

Eyeing the Big Play, Not Just the Moves

Advancing the Interpretability of RL Agents through Temporal Abstraction via Options

Master thesis by Mark Rothermel

Date of submission: October 31, 2023

1. Review: Quentin Delfosse
2. Review: Prof. Dr. Kristian Kersting
Darmstadt



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Informatik

Computer Science
Department

Artificial Intelligence and
Machine Learning

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

Hiermit erkläre ich, Mark Rothermel, 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, 31. Oktober 2023



M. Rothermel

Contents

1. Introduction	7
2. Background	12
2.1. Foundations of Reinforcement Learning (RL)	12
2.1.1. (Semi-)Markov Decision Process	14
2.1.2. Training Agents: The Actor-Critic Framework	15
2.1.3. Proximal Policy Optimization (PPO) as the State-of-the-Art Training Algorithm	16
2.2. Explainable RL: How to Make Agents Understandable and Trustworthy?	16
2.3. Temporal Abstraction via High-Level Actions, a.k.a. Options	17
2.4. NUDGE: A Logic Framework for Learning Symbolic Policies	20
3. Method	22
3.1. How to: Improve Policy Interpretability via Options	22
3.2. Combining PPO with the Option-Critic	24
3.3. Generalized Advantage Estimation for Options	26
3.4. The Options Training Algorithm	28
3.5. Expectations and Final Theoretical Remarks	30
4. Experiments and Results	33
4.1. Introducing the MEETINGROOM Environment	33
4.2. Experimental Setup	36
4.3. Results	37
4.3.1. ω -NUDGE Performs Better than NUDGE	40
4.3.2. ω -NUDGE Has Better High-Level Interpretability than NUDGE	40
4.3.3. Even Without Pretraining ω -NUDGE Learns Faster than Neural Hierarchies	41
4.3.4. ω -NUDGE Finds More Meaningful Options than Neural Hierarchies	41
4.3.5. Termination Function Training is Difficult	43

4.4. Side Contributions	44
5. Related Work	46
5.1. Hierarchical RL with Logic	46
5.2. Symbolic Planning with Hierarchies	47
5.3. Interpretable Neural HRL	48
6. Discussion	49
6.1. ω -NUDGE versus NUDGE versus Neural Hierarchies	49
6.2. Limitations	51
6.3. Future Directions	52
6.4. Ethical Implications	53
7. Conclusion	55
Acronyms	56
Bibliography	58
A. Appendix	63
A.1. Pitfalls of the GAEO	63
A.2. Learning a Reasonable Termination Function is Non-Trivial	65
A.3. Implementation	65
A.3.1. Repository Structure	65
A.3.2. How to Use	67
A.4. Graphical User Interface for the RAM Extraction Method in OCArari	68
A.5. State Abstraction with OCArari and SCoBots	69
A.5.1. Seeing the Forest, not Just the Trees	69

Abstract

In contexts where Reinforcement Learning (RL) agents are contemplated for critical applications such as medical diagnostics or autonomous vehicular control, model interpretability is imperative. However, the most successful agents are typically the least interpretable ones. The advent of symbolic policies, exemplified by NUDGE (Delfosse et al., 2023b), endeavors to establish inherently interpretable models. Interpretability of these models holds true for rudimentary environments with a limited action space and succinct episodes. But, it does not for complex RL tasks like those found in the real world where logical rule sets inevitably grow inscrutably large.

This thesis advocates the incorporation of temporal abstraction into symbolic policies to maintain interpretability even within complex tasks. ω -NUDGE is proposed which is a neuro-symbolic, hierarchical policy extension of NUDGE, building up on the options framework and the option-critic. The MEETINGROOM environment is introduced to evaluate hierarchical agents like ω -NUDGE. Empirical results within MEETINGROOM show that ω -NUDGE outperforms NUDGE significantly in both performance and interpretability. The code was made publicly available on GitHub¹.

¹<https://github.com/MaggiR/logic-options>

Acknowledgements

I want to thank Quentin Delfosse for the exceptional supervision of this Master's thesis! The time he invested, providing countless tips and valuable advice for my future as a PhD candidate, is deeply appreciated. It's a pleasure to continue working with him as cooperating colleagues following this thesis.

Many thanks go to my friends and my family who supported me during my eight years of studies. Special thanks go to my boyfriend Jannis Rösner, who beared my physical and mental absence dedicated to this thesis, especially during the last couple of weeks.

Furthermore, my gratitude also goes to the student council of the Computer Science Department at this university. As a member there, I spent countless days of engagement and activity that filled me with happiness. I also found many friends there, including Jannis.

Finally, I also thank the *Friedrich-Naumann Stiftung für die Freiheit* (FNF) for the scholarship. This financial support made overall student life significantly easier.

1. Introduction

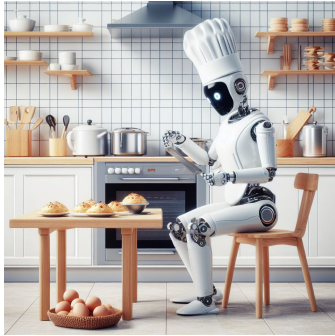
Reinforcement Learning (RL) is the field of Machine Learning (ML) that deals with sequential decision-making. RL practitioners showed remarkable results on games like DotA 2 (Berner et al., 2019) and Go (Silver et al., 2016), but also in real-world applications like robotics (Mahmood et al., 2018), autonomous driving (You et al., 2017), recommender systems (Zhang et al., 2019), and fleet management (Lin et al., 2018).

Most of these achievements were accomplished through the usage of deep Neural Networks (NNs), a standard model architecture in AI. NNs proved to have outstanding flexibility when it comes to learning high-dimensional, complex functions. Nevertheless, NNs incorporate millions to even trillions of parameters, making their inner workings and reasoning obscure humans. Model interpretability is a requirement in high-stakes applications where failures incur high cost, like in medicine or public traffic. The reason is that interpretable models are better verifiable and, consequently, more trustworthy. In decision processes with "significant" effect, even European law requires the used AI models to be fully explainable. (Goodman and Flaxman, 2016)

Interpretability in RL is an unsolved problem. (Casper et al., 2023) RL models can impressively solve tasks, but there is still a long way to go to ultimately train *and* understand well performing real-world RL models. The rapidly evolving field of Explainable AI (XAI) deals with the development of *intrinsically* understandable AI models or methods that make black-box models transparent *post-hoc*. In RL, symbolic reasoning methods were developed to support intrinsic model interpretability. One of them is a recently proposed framework called. Neurally gUided Differentiable loGic policiEs (NUDGE) (Delfosse et al., 2023b) It achieves model interpretability through the usage of First-Order Logic (FOL) and differentiable forward reasoning. Such symbolic frameworks not only enable model interpretability but also, for example, promote objective robustness (Koch et al., 2021).

Interpretability and model size follow an anti-correlation: Large models are harder to understand than smaller ones, even when consisting of inherently interpretable components. The reason is that humans have limited capacity to digest information—if confronted with

thousands of FOL, a human wouldn't be able to look through. Unfortunately, large models are a typical result of learning real-world tasks which is due to the task's complexity coming from, e.g., high dimensionality and long action sequences.



Created with DALL-E 3.

For example, think of a robot that learned to bake a cake. The robot relies on its servo joints to grasp and move objects, to manipulate them, etc. For a human, baking a cake is a rather simple task and consists of algorithmically following a baking recipe—without the inefficient thinking of how to exactly move their fingers in order to, e.g., crack an egg. In other words, humans learned a *temporal abstraction* of their actions they perform: Pouring milk into the bowl is an abstraction of getting the milk package out of the fridge, opening it and turning it so that the milk can flow. Opening the package, in turn, is an abstraction of moving the hand to the lid and performing a movement that rotates the lid, and so on. In essence, temporal abstraction is a natural concept

that makes processing information in the world more efficient. In fact, in very high-dimensional problems, RL researchers argue that temporal abstraction is vital. (Konidaris and Barto, 2009; Shu et al., 2017) However, most RL methods do not incorporate temporal abstraction. If we were to ask the (preferably logic) cake-baking robot what recipe it follows, it would either return an extremely long list of servo joint commands or an uninterpretable large set of logic rules.

Temporal abstraction in RL can be achieved through a hierarchical policy structure as depicted in Figure 1.1. The idea comes from the *Hierarchical Reinforcement Learning (HRL)* domain which was originally introduced to efficiently solve problems with very long decision sequences and sparse feedback. (Hutsebaut-Buysse et al., 2022) HRL models consist of a collection of specialized sub-policies, structured in a hierarchy, with a meta policy at the top. See also Figure 1.1. The meta policy π_{meta} determines the current subgoal and selects the proper sub-policy π_i to solve that goal, making use of the divide-and-conquer principle. The hierarchy of the model just needs to be chosen according to the *goal hierarchy* that underlies in the actual problem. For instance, baking a cake involves subgoals like having all the necessary ingredients gathered or like having added a sweet cream topping. In fact, a hierarchical subgoal structure is a common property shared among real-world problems. (Jong et al., 2008; Hutsebaut-Buysse et al., 2022) Each such subgoal can be addressed through a dedicated sub-policy. Sub-policies execute a sequence of lower-level actions until the subgoal is reached, cf. Figure 1.2. In a sense, a

sub-policy unifies the sequence of tasks it performs into one "high-level" task. In other words, a sub-policy is an element of temporal abstraction.

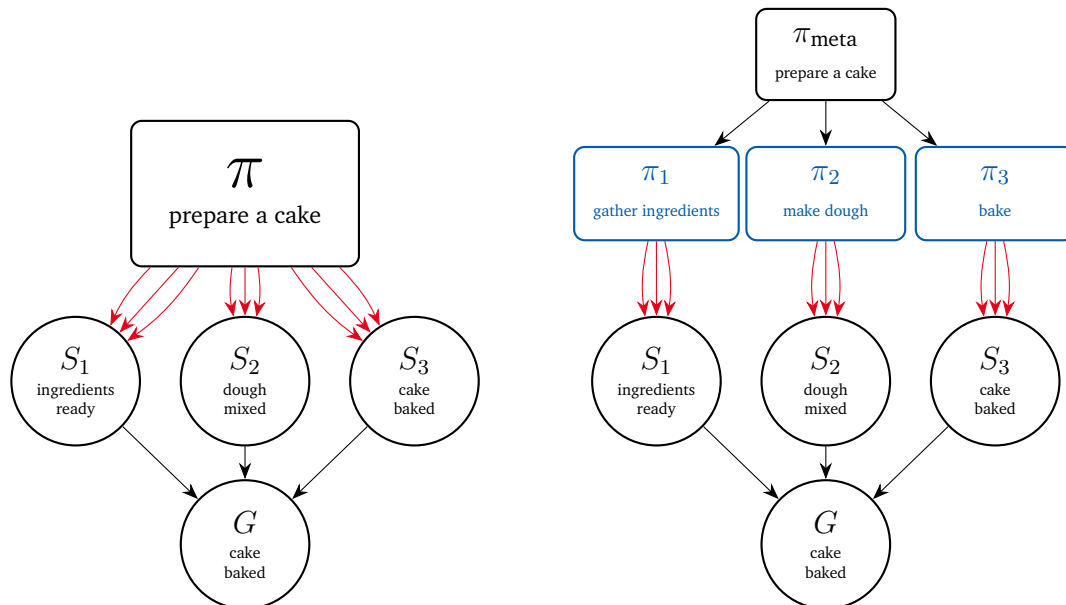


Figure 1.1.: The benefit of temporal abstraction, here with the cake baking example. The goal G of baking a cake has several subgoals S_i . In the typical RL case (left), a single atomic policy π deals with each subgoal S_i via **actions**. By introducing specialized **lower-level policies** π_i , the resulting higher-level policy π_{meta} now only needs to choose the right sub-policy for the right subgoal—making π_{meta} significantly simpler compared to π .

Interpretability of NUDGE is evident on simple tasks like in the GETOUT environment. (Delfosse et al., 2023b) However, it is reasonable to expect that, just as in the cake-baking robot example, the size of the policy rule set explodes and, thus, is inscrutable to humans. The core idea of this thesis lies in the application of hierarchical sub-policies to make NUDGE better interpretable. Though HRL was already identified as a promising direction for better interpretability, it is yet an under-explored direction with a lot of room for further examination. (Hutsebaut-Buysse et al., 2022)

Usually, we are not interested in the low-level action movements but rather in the high-level strategy—the big play—of how to solve a problem. Just like humans rationalize about where to spend the next vacation and not how to move the mouse at the computer in order

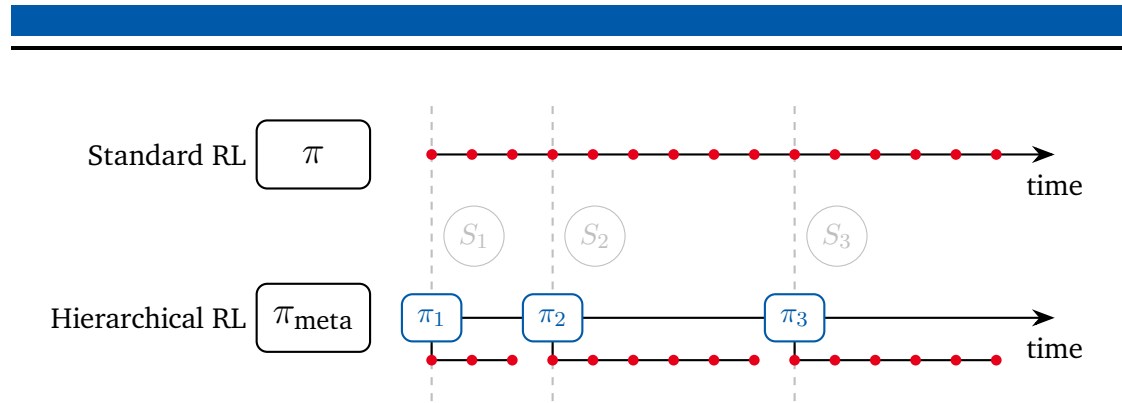


Figure 1.2.: The example from Figure 1.1 on a time dimension, showing that π needs to take an action at every time step while π_{meta} can focus on subgoal changes.

to reserve a hotel. Therefore, it is not detrimental to use black-box sub-policies like NNs in the hierarchy. Allowing them in the architecture, we can combine the strength of neural flexibility with symbolic interpretability. Neural sub-policies also overcome two of NUDGE's shortcomings: The generally limited expressiveness and the inapplicability to continuous action spaces. Moreover, the way how humans learn to interact with their environment can serve as another role model. More specifically, humans learn incrementally: first, as a baby, how to move fingers, hands and feet, touch and grasp objects, then, as a child, how to move objects and later how to use objects as tools for a higher purpose. This analogy invites to exploit the composability of hierarchical policies for curriculum learning, i.e., pretraining the sub-policies on specific sub-tasks before assembling them together into the hierarchy.

Putting the aforementioned ideas into practice, this work introduces a hierarchical extension of NUDGE, called ω -NUDGE. It builds upon the options framework (Sutton et al., 1999), a very popular specification of hierarchical sub-policies. Furthermore, the options-critic architecture (Bacon et al., 2016) is used as the model's base structure where NUDGE serves as the meta policy. Overall, this work seeks to answer the question:

Can we use options to make NUDGE more interpretable, even when we use black-box models as options?

On a sidenote, the literature lacks a unified definition of interpretability and explainability. (Molnar et al., 2020) Without going into the details, this work regards interpretability as the intrinsic property of a model to be understandable to the user by design. Explainability, on the other hand, is the model's ability to present convincing and understandable *post-hoc* explanations of its rationale or decisions to a human. (Arrieta et al., 2019)

Overall, this work makes the following main contributions:

- A novel, neuro-symbolic RL model called ω -NUDGE is proposed that extends Neurally gUided Differentiable loGic policiEs (NUDGE) (Delfosse et al., 2023b) by a hierarchy of sub-policies.
- Empirical evidence is delivered that shows the improved interpretability of ω -NUDGE over NUDGE.
- A new environment called MEETINGROOM is introduced as a proof-of-concept playground for hierarchical models.
- The options framework is implemented, including an option-version of the state-of-the-art Proximal Policy Optimization (PPO) (Schulman et al., 2017) learning algorithm. The code was made publicly available on GitHub, see also Appendix A.3.

