) **Skill Returns** to test whether the skills were learned successfully, (2) **Human-Normalized Score** (HNS) for absolute performance relative to human and random baselines and (3) **Aligned Environment Goal Scores** that measure performance based on the main goals described in the game's manual.

Human Normalized Score. Human-Normalized Score standardizes agent performance across Atari environments by accounting for differences in reward scales (Mnih et al., 2015; Machado et al., 2018). Given the average agent score A, human score H, and random score R, HNS is defined as:

$$\text{HNS} = \frac{A - R}{|H - R|}$$

A value of 1.0 indicates human-level performance, values greater than 1.0 indicate superhuman performance, and values below 0 denote sub-random behavior. We adopt the human and random baselines from Badia et al. (2020), derived from professional human play.

Aligned Environment Goal Scores. Environment reward signals may not always align with the intended task objectives and can be susceptible to reward hacking. In such cases, agents may learn high-reward behaviors that are non-intuitive and deviate from human-like solutions. To better capture progress toward the actual environment goals, we define two aligned goal-based metrics grounded in the objectives stated in the game manuals.

For *Seaquest*, the goal is to rescue as many divers as possible; for *Kangaroo*, it is to help the mother kangaroo reach and rescue her baby, located on the topmost platform. Accordingly, we track the number of divers retrieved and the number of platforms reached, respectively.

C Additional Results

To assess the generality of LENS, we extend our evaluation to three additional Atari environments: *Pong*, *Breakout*, and *Freeway*. Skill learning curves for these environments are shown in Figure 8, while comparisons to baseline agents and ablations—evaluated via human-normalized scores—are presented in Figure 9. Note that these games are generally less complex than *Kangaroo* and *Seaquest*, and the learned skills are not strictly necessary to achieve the environment goals. In particular, the skills "Move Up" and "Avoid Crash" in *Freeway* largely correspond to atomic actions such as

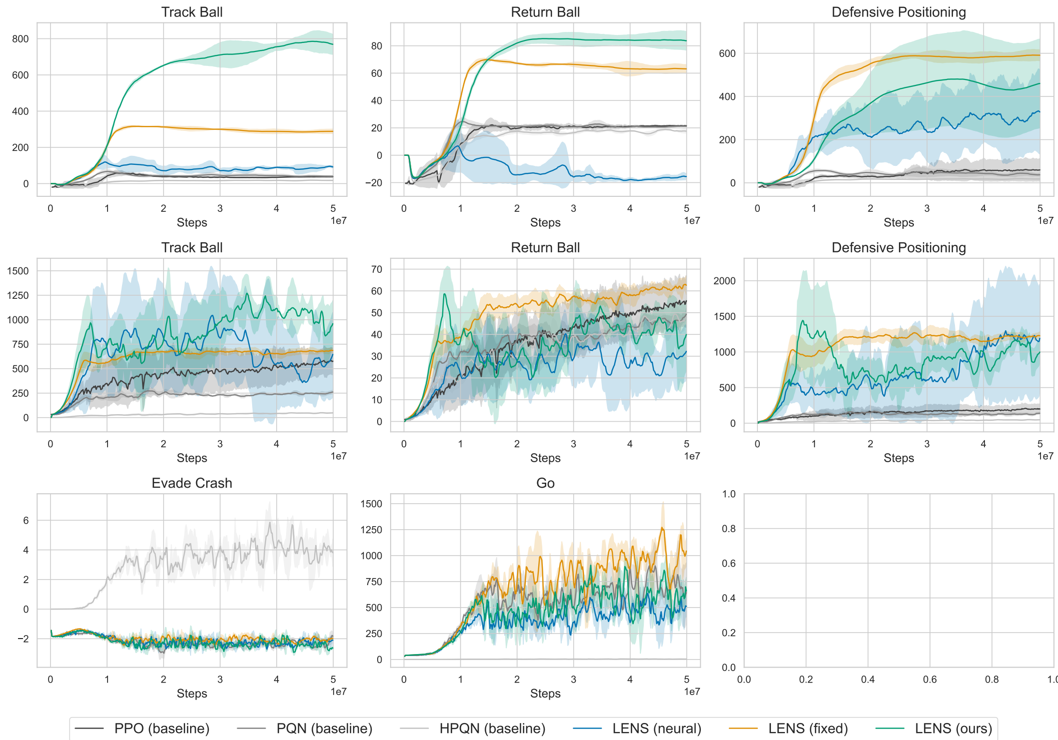

Figure 8: LENS successfully learns skills in *Pong* (top), *Breakout* (middle), and *Freeway* (bottom).