2. Background

We begin with the theoretical introduction of the concepts used in this work. Section 2.1 reiterates the fundamental RL notions and defines commonly used symbols, substantially following the book of Sutton and Barto (2017). Well-versed readers can skip the first two sections and begin with the section about options, Section 2.3, followed by a closer look at NUDGE.

2.1. Foundations of Reinforcement Learning (RL)

We start off with an example. Imagine a talented pizza chef opening his first own restaurant. He has a family he provides for and the loan for purchasing the establishment's premises needs to be paid off. Therefore, he wants to generate high revenue quickly. Revenue, in turn, comes from popularity and publicity. Since the pizza chef has no experience in running a restaurant, he needs to learn and find a good strategy how to successfully maximize his revenue.

Speaking in terms of Reinforcement Learning, the pizza chef is the *agent*. Everything and everyone he interacts with (restaurant, guests, etc.) are part of the *environment*. The pizza chef has a range of *actions* he can execute, for example, baking a tasty pizza according to the guest's preferences, entertaining the guests, cleaning up the place, etc. For each pizza he serves, he earns a small amount of money, referred to as *reward*. Pleased guests increase the reward by providing a tip, returning regularly, or telling other people about the nice restaurant. Not every action is appropriate all the time. For example, cleaning up an already tidy restaurant is a mere time waste. Or, if there are more guests asking for pizza than



Created with DALL-E 3.

he can handle, he needs to hire an employee. In other words, the action choice depends on the *state* of the environment. The heart of *Reinforcement Learning (RL)* lies in finding an action choice strategy that maximizes the sum of all rewards (the pizza chef’s revenue). This learning paradigm draws inspiration from behavioral psychology, aiming to mimic the way how living creatures learn from the consequences of their interactions with the surrounding environment.

More generally, RL is an ML optimization method for maximizing the dynamic objective of a sequential task. Figure 2.1 showcases the general RL setting. An agent interacts with an environment by observing a state S and taking a proper action A , receiving a reward R , repeating the procedure forever or until termination.¹

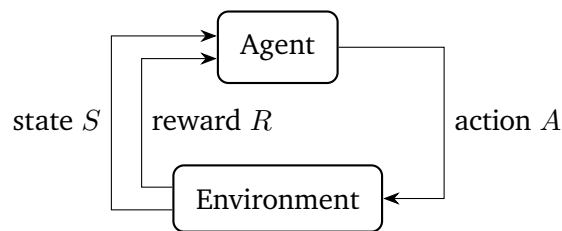


Figure 2.1.: The fundamental RL setting: An agent interacts with an environment by taking actions and observing next states and rewards.

Indices represent time steps, i.e., R_t is the reward at time step t , analogously for states and actions. Repeating the interaction loop yields a sequence $S_1, A_1, R_1, S_2, A_2, \dots$ which is called *trajectory*. If there is a notion of termination like in tennis (the player winning or losing a game) the (finite) trajectory part that represents one tennis game is called *episode*. That is, trajectories can consist of multiple episodes (the tennis player playing multiple tennis games or the pizza chef starting anew with a different restaurant). The sum $G_t := R_t + R_{t+1} + \dots$ of all rewards gathered from time step t on is referred to as the *return* (at time step t). Furthermore, the *discounted return* is defined as

$$G_t := R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$$

where $0 \leq \gamma \leq 1$ is the *discount factor*.

The agent’s strategy, which action to take at which state, is called *policy*. Formally, a policy is a mapping, denoted with π , that assigns an action A_t to a state S_t . If we denote

¹Here, we use capital letters to denote random variables while lowercase letters represent value instances.

the set of all possible states, the *state space*, with \mathcal{S} and the set of actions, the *action space*, with \mathcal{A} , then $\pi : \mathcal{S} \rightarrow \mathcal{A}$. In the case where the policy π is non-deterministic, we write $\pi(a|s) := \pi(A_t = a | S_t = s)$ for the probability that action a is selected at state s during time step t . From here on, we assume finite episodes of length T for simplicity and consider the general case where π is non-deterministic.

Given π , it is useful to predict the return one can expect at a given state s . We define

$$V^\pi(s) := \mathbb{E}[G_t | S_t = s] = \mathbb{E} \left[\sum_{k=0}^{T-t} \gamma^k R_{t+k} \mid S_t = s \right]$$

and call it the *value* of s under policy π . In words, the value is the expected discounted sum of rewards. Besides, V^π is called the *state-value function*. The symbol π is dropped if the policy is clear from the context. Similarly,

$$Q^\pi(s, a) := \mathbb{E}[G_t | S_t = s, A_t = a] = \mathbb{E} \left[\sum_{k=0}^{T-t} \gamma^k R_{t+k} \mid S_t = s, A_t = a \right]$$

is the value of taking action a at state s , the function of which is referred to as the *action-value function*.

The functions V^π or Q^π have comparably high variance, rendering them less appropriate for training. Instead, the *advantage* $\hat{A}^\pi(s, a) := Q^\pi(s, a) - V^\pi(s)$ is a better estimate with less variance. The advantage quantifies the benefit of taking action a at state s compared to the best action available in s , all under policy π .

2.1.1. (Semi-)Markov Decision Process

Typically, the environment's underlying transition logic can be modeled as a finite *Markov Decision Process (MDP)* which is a tuple $(\mathcal{S}, \mathcal{A}, P, \mathcal{S}_0, \gamma)$ where \mathcal{S} is the set of states, \mathcal{A} is the set of actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathbb{R} \rightarrow [0, 1]$ is the transition function that maps (s, a, s', r) to the probability that taking action a in state s results in next state s' and reward r , denoted with $P(s', r | s, a)$, \mathcal{S}_0 is the set of initiation states, and γ is the discount factor.

A core property of an MDP is that the transition P only depends on the current state and the action taken. In particular, all previous states don't have any influence on the next transition when s is given.

2.1.2. Training Agents: The Actor-Critic Framework

RL agents can be trained in various ways, divided into policy gradients, dynamic programming, Monte-Carlo approaches, among others. One of the most popular training schemes is the actor-critic framework. The critic learns the state-value function while the actor embodies the policy which is trained using the critic. See Figure 2.2 for a visual summary.

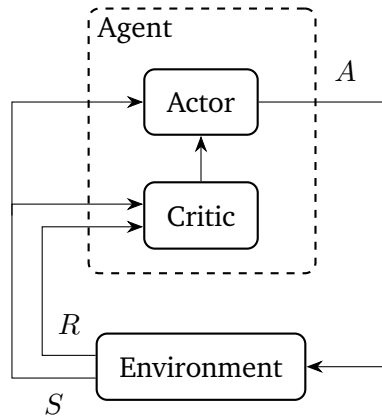


Figure 2.2.: The architecture of an actor-critic. The actor takes state S and chooses action A to interact with the environment. As a result, reward R is submitted to the critic which uses it to train the actor.

The arguably most important formal finding is the Bellman optimality equation which puts the state-value and the action-value function into a self-recursive relation as follows:

$$V(s) = \max_{a \in \mathcal{A}} \sum_{s', r} P(s', r | s, a) [r + \gamma V(s')], \quad (2.1)$$

$$Q(s, a) = \sum_{s', r} P(s', r | s, a) \left[r + \gamma \max_{a' \in \mathcal{A}} Q(s', a') \right]. \quad (2.2)$$

This fact is used by Temporal Difference (TD) learning which updates the state-value estimator via the update step

$$V(S) \leftarrow V(S) + \alpha \underbrace{\left[R + \gamma V(S') - V(S) \right]}_{\text{TD error}}$$

We also say that TD *bootstraps* the target value by reusing estimates of next state values. Besides, RL models can also learn the environment’s dynamics in order to become explicitly able to anticipate the consequences of an action. Such models are called *model-based*, in contrast to *model-free*.

2.1.3. Proximal Policy Optimization (PPO) as the State-of-the-Art Training Algorithm

This work uses the Proximal Policy Optimization (PPO) by Schulman et al. (2017). It is the state-of-the-art for training RL models. If we parameterize the policy π with parameters θ , we write π_{θ} *heta*. PPO incorporates the *PPO loss*

$$L^{\text{PPO}}(\theta) := -\hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right],$$

where $0 \leq \epsilon$ and where

$$r_t(\theta) := \frac{\pi_{\theta}(a_t, s_t)}{\pi_{\theta_{\text{old}}}(a_t, s_t)}.$$

ϵ is the clipping factor used to limit the extend of the policy update to reduce the chance of harmful policy updates.

2.2. Explainable RL: How to Make Agents Understandable and Trustworthy?

This section presents an overview of the state-of-the-art in *Explainable RL (XRL)*, the XAI subfield for RL.

As mentioned in the introduction, AI model architects can achieve explainability *post-hoc* or *intrinsically*. Furthermore, explanations can be *local* or *global*. A local explanation describes the model’s decision for a single, specific input instance whereas global explanations capture the model’s overall prediction behavior. (Milani et al., 2022)

Since recently, XRL draws an extensive attention and research. (Milani et al., 2022; Vouros, 2022; Glanois et al., 2021; Puiutta and Veith, 2020; Alharin et al., 2020; Krajna et al., 2022; Wells and Bednarz, 2021; Dazeley et al., 2022; Qing et al., 2022; Heuillet

et al., 2020; Hickling et al., 2022). According to Vouros (2022), the amount of XRL work is already large and the achieved progress towards full transparency is significant. Nevertheless, the authors point out a whole range of unsolved fundamental issues inside that field: For example, work on the qualities of a "good" explanation for an RL agent is missing and there is lack of a comparative evaluation benchmark for explainability and interpretability.

Dazeley et al., 2021 suggest different "levels of explanation" based on the agent's intentions. For example, first-order explanations explain agent decisions based on, inter alia, "the agent's underlying internal disposition towards the environment." In contrast, second-order explanations are generated based on the agent's belief of its own or other actors' mental states.

Milani et al., 2022 introduce a taxonomy that divides XRL methods into three categories: (1) Feature importance methods which generate explanations that point out decision-relevant input features, (2) learning process & MDP methods present instances of the past experience or components of the MDP that imply the current behavior, and (3) policy-level methods which describe the agent's long-term behavior.

Qing et al., 2022 categorize XRL explanation frameworks into four different types: (1) *Model-explaining* methods aim to produce an agent that has an interpretable operation in its inner structure. (2) *Reward-explaining* methods either decompose the reward to understand the individual component's influence on the agent's decisions or they directly obtain an understandable reward function. (3) *State-explaining* methods locally quantify the influence of state features or past observations on the agent's decision. Finally, (4) *task-explaining* methods are used in the context of hierarchical agents to decompose the main goal into (discovered) sub-goals and describe the agent's behavior regarding the sub-goals.

2.3. Temporal Abstraction via High-Level Actions, a.k.a. Options

Revisit the cake-baking robot example from the introduction. We want to abstract away from the long series of servo joint commands. *Hierarchical Reinforcement Learning (HRL)* is the key tool to help. HRL agents don't have just a single policy but multiple ones, arranged in a hierarchy. Higher-level policies execute higher-level tasks, like cracking an egg or completing a recipe step, while lower-level policies deal with lower-level tasks such as robot body maneuvering or arm positioning.

Options are an HRL approach introduced by Sutton et al. (1999). The authors define an option as follows:

Definition 2.3.1 (Option). An *option* ω is a triple $(\mathcal{I}^\omega, \pi^\omega, \beta^\omega)$ in which

- $\mathcal{I}^\omega \subseteq \mathcal{S}$ is an *initiation set*,
- $\pi^\omega : \mathcal{S} \rightarrow \mathcal{A}$ is a policy, the *intra-option policy*, and
- $\beta^\omega : \mathcal{S} \rightarrow [0, 1]$ is a *termination function*.

The set of all options is denoted with Ω .

An option can be seen as a temporally extended action. A higher-level policy π_{meta} , called *meta policy*, selects among options Ω . Each option $\omega \in \Omega$ is available only at states $S \in \mathcal{S}$ that are in the option's initiation set, i.e., $S \in \mathcal{I}^\omega$. If ω is selected by π_{meta} , that option executes its own policy π to choose the actual environment actions. The option ω is terminated depending on the termination probability determined by β . More specifically, the higher $\beta(s)$, the more likely ω terminates when approaching state s .

Besides, we can consider actions as a special case of options: Let $a \in \mathcal{A}$ be an action. Then, the corresponding option is $\omega = (\mathcal{S}, a, \mathbb{1}_{\mathcal{S}})$ which is the option that is available everywhere, selects action a always, and terminates immediately. Such an option is called *primitive*.

The action space of the intra-option policy does not necessarily consist of the environment actions but, instead, can comprise of another set of (lower-level) options. Exploiting this fact, we can construct a hierarchy with an arbitrary number of levels.

Definition 2.3.2 (Option hierarchy). Let $\Omega_0 := \mathcal{A}$ be a set of actions and $\Omega_1, \dots, \Omega_{L-1}$ be sets of options where $L \in \mathbb{N}$ and each option $\omega^{(l)} \in \Omega_l$ with policy $\pi^{(l)}$ chooses over options Ω_{l-1} . Furthermore, let π_{meta} be a meta policy choosing from options Ω_{L-1} . Then, $\mathcal{H} := (\Omega_0, \Omega_1, \dots, \Omega_{L-1}, \pi_{\text{meta}})$ is called (*option*) *hierarchy* with *size* L . If $L = 1$, we call the hierarchy *flat*.

In the case $L = 1$, the model consists only of the meta policy choosing actions. A model with $L \geq 2$ has one meta policy and $L - 1$ levels consisting of options (actions). With Ω_l we denote the set of options of level $0 \leq l < L$. Similarly, $\omega^{(l)} \in \Omega_l$ is an option of level l , its policy is denoted with $\pi^{(l)}$. In the special case $l = 0$, i.e., the lowest level, the set of options is the set of environment actions \mathcal{A} . For $l < L$, the intra-option policy $\pi^{(l+1)}$ chooses among the options Ω_l . See Table 2.1 for an exemplary summary.

Level	Options	Policy	Description	Examples
4	–	π_{meta}	the global policy	bake a cake
3	Ω_3	$\pi_j^{(3)}$	complete a recipe step	mix the dough, weigh 250g of butter
2	Ω_2	$\pi_j^{(2)}$	grab, place, manipulate objects	crack an egg, turn the milk carton
1	Ω_1	$\pi_j^{(1)}$	position body, arm, finger	move to the fridge, grab sugar package
0	$\Omega_0 := \mathcal{A}$	–	environment actions	servo joint commands

Table 2.1.: Exemplary option hierarchy in the cake baking example with $L = 4$ levels.

Each executing option has an executing caller from a higher level, inducing a chain of active options, giving rise to the following

Definition 2.3.3. Let $\mathcal{H} := (\Omega_0, \Omega_1, \dots, \Omega_{L-1}, \pi_{\text{meta}})$ be an option hierarchy. If option $\omega^l \in \Omega_l$ calls/selects option $\omega^{l-1} \in \Omega_{l-1}$ we say ω^l *invokes* ω^{l-1} and write $\omega^l \rightarrow \omega^{l-1}$. Moreover, we say that ω^{l-1} executes *in the context of* ω^l . Analogous for policies, especially the meta policy π_{meta} . We call a full sequence $\pi_{\text{meta}} \rightarrow \omega^{l-1} \rightarrow \dots \rightarrow \omega^1$ the *invocation trace*.

The termination of a higher-level option enforces termination of all lower-level options down the invocation trace. A visualization of the invocation trace can be found in Figure 2.3. It shows the entire option hierarchy and indicates the chain of invoked options.

The introduction of options invalidates the underlying Markov Decision Process because the ultimate action choice does not only depend on the current state (as required by the Markov assumption) but also on the current invocation trace. The MDP augmented by options is a Semi-Markov Decision Process (SMDP). (Sutton et al., 1999; Araki et al., 2021) From the MDP perspective, options can be seen as shortcuts between states, cf. Figure 2.4.

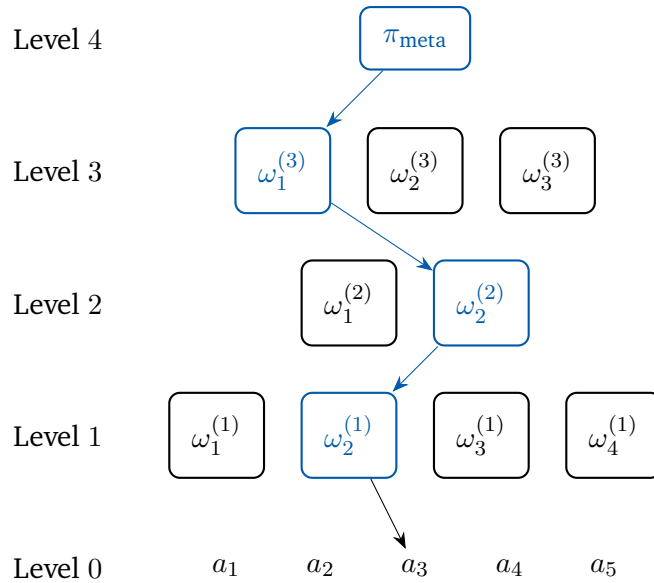


Figure 2.3.: The **invocation trace** inside a 4-level option hierarchy. Arrows indicate which option invoked which lower-level option/action.

2.4. NUDGE: A Logic Framework for Learning Symbolic Policies

This work uses *Neurally gUided Differentiable loGic policiEs (NUDGE)* (Delfosse et al., 2023b) because it is capable to train interpretable and explainable policies that are competitive with neural baselines on simple tasks. NUDGE is a framework that, with the help of a neural value network, learns a symbolic actor through differentiable forward reasoning.

More specifically, NUDGE takes a logic state representation as input and grounds the facts by computing weighted object relations, such as `obj1` being close to `obj2`. Next, NUDGE applies differentiable forward reasoning by means of the grounded relations and First-Order Logic (FOL) rules to determine the probabilities of the actions to use. Figure 2.5 provides a visual summary of the reasoning process. The policy can then be interpreted through the set of weighted action rules. NUDGE generates explanations for a given action via input gradients. The user defines the relevant objects, the object relations, and the initial FOL action rules, introducing prior knowledge to the agent.

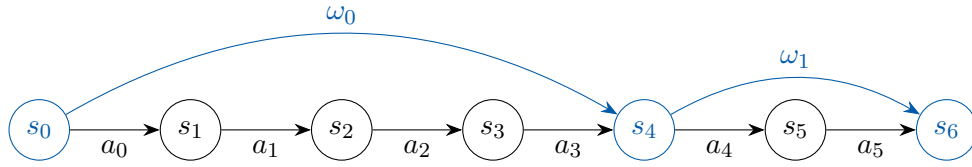


Figure 2.4.: The **SMDP** induced by options ω_t over an MDP with states s_t and actions a_t .

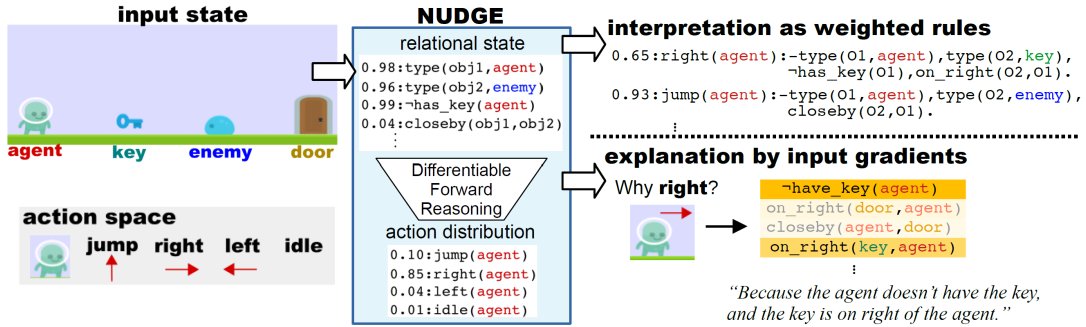


Figure 2.5.: NUDGE in a nutshell, taken from Delfosse et al. (2023b).

The NUDGE model architecture is defined as an actor-critic. The actor is the logic forward reasoner conducting the steps described above while the critic is a fully flexible NN. During training, the latter guides the former towards optimality. NUDGE uses standard PPO to train the neural critic. In contrast, the actor with policy $\pi_{\mathbf{W}}$ and weights \mathbf{W} is updated via simple gradient ascent, i.e.,

$$\mathbf{W} \leftarrow \mathbf{W} + \delta \nabla_{\mathbf{W}} \ln \pi_{\mathbf{W}}(s_t),$$

where δ is the TD(0) error of the neural critic. With this weight update, the logic policy $\pi_{\mathbf{W}}$ learns to maximize the expected return.

3. Method

Building up on the theoretical foundations from the last chapter, this chapter is going to further lay out the core idea of this work, specify the methodological approach how to answer the research question and outline the needed experiments to test the proposed ω -NUDGE model. Furthermore, this chapter offers a new algorithm how to learn options using PPO and GAE and, finally, examines the expectations and additional theoretical implications of applying the outlined method.

3.1. How to: Improve Policy Interpretability via Options

We want to investigate the central research question, namely whether options can help to make a logic policy more understandable. Or, in short, if ω -NUDGE is better interpretable than NUDGE. There is good reason to believe that this is true, at least for environments with long episodes and heterogeneous subtasks: If for each subtask there is a specialized option that can solve it, the meta policy merely needs to identify the current subtask from the observation and choose the proper option to solve it. The subtask-option-assignment problem is substantially simpler than the entire RL problem since the former is a strict subset of the latter. In other words, the meta policy can focus on macro-strategic planning rather than micro-management. Thus, it is reasonable to expect that the number of FOL rules to build up the meta policy program will be smaller, hence better interpretable, than in the flat counterpart. Of course, parts of the agent will then be hidden in black-box NN options. However, we are interested in the transparency of the high-level thinking of the agent, not the low-level execution. Therefore, we can tolerate that the lower-level policy remains non-interpretable to us.

Since each option corresponds to a subgoal, each meta policy decision inherently incorporates the information which subgoal the agent currently wants to achieve. Subgoal descriptions are encoded as human-readable predicate names. Subgoals typically are

states that the agent must pass in order to reach the final goal. Such states are also called *access states*.

Pretrained options should be used whenever possible because it improves the learning success chance—and follows the way how humans learn. Therefore, the experiments with ω -NUDGE should include runs with pretrained options. Pretraining options also avoids the necessity to find out which option learned to solve which subgoal.

For simplicity, we chose a model-free and state-free feed-forward agent architecture. Generally, the choice of the model hierarchy should reflect the goal hierarchy of the RL problem. In particular, each subgoal should be addressed by exactly one option. The state-of-the-art PPO algorithm (Schulman et al., 2017) with GAE (Schulman et al., 2015) will be used but need to be extended to options first.

Hierarchical agents bring a number of additional new metrics that can be exploited for explanations. They include:

- The option initiation set \mathcal{I} and the option choice probabilities returned by π_{meta} . Knowing which option is available and invoked in which state reveals information about which option is used to address which subtask.
- The execution length of options, i.e., the average time between option invocation and termination. One would anticipate that the option execution length correlates with the complexity of a subtask.
- The termination probability β : Termination of an option should correlate with the achievement of the corresponding subgoal. Additionally, options ideally also terminate in situations where the subgoal achievement is utterly hopeless or the state is simply out of distribution for what the option is trained for.
- The invocation frequency or the share in the total execution time: Frequently occurring subtasks obviously are solved by frequently invoked options, given that options do not terminate unnecessarily before achieving the targeted subtask.
- Finally, the overall behavior of the option and the action probability distribution of the intra-option policy. Qualitative analysis of the trajectories rolled out by an option can disclose further insight into the purpose of the respective option.

Methodologically, the following roadmap will guide the investigation. First, we learn three baselines: a NUDGE flat, a neural flat, and a neural hierarchy baseline model on a selected environment. The logic version might perform worse than the neural ones due to the limited expressiveness of FOL rules.

Second, on the same environment, we learn a hierarchical model with a logic meta policy and compare it with the flat logic version in terms of the number of FOL rules and the number of atoms contained in the rules. We will also compare the performance in terms of average return. Since it is not guaranteed that the options indeed will learn to solve the desired subtasks, there is danger that the option predicates won't match the intended option semantics, invalidating the interpretability of the entire logic meta policy. Therefore, additionally, the aforementioned training and comparisons will be conducted with pre-trained options.

3.2. Combining PPO with the Option-Critic

First, we need to redefine the state-value and action-value functions and extend them to options. Here, we follow Sutton et al. (1999) and Bacon et al. (2016) but with adjusted notation.

Definition 3.2.1 (Value functions in context of ω). Let $\hat{\omega} \rightarrow \bar{\omega}$ be two options where option $\bar{\omega}$ chooses over options $\check{\Omega}$.

- (1) The *state-value function in the context of $\bar{\omega}$* is the function $V^{\bar{\omega}} : \mathcal{S} \rightarrow \mathbb{R}$ defined as

$$V^{\bar{\omega}}(s) := \sum_{\check{\omega} \in \check{\Omega}} \pi_{\bar{\omega}}(\check{\omega} | s) Q^{\bar{\omega}}(s, \check{\omega}).$$

- (2) The *option-value function in the context of $\bar{\omega}$* is the function $Q^{\bar{\omega}} : \mathcal{S} \times \check{\Omega} \rightarrow \mathbb{R}$ defined as

$$Q^{\bar{\omega}}(s, \check{\omega}) := r(s, \check{\omega}) + \gamma \sum_{s'} P(s' | s, \check{\omega}) U^{\hat{\omega}}(\bar{\omega}, s').$$

- (3) The *option-value function upon arrival in the context of $\bar{\omega}$* is the function $U^{\bar{\omega}} : \hat{\Omega} \times \mathcal{S} \rightarrow \mathbb{R}$ defined as

$$U^{\bar{\omega}}(\hat{\omega}, s') := \beta_{\bar{\omega}}(s') V^{\hat{\omega}}(s') + (1 - \beta_{\bar{\omega}}(s')) V^{\bar{\omega}}(s').$$

The new definition of the state-value function is analogous to the native state-value function $V_{\pi_{\bar{\omega}}}(s)$. $Q^{\bar{\omega}}(s, \check{\omega})$ is the value of executing $\check{\omega}$ in state s in context of $\bar{\omega}$. $U^{\bar{\omega}}(\hat{\omega}, s')$ reflects the value of executing $\bar{\omega}$ upon entering the state s' . We can canonically extend the value function definitions to

Definition 3.2.2 (Advantage in context of ω). Let again $\hat{\omega} \rightarrow \bar{\omega} \rightarrow \check{\omega}$. The advantage $\hat{A}^{\hat{\omega}}(\check{\omega}_t, s_t)$ of option $\check{\omega}_t$ in state s_t is called the *advantage in context of $\bar{\omega}$* and is defined as

$$\hat{A}^{\bar{\omega}}(\check{\omega}_t, s_t) := Q^{\bar{\omega}}(s_t, \check{\omega}_t) - V^{\bar{\omega}}(s_t).$$

Furthermore, we write

$$A(\bar{\omega}, s_t) := V^{\bar{\omega}}(s_t) - V^{\hat{\omega}}(s_t)$$

for the *continuation advantage of ω* .

The option-critic (Bacon et al., 2016) is an extension of the actor-critic framework, cf. Figure 3.1. It is the first end-to-end HRL architecture, also with the first proven policy gradient theorem for options.

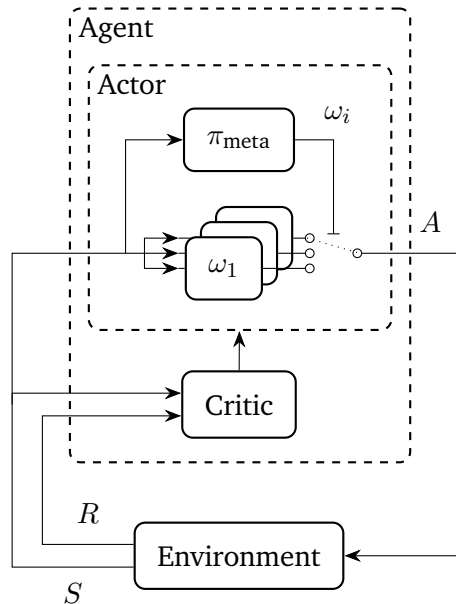


Figure 3.1.: The option-critic architecture. It is an extension of the actor-critic, cf. Figure 2.2. The meta policy π_{meta} selects an option ω_i which is executed until termination. The critic submits feedback to the meta policy and do the currently active option. Figure inspired by Bacon et al., 2016.

Options update their policy and their termination function only with transitions where they were active. Let ω be an option and $\theta := (\theta_\pi, \theta_\beta)$ the parameters of its policy π and termination function β . The loss function applied by Bacon et al. for option training is

$$L^{\text{OPTION}}(\theta) := \underbrace{-\hat{\mathbb{E}}_t \left[\log \pi_{\omega, \theta}(a_t, s_t) \hat{A}_t^\omega \right]}_{\text{policy loss}} + \underbrace{\hat{\mathbb{E}}_t \left[\beta_{\omega, \theta}(s_{t+1}) \cdot (\hat{A}_{t+1} + \xi) \right]}_{\text{terminator loss}},$$

where ξ is the termination regularizer, used to adjust the general execution duration of options. \hat{A}_t the advantage of ω in the higher-level context estimated by $V_\omega(s_{t+1})$.

This work proposes the following option-specific loss function to be used with PPO in place of L^{OPTION} .

Definition 3.2.3 (PPO loss for options). The loss used for PPO training with options is defined as

$$\begin{aligned} L^\pi(\theta) &:= -\hat{\mathbb{E}}_t \left[\text{PPO} \left(\rho_t^\pi(\theta), \hat{A}_t^\omega \right) \right] & \text{where } \rho_t^\pi(\theta) &:= \frac{\pi_{\omega, \theta}(a_t, s_t)}{\pi_{\omega, \theta_{\text{old}}}(a_t, s_t)}, \\ L^\beta(\theta) &:= \hat{\mathbb{E}}_t \left[\text{PPO} \left(\rho_{t+1}^\beta(\theta), \hat{A}_{t+1} + \xi \right) \right] & \text{where } \rho_t^\beta(\theta) &:= \frac{\beta_{\omega, \theta}(s_t)}{\beta_{\omega, \theta_{\text{old}}}(s_t)}, \end{aligned}$$

and where

$$\text{PPO}(x, y) := \min(xy, \text{clip}(x, 1 - \varepsilon, 1 + \varepsilon)y).$$

This proposed loss function is a natural extension of the PPO loss to both the intra-option policy and the termination function.

3.3. Generalized Advantage Estimation for Options

This work also proposes to use an option-specific Generalized Advantage Estimator (GAE), called Generalized Advantage Estimator for Options (GAEO). The experiments, however, revealed that the usage of GAE was counterproductive.

Let $\tau = (s_1, (\omega_1^{L-1}, \dots, \omega_1^1, a_1), r_1, s_2, \dots, r_{T-1}, s_T)$ be a trajectory from the agent's experience. For simpler notation, we write $V_t^\omega := V^\omega(s_t)$, analogously $Q_t^\omega := Q^{\omega_t}(s_t, \hat{\omega}_t)$, and $U_t^\omega := U^{\omega_t}(\hat{\omega}_t, s_t)$, where $\hat{\omega}_t \rightarrow \omega_t$.

Recall that δ_t denotes the TD(0) error at time step t (see also Section 2.1.2).

Definition 3.3.1 (*n*-step advantage). Let

$$\delta_t := r_t + \gamma V_{t+1} - V_t$$

denote the TD(0) error. The *n*-step advantage is defined as

$$\hat{A}_t^{(n)} := \sum_{k=0}^{n-1} \gamma^k \delta_{t+k}.$$

The *n*-step advantage is a collection of several different advantage estimators: The 1-step advantage is equal to the TD(0) error δ_t which has low variance but high bias. The counterpart is the Monte-Carlo advantage estimate which is the ∞ -step advantage $\hat{A}_t^{(\infty)} = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$. It is the discounted return and does not include any other estimators, hence, has low bias but high variance.

In order to balance bias and variance, this work tests the Generalized Advantage Estimator by Schulman et al. (2015):

Definition 3.3.2 (GAE). For $0 \leq \lambda \leq 1$, the *Generalized Advantage Estimator* (GAE) is

$$\begin{aligned} \hat{A}_t^{\text{GAE}} &:= (1 - \lambda) \cdot \left(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \dots \right) \\ &= \sum_{k=0}^{\infty} (\gamma \lambda)^k \delta_{t+k}. \end{aligned}$$

In words, the GAE is a geometric sum of all (available) *n*-step advantages. The weighting with $(\gamma \lambda)^k$ reflects a compromise in the bias-variance trade-off between the high-variance Monte-Carlo return estimate and the high-bias 1-step advantage.