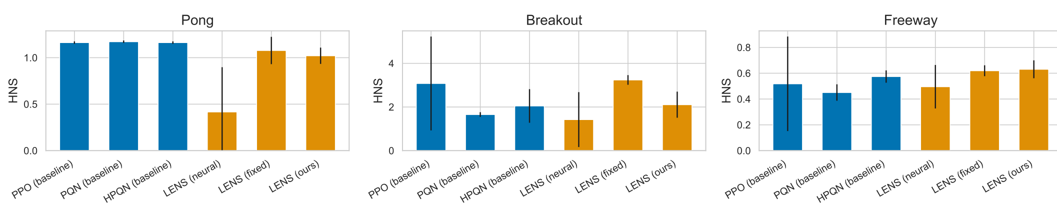

Figure 9: LENS achieves performance comparable to baseline methods across the three evaluated environments.

`forward` or `noop`. As such, a simple fixed meta-policy operating directly on primitive actions could suffice for solving this task.

We further validate the applicability of LENS to more complex domains by evaluating on *Crafter* (Hafner, 2021; Matthews et al., 2024). Using the game manual and state description, we query an LLM for essential skills. LENS is then trained explicitly on these skills and compared to the PQN baseline. Results are presented in Figure 10.

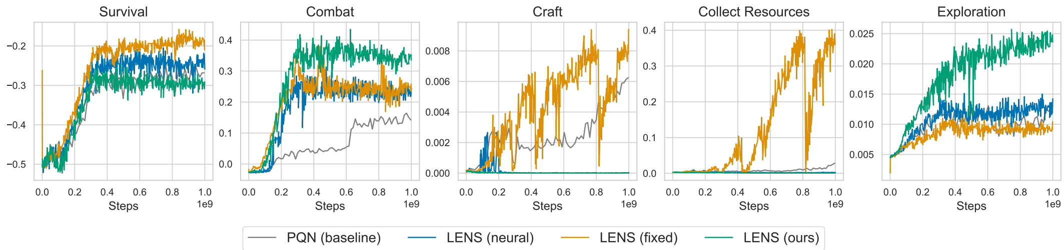

Figure 10: LENS succesfully learns LLM-identified skills in *Crafter*.

D LLM Interaction

We outline the procedure for leveraging an LLM to generate task-relevant skills, associated reward functions, and a fixed meta-policy rule set. The LLM is conditioned on the game's original manual and structured type information describing the object-centric observations available to the agent.

Initially, the LLM is queried for a set of skills and corresponding reward functions. These outputs can be manually refined before querying the LLM for a meta-policy function that selects which skill to execute. Prior prompts and responses are retained to maintain conversational context, consistent with standard chat behavior. Prompt details are provided in Figure 11.

Unedited responses from GPT-4o (via chatgpt.com on 21-07-2025) for the games Kangaroo and Seaquest are included in Figure 12–Figure 15. Final implementations were modified to align with our framework constraints (e.g., JAX compatibility, indexing conventions).

You are a RL expert and develop a hierarchical agent to play the atari game `<GameName>`. Below, I have provided a detailed description of the game. Your task is to come up with a short list of essential skills (3-4) that are needed to solve the game.
Game Description: — `<Game manual>` —
Think about what the main goal of this game is, then come up with a list of the required skills to solve the game and provide it in the output. Finally, create a pseudo reward function for each of these skills that a RL agent can use to learn the skills.
The reward functions are called at each step in the environment and retrieve the current and previous symbolic observation of the game. Here is the type definition for the observation:
`<ObservationTypeDefinition>`
Each skill and it's reward function should be as simple as possible that achieves the wanted behavior. Provide: List of skills and description, pseudo-code reward functions for each skill.