In the options framework, we need to further specify the GAE computation for two reasons: First, options can terminate at any point of time in the episode, forbidding to use *n*-step advantages that reach beyond option termination. Second, any policy on level 2 or higher does not always make a decision because it needs to wait until the lower-level option has terminated.

This work addresses the aforementioned situations by proposing

Definition 3.3.3 (GAEO). Let $\tau := (s_1, \omega_1, r_1, s_2, \dots, r_{1-T}, s_t)$ be a trajectory in the option-induced SMDP. For $0 \leq \lambda \leq 1$, the *Generalized Advantage Estimator for Options (GAEO)* is

$$\hat{A}_t^{\text{GAEO}} := \sum_{k=0}^{\infty} (\gamma\lambda)^k \delta_{t+k}^\omega,$$

where

$$\delta_t^\omega := r_t + \gamma U_{t+1}^\omega - V_t^\omega$$

is the option-specific TD(0) error.

Changes to the GAE are highlighted in blue. In a nutshell, the GAEO is the GAE over the option-induced SMDP where, for all options ω , their value estimates V_t^ω are replaced by the value upon arrival U_t^ω . Moreover, termination is handled with bootstrapping using U_t^ω , too. Note that t does *not* refer to actual time steps as in the original MDP (where transitions are done by actions) but rather to an "abstracted" time step in the higher-level SMDP. In particular, r_t refers here to the sum of rewards gathered during execution of the respective lower-level option. Moreover, each level in the options hierarchy has its own GAEO estimate whereas the estimate for π_{meta} remains the GAE.

The inclusion of U_t^ω in the GAEO adheres the fact that an option can terminate at any (SMDP) time step. However, the usage of U_t^ω also has its pitfalls, as the experiments show. For a detailed discussion please refer to the Appendix A.1.

3.4. The Options Training Algorithm

Overall, the training algorithm applied in this work is a modification of the state-of-the-art PPO algorithm by Schulman et al. (2017). Algorithm 1 summarizes the training process from a global point of view. The model gathers on-policy experience by rolling out a trajectory in the environment for a fixed number of steps n_{rollout} . Here, the model acts non-deterministically by random choice according to the probabilities returned by π^ω and π_{meta} , respectively. After rollout, the algorithm computes the GAEO for all experienced time steps using Equation A.1. Next, π_{meta} trains on experience \mathcal{D} following the PPO algorithm. After π_{meta} , the algorithm traverses the option hierarchy and trains each option individually following a proposed, option-specific modification of the PPO algorithm for both, the

option-critic (Algorithm 2) and the terminator (Algorithm 3). The overall sequence—gathering rollouts, computing GAEs, and training—repeats for n_{total} iterations (or until convergence).

Algorithm 1: Option Hierarchy Training

Input : Hierarchy $\mathcal{H} = (\Omega_0, \dots, \Omega_{L-1})$ with meta policy π_{meta} , total time steps n_{total} , time steps between updates n_{rollout} , maximum episode length n_{maxlen}

Output : Trained hierarchy \mathcal{H}'

for iteration $1, 2, \dots, n_{\text{total}}$ **do**

for rollout step $1, 2, \dots, n_{\text{rollout}}$ **do**

 Perform environment action

 Add observed transition to experience \mathcal{D}

 Reset environment if goal achieved or number of steps reached n_{maxlen}

 Compute GAE \hat{a}_t for all time steps t

 Train meta policy π_{meta} with PPO

 see Schulman et al. (2017)

for each hierarchy level $1 \leq l < L$ **do**

for each option $\omega \in \Omega_l$ **do**

 Train inter-option policy π_ω with PPO

 see Algorithm 2

 Train terminator β_ω with PPO

 see Algorithm 3

return trained hierarchy \mathcal{H}'

Algorithm 2 depicts the PPO-based training process for the option-critic of an option ω . It applies stochastic gradient descent with batches $\mathcal{D}_{\text{batch}}^\omega$ of size b for n_{epochs} repetitions. Here, $\mathcal{D}_{\text{batch}}^\omega$ is a set of those augmented SMDP transitions that occurred right after ω took a decision. Optionally, the GAEs can be normalized to have zero mean and standard deviation of 1. Unlike ε -greedy where exploration is controlled via ε , here the entropy coefficient c_{entropy} trades off between exploration and exploitation. A higher coefficient increases the impact of the entropy $S(\pi_{\theta'})$ in the overall loss, enforcing $\pi_{\theta'}$ to balance probabilities among the options. The overall loss also includes the value loss

$$L^{\text{VF}}(\theta') := \frac{1}{b} \sum_{s_t \in \mathcal{D}_{\text{batch}}^\omega} (V_{\theta'}^\omega(s_t) - g_t)^2,$$

which is the mean squared error between the parameterized critic $V_{\theta'}$ and the update target g_t , here the discounted return at time step t . The value coefficient c_{value} controls the overall loss share of the value loss and is used to even out actor and critic loss for uniform training progress.

Algorithm 2: Option-Critic PPO

Input : Intra-option policy π_θ of option ω with parameters θ deciding over options $\tilde{\omega} \in \tilde{\Omega}$, number of epochs n_{epochs} , rollout data \mathcal{D} , batch size b , value coefficient c_{value} , entropy coefficient c_{entropy}

Output : $\pi_{\theta'}$ trained on \mathcal{D}

$\theta' \leftarrow \theta$

for epoch $1, 2, \dots, n_{\text{epochs}}$ **do**

for each batch $\mathcal{D}_{\text{batch}}^\omega \subseteq \mathcal{D}$ of size b **do**

 Compute ratio $\rho_t^\pi = \frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(\tilde{\omega}_t|s_t)}$ for each option-state pair $\tilde{\omega}_t, s_t \in \mathcal{D}_{\text{batch}}^\omega$ Actor

 Optionally, normalize all \hat{A}_t^{GAEO} in $\mathcal{D}_{\text{batch}}$

 Update θ' by gradient descent step

$\nabla_{\theta'} (L^{\text{PPO}}(\theta') + c_{\text{value}}L^{\text{VF}}(\theta') + c_{\text{entropy}}S(\pi_{\theta'}))$

return $\pi_{\theta'}$

Finally, Algorithm 3 applies PPO to the terminator. It is analogous to the option-critic PPO except for two differences: There is no critic that needs to be trained and a termination regularizer ξ , as suggested by Bacon et al. (2016), can be used to influence the execution length of options. More precisely, increasing ξ leads to longer options because ξ adds artificial advantage, suggesting higher value in the option's execution, i.e., lower loss in terminating it.

3.5. Expectations and Final Theoretical Remarks

The hope is that ω -NUDGE is better interpretable than NUDGE. This section elaborates further theoretical consequences of combining NUDGE with options.

The options training algorithm (Algorithm 1) trains the meta policy and all options concurrently. From the viewpoint of the meta policy, the available action (option) space changes during training. It's like a train driver who learns to drive a train, but the effect of all the buttons and knobs changes over time. Accordingly, training with simultaneous option updates is expected to become difficult. There are at least two remedies for this issue: First, the option training can be "cooled" down early by reducing the learning rate or the PPO clip factor only for options. In essence, the options converge to a solution and the meta policy receives enough extra time to get used to the options. Second, one could

Algorithm 3: Terminator PPO

Input : Termination function β_θ of option ω with parameters θ , number of epochs n_{epochs} , rollout data \mathcal{D} , batch size b , termination regularizer ξ

Output : $\beta_{\theta'}$ trained on \mathcal{D}

$\theta' \leftarrow \theta$

for epoch $1, 2, \dots, n_{\text{epochs}}$ **do**

for each batch $\mathcal{D}_{\text{batch}}^\omega \subseteq \mathcal{D}$ of size b **do**

 Compute ratio $\rho_t^\beta(\theta') = \frac{\beta_{\theta'}(s_t)}{\beta_\theta(s_t)}$ for each state $s_t \in \mathcal{D}_{\text{batch}}^\omega$

 Optionally, normalize all \hat{A}_t in $\mathcal{D}_{\text{batch}}$

 Update θ' by gradient descent step $\nabla_{\theta'} \text{PPO} \left(\rho_t^\beta(\theta'), -(\hat{A}_t + \xi) \right)$

return $\beta_{\theta'}$

separate the learning of all involved policies through curriculum learning. That is, each option individually learns to solve a simple subtask, as proposed above. Afterwards, the meta policy learns to solve the RL task by using the fixed, pre-trained options. The latter solution requires the environment together with the reward function to be modifiable.

Logic policies like NUDGE don't apply readily to environments that have a continuous action space. Options can have any arbitrary architecture and, thus, can deal with any form of action space. Moreover, the meta policy's choice over options is always discrete. Thus, options can also be seen as a translation from discrete actions to continuous (sequences of) actions, enabling the usage of logic policies for continuous action domains.

In NUDGE, a domain expert provides a priori knowledge in form of predicates that encode the observation and FOL rules for choosing actions. When combining NUDGE with options, the domain expert now is required to provide FOL rules for *options* instead of actions. If the options are not pre-trained, the expert needs to anticipate which option is going to solve which subtask, seemingly impossible. Fortunately, through the provision of FOL rules, the expert can influence the future development of the options: Since FOL rules determine under which conditions a certain option is taken the expert can, to some extent, define the initiation set \mathcal{I} for each option. Options that are invoked in only specific, similar situations are going to specialize on these situations and learn to adequately react. Since the options mostly see the situations they are trained for, they perform bad on out-of-distribution MDP regions, giving an incentive for the meta policy to remain the option invocations at the states implied by the FOL rules. In order to avoid suboptimal results due to exaggerated restriction, the expert can add "backup" options to the hierarchy that underlie no specific

prior and can evolve freely.

Where NUDGE fails, ω -NUDGE might be successful. Especially in real-world problems where state and action space typically have large dimensions, the restriction of a policy to FOL rules could be too limiting in order to find a satisfying policy. The ML practitioner can choose especially flexible architectures like deep NNs that deliver the required expressiveness to solve sub-problems where interpretability isn't as crucial as on higher levels. In other words, by choosing the hierarchy and the option architecture, the ML practitioner can trade-off between interpretability and flexibility.

Finally, as usual with options (Bacon et al., 2016), ω -NUDGE is expected to perform better and learn faster than NUDGE

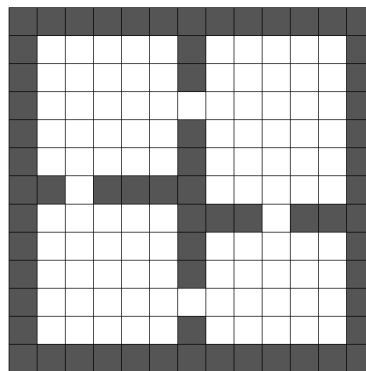
4. Experiments and Results

This work introduces a novel environment: the `MEETINGROOM` environment, explicitly designed for hierarchical agents. It is a simple environment exhibiting opportunities for abstraction. However, it is not comparable with real-world applications as the latter are much more complicated. The experiments on `MEETINGROOM` should be considered as a proof of concept.

The interested reader finds the implementation details and a brief documentation in the Appendix A.3.

4.1. Introducing the `MEETINGROOM` Environment

A popular environment (Hutsebaut-Buysse et al., 2022) used for HRL benchmarking is the *FOURROOMS* environment by Sutton et al. (1999). *FOURROOMS* is a grid world of four rooms connected via doorways, see the figure on the right. The agent starts from a random position and is tasked to navigate to a random target position. The four doorways are key positions because moving from one room to another requires the agent to move through the doorway. Sutton et al. (1999) defined options that move the agent to one of the doorways. The authors showed that fewer iterations were needed to learn the optimal value function. *FOURROOMS* is only useful for 2-level hierarchies because there is no possibility to further abstract beyond doorways.



FOURROOMS

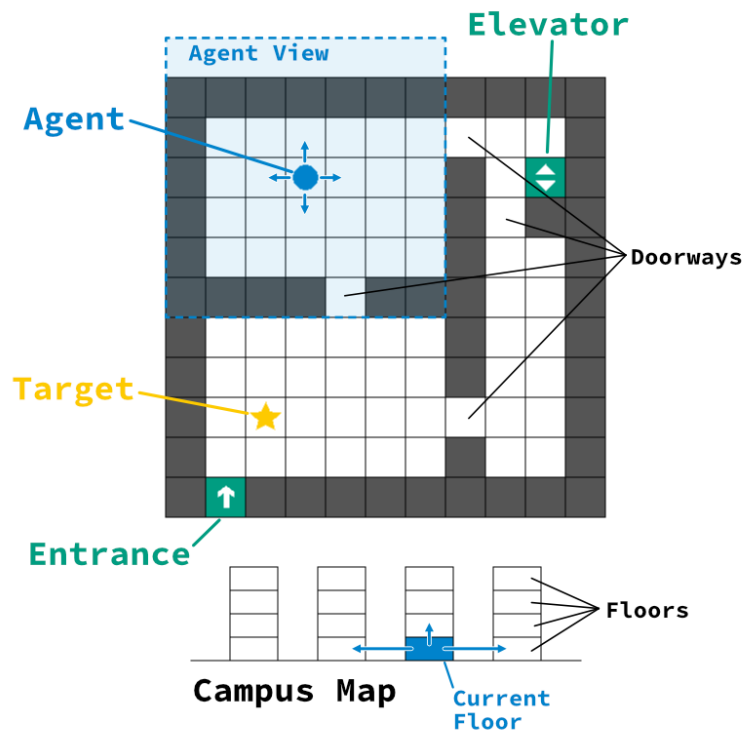


Figure 4.1.: The MEETINGROOM environment explained. Dark fields represent walls which can't be passed by the agent. The campus map on the bottom indicates in which building and which floor the agent currently is. In this example, the agent is on the ground floor of the third building.

Therefore, this work proposes an extension of the FOURROOMS environment. Two new dimensions are added to the grid world: floors and buildings. The agent can switch floors via an elevator. Moreover, each building has an entrance at the ground floor used to switch buildings. The four move actions north, east, south, and west are extended by the actions *next* and *previous* utilized to switch floors (when in an elevator) or buildings (when at an entrance). The task can be compared with the problem of navigating to a meeting room somewhere on a large corporate campus. Therefore, this environment is referred to as the *MEETINGROOM* environment. See Figure 4.1 for a visualization of the environment.

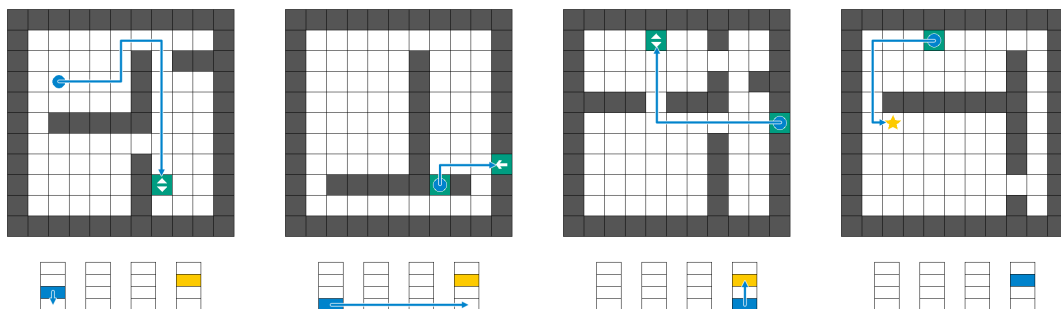


Figure 4.2.: Exemplary trajectory in the MEETINGROOM environment. Since the target is in a different building, the Agent first needs to take the elevator to get to the ground floor before moving to the target’s building. After reaching the target building, it can take the other elevator to the target’s floor.

The observation returned by the environment is a vector consisting of

- the agent’s relative position to the target and to the elevator as well as the entrance of the current building
- the number of the current floor
- a flattened 7×7 Boolean-encoded view of the locally surrounding walls

MEETINGROOM is designed to offer a multi-level goal hierarchy. In the most general case, the target is in a different building than the agent. Consequently, the agent not only needs to find the target’s room but also the target’s floor and building, navigating through entrances and elevators. The latter two induce access states which the agent must pass before reaching the target, affording opportunities for temporal abstraction. Figure 4.2 displays an exemplary trajectory.

Note that, just like in FOURROOMS, each floor is divided into four rooms. The floor plan, however, is randomly generated for each floor and for each new episode. Re-generation prevents the agent from overfitting to one single floor by simply rote-learning the direct path to the target. Instead, the agent is forced to generalize to any floor plan, incentivizing policies that take not only the agent’s current position into account but also the local surroundings.

4.2. Experimental Setup

This work evaluates ω -NUDGE on MEETINGROOM with one building and four floors and a floor size of 11×11 . Multiple buildings turned out to be surprisingly difficult to learn and, thus, remain for future work. In the setting with one building, a two-level hierarchy with three options is evident: One option for targeting the elevator, one option for the star an one option for—when inside the elevator—moving to the target floor.

Two of the three neural options of ω -NUDGE are pretrained: The first to navigate to the star, the second to navigate to the elevator. The third option, which is for floor switching if already inside an elevator, was not pretrained as it is expected that this simple task will be learned easily by the agent.

For the sake of simplicity, we choose the initiation set $\mathcal{I} := \mathcal{S}$ to be the entire state space, i.e., each option is available to its higher-level policy everywhere. A human expert provides the FOL rules as well as action (flat) and option (hierarchical) predicates to choose from.

In all experiments, episodes are truncated at 100 steps to limit evaluation time. Rewards and advantages are normalized while observations are normalized only for neural policies (logic policies receive a symbolic state representation where normalization doesn't make sense).

The neural policies use two layers of dense NNs for all three, actors, critics, and terminators. Except for the neural flat baseline, which has 64 neurons per layer, each neural option policy has a small dense layer width of 16 units, enforcing specialization and avoiding option mode collapse.

To accelerate learning, the reward function not only returns +1 for reaching the goal but it also awards movement towards the goal with a small signal while penalizing the opposite.

The most important hyperparameters can be found in Table 4.1. The learning rate uses an exponential decay with half-life period of 25% of the total training time if the policy is a meta policy, 10% otherwise, i.e., options slow down learning early to allow the meta policy to adjust. For a full specification including used seeds etc., please refer to the experiment configuration files saved in the GitHub repository, linked in Appendix A.3.

Note that no systematic hyperparameter search was conducted, implying that there might be room for improvement for any of the used models. In particular, these experiments do not claim to be fully representative but rather serve as indication.

Hyperparameter	Value	
	Neural	Logic
Total steps n_{steps}	10 ⁷	
Steps n_{rollout} between updates	512	
Number of epochs n_{epochs}	6	
Batch size b	256	
Maximum episode length n_{maxlen}	100	
Optimizer	Adam	
Learning rate	0.002	0.001
Value coefficient c_{value}	0.05	0.0025
Entropy coefficient c_{entropy}	0.1	0.00001
Discount factor γ	0.9	
GAE(O) λ	0	0
Termination entropy coefficient c_{entropy}	0.1	n.a.
Termination regularizer ξ	-0.2	n.a.

Table 4.1.: The hyperparameters used for training the neural and the logic policies on MEETINGROOM, respectively. See Algorithm 1 for the usage of the parameters.

Furthermore, the FOL rules provided to the logic policies are listed in Figures 4.3 and 4.4. Each rule is structured following the scheme `consequence :- requirements` where the consequence is the action (predicate) and requirements are a conjunction of atomic facts. If all requirements of a rule are fulfilled, the corresponding action is more likely to be selected.

4.3. Results

Experiments were run for ω -NUDGE, NUDGE and two neural baselines: a flat and a hierarchical one. The results are depicted in Figure 4.5. The following sections are going to interpret and elaborate the results.

```

north(X) :- north_free(X).
east(X) :- east_free(X).
south(X) :- south_free(X).
west(X) :- west_free(X).
north_to_target(X) :- south_of_target(X).
east_to_target(X) :- west_of_target(X).
south_to_target(X) :- north_of_target(X).
west_to_target(X) :- east_of_target(X).
north_to_elevator(X) :- not_on_target_floor(X), south_of_elevator(X).
east_to_elevator(X) :- not_on_target_floor(X), west_of_elevator(X).
south_to_elevator(X) :- not_on_target_floor(X), north_of_elevator(X).
west_to_elevator(X) :- not_on_target_floor(X), east_of_elevator(X).
next_upper_floor_to_target(X) :- in_elevator(X), below_target(X).
prev_lower_floor_to_target(X) :- in_elevator(X), above_target(X).
north_bypass_wall(X) :- wall_west(X), east_of_target(X).
south_bypass_wall(X) :- wall_east(X), west_of_target(X).
south_bypass_wall(X) :- wall_west(X), east_of_target(X).
east_bypass_wall(X) :- wall_north(X), south_of_target(X).
east_bypass_wall(X) :- wall_south(X), north_of_target(X).
west_bypass_wall(X) :- wall_north(X), south_of_target(X).
west_bypass_wall(X) :- wall_south(X), north_of_target(X).

```

Figure 4.3.: The FOL rule set used for NUDGE.

```

opt0_to_target(X) :- on_target_floor(X).
opt1_to_elevator(X) :- below_target(X).
opt1_to_elevator(X) :- above_target(X).
opt2_switch_floor(X) :- in_elevator(X), not_on_target_floor(X).

```

Figure 4.4.: The FOL rule set used for ω -NUDGE. optX identifies the target option to choose.

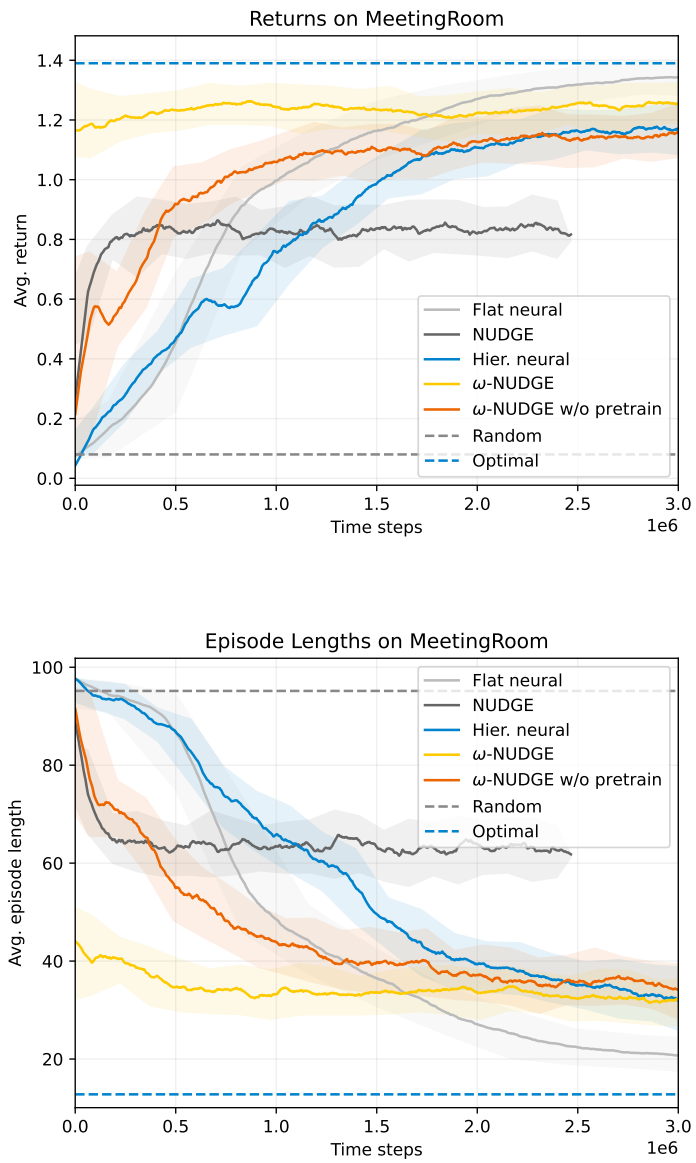


Figure 4.5.: The training results for ω -NUDGE, NUDGE, a neural flat and a neural hierarchical baseline. Highlighted areas show the 90% confidence interval.

4.3.1. ω -NUDGE Performs Better than NUDGE

The most important observation from the results in Figure 4.5 is the striking performance of ω -NUDGE: First, it developed a decent overall performance very quickly and, second, outperforms NUDGE by far. Arguably, the two main reasons for the comparably outstanding performance are (1) the fact that ω -NUDGE had access to pretrained policies that already know how to solve the environment partially and (2) that the pretrained policies are neural, i.e., have good fitting capabilities. The training curve for ω -NUDGE without pretraining supports both arguments: First, ω -NUDGE with pre-training reaches its long-term performance very early whereas ω -NUDGE without pretraining needs more time to reach the same performance. Second, ω -NUDGE without pretraining performs better than NUDGE, indication for the strength of NNs.

Furthermore, NUDGE was originally tested on low-dimensional environments like `GETOUT` where, on just a single dimension, the agent has to collect a key in order to open a door while evading enemies. (Delfosse et al., 2023b) In contrast, `MEETINGROOM` has three dimensions (if recognizing the elevator as the gate to the third dimension), yielding six different actions, making `MEETINGROOM` policies more complex to be represented in terms of rules.

4.3.2. ω -NUDGE Has Better High-Level Interpretability than NUDGE

ω -NUDGE has a far smaller rule set than NUDGE cf. Figures 4.3 and 4.4. More precisely, NUDGE has 22 rules while ω -NUDGE has only 4 rules, clearly showing a significantly improved high-level interpretability. Yet, ω -NUDGE is able to beat NUDGE by far in terms of performance.

Moreover, the perception of complex structures like walls is problematic for logic policies. Logic rules require a logic state representation. However, representing walls in terms of rule sets is a non-trivial task since walls can have arbitrary shape. One could model each single field in the local 7×7 view of the agent as an object that is either a wall or an empty field. This, however, would lead to $7 \cdot 7 = 49$ different predicates only for perceiving the local surroundings. As a consequence, learned rule sets would become inscrutably large, much larger than the one presented in Figure 4.3.

Options allow the logic meta policy to abstract away the walls and concentrate on the higher-level strategy: going to the elevator when on the wrong floor, going to the star

when on the right floor. Neural options then use their capable perception of the gridworld to actually navigate to the current subgoal.

4.3.3. Even Without Pretraining ω -NUDGE Learns Faster than Neural Hierarchies

Consulting the learning curves in Figure 4.5 of ω -NUDGE without pretraining versus the neural hierarchy, the former clearly learns the problem faster than the latter. This outcome is exceptionally remarkable because the only difference between these two agents is that ω -NUDGE incorporates only 4 high-level rules. In other words, even though all sub-policies are randomly initialized, providing just a few high-level rules accelerates training strongly. One could even argue that using a neural meta policy has no performance benefits over a logic meta policy here. This conjecture, however, would need further empirical investigation.

The likely reason for the faster learning progress by ω -NUDGE is that the FOL rules incentivize an early development of the options to the desired subtask specialization. As we will see in the next subsection, the options indeed learned the subtasks as intended. However, note that, due to their expressiveness, the neural baselines both have a better long-term performance (omitted in the figure) than ω -NUDGE.

4.3.4. ω -NUDGE Finds More Meaningful Options than Neural Hierarchies

Figure 4.6 shows additional evaluation results regarding the option selection probabilities depending on the state. For ω -NUDGE, indeed, the probabilities match the expectations one would have when considering the option names. The heatmaps also show that option activation is uniform for the entire respective floor, except for the elevator which is reserved for option ω_2 for switching the floor. In contrast, the heatmaps for the neural hierarchy are much less meaningful. Not only that: Option 2 is used for both, going to the elevator *and* going to the target. Hence, the neural hierarchy didn't even split up the task into subgoals but, instead, let one single option do the job.

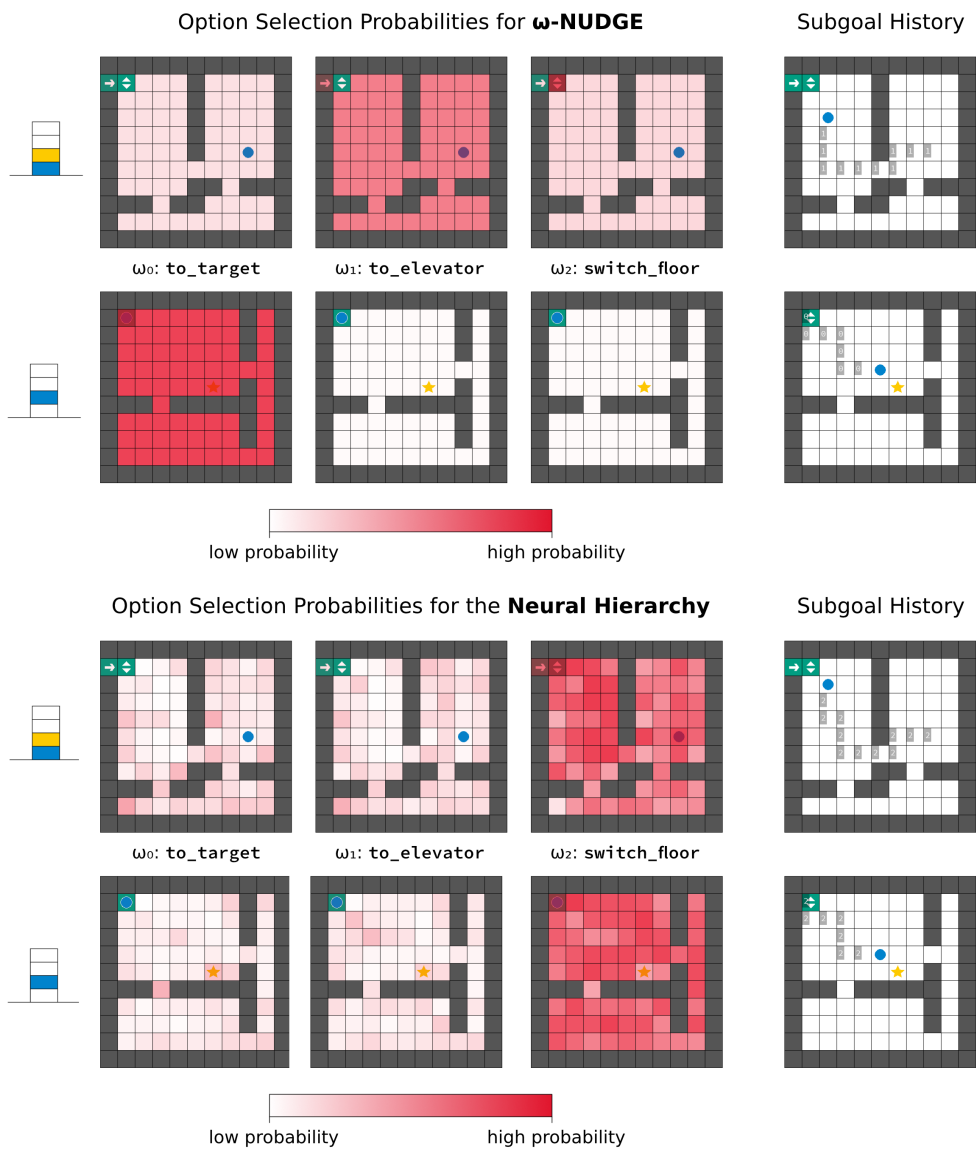


Figure 4.6.: Option selection probabilities visualized as heatmaps. Strong color indicates high probability that the corresponding option is chosen by π_{meta} when the agent is on that field. The rightmost column shows the history of chosen options, i.e., subgoals. The building pictograms on the left show the current floor level.

Generally, even if the neural hierarchy had learned meaningful options, the option semantics are unknown until after training where a human investigator needs to analyze the options and identify the corresponding subtask. This task of option labeling becomes the harder the more diffuse the action probability heatmaps are.

Figure 4.7 shows the activity share of each option w.r.t. the total execution time. Surprisingly, for ω -NUDGE, ω_0 (going to the target) is invoked much more often than ω_1 (going to the elevator). One would expect both activity shares to be roughly equal (ω_0 with a slightly larger share since there is a chance of 1/4 that the agent starts from the target floor). This expectation is matched well by ω -NUDGE without pretraining. In other words, using pre-trained options lead to an overestimation of ω_0 , maybe because it is the option which safely reaches the goal. This observation needs further investigation.