You are now creating a function that decides at each time step in the game, which skill to choose. The skills are: `<skills from previous prompt>` The function again has access to the current and previous observation.
Think about at what point in the game you want to use which skill, then provide the function.

Figure 11: Prompts to generate a skill-list and corresponding reward functions (top) and meta-policy function (bottom).

Essential Skills for Solving *Kangaroo* (Atari 2600)

A hierarchical RL agent for *Kangaroo* can be built upon the following four primitive skills:

1. Obstacle Avoidance
Description: Avoid contact with apples or monkeys via dodging or punching.
Reward Function:

```
def reward_obstacle_avoidance(obs, prev_obs):
    collided = any(np.array_equal(obs.player_x, x) and
                   np.array_equal(obs.player_y, y)
                   for (x, y) in obs.apple_positions + obs.monkey_positions)
    return -1.0 if collided else 0.0
```

2. Vertical Navigation
Description: Climb ladders to ascend levels and progress.
Reward Function:

```
def reward_vertical_navigation(obs, prev_obs):
    dy = prev_obs.player_y - obs.player_y
    return 0.1 if dy > 0 else 0.0
```

3. Fruit Collection
Description: Collect fruits and optionally ring a bell to regenerate them.
Reward Function:

```
def reward_fruit_collection(obs, prev_obs):
    reward = 0.0
    if len(obs.fruit_positions) < len(prev_obs.fruit_positions):
        reward += 0.5
    if np.array_equal(obs.player_x, obs.bell_position[0]) and \
       np.array_equal(obs.player_y, obs.bell_position[1]):
        reward += 0.2
    return reward
```

4. Goal Reaching (Child Rescue)
Description: Reach the child's location to complete the level.
Reward Function:

```
def reward_reach_child(obs, prev_obs):
    return 1.0 if np.array_equal(obs.player_x, obs.child_position[0]) and \
                  np.array_equal(obs.player_y, obs.child_position[1]) else 0.0
```

These modular rewards enable a high-level controller to invoke appropriate behaviors depending on context.

Figure 12: Generated reward functions for Kangaroo by GPT4-O

Kangaroo Skill Selection Function

To select appropriate skills, the agent follows a fixed priority:

(1) **Obstacle Avoidance**: If any apple or monkey is in close proximity.

(2) **Goal Reaching**: If the player is near the child on the top platform.

(3) **Fruit Collection**: If fruit is visible and reachable.

(4) **Vertical Navigation**: Default behavior to explore or ascend.

Skill Selector Function:

```
def select_skill(obs, prev_obs):
    player_pos = np.array([obs.player_x, obs.player_y])

    def in_danger():
        danger_entities = obs.apple_positions + obs.monkey_positions
        return any(np.linalg.norm(player_pos - np.array([x, y])) <= 1.0
                   for (x, y) in danger_entities)

    def near_child():
        child_pos = np.array([obs.child_position[0], obs.child_position[1]])
        return np.linalg.norm(player_pos - child_pos) <= 1.5

    def fruit_visible():
        return any(abs(y - obs.player_y) <= 1 and
                   abs(x - obs.player_x) <= 3
                   for (x, y) in obs.fruit_positions)

    if in_danger():
        return obstacle_avoidance
    elif near_child():
        return goal_reaching
    elif fruit_visible():
        return fruit_collection
    else:
        return vertical_navigation
```

This rule-based selection policy balances safety, progression, and reward acquisition, enabling effective hierarchical control.