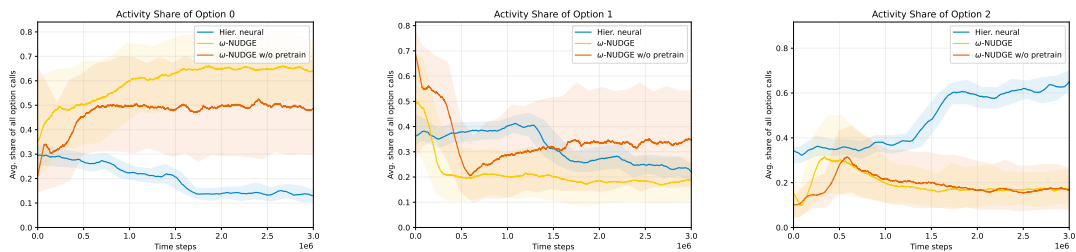


Figure 4.7.: The activity shares of the three options for all three hierarchical models.

4.3.5. Termination Function Training is Difficult

When running the experiments, it turned out to be difficult to achieve a satisfying option execution length: Typically, option execution falls too short, i.e., the learned terminators are too restrictive, see also Figure 4.8 as well as Figure A.1 in the appendix. More effort in exploring a suitable terminator regularizer ξ and a terminator entropy coefficient is required.

A hypothetical reason for the short execution length could be that π_{meta} has too high entropy. That is, π_{meta} invokes the options too often on out-of-distribution data leading to a bad overall performance of the agent, incentivizing the termination function to let options terminate early so that π_{meta} can correct its decision.

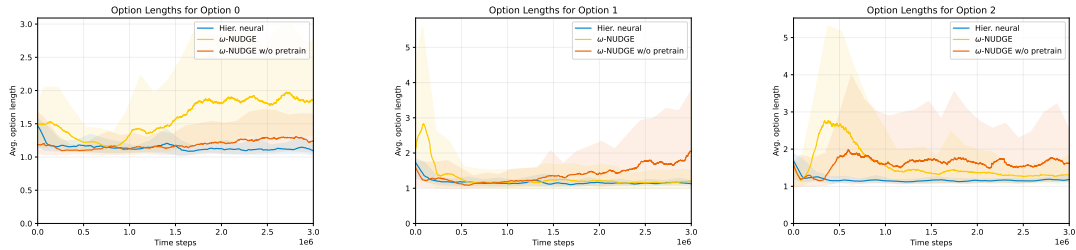


Figure 4.8.: The execution lengths of the three options for all three hierarchical models.

4.4. Side Contributions

In preparation for the upcoming experiments on OCArari with SCoBots, a Graphical User Interface (GUI) was implemented to support the manual RAM information extraction for Atari games. The tool is called *RAM Extraction Method GUI (REM GUI)*. It enables the user to interactively explore the RAM of the Atari game state and find correlations between objects and RAM entries. See Appendix A.4 for a screenshot and a brief documentation. Using this tool, the RAM extraction method for the games Pong, Seaquest, and Kangaroo was fully reworked, fixing any incorrect object detection and completing the detection by previously missed objects.

Additionally, an interface for aggregation functions in SCoBots was implemented. Aggregation functions enable the user to add, for example, total object counts, to the concept bottleneck. Experiments on Pong, Seaquest, and Kangaroo showed that normalizing the concept bottleneck values is vital to the learning success of the agent. Furthermore, it was found that there are use cases like in Seaquest where freezing the properties of disappeared objects is detrimental to the agent’s learned behavior. Instead, setting the properties of invisible objects to zero turned out to produce better results on Seaquest. Therefore, the ability to perform observation normalization and deactivate object freezes was implemented. All the aforementioned improvements enabled the successful training of flat neural agents that achieve remarkable performance on both, Kangaroo and Seaquest, used as results in the SCoBots publication. These agents will serve as a baseline for the upcoming ω -NUDGE experiments.

Furthermore, an opportunity to increase the training speed of NUDGE was discovered. Previously, the FOL rules compared predicates against any game object, leading to a memory complexity of $O(n^k)$ where n is the number of constants and k is the maximum

number of variables that a predicate accepts in the rule set. The number of constants correlates with the number of game objects. A high number of game objects quickly lead to GPU out-of-memory errors. As a solution, the feature was introduced to replace variables with constants wherever applicable. This way, the maximum number k of variables inside a predicate can be effectively reduced to 1, resulting in a memory complexity linear in the number of game objects instead of polynomial. Consequently, the out-of-memory errors vanished and also the training speed of NUDGE increased greatly by a factor of 5 to 10, depending on the use case.

5. Related Work

5.1. Hierarchical RL with Logic

The approach most similar to this thesis was published by Araki et al. (2021). The authors propose the *Logical Options Framework (LOF)* which combines Linear Temporal Logic (LTL) (Clarke and Schlingloff, 1996) with options. The resulting model has a hierarchy of two levels: The meta policy in the upper level is a finite state machine, specified through LTL. Each option in the lower level solves a subgoal, explicitly specified by the user through LTL formulae. Araki et al. argue that they are the first who created a framework that, under certain conditions, learns policies that meet the three properties satisfaction, optimality, and composability. *Satisfaction* means that the user-specified LTL rules are all satisfied by the trained agent, always. *Optimality* is reached when the agent maximizes the expected cumulative reward. Optimality can be guaranteed only if observations are complete and the environment is discrete. Finally, an agent is *composable* if it is modular and the modules can be rearranged and reused on new tasks. There are several important differences between their and this work. First, the meta policy is represented by LTL formulae that can be converted to a finite state machine. This is different to ω -NUDGE the meta policy π_{meta} of which is learned with NUDGE through neural guidance, resulting in a logic program, consisting of FOL rules. Second, LOF learns each option individually and independently, requiring a clearly specified subgoal per option. Subgoals are given a priori by the user through LTL formulae. ω -NUDGE can also be used to automatically discover subgoals. Third, an LOF option terminates if and only if it reaches the subgoal. In particular, the termination function is given and not learned as in ω -NUDGE. Fourth, LOF is defined only for two-level hierarchy models whereas ω -NUDGE (and the implementation) support arbitrarily many levels. Finally, and the most crucial difference, LOF was primarily introduced with the intention to meet the three aforementioned model properties. The focus of LOF was not to improve model interpretability or explainability nor was it evaluated regarding these properties. The restriction of LOF to a two-level hierarchy

limits the ability of temporal abstraction significantly at the cost of interpretability for RL problems that have goal hierarchies larger than two levels.

Later, the same authors propose *Hierarchical Inference with Logical Options (HILO)* to learn a hierarchical model through expert imitation. (Araki et al., 2022) Similar to above, LTL subgoals are provided by the user as LTL formulae and readily-trained options are given. Additionally, a set of expert trajectories is given as input to HILO which then learns a distribution over LTL formulae, inducing a distribution over policies that best imitates the expert.

Xu and Fekri (2021) present a hierarchy model with a neural high-level agent, a neural subtask solver, and a logic-based transition model. The high-level agent selects subgoals and submits them to the subtask solver. The transition model learns via inductive logic programming the rules to model the MDP transition function. In contrast, ω -NUDGE has a logic meta policy and forgoes any transition model.

5.2. Symbolic Planning with Hierarchies

A series of publications combines classical planning techniques with HRL. (Jin et al., 2022; Lyu et al., 2018; Illanes et al., 2019; Illanes et al., 2020; Achterhold et al., 2022; Konidaris, 2019; Prakash et al., 2022) They follow a similar principle: From a symbolic state representation, a high-level planner generates a symbolic action plan, i.e., a series of subgoals, which is then executed by lower level policies. Planning is applied in environments where the transition function is known. ω -NUDGE does not rely on that assumption and can be applied also to environments with unknown dynamics.

One of the most interesting works in the symbolic planning HRL domain comes from Lyu et al. (2018). They use a meta controller that defines goals for the sub-policies: It modifies the reward signal to align with a new, "intrinsic" goal the respective sub-policy is supposed to reach. The meta controller also evaluates them. For planning, they use special action description language. The most critical drawback is that their model is not composable. (Araki et al., 2021) ω -NUDGE supports composability inherently due to the modularity of options.

Another noteworthy symbolic planning framework was introduced by Illanes et al. (2019) and Illanes et al. (2020) that incorporates action models that do both, symbolic state and temporal abstraction. They propose to explicitly not communicate subgoals to the lower-level policies but instead high-level plans that guide the subgoals. Rewards are

generated via reward machines which are finite-state machines that specify temporally extended reward signals. Moreover, terminators are pre-defined unlike in ω -NUDGE. In particular, also their framework is not composable. (Araki et al., 2021)

5.3. Interpretable Neural HRL

Interpretable HRL methods seek for goal-based interpretability, i.e., where explanations for actions don't rely on input features but on the currently pursued subgoal, depending on the meta policy's choice. Several such neural frameworks were proposed. (Shu et al., 2017; Bonaert et al., 2021; Munoz et al., 2022; Rietz et al., 2022) Since the models are neural, in particular not intrinsically explainable, sometimes some extra post-hoc explanation methods are applied. As an instance, Beyret et al. (2019) explain their model simply via Q -values drawn from the (single and only) sub-policy w.r.t. a given subgoal. Another example is the work of Munoz et al. (2022) who propose a memory-based explanation method which uses statistics from the agent's past experience to infer success probabilities of subgoals and, hence, justify the agent's behavior. Other work even introduced the ability for the meta policy to decide when to learn and add a new sub-policy. (Shu et al., 2017) Finally, Rietz et al. (2022) are the first to combine HRL subgoal context with one-step explanations. That is, they don't use the mere current subgoal as an explanation but rather as an additional context for one-step explanations. They couple it with reward decomposition.

6. Discussion

Overall, the experiments confirmed the expectation that the interpretability of NUDGE can be improved via options. Additionally, also the performance was successfully increased. Nevertheless, ω -NUDGE also has its disadvantages, more discussed in this chapter. Future directions including will be pointed out, followed by a short ethical assessment.

6.1. ω -NUDGE versus NUDGE versus Neural Hierarchies

In contrast to NUDGE which uses a neural model to only guide the learning of the symbolic policy, ω -NUDGE combines a symbolic policy with neural sub-policies and can be, therefore, considered as a neuro-symbolic AI (Sarker et al., 2021). As other neuro-symbolic AI models, NUDGE and ω -NUDGE want to combine the advantages of both neural and symbolic approaches. ω -NUDGE uses neural components to a higher degree than NUDGE. Naturally, a better flexibility of ω -NUDGE was observed. In fact, ω -NUDGE decouples flexibility from high-level interpretability, loosening the trade-off between interpretability and flexibility. Of course, low-level interpretability remains a limiting factor for the overall interpretability of ω -NUDGE. Fortunately, typically, we are more interested in the high-level behavior of an agent rather than its fine-grained, one-step tactics.

Generally, pre-training should be used whenever possible. Due to its composability, ω -NUDGE allows to integrate readily trained sub-policies or even specific algorithms like, for example, path finding solutions. This is not possible with NUDGE alone due to its atomic nature.

HRL practitioners demonstrated that HRL models are capable to overcome the credit assignment problem in long-horizon, sparse-reward tasks and to improve exploration. Although not tested on these criteria, the scientific evidence gives good reason to expect

that ω -NUDGE can cope with such problems since ω -NUDGE takes full advantage of the HRL paradigm.

One more important drawback of NUDGE solved by ω -NUDGE is the missing applicability of NUDGE to environments with continuous action spaces. ω -NUDGE can use any arbitrary, even a heterogeneous choice of architectures as options. In particular, ω -NUDGE applies to any kind of action space as long as there exists any model architecture that can deal with it. In this context, options can be seen as a means to transform a continuous action space into a discrete one for the logic meta policy.

NUDGE and ω -NUDGE both allow the ML expert to include prior knowledge into the (meta) policy. This is a crucial advantage that neural hierarchy models do not have. As seen in the experiments, such prior knowledge accelerates learning progress and increases the chances to get meaningful options. In fact it was proven that the automatic discovery of subgoals for a limited number of time steps is NP-hard. (Jinnai et al., 2018) With ω -NUDGE the user can evade the difficulty of automatically discovering subgoals by providing proper rules that drive the options to quickly learn the subgoals.

Arguably, higher-level concepts are more easily encodeable into logic state representations than direct, environmental perceptions. This might be an additional, decisive factor why high-level logic policies have great chances to be much smaller than flat logic policies.

A summary of the discussed advantages and disadvantages can be found in Table 6.1.

	NUDGE	ω -NUDGE	Neural hierarchy
High-level interpretability	✓	✓✓	-
Low-level interpretability	✓	-	✗
Flexibility	-	✓	✓
Composable	-	✓	✓
Low sample complexity	-	✗	✗
Suited for sparse-reward problems	✗	✓	✓
Accepts continuous action space	✗	✓	✓
Takes prior knowledge (through FOL)	✓	✓	✗
Training stability	✓	-	-

Table 6.1.: Summarized comparison between the original NUDGE, the proposed ω -NUDGE, and a purely neural hierarchy.

We conclude this section with discussing the following question: Does the core idea of NUDGE get invalidated when learning it jointly with untrained neural options? The idea is that a neural model guides the differential logic training process without actually performing actions. This principle still applies in ω -NUDGE even when the lower-level policies are neural—the NUDGE principle is only applied on higher, temporally abstracted layer. It is irrelevant what kind of architecture is used in the lower-level options as they do not actively interfere with the NUDGE learning process. The only difference is that the NUDGE meta policy has an action space consisting of options rather than the actual environment action. However, since the effect of the options can change at training time when learned jointly, NUDGE needs to adapt to the change, likely increasing training time—but not fundamentally undermining NUDGE’s core idea.

6.2. Limitations

Of course, ω -NUDGE also has flaws, some of which are inherent and others which can be addressed through future work. The probably biggest issue of ω -NUDGE is an overall lack of training stability: The most crucial failure case is the option collapse mode where a single or only a few options solve the entire RL problem. In this case, one option can pursue a large number of different subgoals. There is no possibility anymore to differentiate between the subgoals and the meta policy becomes redundant. Another factor in the model training stability are the termination functions which are not guaranteed to learn reasonable termination probabilities. Although this can be influenced by the termination regularizer, increasing this hyperparameter lead to extremely long option executions. The effect of this hyperparameter is strong and needs further investigation. Generally, the dimension of the hyperparameter space is huge since, in theory, the meta policy and each option can all have individual hyperparameters. Consequently, hyperparameter search is costly to conduct. At least, the user can cope with this situation by using pretrained options.

One other drawback of ω -NUDGE is that it only accepts a discrete option space. However, the nature of real-world problems is typically continuous and high-dimensional. (Hutsebaut-Buyse et al., 2022) Also subgoals can be continuous, think of shooting a football at a goal which requires the specification of kick angle and intensity. Learning parameterized options is an open field of research, but some solutions exist. (Silva et al., 2012)

Furthermore, option training inevitably requires more computational resources. Multiple policies need to be trained at once. Due to the heterogeneous nature of hierarchical models, the advantages of parallelization cannot be used as efficiently as, e.g., in deep NNs. In the experiments of this work, flat neural policies were trained with an average throughput of about 1800 transitions per second whereas neural hierarchical models learned only about 800 samples during the same time period on the same hardware.

The experiments revealed that the option execution time is very short. From a mathematical perspective, short options are suboptimal: Frequent option termination increases the necessity to bootstrap value functions, introducing more bias. Additionally, short options are bad for the explainability of the model. More specifically, a user wants to trust the subgoal provided by the agent, but the communicated subgoal is not trustworthy if it can change right in the next step with high probability. Instead, in order to align with the user’s expectations, the model should learn to “commit” to subgoals, i.e., have longer option execution times. To this end, terminator hyperparameters need to be tuned in future experiments.

6.3. Future Directions

In this work, ω -NUDGE was only evaluated on MEETINGROOM. More experimental evaluation is needed and will be conducted on other environments, including Seaquest and Kangaroo. Moreover, the environments “Montezuma’s Revenge”, “Pitfall”, and “Private Eye” are considered suitable HRL benchmarks (Hutsebaut-Buysse et al., 2022) and should be, therefore, included as well. To this end, RAM extraction in OCArari needs to be prepared for this.

The current ω -NUDGE implementation is sample-inefficient. The main reason is that policies are only trained on transitions where they actively made a decision. Instead, in a first step, all policies that are part of the invocation trace can use the experienced transitions for learning—even if the policy did only wait until the lower-level option is finished executing. It can be assumed as if the waiting policy *had actually chosen* the currently active lower-level option. This would drastically increase the available training data in cases where options are executed for a long number of transitions. In a second step, as proposed by Sutton et al. (1999), similar options from the same level that pursue similar tasks could reuse the same experience from other options, allowing even the training of options for time steps where they were not part of the invocation trace.

Another interesting direction is the introduction of reward decomposition (Juozapaitis et al., 2019) where the reward signal is split up according to the subgoals. A decomposed reward signal is more interpretable to the human as each entry in the reward vector has a specific meaning. Moreover, the options can be trained to explicitly only follow the reward of the respective subgoal, incentivizing stronger task-specialization.

One further improvement of ω -NUDGE could be achieved through the introduction of attention. It requires only a simple extension to the already existing Attention Option-Critic (Chunduru and Precup, 2022). Attention fosters specialization of options and adds another property serving for interpreting neural options.


Another opportunity lies in the benefit of making lower-level options available to high-level policies beyond the next higher level. This enables high-level policies to access low-level actions in cases where needed.

Option *interruption* (Sutton et al., 1999) is the concept where an option does not only decide on its own when to terminate but also where an external policy can abort the execution of an option. It was formally shown by Sutton et al. that interruption is beneficial in situations where other options become more valuable during the execution of one option. This would require a redefinition of the termination mechanism. An alternative definition is proposed in Appendix A.1 in the context of the GAEO.

6.4. Ethical Implications

On a final note, since the development of AI tools comes with foreseeably huge societal impact, an ethical assessment of the proposed technology falls in the responsibility of its inventor. ω -NUDGE is a general, foundational framework with no pre-defined use cases. However, it can be further developed to solve specific real-world tasks. The interpretability of ω -NUDGE is supposed to be a contribution to more trustworthy AI so that human users can understand and better interact with AI models. Nevertheless, better interpretability also means better ability to intervene in the model's decision process, enabling attackers to dictate AI models their own rules.

Since complex real-world problems require hierarchical models, ω -NUDGE is also a step towards real-world application. Modularity and interpretability of ω -NUDGE also help to generally further optimize models, making their deployment to the real-world more attractive. While this work strongly encourages to strive towards highly-capable AI models



that are applied in the real world, there is always the abstract danger of dual use, i.e., that an ML practitioner—intentionally or not—can apply AI to do harm.

7. Conclusion

This work introduced ω -NUDGE, a neuro-symbolic policy framework which extends the Neurally gUided Differentiable loGic policiEs (NUDGE) framework to hierarchical models that perform temporal abstraction. It incorporates the option framework as well as the option-critic, combined with Proximal Policy Optimization (PPO). By means of the newly proposed environment MEETINGROOM, it was empirically shown that ω -NUDGE improves high-level interpretability *and* performance of NUDGE significantly. Moreover, the experiments also indicated that the rules provided to ω -NUDGE can guide learning well when the meta policy and the sub-policies are trained jointly.

The rather limited experiments, however, only cover the tip of an iceberg and more thorough investigation on other environments is needed. The most important limitation of meaningless termination functions is an open issue that should be addressed in future work as well.

Acronyms

AI Artificial Intelligence.

ALE Arcade Learning Environment.

FOL First-Order Logic.

GAE Generalized Advantage Estimator.

GAEO Generalized Advantage Estimator for Options.

HRL Hierarchical Reinforcement Learning.

MDP Markov Decision Process.

ML Machine Learning.

NN Neural Network.

NUDGE Neurally gUided Differentiable loGic policiEs.

OCAtari Object-Centric Atari.

OCOC Object-Centric Option-Critic.

PPO Proximal Policy Optimization.

RL Reinforcement Learning.

SCoBots Successive Concept Bottleneck Agents.



SMDP Semi-Markov Decision Process.

TD Temporal Difference.

XAI Explainable AI.

XRL Explainable RL.

Bibliography

- Abel, David (2022). “A Theory of Abstraction in Reinforcement Learning”. In: *ArXiv abs/2203.00397*.
- Achterhold, Jan, Markus Krimmel, and Joerg Stueckler (2022). “Learning Temporally Extended Skills in Continuous Domains as Symbolic Actions for Planning”. In: *ArXiv abs/2207.05018*.
- Alharin, Alnour, Thanh-Nam Doan, and Mina Sartipi (2020). “Reinforcement Learning Interpretation Methods: A Survey”. In: *IEEE Access* 8, pp. 171058–171077.
- Araki, Brandon, Jeana Choi, Lillian Chin, Xiao Li, and Daniela Rus (2022). “Learning Policies by Learning Rules”. In: *IEEE Robotics and Automation Letters* 7, pp. 1284–1291.
- Araki, Brandon, Xiao Li, Kiran Vodrahalli, Jonathan A. DeCastro, Micah J. Fry, and Daniela Rus (2021). “The Logical Options Framework”. In: *ArXiv abs/2102.12571*.
- Arrieta, Alejandro Barredo et al. (2019). “Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI”. In: *ArXiv abs/1910.10045*.
- Bacon, Pierre-Luc, Jean Harb, and Doina Precup (2016). “The Option-Critic Architecture”. In: *ArXiv abs/1609.05140*.
- Bellemare, Marc G., Yavar Naddaf, Joel Veness, and Michael Bowling (2012). “The Arcade Learning Environment: An Evaluation Platform for General Agents (Extended Abstract)”. In: *International Joint Conference on Artificial Intelligence*.
- Berner, Christopher et al. (2019). “Dota 2 with Large Scale Deep Reinforcement Learning”. In: *ArXiv abs/1912.06680*.
- Beyret, Benjamin, Ali Shafti, and A. Faisal (2019). “Dot-to-Dot: Explainable Hierarchical Reinforcement Learning for Robotic Manipulation”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5014–5019.
- Bonaert, Gregory, Youri Coppens, Denis Steckelmacher, and Ann Nowe (2021). *Explainable Reinforcement Learning Through Goal-Based Interpretability*.
- Casper, Stephen et al. (2023). “Open Problems and Fundamental Limitations of Reinforcement Learning from Human Feedback”. In: *ArXiv abs/2307.15217*.

-
- Chunduru, Raviteja and Doina Precup (2022). “Attention Option-Critic”. In: *ArXiv* abs/2201.02628.
- Clarke, Edmund M. and Holger Schlingloff (1996). “Model checking”. In: *Communications of the ACM* 52, pp. 74–84.
- Dazeley, Richard, Peter Vamplew, and Francisco Cruz (2022). “Explainable reinforcement learning for broad-XAI: a conceptual framework and survey”. In: *Neural Computing and Applications* 35, pp. 16893–16916.
- Dazeley, Richard, Peter Vamplew, Cameron Foale, Charlotte Young, Sunil Aryal, and Francisco M. Torres Cruz (2021). “Levels of explainable artificial intelligence for human-aligned conversational explanations”. In: *Artif. Intell.* 299, p. 103525.
- Delfosse, Quentin, Jannis Blüml, Bjarne Gregori, Sebastian Sztwiertnia, and Kristian Kersting (2023a). *OCArari: Object-Centric Atari 2600 Reinforcement Learning Environments*.
- Delfosse, Quentin, Hikaru Shindo, Devendra Singh Dhama, and Kristian Kersting (2023b). “Interpretable and Explainable Logical Policies via Neurally Guided Symbolic Abstraction”. In: *ArXiv* abs/2306.01439.
- Delfosse, Quentin, Sebastian Sztwiertnia, Wolfgang Stammer, Mark Rothermel, and Kristian Kersting (2023c). *Interpretable Concept Bottlenecks to Align Reinforcement Learning Agents*.
- Doshi-Velez, Finale and Been Kim (2017). “Towards A Rigorous Science of Interpretable Machine Learning”. In: *arXiv: Machine Learning*.
- Glanois, Claire, P. Weng, Matthieu Zimmer, Dong Li, Tianpei Yang, Jianye Hao, and Wulong Liu (2021). “A Survey on Interpretable Reinforcement Learning”. In: *ArXiv* abs/2112.13112.
- Goodman, Bryce and Seth Flaxman (2016). “European Union Regulations on Algorithmic Decision-Making and a ”Right to Explanation””. In: *AI Mag.* 38, pp. 50–57.
- Heuillet, Alexandre, Fabien Couthouis, and Natalia Díaz Rodríguez (2020). “Explainability in Deep Reinforcement Learning”. In: *Knowl. Based Syst.* 214, p. 106685.
- Hickling, Tom, Abdelhafid Zenati, Nabil Aouf, and Phillippa Spencer (2022). “Explainability in Deep Reinforcement Learning, a Review into Current Methods and Applications”. In: *ArXiv* abs/2207.01911.
- Hutsebaut-Buysse, Matthias, Kevin Mets, and Steven Latré (2022). “Hierarchical Reinforcement Learning: A Survey and Open Research Challenges”. In: *Mach. Learn. Knowl. Extr.* 4, pp. 172–221.
- Illanes, León, Xi Yan, Rodrigo Toro Icarte, and Sheila A. McIlraith (2019). “Leveraging Symbolic Planning Models in Hierarchical Reinforcement Learning”. In: – (2020). “Symbolic Plans as High-Level Instructions for Reinforcement Learning”. In: *International Conference on Automated Planning and Scheduling*.

-
- Jin, Muyang, Zhihao Ma, Kebin Jin, Hankz Hankui Zhuo, Chen Chen, and Chao Yu (2022). “Creativity of AI: Automatic Symbolic Option Discovery for Facilitating Deep Reinforcement Learning”. In: *AAAI Conference on Artificial Intelligence*.
- Jinnai, Yuu, David Abel, Michael L. Littman, and George Dimitri Konidaris (2018). “Finding Options that Minimize Planning Time”. In: *ArXiv* abs/1810.07311.
- Jong, Nicholas K., Todd Hester, and Peter Stone (2008). “The utility of temporal abstraction in reinforcement learning”. In: *Adaptive Agents and Multi-Agent Systems*.
- Juozapaitis, Zoe, Anurag Koul, Alan Fern, Martin Erwig, and Finale Doshi-Velez (2019). “Explainable Reinforcement Learning via Reward Decomposition”. In: *AAAI Conference on Artificial Intelligence*.
- Koch, Jack, Lauro Langosco di Langosco, Jacob Pfau, James Le, and Lee D. Sharkey (2021). “Objective Robustness in Deep Reinforcement Learning”. In: *ArXiv* abs/2105.14111.
- Konidaris, George Dimitri (2019). “On the necessity of abstraction”. In: *Current Opinion in Behavioral Sciences* 29, pp. 1–7.
- Konidaris, George Dimitri and Andrew G. Barto (2009). “Efficient skill learning using abstraction selection”. In: *International Joint Conference on Artificial Intelligence*.
- Krajna, Agneza, Mario Brčić, Tomislav Lipić, and Juraj Doncevic (2022). “Explainability in reinforcement learning: perspective and position”. In: *ArXiv* abs/2203.11547.
- Lin, Kaixiang, Renyu Zhao, Zhe Xu, and Jiayu Zhou (2018). “Efficient Large-Scale Fleet Management via Multi-Agent Deep Reinforcement Learning”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.
- Lyu, Daoming, Fangkai Yang, Bo Liu, and Steven M. Gustafson (2018). “SDRL: Interpretable and Data-efficient Deep Reinforcement Learning Leveraging Symbolic Planning”. In: *ArXiv* abs/1811.00090.
- Mahmood, Ashique Rupam, Dmytro Korenkevych, Gautham Vasan, William Ma, and James Bergstra (2018). “Benchmarking Reinforcement Learning Algorithms on Real-World Robots”. In: *ArXiv* abs/1809.07731.
- Milani, Stephanie, Nicholay Topin, Manuela M. Veloso, and Fei Fang (2022). “A Survey of Explainable Reinforcement Learning”. In: *ArXiv* abs/2202.08434.
- Miller, Tim (2017). “Explanation in Artificial Intelligence: Insights from the Social Sciences”. In: *Artif. Intell.* 267, pp. 1–38.
- Molnar, Christoph, Giuseppe Casalicchio, and B. Bischl (2020). “Interpretable Machine Learning - A Brief History, State-of-the-Art and Challenges”. In: *PKDD/ECML Workshops*.
- Munoz, Hugo, Ernesto Portugal, Angel Ayala, Bruno Fernandes, and Francisco Cruz (2022). “Explaining Agent’s Decision-making in a Hierarchical Reinforcement Learning Scenario”. In: *2022 41st International Conference of the Chilean Computer Science Society (SCCC)*, pp. 1–8.

-
- Prakash, Bharat, Nicholas R. Waytowich, Tim Oates, and Tinoosh Mohsenin (2022). “Towards an Interpretable Hierarchical Agent Framework using Semantic Goals”. In: *ArXiv* abs/2210.08412.
- Puiutta, Erika and Eric M. S. P. Veith (2020). “Explainable Reinforcement Learning: A Survey”. In: *ArXiv* abs/2005.06247.
- Qing, Yunpeng, Shunyu Liu, Jie Song, and Mingli Song (2022). “A Survey on Explainable Reinforcement Learning: Concepts, Algorithms, Challenges”. In: *ArXiv* abs/2211.06665.
- Rietz, Finn, Sven Magg, Fredrik Heintz, Todor Stoyanov, Stefan Wermter, and Johannes Andreas Stork (2022). “Hierarchical goals contextualize local reward decomposition explanations”. In: *Neural Computing and Applications* 35, pp. 16693–16704.
- Sarker, Md Kamruzzaman, Lu Zhou, Aaron Eberhart, and Pascal Hitzler (2021). “Neuro-Symbolic Artificial Intelligence: Current Trends”. In: *AI Commun.* 34, pp. 197–209.
- Schulman, John, Philipp Moritz, Sergey Levine, Michael I. Jordan, and P. Abbeel (2015). “High-Dimensional Continuous Control Using Generalized Advantage Estimation”. In: *CoRR* abs/1506.02438.
- Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). “Proximal Policy Optimization Algorithms”. In: *ArXiv* abs/1707.06347.
- Shu, Tianmin, Caiming Xiong, and Richard Socher (2017). “Hierarchical and Interpretable Skill Acquisition in Multi-task Reinforcement Learning”. In: *ArXiv* abs/1712.07294.
- Silva, Bruno C. da, George Dimitri Konidaris, and Andrew G. Barto (2012). “Learning Parameterized Skills”. In: *International Conference on Machine Learning*.
- Silver, David et al. (2016). “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529, pp. 484–489.
- Sutton, Richard S. and Andrew G. Barto (2017). *Reinforcement Learning: An Introduction*. 2nd ed. MIT Press.
- Sutton, Richard S., Doina Precup, and Satinder Singh (1999). “Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning”. In: *Artif. Intell.* 112, pp. 181–211.
- Vaswani, Ashish, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin (2017). “Attention is All you Need”. In: *Neural Information Processing Systems*.
- Vouros, George A. (2022). “Explainable Deep Reinforcement Learning: State of the Art and Challenges”. In: *ACM Computing Surveys* 55, pp. 1–39.
- Wells, Lindsay and Tomasz Bednarz (2021). “Explainable AI and Reinforcement Learning—A Systematic Review of Current Approaches and Trends”. In: *Frontiers in Artificial Intelligence* 4.
- Xu, Duo and Faramarz Fekri (2021). “Interpretable Model-based Hierarchical Reinforcement Learning using Inductive Logic Programming”. In: *ArXiv* abs/2106.11417.

-
- You, Yurong, Xinlei Pan, Ziyang Wang, and Cewu Lu (2017). “Virtual to Real Reinforcement Learning for Autonomous Driving”. In: *ArXiv* abs/1704.03952.
- Zhang, Jing, Bowen Hao, Bo Chen, Cuiping Li, Hong Chen, and Jimeng Sun (2019). “Hierarchical Reinforcement Learning for Course Recommendation in MOOCs”. In: *AAAI Conference on Artificial Intelligence*.

A. Appendix

A.1. Pitfalls of the GAEO

The vanilla definition of GAE doesn't apply to options, see Section 3.3. Therefore, the Generalized Advantage Estimator for Options (GAEO) was introduced. It uses the value upon arrival U^ω as a surrogate for the value V^ω . Although having a sound intuition, this section shows a major drawback of that definition.