Figure 13: Generated fixed meta-policy function for Kangaroo by GPT4-O

Essential Skills for Solving *Seaquest* (Atari 2600)

To solve *Seaquest* effectively, a hierarchical RL agent must master the following four core skills:

1. Avoid Obstacles (Survival Navigation)
Description: Avoid collisions with sharks, enemy submarines, missiles, and the patrol sub, both underwater and when surfacing.
Reward Function (pseudo-code):

```
def avoid_obstacles_reward(prev_obs, curr_obs):
    if player_collided(curr_obs):
        return -1.0
    return 0.0

def player_collided(obs):
    return check_collision_with_sharks(obs) or \
           check_collision_with_subs(obs) or \
           check_collision_with_missiles(obs) or \
           check_collision_with_surface_sub(obs)
```

2. Collect Divers
Description: Navigate to and collect divers to accumulate score and progress.
Reward Function (pseudo-code):

```
def collect_divers_reward(prev_obs, curr_obs):
    prev_count = count_nearby_divers(prev_obs)
    curr_count = count_nearby_divers(curr_obs)
    return float(prev_count > curr_count) * 1.0

def count_nearby_divers(obs):
    return sum(is_near(obs.player_x, obs.player_y, dx, dy)
               for dx, dy in obs.diver_positions)
```

3. Surface Strategically
Description: Surface safely when carrying six divers or when oxygen is low, to drop off divers and replenish oxygen.
Reward Function (pseudo-code):

```
def surface_reward(prev_obs, curr_obs):
    surfaced_now = prev_obs.player_y > 0 and curr_obs.player_y == 0
    has_6_divers = num_divers_collected(prev_obs) == 6
    low_oxygen = prev_obs.oxygen < OXYGEN_THRESHOLD
    if surfaced_now and (has_6_divers or low_oxygen):
        return 2.0 if has_6_divers else 0.5
    return 0.0

def num_divers_collected(obs):
    return 6 - sum(is_valid(dx, dy) for dx, dy in obs.diver_positions)
```

4. Eliminate Threats (Combat)
Description: Fire missiles to destroy sharks and enemy submarines, clearing paths and increasing score.
Reward Function (pseudo-code):

```
def eliminate_threats_reward(prev_obs, curr_obs):
    num_enemies_prev = count_enemies(prev_obs)
    num_enemies_curr = count_enemies(curr_obs)
    return float(num_enemies_curr < num_enemies_prev) * 0.5

def count_enemies(obs):
    return sum(is_valid(dx, dy) for dx, dy in obs.shark_positions) + \
           sum(is_valid(dx, dy) for dx, dy in obs.sub_positions)
```


Figure 14: Generated reward functions for Seaquest by GPT4-O

Seaquest Skill Selection Function

Here is a skill selection function that chooses the appropriate control skill at each time step based on game context.

Skill Selection Function (pseudo-code):

```
def select_skill(prev_obs, curr_obs):
    # Skill names
    AVOID = "obstacle_avoidance"
    COLLECT = "collect_divers"
    SURFACE = "surface_strategically"
    COMBAT = "eliminate_threats"

    # === Utility Functions ===
    def is_threat_near(obs: SeaquestObservation) -> bool:
        return check_collision_with_sharks(obs) or \
               check_collision_with_subs(obs) or \
               check_collision_with_missiles(obs) or \
               check_collision_with_surface_sub(obs)

    def num_divers_collected(obs: SeaquestObservation) -> int:
        return 6 - sum(is_valid(x, y) for x, y in obs.diver_positions)

    def low_oxygen(obs: SeaquestObservation) -> bool:
        return obs.oxygen < 20 # adjustable threshold

    def at_surface(obs: SeaquestObservation) -> bool:
        return obs.player_y == 0

    # === Skill Selection Logic ===
    if is_threat_near(curr_obs):
        return AVOID

    if (num_divers_collected(curr_obs) == 6 or low_oxygen(curr_obs)) and \
        not at_surface(curr_obs):
        return SURFACE

    if has_nearby_enemies(curr_obs): # e.g. visible on screen in front
        return COMBAT

    return COLLECT
```

Explanation of Logic:

- **Obstacle Avoidance** is triggered immediately when any nearby threat is detected.
- **Surface Strategically** is invoked if the agent has collected 6 divers or is low on oxygen, and is not already at the surface.
- **Eliminate Threats** is used when enemies are visible but not an immediate threat.
- **Collect Divers** is the default skill when no critical conditions are active.

This rule-based selection mechanism allows a high-level controller to choose among primitive skills in a safety-first manner while still enabling reward-driven exploration and progress.

Figure 15: Generated fixed meta-policy for Seaquest by GPT4-O