To begin with, note that the GAEO (like the GAE) exhibits a recursive relation

$$\hat{A}_t^{\text{GAEO}} = \delta_t^\omega + \gamma \lambda \hat{A}_{t+1}^{\text{GAEO}}. \quad (\text{A.1})$$

This relation helps to understand the intuition behind the estimator and, besides, it is used to efficiently and iteratively compute the GAE of an entire trajectory. Recall that

$$\delta_t^\omega = r_t + \gamma U_{t+1}^\omega - V_t^\omega$$

and

$$U_t^\omega = \beta_t^\omega V_t^{\hat{\omega}} + (1 - \beta_t) V_t^\omega,$$

where $\hat{\omega} \rightarrow \omega$. In contrast to \hat{A}_t^{GAE} which uses one single value estimator V_t for time step t , \hat{A}_t^{GAEO} depends on the two estimators $V_t^{\hat{\omega}}$ and V_t^ω . First, this introduces quite some bias: The target of the option's value function receives a signal not only from itself but also from the higher-level value function. The latter changes over time if not fixed, posing a concept drift for the lower-level value function and decreasing training stability crucially—in theory at least. The noise created by $V_t^{\hat{\omega}}$ may be magnitudes stronger than the actual reward signal by r_t , effectively disturbing training of V_t^ω .

Second, the GAEO may "explode." In order to see this, note that we cannot assume that $V_t^{\hat{\omega}}$ and V_t^ω follow similar statistics. In particular, the mean $\mathbb{E}_t[V_t^{\hat{\omega}}]$ can be significantly

different from $\mathbb{E}_t[V_t^\omega]$. In that case, the expected value upon arrival, $\mathbb{E}_t[U_t^\omega]$ will be also different from $\mathbb{E}_t[V_t^\omega]$. Hence, even with a normalized reward signal, the TD(0) error δ_t^ω then has a mean that deviates significantly from zero. Consequently, at each iteration (Equation A.1), an estimator with non-zero mean is added to the running advantage estimate. As a result, \hat{A}_t^{GAE0} diverges with decreasing t . The divergence increases linearly in the execution length of the option ω and is proportional to the difference $\mathbb{E}_t[V_t^\omega] - \mathbb{E}_t[V_t^{\hat{\omega}}]$. Since we have no guarantees on these two estimators, the difference can be arbitrarily large and cause the GAE0 to be an inadequate estimator.

There exist at least two possible solutions to this issue. However, due to the scope constraints of this work, the workaround $\lambda := 0$ was used, i.e., instead of the GAE0, the TD(0) error served as an advantage estimate. The possible solutions are left as future work and include:

- (1) One could find regularization methods that enforce the means of $V^{\hat{\omega}}$ and V^ω to be similar. This might, however, affect the expressiveness of these two value functions and hinder convergence to an optimal solution.
- (2) Instead of the value upon arrival, U_t^ω , the TD(0) error for options could use the normal value function V_t^ω for all time steps t . The resulting estimator is a mere canonical extension of GAE where all theoretical guarantees, as worked out by Schulman et al. (2015) and studied afterwards, apply. With this solution, V^ω doesn't incorporate higher-level value estimation anymore and estimates the value as if ω would never terminate. V^ω then solely depends on what ω sees during its execution. Generally, this may be a narrow selection of states in the MDP, especially if ω is specialized to solve a specific task. V^ω became a subjective value function that evaluates and learns the experience only in the temporally restricted context of ω . In particular, V^ω semantically describes a different valuation concept than $V^{\hat{\omega}}$. Consequently, it doesn't make sense to compare V^ω against $V^{\hat{\omega}}$ anymore, hence, the termination function β^ω loses any point of comparison and needs to be redefined. Instead of taking the value functions as criteria for termination, the higher-level actor probabilities could be used as a point of reference. For example, the termination probability of option ω is set to be high if the higher-level actor sees other options as more probably useful.

A.2. Learning a Reasonable Termination Function is Non-Trivial

The experiments showed that training a reasonable and meaningful termination function turns out to be difficult. Even slight changes in the termination regularizer ξ affect options to be either very short or very long. Figure A.1 shows the termination probabilities learned by ω -NUDGE. They do not reflect what one would expect when the respective options should terminate. However, the heatmaps share one noticeable common characteristic: The options tend to have low termination probabilities inside doorways. Maybe this is due to the fact that there is no point in choosing a new option since the new option would just continue to step through the doorway.

A.3. Implementation

As part of this work, the options framework with state-of-the-art training algorithms together with a NUDGE integration was implemented. The code is written in Python using `pytorch` and can be found publicly on GitHub under

<https://github.com/MaggiR/logic-options>.

Where applicable, already existing implementations of algorithms and model components were reused from the `stable-baselines3` (SB3) package. This includes PPO and the GAE computation which got adjusted to options, accordingly. The resulting implementation is, to the best of the author's knowledge, the first repository that offers RL training with agents that have an arbitrary option hierarchy. Especially, this repository is also the first that offers option training with PPO and GAE. It offers a wide range of experiment setup parameters, including observation, reward, and advantage normalization, pre-trained model loading, learning rate scheduling, etc.

A.3.1. Repository Structure

Termination Probabilities for ω -NUDGE

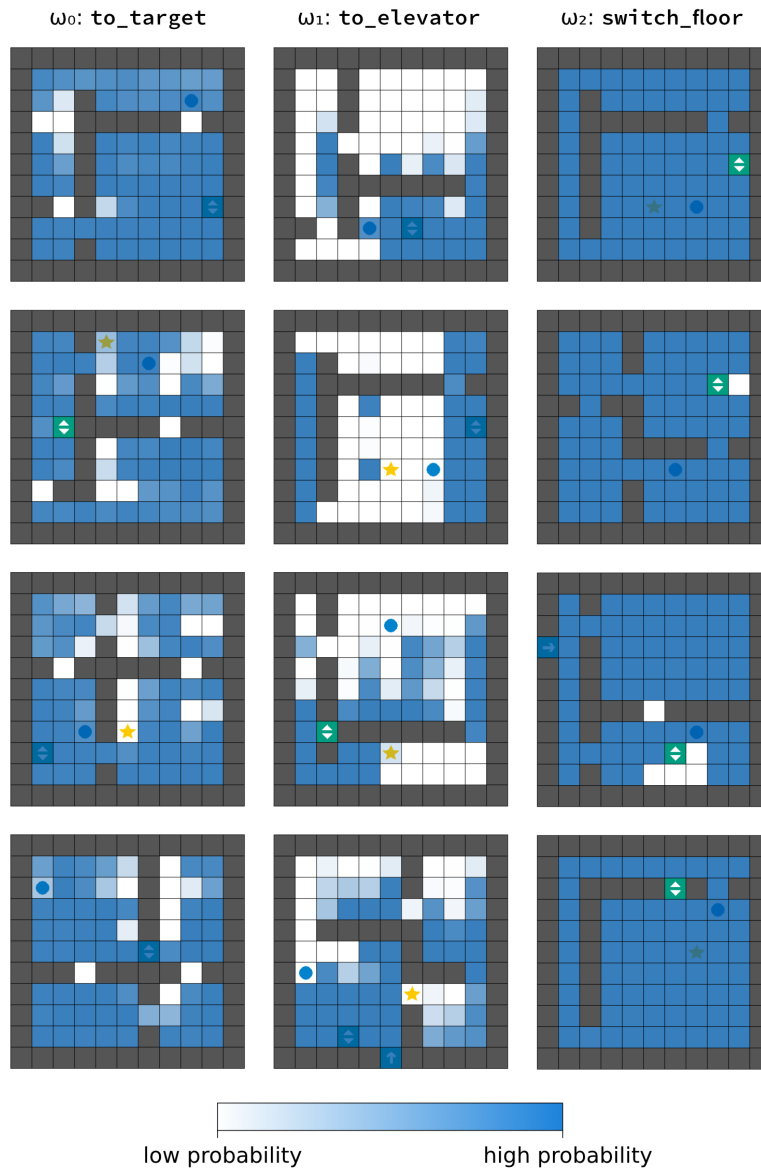


Figure A.1.: Termination heatmaps showing the termination probabilities for each free position on the gridworld. Each column shows the probabilities for the same option on four different examples.

The source code folder is divided into the following modules: The **options** module contains the implementation of the option hierarchy and individual options. Together with the meta policy, they are summarized in the `OptionsAgent` class which manages forwarding, value estimation, and predicate conversion. Also, the PPO algorithm (from Section 3.4) for both, actor-critics and terminators, can be found in this module. Finally, GAEO (from Section 3.3) is implemented in the `RolloutBuffer` class.

All NUDGE-related implementations are located in **logic**. The class `NudgePolicy` is the logic actor-critic where the `NSFReasoner` class from the NUDGE package is wrapped as an actor. Environments are wrapped, too, because predicates need to be translated into actions, next to logic state extraction from the observations so that the states can be read and evaluated by the environment-specific valuation functions saved in `valuation`. Furthermore, the user can specify FOL rules, atoms, and predicates inside `in/logic`.

In the **envs** module, the `MEETINGROOM` implementation is located, together with environment initialization helper functions. Any other auxiliary functions are included in **utils**, along with the agent evaluation function.

A.3.2. How to Use

Experiments are started via the `run.py` file. Each experiment requires a sufficiently specified YAML configuration, placed into `in/queue`. Exemplary configuration file including those used for the experiments can be found inside `in/config`. The user can observe trained models by running the `play.py` script where the respective model name and environment need to be specified. Finally, the `eval.py` script tests the model and measures return and episode lengths. With `train_continue.py`, the user can resume interrupted training processes.

```
in/  
- config/  
- logic/  
src/  
- envs/  
- logic/  
- options/  
- utils/  
eval.py  
play.py  
train.py  
train_continue.py
```

A.4. Graphical User Interface for the RAM Extraction Method in OCArari

As a side contribution, this work also ships with a Graphical User Interface (GUI) for improving RAM extraction in OCArari. This GUI is used to easily analyze and alter the RAM while manually playing Atari. Furthermore, it optionally renders an overlay of the detected objects, helping to verify if the RAM extraction is correct. A screenshot of the GUI is depicted in Figure A.2. The code is already included in the OCArari package. The below text is from the included documentation.



Figure A.2.: The GUI for OCArari RAM Extraction: On the left, the Atari game screen is rendered together with an overlay showing the extraction results. The right panel visualizes the entire game RAM as a grid where each cell shows the ID and the current value of a RAM cell.

Run the GUI via `scripts/rem_gui.py`. In the `__main__` method of `rem_gui.py`, you define the Atari game to run. Start the GUI by executing that script file. You play the Atari game by using the keys W, A, S, D, Space, and Esc. Pause/resume the game with P, reset it with R.

Investigating and manipulating the RAM is done as follows: By mouse-clicking any object on the game screen, all (presumably) relevant RAM cells for that object will be highlighted blue. You alter a RAM cell by clicking the respective cell (it will be highlighted yellow). You can then enter a new number (between 0 and 255), confirm with **Enter**. Changes become visible only if the game is resumed. Besides, rotating the mouse wheel in- or decrements the currently hovered cell's value.

If you click a pixel on the game screen, you execute a causative RAM analysis for that pixel. The analysis identifies all RAM cells that (individually) affect the appearance of the clicked pixel by altering the cell's value and comparing the resulting screen image with the original one.

A.5. State Abstraction with OCArari and SCoBots

Abstraction is key to interpretability. Leaving out overwhelming minutiae and aggregating information to more compact, symbolic knowledge makes a model, the processed concepts or an explanation more understandable to humans. For instance, humans inherently use abstractions in their language to elucidate complex concepts. (Miller, 2017) In fact, many XAI methods can be understood as abstractions. For example, feature importance methods rank input features based on their importance to the model's decision, simplifying the view of what matters the most. Techniques like LIME (Local Interpretable Model-agnostic Explanations) (Doshi-Velez and Kim, 2017) work by approximating complex models with simpler, interpretable models for specific instances. This acts as an abstraction layer that can be more readily understood by humans. Abstraction in RL is useful to drastically reduce the state space, i.e., the MDP, simplifying the RL problem and reducing sample complexity. Consequently, abstraction received large interest and various abstraction methods were proposed and studied. (Abel, 2022) In RL, abstraction can be done for states, actions, and both intertwined. This work prepares ω -NUDGE to be applied to the OCArari, an Atari game environment with object-centric state representations.

A.5.1. Seeing the Forest, not Just the Trees

In many visual applications, states are represented as RGB pixel matrices. Reasoning on such data, however, affords only limited potential for interpretability for two main reasons. First, high-dimensional data typically requires large models like convolutional neural nets

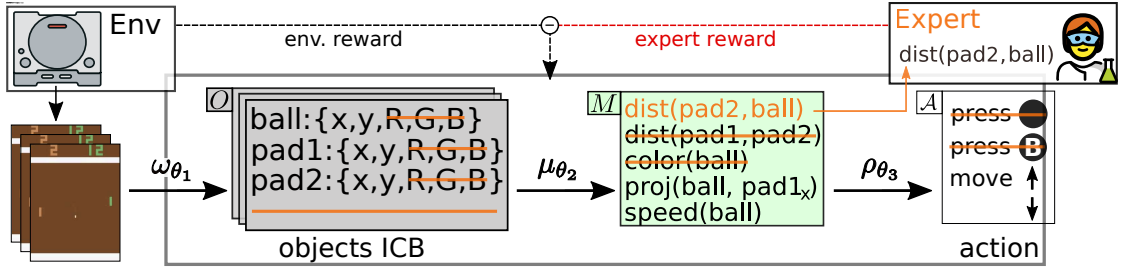


Figure A.3.: The idea of SCoBots: First, an object detector ω extracts objects and their properties from the raw environment observation. Second, a feature selector μ computes object relations. Finally, an action selector ρ decides upon the next action based on the relations. The expert prunes out unneeded objects or properties, defines relations, removes useless actions, and modifies the reward signal, making use of the relations.

to process–model size opposes interpretability. Second, symbolic reasoning, for example with First-Order Logic (FOL), requires symbolic state representation. Fortunately, most of the information in the high-dimensional input is redundant, allowing us to extract the relevant core aspects and represent it in a low-dimensional way. This *state abstraction* process is formally represented by a function $\varphi : \mathcal{S} \rightarrow \mathcal{S}_\varphi$ mapping the raw state $s \in \mathcal{S}$ onto a lower-dimensional state $s_\varphi \in \mathcal{S}_\varphi$. (Abel, 2022) Moreover, multiple raw states can have the same abstracted representation. For this reason, φ induces a new decision process \mathcal{M}_φ called *abstract MDP*. State abstraction is slightly different from the attention mechanism (Vaswani et al., 2017) in that state abstraction deterministically and explicitly maps the entire high-dimensional state space into a low-dimensional space while attention is a rather implicit, learned, and not necessarily interpretable information extraction function, also potentially depending on the internal model state.

OCAtari The Arcade Learning Environment (ALE) (Bellemare et al., 2012) is a set of Atari 2600 video games and serves as an entrenched RL benchmark, also in this work. Natively, the Atari game states are represented as RGB pixel matrices. Therefore, this work uses the Object-Centric Atari (OCAtari) (Delfosse et al., 2023a) environment which, instead of a pixel image, returns a list of objects with corresponding properties like position, size, color, speed, inherent value, etc. OCAtari reconstructs that symbolic state representation from the game RAM.

SCoBots Successive Concept Bottleneck Agents (SCoBots), an ongoing work by Delfosse et al. (2023c), is an interactive framework which translates high-dimensional, raw environment observations into low-dimensional relational representations, maintaining interpretability. The expert user defines the relations to use, for example, relative distance or speed between two objects, total object counts, etc. Furthermore, the expert prunes out redundant or unnecessary information and excludes unused actions from the action space. By successively adding more concept layers, the user can define even higher-level, abstract relations such as the averaged speed of a swarm of objects. See A.3 for a visual summary.

The low-dimensional, interpretable concept bottleneck improves overall model interpretability and explainability. Furthermore, the relational concept bottlenecks offer opportunities to the expert to guide model training, for example through reward shaping using the relations. SCoBots also help uncover misalignment in the agent, i.e., making the right actions for the wrong reasons. In this work, SCoBots serve as the state abstraction function for ALE experiments.